

CONSTRAINTS ON SIGNALING OF HYPOTHETICAL REFERENCE DECODER PARAMETERS IN VIDEO BITSTREAMS

CROSS-REFERENCE TO RELATED APPLICATIONS

[001] [001] Under the applicable patent law and/or rules pursuant to the Paris Convention, this application is made to timely claim the priority to and benefits of US Provisional Application No. 62/953,854, filed on December 26, 2019 and US Provisional Application No. 62/955,185, filed on December 30, 2019. For all purposes under the law, the entire disclosure of the aforementioned applications are incorporated by reference as part of the disclosure of this application.

TECHNICAL FIELD

[002] This patent document relates to image and video coding and decoding.

BACKGROUND

[003] Digital video accounts for the largest bandwidth use on the internet and other digital communication networks. As the number of connected user devices capable of receiving and displaying video increases, it is expected that the bandwidth demand for digital video usage will continue to grow.

SUMMARY

[004] The present document discloses techniques that can be used by video encoders and decoders to perform video encoding or decoding.

[005] In one example aspect, a video processing method is disclosed. The method includes performing a conversion between a video comprising one or more scalable video layers and a bitstream of the video. The video comprises one or more video pictures comprising one or more slices. The bitstream conforms to a format rule. The format rule specifies that a value of a field indicative of a slice type of a slice is set to indicate a type of intra slice in case that a corresponding network abstraction layer unit type is in a predetermined range and that a corresponding video layer flag indicates that a video layer corresponding to the slice does not use inter-layer prediction.

[006] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video comprising a plurality of video layers and a bitstream of the video, wherein the bitstream comprises a plurality of output layer sets (OLSs), each comprising one or more of the plurality of scalable video layers, and the bitstream conforms to a format rule, wherein the format rule specifies that, for an OLS having a single layer, a profile-tier-level (PTL) syntax structure that indicates a profile, a tier and a level for the OLS is included in a video parameter set for the bitstream, and the PTL syntax structure for the OLS is also included in a sequence parameter set coded in the bitstream.

[007] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video comprising a plurality of video layers and a bitstream of the video, wherein the bitstream comprises a plurality of output layer sets (OLSs), each of which comprises one or more of the plurality of video layers, and the bitstream conforms to a format rule, wherein the format rule specifies a relationship between occurrence of a number of profile-tier-level (PTL) syntax structures in a video parameter set for the bitstream and a byte alignment syntax field in the video parameter set; wherein each PTL syntax structure indicates a profile, a tier and a level of one or more of the plurality of OLS.

[008] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video comprising a plurality of scalable video layers and a bitstream of the video, wherein the bitstream comprises a plurality of output layer sets (OLSs), each comprising one or more of the plurality of scalable video layers, and the bitstream conforms to a format rule, wherein the format rule specifies that: during encoding, a syntax element indicative of an index to a syntax structure describing a profile, a tier and a level of one or more of the plurality of OLSs is excluded from a video parameter set for the bitstream in case that a value of the index is zero, or, during decoding, the value is inferred to be zero in case that the syntax element is not present in the bitstream.

[009] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video comprising a plurality of video layers and a bitstream of the video, wherein the bitstream comprises a plurality of output layer sets (OLSs), each of which comprises one or more of the plurality of video layers, and the bitstream conforms to a format rule, wherein the format rule specifies that, for a layer i , where i is an integer, the bitstream includes a

set of first syntax element indicative of a first variable indicating whether the layer *i* is included in at least one of the plurality of OLSs.

[0010] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video and a bitstream of the video, wherein the bitstream includes one or more output layer sets each comprising one or more video layers; wherein the bitstream conforms to a format rule, wherein the format rule specifies that a number of decoded picture buffer parameter syntax structures included in a video parameter set for the bitstream is equal to: zero, in a case that each output layer set includes a single video layer; or one plus a value of a syntax element, in a case that each output layer set includes a single layer is not true.

[0011] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video and a bitstream of the video, wherein the bitstream includes a coded video sequence (CVS) comprising one or more coded video pictures of one or more video layers; and wherein the bitstream conforms to a format rule that specifies that one or more sequence parameter sets (SPS) indicative of conversion parameters that are referred to by the one or more coded pictures of the CVS have a same reference video parameter set (VPS) identifier indicative of a referenced VPS.

[0012] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video and a bitstream of the video, wherein the bitstream comprises one or more output layer sets (OLSs) each comprising one or more video layers, wherein the bitstream conforms to a format rule; wherein the format rule specifies whether or how a first syntax element indicating whether a first syntax structure descriptive of parameters of a hypothetical reference decoder (HRD) used for the conversion is included in a video parameter set (VPS) of the bitstream.

[0013] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video and a bitstream of the video, wherein the bitstream comprises one or more output layer sets (OLSs) each comprising one or more video layers, wherein the bitstream conforms to a format rule; wherein the format rule specifies whether or how a first syntax structure descriptive of general hypothetical reference decoder (HRD) parameters and a number of second syntax structures descriptive of OLS-specific HRD parameters are included in a video parameter set (VPS) for the bitstream.

[0014] In yet another example aspect, a video encoder apparatus is disclosed. The video encoder comprises a processor configured to implement above-described methods.

[0015] In yet another example aspect, a video decoder apparatus is disclosed. The video decoder comprises a processor configured to implement above-described methods.

[0016] In yet another example aspect, a computer readable medium having code stored thereon is disclosed. The code embodies one of the methods described herein in the form of processor-executable code.

[0017] In yet another example aspect, a method for writing a bitstream generated according to one of the above methods to a computer readable medium is disclosed.

[0018] In another example aspect, a computer readable medium storing a bitstream of a video generated according to an above-described method is disclosed.

[0019] These, and other, features are described throughout the present document.

BRIEF DESCRIPTION OF DRAWINGS

[0020] FIG. 1 is a block diagram that illustrates a video coding system in accordance with some embodiments of the present disclosure.

[0021] FIG. 2 is a block diagram of an example hardware platform used for video processing.

[0022] FIG. 3 is a flowchart for an example method of video processing.

[0023] FIG. 4 is a block diagram that illustrates an example video coding system.

[0024] FIG. 5 is a block diagram that illustrates an encoder in accordance with some embodiments of the present disclosure.

[0025] FIG. 6 is a block diagram that illustrates a decoder in accordance with some embodiments of the present disclosure.

[0026] FIGS. 7A - 7I are flowchart for various video processing method examples.

DETAILED DESCRIPTION

[0027] Section headings are used in the present document for ease of understanding and do not limit the applicability of techniques and embodiments disclosed in each section only to that section. Furthermore, H.266 terminology is used in some description only for ease of understanding and

not for limiting scope of the disclosed techniques. As such, the techniques described herein are applicable to other video codec protocols and designs also.

1. Summary

[0028] This document is related to video coding technologies. Specifically, it is about various improvements in scalable video coding, wherein a video bitstream can contains more than one layer. The ideas may be applied individually or in various combination, to any video coding standard or non-standard video codec that supports multi-layer video coding, e.g., the being-developed Versatile Video Coding (VVC).

2. Abbreviations

APS	Adaptation Parameter Set
AU	Access Unit
AUD	Access Unit Delimiter
AVC	Advanced Video Coding
CLVS	Coded Layer Video Sequence
CPB	Coded Picture Buffer
CRA	Clean Random Access
CTU	Coding Tree Unit
CVS	Coded Video Sequence
DPB	Decoded Picture Buffer
DPS	Decoding Parameter Set
EOB	End Of Bitstream
EOS	End Of Sequence
GDR	Gradual Decoding Refresh
HEVC	High Efficiency Video Coding
HRD	Hypothetical Reference Decoder
IDR	Instantaneous Decoding Refresh
JEM	Joint Exploration Model
MCTS	Motion-Constrained Tile Sets
NAL	Network Abstraction Layer
OLS	Output Layer Set
PH	Picture Header
PPS	Picture Parameter Set
PTL	Profile, Tier and Level

PU	Picture Unit
RBSP	Raw Byte Sequence Payload
SEI	Supplemental Enhancement Information
SPS	Sequence Parameter Set
SVC	Scalable Video Coding
VCL	Video Coding Layer
VPS	Video Parameter Set
VTM	VVC Test Model
VUI	Video Usability Information
VVC	Versatile Video Coding

3. Initial Discussion

[0029] Video coding standards have evolved primarily through the development of the well-known ITU-T and ISO/IEC standards. The ITU-T produced H.261 and H.263, ISO/IEC produced MPEG-1 and MPEG-4 Visual, and the two organizations jointly produced the H.262/MPEG-2 Video and H.264/MPEG-4 Advanced Video Coding (AVC) and H.265/HEVC [1] standards. Since H.262, the video coding standards are based on the hybrid video coding structure wherein temporal prediction plus transform coding are utilized. To explore the future video coding technologies beyond HEVC, the Joint Video Exploration Team (JVET) was founded by VCEG and MPEG jointly in 2015. Since then, many new methods have been adopted by JVET and put into the reference software named Joint Exploration Model (JEM) [2]. The JVET meeting is concurrently held once every quarter, and the new coding standard is targeting at 50% bitrate reduction as compared to HEVC. The new video coding standard was officially named as Versatile Video Coding (VVC) in the April 2018 JVET meeting, and the first version of VVC test model (VTM) was released at that time. As there are continuous effort contributing to VVC standardization, new coding techniques are being adopted to the VVC standard in every JVET meeting. The VVC working draft and test model VTM are then updated after every meeting. The VVC project is now aiming for technical completion (FDIS) at the July 2020 meeting.

3.1. Scalable video coding (SVC) in general

[0030] Scalable video coding (SVC) refers to video coding in which a base layer (BL), sometimes referred to as a reference layer (RL), and one or more scalable enhancement layers (ELs) are used. In SVC, the base layer can carry video data with a base level of quality. The one or more enhancement layers can carry additional video data to support, for example, higher spatial,

temporal, and/or signal-to-noise (SNR) levels. Enhancement layers may be defined relative to a previously encoded layer. For example, a bottom layer may serve as a BL, while a top layer may serve as an EL. Middle layers may serve as either ELs or RLs, or both. For example, a middle layer (e.g., a layer that is neither the lowest layer nor the highest layer) may be an EL for the layers below the middle layer, such as the base layer or any intervening enhancement layers, and at the same time serve as a RL for one or more enhancement layers above the middle layer. Similarly, in the Multiview or 3D extension of the HEVC standard, there may be multiple views, and information of one view may be utilized to code (e.g., encode or decode) the information of another view (e.g., motion estimation, motion vector prediction and/or other redundancies).

[0031] In SVC, the parameters used by the encoder or the decoder are grouped into parameter sets based on the coding level (e.g., video-level, sequence-level, picture-level, slice level, etc.) in which they may be utilized. For example, parameters that may be utilized by one or more coded video sequences of different layers in the bitstream may be included in a video parameter set (VPS), and parameters that are utilized by one or more pictures in a coded video sequence may be included in a sequence parameter set (SPS). Similarly, parameters that are utilized by one or more slices in a picture may be included in a picture parameter set (PPS), and other parameters that are specific to a single slice may be included in a slice header. Similarly, the indication of which parameter set(s) a particular layer is using at a given time may be provided at various coding levels.

3.2. Parameter sets

[0032] AVC, HEVC, and VVC specify parameter sets. The types of parameter sets include SPS, PPS, APS, VPS, and DPS. SPS and PPS are supported in all of AVC, HEVC, and VVC. VPS was introduced since HEVC and is included in both HEVC and VVC. APS and DPS were not included in AVC or HEVC, but are included in the latest VVC draft text.

[0033] SPS was designed to carry sequence-level header information, and PPS was designed to carry infrequently changing picture-level header information. With SPS and PPS, infrequently changing information need not to be repeated for each sequence or picture, hence redundant signalling of these information can be avoided. Furthermore, the use of SPS and PPS enables out-of-band transmission of the important header information, thus not only avoiding the need for redundant transmissions but also improving error resilience.

[0034] VPS was introduced for carrying sequence-level header information that is common for all layers in multi-layer bitstreams.

[0035] APS was introduced for carrying such picture-level or slice-level information that needs quite some bits to code, can be shared by multiple pictures, and in a sequence there can be quite many different variations.

[0036] DPS was introduced for carrying bitstream-level information that indicates the highest capability needed for decoding the entire bitstream.

3.3. VPS syntax and semantics in VVC

[0037] VVC supports scalability, also known as scalable video coding, wherein multiple layers can be encoded in one coded video bitstream.

[0038] In the latest VVC text, the scalability information is signalled in the VPS, the syntax and semantics are as follows.

7.3.2.2 Video parameter set 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_all_layers_same_num_sublayers_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])	
for(j = 0; j < i; j++)	
vps_direct_ref_layer_flag[i][j]	u(1)
}	
}	
if(vps_max_layers_minus1 > 0) {	
if(vps_all_independent_layers_flag)	
each_layer_is_an_ols_flag	u(1)
if(!each_layer_is_an_ols_flag) {	
if(!vps_all_independent_layers_flag)	
ols_mode_idc	u(2)
if(ols_mode_idc == 2) {	
num_output_layer_sets_minus1	u(8)

for(i = 1; i <= num_output_layer_sets_minus1; i ++)	
for(j = 0; j <= vps_max_layers_minus1; j ++)	
ols_output_layer_flag[i][j]	u(1)
}	
}	
}	
vps_num_ptls	u(8)
for(i = 0; i < vps_num_ptls; i ++) {	
if(i > 0)	
pt_present_flag[i]	u(1)
if(vps_max_sublayers_minus1 > 0 && !vps_all_layers_same_num_sublayers_flag)	
ptl_max_temporal_id[i]	u(3)
}	
while(!byte_aligned())	
vps_ptl_byte_alignment_zero_bit /* equal to 0 */	u(1)
for(i = 0; i < vps_num_ptls; i ++)	
profile_tier_level(pt_present_flag[i], ptl_max_temporal_id[i])	
for(i = 0; i < TotalNumOls; i ++)	
if(NumLayersInOls[i] > 1 && vps_num_ptls > 1)	
ols_ptl_idx[i]	u(8)
if(!vps_all_independent_layers_flag)	
vps_num_dpb_params	ue(v)
if(vps_num_dpb_params > 0) {	
same_dpb_size_output_or_nonoutput_flag	u(1)
if(vps_max_sublayers_minus1 > 0)	
vps_sublayer_dpb_params_present_flag	u(1)
}	
for(i = 0; i < vps_num_dpb_params; i ++) {	
dpb_size_only_flag[i]	u(1)
if(vps_max_sublayers_minus1 > 0 && !vps_all_layers_same_num_sublayers_flag)	
dpb_max_temporal_id[i]	u(3)
dpb_parameters(dpb_size_only_flag[i], dpb_max_temporal_id[i], vps_sublayer_dpb_params_present_flag)	
}	
for(i = 0; i < vps_max_layers_minus1 && vps_num_dpb_params > 1; i ++) {	
if(!vps_independent_layer_flag[i])	
layer_output_dpb_params_idx[i]	ue(v)
if(LayerUsedAsRefLayerFlag[i] && !same_dpb_size_output_or_nonoutput_flag)	
layer_nonoutput_dpb_params_idx[i]	ue(v)
}	
vps_general_hrd_params_present_flag	u(1)
if(vps_general_hrd_params_present_flag) {	
general_hrd_parameters()	

if(vps_max_sublayers_minus1 > 0)	
vps_sublayer_cpb_params_present_flag	u(1)
if(TotalNumOlss > 1)	
num_ols_hrd_params_minus1	ue(v)
for(i = 0; i <= num_ols_hrd_params_minus1; i++) {	
if(vps_max_sublayers_minus1 > 0 && !vps_all_layers_same_num_sublayers_flag)	
hrd_max_tid[i]	u(3)
firstSubLayer = vps_sublayer_cpb_params_present_flag ? 0 : hrd_max_tid[i]	
ols_hrd_parameters(firstSubLayer, hrd_max_temporal_id[i])	
}	
if(num_ols_hrd_params_minus1 > 0)	
for(i = 1; i < TotalNumOlss; i++)	
ols_hrd_idx[i]	ue(v)
}	
vps_extension_flag	u(1)
if(vps_extension_flag)	
while(more_rbsp_data())	
vps_extension_data_flag	u(1)
rbsp_trailing_bits()	
}	

7.4.3.2 Video parameter set RBSP semantics

A VPS RBSP shall be available to the decoding process prior to it being referenced, included in at least one AU with TemporalId equal to 0 or provided through external means.

All VPS NAL units with a particular value of vps_video_parameter_set_id in a CVS shall have the same content.

vps_video_parameter_set_id provides an identifier for the VPS for reference by other syntax elements. The value of vps_video_parameter_set_id shall be greater than 0.

vps_max_layers_minus1 plus 1 specifies the maximum allowed number of layers in each CVS referring to the VPS.

vps_max_sublayers_minus1 plus 1 specifies the maximum number of temporal sublayers that may be present in each CVS referring to the VPS. The value of vps_max_sublayers_minus1 shall be in the range of 0 to 6, inclusive.

vps_all_layers_same_num_sublayers_flag equal to 1 specifies that the number of temporal sublayers is the same for all the layers in each CVS referring to the VPS. vps_all_layers_same_num_sublayers_flag equal to 0 specifies that the layers in each CVS referring to the VPS may or may not have the same number

of temporal sublayers. When not present, the value of `vps_all_layers_same_num_sublayers_flag` is inferred to be equal to 1.

`vps_all_independent_layers_flag` equal to 1 specifies that all layers in the CVS are independently coded without using inter-layer prediction. `vps_all_independent_layers_flag` equal to 0 specifies that one or more of the layers in the CVS may use inter-layer prediction. When not present, the value of `vps_all_independent_layers_flag` is inferred to be equal to 1. When `vps_all_independent_layers_flag` is equal to 1, the value of `vps_independent_layer_flag[i]` is inferred to be equal to 1. When `vps_all_independent_layers_flag` is equal to 0, the value of `vps_independent_layer_flag[0]` is inferred to be equal to 1.

`vps_layer_id[i]` specifies the `nuh_layer_id` value of the *i*-th layer. For any two non-negative integer values of *m* and *n*, when *m* is less than *n*, the value of `vps_layer_id[m]` shall be less than `vps_layer_id[n]`.

`vps_independent_layer_flag[i]` equal to 1 specifies that the layer with index *i* does not use inter-layer prediction. `vps_independent_layer_flag[i]` equal to 0 specifies that the layer with index *i* may use inter-layer prediction and the syntax elements `vps_direct_ref_layer_flag[i][j]` for *j* in the range of 0 to *i* – 1, inclusive, are present in VPS. When not present, the value of `vps_independent_layer_flag[i]` is inferred to be equal to 1.

`vps_direct_ref_layer_flag[i][j]` equal to 0 specifies that the layer with index *j* is not a direct reference layer for the layer with index *i*. `vps_direct_ref_layer_flag[i][j]` equal to 1 specifies that the layer with index *j* is a direct reference layer for the layer with index *i*. When `vps_direct_ref_layer_flag[i][j]` is not present for *i* and *j* in the range of 0 to `vps_max_layers_minus1`, inclusive, it is inferred to be equal to 0. When `vps_independent_layer_flag[i]` is equal to 0, there shall be at least one value of *j* in the range of 0 to *i* – 1, inclusive, such that the value of `vps_direct_ref_layer_flag[i][j]` is equal to 1.

The variables `NumDirectRefLayers[i]`, `DirectRefLayerIdx[i][d]`, `NumRefLayers[i]`, `RefLayerIdx[i][r]`, and `LayerUsedAsRefLayerFlag[j]` are derived as follows:

```

for( i = 0; i <= vps_max_layers_minus1; i++ )
    for( j = 0; j <= vps_max_layers_minus1; j++ ) {
        dependencyFlag[ i ][ j ] = vps_direct_ref_layer_flag[ i ][ j ]
        for( k = 0; k < i; k++ )
            if( vps_direct_ref_layer_flag[ i ][ k ] && dependencyFlag[ k ][ j ] )
                dependencyFlag[ i ][ j ] = 1
    }
for( i = 0; i <= vps_max_layers_minus1; i++ ) {

```

```

    LayerUsedAsRefLayerFlag[ j ] = 0
    for( j = 0, d = 0, r = 0; j <= vps_max_layers_minus1; j++ ) {
        if( direct_ref_layer_flag[ i ][ j ] ) {
            DirectRefLayerIdx[ i ][ d++ ] = j
            LayerUsedAsRefLayerFlag[ j ] = 1
        }
        if( dependencyFlag[ i ][ j ] )
            RefLayerIdx[ i ][ r++ ] = j
    }
    NumDirectRefLayers[ i ] = d
    NumRefLayers[ i ] = r
}

```

(37)

The variable `GeneralLayerIdx[ i ]`, specifying the layer index of the layer with `nuh_layer_id` equal to `vps_layer_id[ i ]`, is derived as follows:

```

for( i = 0; i <= vps_max_layers_minus1; i++ )
    GeneralLayerIdx[ vps_layer_id[ i ] ] = i

```

(38)

each_layer_is_an_ols_flag equal to 1 specifies that each output layer set contains only one layer and each layer itself in the bitstream is an output layer set with the single included layer being the only output layer. **each_layer_is_an_ols_flag** equal to 0 that an output layer set may contain more than one layer. If `vps_max_layers_minus1` is equal to 0, the value of the value of **each_layer_is_an_ols_flag** is inferred to be equal to 1. Otherwise, when **vps_all_independent_layers_flag** is equal to 0, the value of **each_layer_is_an_ols_flag** is inferred to be equal to 0.

ols_mode_idc equal to 0 specifies that the total number of OLSs specified by the VPS is equal to `vps_max_layers_minus1 + 1`, the *i*-th OLS includes the layers with layer indices from 0 to *i*, inclusive, and for each OLS only the highest layer in the OLS is output.

ols_mode_idc equal to 1 specifies that the total number of OLSs specified by the VPS is equal to `vps_max_layers_minus1 + 1`, the *i*-th OLS includes the layers with layer indices from 0 to *i*, inclusive, and for each OLS all layers in the OLS are output.

ols_mode_idc equal to 2 specifies that the total number of OLSs specified by the VPS is explicitly signalled and for each OLS the output layers are explicitly signalled and other layers are the layers that are direct or indirect reference layers of the output layers of the OLS.

The value of `ols_mode_idc` shall be in the range of 0 to 2, inclusive. The value 3 of `ols_mode_idc` is reserved for future use by ITU-T | ISO/IEC.

When `vps_all_independent_layers_flag` is equal to 1 and `each_layer_is_an_ols_flag` is equal to 0, the value of `ols_mode_idc` is inferred to be equal to 2.

`num_output_layer_sets_minus1` plus 1 specifies the total number of OLSs specified by the VPS when `ols_mode_idc` is equal to 2.

The variable `TotalNumOlss`, specifying the total number of OLSs specified by the VPS, is derived as follows:

```

if( vps_max_layers_minus1 == 0 )
    TotalNumOlss = 1
else if( each_layer_is_an_ols_flag || ols_mode_idc == 0 || ols_mode_idc == 1 )
    TotalNumOlss = vps_max_layers_minus1 + 1
else if( ols_mode_idc == 2 )
    TotalNumOlss = num_output_layer_sets_minus1 + 1

```

(39)

`ols_output_layer_flag[i][j]` equal to 1 specifies that the layer with `nuh_layer_id` equal to `vps_layer_id[j]` is an output layer of the *i*-th OLS when `ols_mode_idc` is equal to 2. `ols_output_layer_flag[i][j]` equal to 0 specifies that the layer with `nuh_layer_id` equal to `vps_layer_id[j]` is not an output layer of the *i*-th OLS when `ols_mode_idc` is equal to 2.

The variable `NumOutputLayersInOls[i]`, specifying the number of output layers in the *i*-th OLS, and the variable `OutputLayerIdInOls[i][j]`, specifying the `nuh_layer_id` value of the *j*-th output layer in the *i*-th OLS, are derived as follows:

```

NumOutputLayersInOls[ 0 ] = 1
OutputLayerIdInOls[ 0 ][ 0 ] = vps_layer_id[ 0 ]
for( i = 1; i < TotalNumOlss; i++ ) {
    if( each_layer_is_an_ols_flag || ols_mode_idc == 0 ) {
        NumOutputLayersInOls[ i ] = 1
        OutputLayerIdInOls[ i ][ 0 ] = vps_layer_id[ i ]
    } else if( ols_mode_idc == 1 ) {
        NumOutputLayersInOls[ i ] = i + 1
        for( j = 0; j < NumOutputLayersInOls[ i ]; j++ )
            OutputLayerIdInOls[ i ][ j ] = vps_layer_id[ j ]
    }
}

```

```

    } else if( ols_mode_idc == 2 ) {
        for( j = 0; j <= vps_max_layers_minus1; j++ )
            layerIncludedFlag[ i ][ j ] = 0
        for( k = 0; j = 0; k <= vps_max_layers_minus1; k++ )
            if( ols_output_layer_flag[ i ][ k ] ) {
                layerIncludedFlag[ i ][ j ] = 1
                OutputLayerIdx[ i ][ j ] = k
                OutputLayerIdInOls[ i ][ j++ ] = vps_layer_id[ k ]
            }
        NumOutputLayersInOls[ i ] = j
        for( j = 0; j < NumOutputLayersInOls[ i ]; j++ ) {
            idx = OutputLayerIdx[ i ][ j ]
            for( k = 0; k < NumRefLayers[ idx ]; k++ )
                layerIncludedFlag[ i ][ RefLayerIdx[ idx ][ k ] ] = 1
        }
    }
}

```

(40)

For each OLS, there shall be at least one layer that is an output layer. In other words, for any value of i in the range of 0 to $\text{TotalNumOls} - 1$, inclusive, the value of $\text{NumOutputLayersInOls}[i]$ shall be greater than or equal to 1.

The variable $\text{NumLayersInOls}[i]$, specifying the number of layers in the i -th OLS, and the variable $\text{LayerIdInOls}[i][j]$, specifying the nuh_layer_id value of the j -th layer in the i -th OLS, are derived as follows:

```

NumLayersInOls[ 0 ] = 1
LayerIdInOls[ 0 ][ 0 ] = vps_layer_id[ 0 ]
for( i = 1; i < TotalNumOls; i++ ) {
    if( each_layer_is_an_ols_flag ) {
        NumLayersInOls[ i ] = 1
        LayerIdInOls[ i ][ 0 ] = vps_layer_id[ i ]
    } else if( ols_mode_idc == 0 || ols_mode_idc == 1 ) {
        NumLayersInOls[ i ] = i + 1
        for( j = 0; j < NumLayersInOls[ i ]; j++ )

```

(41)

```

        LayerIdInOls[ i ][ j ] = vps_layer_id[ j ]
    } else if( ols_mode_idc == 2 ) {
        for( k = 0; j = 0; k <= vps_max_layers_minus1; k++ )
            if( layerIncludedFlag[ i ][ k ] )
                LayerIdInOls[ i ][ j++ ] = vps_layer_id[ k ]
        NumLayersInOls[ i ] = j
    }
}

```

NOTE 1 – The 0-th OLS contains only the lowest layer (i.e., the layer with nuh_layer_id equal to vps_layer_id[0]) and for the 0-th OLS the only included layer is output.

The variable OlsLayerIdx[i][j], specifying the OLS layer index of the layer with nuh_layer_id equal to LayerIdInOls[i][j], is derived as follows:

```

for( i = 0; i < TotalNumOls; i++ )
    for( j = 0; j < NumLayersInOls[ i ]; j++ )
        OlsLayerIdx[ i ][ LayerIdInOls[ i ][ j ] ] = j

```

(42)

The lowest layer in each OLS shall be an independent layer. In other words, for each i in the range of 0 to TotalNumOls – 1, inclusive, the value of vps_independent_layer_flag[GeneralLayerIdx[LayerIdInOls[i][0]]] shall be equal to 1.

Each layer shall be included in at least one OLS specified by the VPS. In other words, for each layer with a particular value of nuh_layer_id nuhLayerId, equal to one of vps_layer_id[k] for k in the range of 0 to vps_max_layers_minus1, inclusive, there shall be at least one pair of values of i and j, where i is in the range of 0 to TotalNumOls – 1, inclusive, and j is in the range of NumLayersInOls[i] – 1, inclusive, such that the value of LayerIdInOls[i][j] is equal to nuhLayerId.

vps_num_ptls specifies the number of profile_tier_level() syntax structures in the VPS.

pt_present_flag[i] equal to 1 specifies that profile, tier, and general constraints information are present in the i-th profile_tier_level() syntax structure in the VPS. pt_present_flag[i] equal to 0 specifies that profile, tier, and general constraints information are not present in the i-th profile_tier_level() syntax structure in the VPS. The value of pt_present_flag[0] is inferred to be equal to 0. When pt_present_flag[i] is equal to 0, the profile, tier, and general constraints information for the i-th profile_tier_level() syntax structure in the VPS are inferred to be the same as that for the (i – 1)-th profile_tier_level() syntax structure in the VPS.

ptl_max_temporal_id[i] specifies the TemporalId of the highest sublayer representation for which the level information is present in the *i*-th `profile_tier_level()` syntax structure in the VPS. The value of `ptl_max_temporal_id[ i ]` shall be in the range of 0 to `vps_max_sublayers_minus1`, inclusive. When `vps_max_sublayers_minus1` is equal to 0, the value of `ptl_max_temporal_id[ i ]` is inferred to be equal to 0. When `vps_max_sublayers_minus1` is greater than 0 and `vps_all_layers_same_num_sublayers_flag` is equal to 1, the value of `ptl_max_temporal_id[ i ]` is inferred to be equal to `vps_max_sublayers_minus1`.

vps_ptl_byte_alignment_zero_bit shall be equal to 0.

ols_ptl_idx[i] specifies the index, to the list of `profile_tier_level()` syntax structures in the VPS, of the `profile_tier_level()` syntax structure that applies to the *i*-th OLS. When present, the value of `ols_ptl_idx[ i ]` shall be in the range of 0 to `vps_num_ptls - 1`, inclusive.

When `NumLayersInOls[ i ]` is equal to 1, the `profile_tier_level()` syntax structure that applies to the *i*-th OLS is present in the SPS referred to by the layer in the *i*-th OLS.

vps_num_dpb_params specifies the number of `dpb_parameters()` syntax structures in the VPS. The value of `vps_num_dpb_params` shall be in the range of 0 to 16, inclusive. When not present, the value of `vps_num_dpb_params` is inferred to be equal to 0.

same_dpb_size_output_or_nonoutput_flag equal to 1 specifies that there is no `layer_nonoutput_dpb_params_idx[ i ]` syntax element present in the VPS. **same_dpb_size_output_or_nonoutput_flag** equal to 0 specifies that there may or may not be `layer_nonoutput_dpb_params_idx[ i ]` syntax elements present in the VPS.

vps_sublayer_dpb_params_present_flag is used to control the presence of `max_dec_pic_buffering_minus1[ ]`, `max_num_reorder_pics[ ]`, and `max_latency_increase_plus1[ ]` syntax elements in the `dpb_parameters()` syntax structures in the VPS. When not present, `vps_sub_dpb_params_info_present_flag` is inferred to be equal to 0.

dpb_size_only_flag[i] equal to 1 specifies that the `max_num_reorder_pics[ ]` and `max_latency_increase_plus1[ ]` syntax elements are not present in the *i*-th `dpb_parameters()` syntax structures in the VPS. **dpb_size_only_flag[i]** equal to 0 specifies that the `max_num_reorder_pics[ ]` and `max_latency_increase_plus1[ ]` syntax elements may be present in the *i*-th `dpb_parameters()` syntax structures in the VPS.

dpb_max_temporal_id[i] specifies the TemporalId of the highest sublayer representation for which the DPB parameters may be present in the *i*-th `dpb_parameters()` syntax structure in the VPS. The value of `dpb_max_temporal_id[ i ]` shall be in the range of 0 to `vps_max_sublayers_minus1`, inclusive. When

`vps_max_sublayers_minus1` is equal to 0, the value of `dpb_max_temporal_id[ i ]` is inferred to be equal to 0. When `vps_max_sublayers_minus1` is greater than 0 and `vps_all_layers_same_num_sublayers_flag` is equal to 1, the value of `dpb_max_temporal_id[ i ]` is inferred to be equal to `vps_max_sublayers_minus1`.

`layer_output_dpb_params_idx[ i ]` specifies the index, to the list of `dpb_parameters( )` syntax structures in the VPS, of the `dpb_parameters( )` syntax structure that applies to the *i*-th layer when it is an output layer in an OLS. When present, the value of `layer_output_dpb_params_idx[ i ]` shall be in the range of 0 to `vps_num_dpb_params - 1`, inclusive.

If `vps_independent_layer_flag[ i ]` is equal to 1, the `dpb_parameters( )` syntax structure that applies to the *i*-th layer when it is an output layer is the `dpb_parameters( )` syntax structure present in the SPS referred to by the layer.

Otherwise (`vps_independent_layer_flag[ i ]` is equal to 0), the following applies:

- When `vps_num_dpb_params` is equal to 1, the value of `layer_output_dpb_params_idx[ i ]` is inferred to be equal to 0.
- It is a requirement of bitstream conformance that the value of `layer_output_dpb_params_idx[ i ]` shall be such that `dpb_size_only_flag[ layer_output_dpb_params_idx[ i ] ]` is equal to 0.

`layer_nonoutput_dpb_params_idx[ i ]` specifies the index, to the list of `dpb_parameters( )` syntax structures in the VPS, of the `dpb_parameters( )` syntax structure that applies to the *i*-th layer when it is a non-output layer in an OLS. When present, the value of `layer_nonoutput_dpb_params_idx[ i ]` shall be in the range of 0 to `vps_num_dpb_params - 1`, inclusive.

If `same_dpb_size_output_or_nonoutput_flag` is equal to 1, the following applies:

- If `vps_independent_layer_flag[ i ]` is equal to 1, the `dpb_parameters( )` syntax structure that applies to the *i*-th layer when it is a non-output layer is the `dpb_parameters( )` syntax structure present in the SPS referred to by the layer.
- Otherwise (`vps_independent_layer_flag[ i ]` is equal to 0), the value of `layer_nonoutput_dpb_params_idx[ i ]` is inferred to be equal to `layer_output_dpb_params_idx[ i ]`.

Otherwise (`same_dpb_size_output_or_nonoutput_flag` is equal to 0), when `vps_num_dpb_params` is equal to 1, the value of `layer_output_dpb_params_idx[ i ]` is inferred to be equal to 0.

`vps_general_hrd_params_present_flag` equal to 1 specifies that the syntax structure `general_hrd_parameters( )` and other HRD parameters are present in the VPS RBSP syntax structure.

vps_general_hrd_params_present_flag equal to 0 specifies that the syntax structure **general_hrd_parameters()** and other HRD parameters are not present in the VPS RBSP syntax structure.

vps_sublayer_cpb_params_present_flag equal to 1 specifies that the *i*-th **ols_hrd_parameters()** syntax structure in the VPS contains HRD parameters for the sublayer representations with TemporalId in the range of 0 to **hrd_max_tid[i]**, inclusive. **vps_sublayer_cpb_params_present_flag** equal to 0 specifies that the *i*-th **ols_hrd_parameters()** syntax structure in the VPS contains HRD parameters for the sublayer representation with TemporalId equal to **hrd_max_tid[i]** only. When **vps_max_sublayers_minus1** is equal to 0, the value of **vps_sublayer_cpb_params_present_flag** is inferred to be equal to 0.

When **vps_sublayer_cpb_params_present_flag** is equal to 0, the HRD parameters for the sublayer representations with TemporalId in the range of 0 to **hrd_max_tid[i] - 1**, inclusive, are inferred to be the same as that for the sublayer representation with TemporalId equal to **hrd_max_tid[i]**. These include the HRD parameters starting from the **fixed_pic_rate_general_flag[i]** syntax element till the **sublayer_hrd_parameters(i)** syntax structure immediately under the condition "if(**general_vcl_hrd_params_present_flag**)" in the **ols_hrd_parameters** syntax structure.

num_ols_hrd_params_minus1 plus 1 specifies the number of **ols_hrd_parameters()** syntax structures present in the **general_hrd_parameters()** syntax structure. The value of **num_ols_hrd_params_minus1** shall be in the range of 0 to 63, inclusive. When **TotalNumOlss** is greater than 1, the value of **num_ols_hrd_params_minus1** is inferred to be equal to 0.

hrd_max_tid[i] specifies the TemporalId of the highest sublayer representation for which the HRD parameters are contained in the *i*-th **ols_hrd_parameters()** syntax structure. The value of **hrd_max_tid[i]** shall be in the range of 0 to **vps_max_sublayers_minus1**, inclusive. When **vps_max_sublayers_minus1** is equal to 0, the value of **hrd_max_tid[i]** is inferred to be equal to 0.

ols_hrd_idx[i] specifies the index of the **ols_hrd_parameters()** syntax structure that applies to the *i*-th OLS. The value of **ols_hrd_idx[i]** shall be in the range of 0 to **num_ols_hrd_params_minus1**, inclusive. When not present, the value of **ols_hrd_idx[i]** is inferred to be equal to 0.

vps_extension_flag equal to 0 specifies that no **vps_extension_data_flag** syntax elements are present in the VPS RBSP syntax structure. **vps_extension_flag** equal to 1 specifies that there are **vps_extension_data_flag** syntax elements present in the VPS RBSP syntax structure.

vps_extension_data_flag may have any value. Its presence and value do not affect decoder conformance to profiles specified in this version of this Specification. Decoders conforming to this version of this Specification shall ignore all **vps_extension_data_flag** syntax elements.

3.4. SPS syntax and semantics in VVC

[0039] In the latest VVC draft text in JVET-P2001-v14, the SPS syntax and semantics that are most relevant to the inventions herein are as follows.

7.3.2.3 Sequence parameter set RBSP syntax

seq_parameter_set_rbsp() {	Descriptor
sps_decoding_parameter_set_id	u(4)
sps_video_parameter_set_id	u(4)
sps_max_sublayers_minus1	u(3)
sps_reserved_zero_4bits	u(4)
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)	
...	

7.4.3.3 Sequence parameter set RBSP semantics

An SPS RBSP shall be available to the decoding process prior to it being referenced, included in at least one AU with TemporalId equal to 0 or provided through external means.

All SPS NAL units with a particular value of sps_seq_parameter_set_id in a CVS shall have the same content.

sps_decoding_parameter_set_id, when greater than 0, specifies the value of dps_decoding_parameter_set_id for the DPS referred to by the SPS. When sps_decoding_parameter_set_id is equal to 0, the SPS does not refer to a DPS and no DPS is referred to when decoding each CLVS referring to the SPS. The value of sps_decoding_parameter_set_id shall be the same in all SPSs that are referred to by coded pictures in a bitstream.

sps_video_parameter_set_id, when greater than 0, specifies the value of vps_video_parameter_set_id for the VPS referred to by the SPS.

When sps_video_parameter_set_id is equal to 0, the following applies:

- The SPS does not refer to a VPS.
- No VPS is referred to when decoding each CLVS referring to the SPS.
- The value of vps_max_layers_minus1 is inferred to be equal to 0.

- The CVS shall contain only one layer (i.e., all VCL NAL unit in the CVS shall have the same value of `nuh_layer_id`).
- The value of `GeneralLayerIdx[ nuh_layer_id ]` is inferred to be equal to 0.
- The value of `vps_independent_layer_flag[ GeneralLayerIdx[ nuh_layer_id ] ]` is inferred to be equal to 1.

When `vps_independent_layer_flag[ GeneralLayerIdx[ nuh_layer_id ] ]` is equal to 1, the SPS referred to by a CLVS with a particular `nuh_layer_id` value `nuhLayerId` shall have `nuh_layer_id` equal to `nuhLayerId`.

`sps_max_sublayers_minus1` plus 1 specifies the maximum number of temporal sublayers that may be present in each CLVS referring to the SPS. The value of `sps_max_sublayers_minus1` shall be in the range of 0 to `vps_max_sublayers_minus1`, inclusive.

`sps_reserved_zero_4bits` shall be equal to 0 in bitstreams conforming to this version of this Specification. Other values for `sps_reserved_zero_4bits` are reserved for future use by ITU-T | ISO/IEC.

`sps_ptl_dpb_hrd_params_present_flag` equal to 1 specifies that a `profile_tier_level()` syntax structure and a `dpb_parameters()` syntax structure are present in the SPS, and a `general_hrd_parameters()` syntax structure and an `ols_hrd_parameters()` syntax structure may also be present in the SPS. `sps_ptl_dpb_hrd_params_present_flag` equal to 0 specifies that none of these syntax structures is present in the SPS. The value of `sps_ptl_dpb_hrd_params_present_flag` shall be equal to `vps_independent_layer_flag[ nuh_layer_id ]`.

If `vps_independent_layer_flag[ GeneralLayerIdx[ nuh_layer_id ] ]` is equal to 1, the variable `MaxDecPicBuffMinus1` is set equal to `max_dec_pic_buffering_minus1[ sps_max_sublayers_minus1 ]` in the `dpb_parameters()` syntax structure in the SPS. Otherwise, `MaxDecPicBuffMinus1` is set equal to `max_dec_pic_buffering_minus1[ sps_max_sublayers_minus1 ]` in the `layer_nonoutput_dpb_params_idx[ GeneralLayerIdx[ nuh_layer_id ] ]`-th `dpb_parameters()` syntax structure in the VPS.

...

3.5. Slice header syntax and semantics in VVC

[0040] In the latest VVC draft text in JVET-P2001-v14, the slice header syntax and semantics that are most relevant to the inventions herein are as follows.

7.3.7.1 General slice header syntax

slice_header() {	Descriptor
slice_pic_order_cnt_lsb	u(v)
if(subpics_present_flag)	
slice_subpic_id	u(v)
if(rect_slice_flag NumTilesInPic > 1)	
slice_address	u(v)
if(!rect_slice_flag && NumTilesInPic > 1)	
num_tiles_in_slice_minus1	ue(v)
slice_type	ue(v)
...	

7.4.8.1 General slice header semantics

...

slice_type specifies the coding type of the slice according to Table 9.

Table 9 – Name association to slice_type

slice_type	Name of slice_type
0	B (B slice)
1	P (P slice)
2	I (I slice)

When nal_unit_type is a value of nal_unit_type in the range of IDR_W_RADL to CRA_NUT, inclusive, and the current picture is the first picture in an access unit, slice_type shall be equal to 2.

...

4. Technical problems solved by described technical solutions

[0041] The existing scalability design in VVC has the following problems:

- 1) The latest VVC draft text includes the following constraint on slice_type:

When nal_unit_type is a value of nal_unit_type in the range of IDR_W_RADL to CRA_NUT, inclusive, and the current picture is the first picture in an AU, slice_type shall be equal to 2.

The value of slice_type equal to 2 for a slice means that the slice is intra coded without using inter prediction from a reference picture.

However, in an AU, not only the first picture that is in the AU and that is an IRAP picture needs to contain intra-coded slices only, but also all the IRAP pictures in all the independent layers need to contain intra-coded slices only. Therefore, the above constraint does needs to be updated.

- 2) When the syntax element `ols_ptl_idx[ i ]` is not present, the value still needs to be used. However, there lacks an inference of the value of `ols_ptl_idx[ i ]` when it is not present.
- 3) The signalling of the syntax element `layer_output_dpb_params_idx[ i ]` is unnecessary when the *i*-layer is not used as an output layer in any of the OLSs.
- 4) The value of the syntax element `vps_num_dpb_params` can be equal to 0. However, there needs to be at least one `dpb_parameters()` syntax structure in the VPS when `vps_all_independent_layers_flag` is equal to 0.
- 5) In the latest VVC draft text, the PTL information for OLSs containing only one layer, such a layer is an independently coded layer without referring to any other layer, is only signalled in the SPS. However, it would be desirable to signal the PTL information for all OLSs in a bitstream in the VPS for session negotiation purposes.
- 6) The `vps_ptl_byte_alignment_zero_bit` syntax element is unnecessarily signalled when the number of PTL syntax structures signalled in the VPS is zero.
- 7) In the semantics of `sps_video_parameter_set_id`, there is the following constraint:

When `sps_video_parameter_set_id` is equal to 0, the CVS shall contain only one layer (i.e., all VCL NAL unit in the CVS shall have the same value of `nuh_layer_id`).

However, this constraint disallows an independent layer that does not refer to a VPS to be included in a multi-layer bitstream. As `sps_video_parameter_set_id` equal to 0 means that the SPS (and the layer) does not refer to a VPS.

- 8) The value of `vps_general_hrd_params_present_flag` can be equal to 1 when `each_layer_is_an_ols_flag` is equal to 1. However, when `each_layer_is_an_ols_flag` is equal to 1, the HRD parameters are only signalled in the SPSs, therefore the value of `vps_general_hrd_params_present_flag` shall not be equal to 1. In some cases, it may not

make sense to signal the `general_hrd_parameter()` syntax structure while signaling zero `old_hrd_parameters()` syntax structures in the VPS.

- 9) The HRD parameters for OLSs containing only one layer are signalled in both the VPS and the SPSs. However, it is not useful to repeat the HRD parameters in the VPS for OLSs containing only one layer.

5. Example embodiments and techniques

[0042] To solve the above problems, and others, methods as summarized below are disclosed. The inventions should be considered as examples to explain the general concepts and should not be interpreted in a narrow way. Furthermore, these inventions can be applied individually or combined in any manner.

- 1) To solve the first problem, the following constraint is specified:

When `nal_unit_type` is in the range of `IDR_W_RADL` to `CRA_NUT`, inclusive, and `vps_independent_layer_flag[ GeneralLayerIdx[ nuh_layer_id ] ]` is equal to 1, `slice_type` shall be equal to 2.

- 2) To solve the second problem, for each value of possible value of `i`, the value of `ols_ptl_idx[ i ]` is inferred to be equal to 0 when the syntax element is not present.
- 3) To solve the third problem, the variable `LayerUsedAsOutputLayerFlag[ i ]` is specified to indicate whether the `i`-layer is used as an output layer in whichever OLS, and when the variable is equal to 0, the signalling of the variable `layer_output_dpb_params_idx[ i ]` is avoided.
 - a. Furthermore, the following constraint may be additionally specified: For each value of `i` in the range of 0 to `vps_max_layers_minus1`, inclusive, the values of `LayerUsedAsRefLayerFlag[ i ]` and `LayerUsedAsOutputLayerFlag[ i ]` shall not be both equal to 0. In other words, there shall be no layer that is neither a direct reference layer of any other layer nor an output layer of at least one OLS.
- 4) To solve the fourth problem, change `vps_num_dpb_params` to `vps_num_dpb_params_minus1`, and specify `VpsNumDpbParams` as follows:

```

if( !vps_all_independent_layers_flag )
    VpsNumDpbParams = vps_num_dpb_params_minus1 + 1
else
    VpsNumDpbParams = 0

```

And replace the use of `vps_num_dpb_params` in the syntax conditions and the semantics with `VpsNumDpbParams`.

- a. Furthermore, the following constraint is additionally specified: When not present, the value of `same_dpb_size_output_or_nonoutput_flag` is inferred to be equal to 1.
- 5) To solve the fifth problem, allow repeating in the VPS the PTL information for OLSs containing only one layer for session negotiation purposes. This can be realized by changing `vps_num_ptls` to `vps_num_ptls_minus1`.
 - a. Alternatively, keep `vps_num_ptls` (without making it `vps_num_ptls_minus1`), but remove "`NumLayersInOls[ i ] > 1 &&` " from the syntax condition for `ols_ptl_idx[ i ]`.
 - a. Alternatively, keep `vps_num_ptls` (without making it `vps_num_ptls_minus1`), but condition `vps_num_ptls` on "`if( !each_layer_is_an_ols_flag )`", or "`if( vps_max_layers_minus1 > 0 && !vps_all_independent_layers_flag )`".
 - b. Alternatively, also allow repeating the DPB parameters information (either just the DPB size or all DPB parameters) for OLSs containing only one layer in the VPS.
- 6) To solve the sixth problem, the `vps_ptl_byte_alignment_zero_bit` syntax element is not signalled as long as the number of PTL syntax structures signalled in the VPS is zero. This can be realized by additionally conditioning `vps_ptl_byte_alignment_zero_bit` on "`if( vps_num_ptls > 0 )`" or by changing `vps_num_ptls` to `vps_num_ptls_minus1`, which effectively disallows the number of PTL syntax structures signalled in the VPS to be equal to zero.
- 7) To solve the seventh problem, remove the following constraint:

When `sps_video_parameter_set_id` is equal to 0, the CVS shall contain only one layer (i.e., all VCL NAL unit in the CVS shall have the same value of `nuh_layer_id`).

And add the following constraint:

The value of `sps_video_parameter_set_id` shall be the same in all SPSs that are referred to by coded pictures in a CVS and that have `sps_video_parameter_set_id` greater than 0.

- a. Alternatively, keep the following constraint:

When `sps_video_parameter_set_id` is equal to 0, the CVS shall contain only one layer (i.e., all VCL NAL unit in the CVS shall have the same value of `nuh_layer_id`).

And specify the following constraint:

The value of `sps_video_parameter_set_id` shall be the same in all SPSs that are referred to by coded pictures in a CVS.

- 8) To solve the eighth problem, the syntax element `vps_general_hrd_params_present_flag` is not signalled when `vps_general_hrd_params_present_flag` is equal to 1, and when not present, the value of `vps_general_hrd_params_present_flag` is inferred to be equal to 0.

- a. Alternatively, the value of `vps_general_hrd_params_present_flag` is constrained to be equal to 0 when `each_layer_is_an_ols_flag` is equal to 1.
- b. Furthermore, the syntax condition for the syntax element `num_ols_hrd_params_minus1`, i.e., "`if( TotalNumOlss > 1 )`", is removed, as it is not needed. This is because when `TotalNumOlss` is equal to 1, the value of `each_layer_is_an_ols_flag` would be equal to 1 and then the value of `vps_general_hrd_params_present_flag` would equal to 0 then no HRD parameters would be signalled in the VPS at all.

In some cases, when there is no `ols_hrd_parameters( )` syntax structure is present in the VPS, the `general_hrd_parameters( )` syntax structure is not signalled in the VPS.

- 9) To solve the ninth problem, the HRD parameters for OLSs containing only one layer are only signalled in SPSs, not in the VPS.

6. Embodiments

[0043] Below are some example embodiments for aspects summarized above in Section 5, which can be applied to the VVC specification. The changed texts are based on the latest VVC text in JVET-P2001-v14. Most relevant parts that have been added or modified are highlighted in **underline boldface**, and some of the deleted parts are highlighted in ***italics boldface***. There are some other changes that are editorial in nature and thus not highlighted.

6.1. First embodiment

6.1.1. VPS syntax and semantics

7.3.2.2 Video parameter set 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_all_layers_same_num_sublayers_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])	
for(j = 0; j < i; j++)	
vps_direct_ref_layer_flag[i][j]	u(1)
}	
}	
if(vps_max_layers_minus1 > 0) {	
if(vps_all_independent_layers_flag)	
each_layer_is_an_ols_flag	u(1)
if(!each_layer_is_an_ols_flag) {	
if(!vps_all_independent_layers_flag)	
ols_mode_idc	u(2)
if(ols_mode_idc == 2) {	
num_output_layer_sets_minus1	u(8)

for(i = 1; i <= num_output_layer_sets_minus1; i++)	
for(j = 0; j <= vps_max_layers_minus1; j++)	
ols_output_layer_flag[i][j]	u(1)
}	
}	
}	
<u>vps_num_ptls_minus1</u>	u(8)
for(i = 0; i <= vps_num_ptls_minus1; i++) {	
if(i > 0)	
pt_present_flag[i]	u(1)
if(vps_max_sublayers_minus1 > 0 && !vps_all_layers_same_num_sublayers_flag)	
ptl_max_temporal_id[i]	u(3)
}	
while(!byte_aligned())	
vps_ptl_byte_alignment_zero_bit /* equal to 0 */	u(1)
for(i = 0; i <= vps_num_ptls_minus1; i++)	
profile_tier_level(pt_present_flag[i], ptl_max_temporal_id[i])	
for(i = 0; i < TotalNumOlss; i++)	
<u>if(vps_num_ptls_minus1 > 0)</u>	
ols_ptl_idx[i]	u(8)
if(!vps_all_independent_layers_flag) {	
<u>vps_num_dpb_params_minus1</u>	ue(v)
same_dpb_size_output_or_nonoutput_flag	u(1)
if(vps_max_sublayers_minus1 > 0)	
vps_sublayer_dpb_params_present_flag	u(1)
}	
for(i = 0; i < <u>VpsNumDpbParams</u> ; i++) {	
dpb_size_only_flag[i]	u(1)
if(vps_max_sublayers_minus1 > 0 && !vps_all_layers_same_num_sublayers_flag)	
dpb_max_temporal_id[i]	u(3)
dpb_parameters(dpb_size_only_flag[i], dpb_max_temporal_id[i], vps_sublayer_dpb_params_present_flag)	
}	
for(i = 0; i < vps_max_layers_minus1 && <u>VpsNumDpbParams</u> > 1; i++) {	
if(<u>LayerUsedAsOutputLayerFlag[i]</u> && !vps_independent_layer_flag[i])	
layer_output_dpb_params_idx[i]	ue(v)
if(LayerUsedAsRefLayerFlag[i] && !same_dpb_size_output_or_nonoutput_flag)	

layer_nonoutput_dpb_params_idx[i]	ue(v)
}	
if(!each_layer_is_an_ols_flag)	
vps_general_hrd_params_present_flag	u(1)
if(vps_general_hrd_params_present_flag) {	
general_hrd_parameters()	
if(vps_max_sublayers_minus1 > 0)	
vps_sublayer_cpb_params_present_flag	u(1)
if(TotalNumOlss > 1)	
num_ols_hrd_params_minus1	ue(v)
for(i = 0; i <= num_ols_hrd_params_minus1; i++) {	
if(vps_max_sublayers_minus1 > 0 && !vps_all_layers_same_num_sublayers_flag)	
hrd_max_tid[i]	u(3)
firstSubLayer = vps_sublayer_cpb_params_present_flag ? 0 : hrd_max_tid[i]	
ols_hrd_parameters(firstSubLayer, hrd_max_tid[i])	
}	
if(num_ols_hrd_params_minus1 > 0)	
for(i = 1; i < TotalNumOlss && NumLayersInOls[i] > 1; i++)	
ols_hrd_idx[i]	ue(v)
}	
vps_extension_flag	u(1)
if(vps_extension_flag)	
while(more_rbsp_data())	
vps_extension_data_flag	u(1)
rbsp_trailing_bits()	
}	

7.4.3.2 Video parameter set RBSP semantics

ols_output_layer_flag[i][j] equal to 1 specifies that the layer with nuh_layer_id equal to vps_layer_id[j] is an output layer of the i-th OLS when ols_mode_idc is equal to 2. **ols_output_layer_flag[i][j]** equal to 0 specifies that the layer with nuh_layer_id equal to vps_layer_id[j] is not an output layer of the i-th OLS when ols_mode_idc is equal to 2.

The variable NumOutputLayersInOls[i], specifying the number of output layers in the i-th OLS, and the variable OutputLayerIdInOls[i][j], specifying the nuh_layer_id value of the j-th output layer in the i-th OLS, are derived as follows:

```

NumOutputLayersInOls[ 0 ] = 1
OutputLayerIdInOls[ 0 ][ 0 ] = vps_layer_id[ 0 ]
LayerUsedAsOutputLayerFlag[ 0 ] = 1
for( i = 1, i <= vps_max_layers_minus1; i++ ) {
    if( each_layer_is_an_ols_flag || ols_mode_idc < 2 )
        LayerUsedAsOutputLayerFlag[ i ] = 1
    if( ols_mode_idc == 2 )
        LayerUsedAsOutputLayerFlag[ i ] = 0
}
for( i = 1; i < TotalNumOlss; i++ ) {
    if( each_layer_is_an_ols_flag || ols_mode_idc == 0 ) {
        NumOutputLayersInOls[ i ] = 1
        OutputLayerIdInOls[ i ][ 0 ] = vps_layer_id[ i ]
    } else if( ols_mode_idc == 1 ) {
        NumOutputLayersInOls[ i ] = i + 1
        for( j = 0; j < NumOutputLayersInOls[ i ]; j++ )
            OutputLayerIdInOls[ i ][ j ] = vps_layer_id[ j ]
    } else if( ols_mode_idc == 2 ) {
        for( j = 0; j <= vps_max_layers_minus1; j++ )
            layerIncludedInOlsFlag[ i ][ j ] = 0
        for( k = 0; j = 0; k <= vps_max_layers_minus1; k++ )
            if( ols_output_layer_flag[ i ][ k ] ) {
                layerIncludedInOlsFlag[ i ][ k ] = 1
                LayerUsedAsOutputLayerFlag[ k ] = 1
                OutputLayerIdx[ i ][ j ] = k
                OutputLayerIdInOls[ i ][ j++ ] = vps_layer_id[ k ]
            }
        NumOutputLayersInOls[ i ] = j
        for( j = 0; j < NumOutputLayersInOls[ i ]; j++ ) {
            idx = OutputLayerIdx[ i ][ j ]
            for( k = 0; k < NumRefLayers[ idx ]; k++ )
                layerIncludedInOlsFlag[ i ][ RefLayerIdx[ idx ][ k ] ] = 1
        }
    }
}

```

(40)

```

    }
}

```

For each value of i in the range of 0 to $vps_max_layers_minus1$, inclusive, the values of $LayerUsedAsRefLayerFlag[i]$ and $LayerUsedAsOutputLayerFlag[i]$ shall not be both equal to 0. In other words, there shall be no layer that is neither a direct reference layer of any other layer nor an output layer of at least one OLS.

For each OLS, there shall be at least one layer that is an output layer. In other words, for any value of i in the range of 0 to $TotalNumOls - 1$, inclusive, the value of $NumOutputLayersInOls[i]$ shall be greater than or equal to 1.

The variable $NumLayersInOls[i]$, specifying the number of layers in the i -th OLS, and the variable $LayerIdInOls[i][j]$, specifying the nuh_layer_id value of the j -th layer in the i -th OLS, are derived as follows:

```

NumLayersInOls[ 0 ] = 1
LayerIdInOls[ 0 ][ 0 ] = vps_layer_id[ 0 ]
for( i = 1; i < TotalNumOls; i++ ) {
    if( each_layer_is_an_ols_flag ) {
        NumLayersInOls[ i ] = 1
        LayerIdInOls[ i ][ 0 ] = vps_layer_id[ i ]
        (41)
    } else if( ols_mode_idc == 0 || ols_mode_idc == 1 ) {
        NumLayersInOls[ i ] = i + 1
        for( j = 0; j < NumLayersInOls[ i ]; j++ )
            LayerIdInOls[ i ][ j ] = vps_layer_id[ j ]
    } else if( ols_mode_idc == 2 ) {
        for( k = 0, j = 0; k <= vps_max_layers_minus1; k++ )
            if( layerIncludedInOlsFlag[ i ][ k ] )
                LayerIdInOls[ i ][ j++ ] = vps_layer_id[ k ]
        NumLayersInOls[ i ] = j
    }
}

```

NOTE – The 0-th OLS contains only the lowest layer (i.e., the layer with nuh_layer_id equal to $vps_layer_id[0]$) and for the 0-th OLS the only included layer is output.

The variable $OlsLayerIdx[i][j]$, specifying the OLS layer index of the layer with nuh_layer_id equal to $LayerIdInOls[i][j]$, is derived as follows:

$$\begin{aligned} & \text{for}(i = 0; i < TotalNumOls; i++) \\ & \quad \text{for}(j = 0; j < NumLayersInOls[i]; j++) \\ & \quad \quad OlsLayerIdx[i][LayerIdInOls[i][j]] = j \end{aligned} \quad (42)$$

The lowest layer in each OLS shall be an independent layer. In other words, for each i in the range of 0 to $TotalNumOls - 1$, inclusive, the value of $vps_independent_layer_flag[GeneralLayerIdx[LayerIdInOls[i][0]]]$ shall be equal to 1.

Each layer shall be included in at least one OLS specified by the VPS. In other words, for each layer with a particular value of nuh_layer_id $nuhLayerId$ equal to one of $vps_layer_id[k]$ for k in the range of 0 to $vps_max_layers_minus1$, inclusive, there shall be at least one pair of values of i and j , where i is in the range of 0 to $TotalNumOls - 1$, inclusive, and j is in the range of $NumLayersInOls[i] - 1$, inclusive, such that the value of $LayerIdInOls[i][j]$ is equal to $nuhLayerId$.

vps_num_ptls_minus1_plus1 specifies the number of `profile_tier_level()` syntax structures in the VPS.

pt_present_flag[i] equal to 1 specifies that profile, tier, and general constraints information are present in the i -th `profile_tier_level()` syntax structure in the VPS. **pt_present_flag[i]** equal to 0 specifies that profile, tier, and general constraints information are not present in the i -th `profile_tier_level()` syntax structure in the VPS. The value of **pt_present_flag[0]** is inferred to be equal to 1. When **pt_present_flag[i]** is equal to 0, the profile, tier, and general constraints information for the i -th `profile_tier_level()` syntax structure in the VPS are inferred to be the same as that for the $(i - 1)$ -th `profile_tier_level()` syntax structure in the VPS.

ptl_max_temporal_id[i] specifies the `TemporalId` of the highest sublayer representation for which the level information is present in the i -th `profile_tier_level()` syntax structure in the VPS. The value of **ptl_max_temporal_id[i]** shall be in the range of 0 to $vps_max_sublayers_minus1$, inclusive. When $vps_max_sublayers_minus1$ is equal to 0, the value of **ptl_max_temporal_id[i]** is inferred to be equal to 0. When $vps_max_sublayers_minus1$ is greater than 0 and **vps_all_layers_same_num_sublayers_flag** is equal to 1, the value of **ptl_max_temporal_id[i]** is inferred to be equal to $vps_max_sublayers_minus1$.

vps_ptl_byte_alignment_zero_bit shall be equal to 0.

ols_ptl_idx[i] specifies the index, to the list of `profile_tier_level()` syntax structures in the VPS, of the `profile_tier_level()` syntax structure that applies to the i -th OLS. When present, the value of **ols_ptl_idx[i]**

shall be in the range of 0 to **vps_num_ptls_minus1**, inclusive. **When vps_num_ptls_minus1 is equal to 0, the value of ols_ptl_idx[i] is inferred to be equal to 0.**

When NumLayersInOls[i] is equal to 1, the profile_tier_level() syntax structure that applies to the i-th OLS is **also** present in the SPS referred to by the layer in the i-th OLS.

vps_num_dpb_params_minus1 plus 1 specifies the number of dpb_parameters() syntax structures in the VPS. When present, the value of **vps_num_dpb_params_minus1** shall be in the range of 0 to 15, inclusive.

The variable VpsNumDpbParams is derived as follows:

if(!vps_all_independent_layers_flag)

VpsNumDpbParams = vps_num_dpb_params_minus1 + 1 (42)

else

VpsNumDpbParams = 0

same_dpb_size_output_or_nonoutput_flag equal to 1 specifies that there is no layer_nonoutput_dpb_params_idx[i] syntax element present in the VPS. **same_dpb_size_output_or_nonoutput_flag** equal to 0 specifies that there may or may not be layer_nonoutput_dpb_params_idx[i] syntax elements present in the VPS. **When not present, the value of same_dpb_size_output_or_nonoutput_flag is inferred to be equal to 1.**

vps_sublayer_dpb_params_present_flag is used to control the presence of max_dec_pic_buffering_minus1[], max_num_reorder_pics[], and max_latency_increase_plus1[] syntax elements in the dpb_parameters() syntax structures in the VPS. When not present, vps_sub_dpb_params_info_present_flag is inferred to be equal to 0.

dpb_size_only_flag[i] equal to 1 specifies that the max_num_reorder_pics[] and max_latency_increase_plus1[] syntax elements are not present in the i-th dpb_parameters() syntax structures in the VPS. **dpb_size_only_flag[i]** equal to 0 specifies that the max_num_reorder_pics[] and max_latency_increase_plus1[] syntax elements may be present in the i-th dpb_parameters() syntax structures in the VPS.

dpb_max_temporal_id[i] specifies the TemporalId of the highest sublayer representation for which the DPB parameters may be present in the i-th dpb_parameters() syntax structure in the VPS. The value of **dpb_max_temporal_id[i]** shall be in the range of 0 to vps_max_sublayers_minus1, inclusive. When vps_max_sublayers_minus1 is equal to 0, the value of **dpb_max_temporal_id[i]** is inferred to be equal to 0. When vps_max_sublayers_minus1 is greater than 0 and vps_all_layers_same_num_sublayers_flag is equal to 1, the value of **dpb_max_temporal_id[i]** is inferred to be equal to vps_max_sublayers_minus1.

layer_output_dpb_params_idx[i] specifies the index, to the list of **dpb_parameters()** syntax structures in the VPS, of the **dpb_parameters()** syntax structure that applies to the i-th layer when it is an output layer in an OLS. When present, the value of **layer_output_dpb_params_idx[i]** shall be in the range of 0 to **VpsNumDpbParams** – 1, inclusive.

If **vps_independent_layer_flag[i]** is equal to 1, the **dpb_parameters()** syntax structure that applies to the i-th layer when it is an output layer is the **dpb_parameters()** syntax structure present in the SPS referred to by the layer.

Otherwise (**vps_independent_layer_flag[i]** is equal to 0), the following applies:

- When **VpsNumDpbParams** is equal to 1, the value of **layer_output_dpb_params_idx[i]** is inferred to be equal to 0.
- It is a requirement of bitstream conformance that the value of **layer_output_dpb_params_idx[i]** shall be such that **dpb_size_only_flag[layer_output_dpb_params_idx[i]]** is equal to 0.

layer_nonoutput_dpb_params_idx[i] specifies the index, to the list of **dpb_parameters()** syntax structures in the VPS, of the **dpb_parameters()** syntax structure that applies to the i-th layer when it is a non-output layer in an OLS. When present, the value of **layer_nonoutput_dpb_params_idx[i]** shall be in the range of 0 to **VpsNumDpbParams** – 1, inclusive.

If **same_dpb_size_output_or_nonoutput_flag** is equal to 1, the following applies:

- If **vps_independent_layer_flag[i]** is equal to 1, the **dpb_parameters()** syntax structure that applies to the i-th layer when it is a non-output layer is the **dpb_parameters()** syntax structure present in the SPS referred to by the layer.
- Otherwise (**vps_independent_layer_flag[i]** is equal to 0), the value of **layer_nonoutput_dpb_params_idx[i]** is inferred to be equal to **layer_output_dpb_params_idx[i]**.

Otherwise (**same_dpb_size_output_or_nonoutput_flag** is equal to 0), when **VpsNumDpbParams** is equal to 1, the value of **layer_output_dpb_params_idx[i]** is inferred to be equal to 0.

vps_general_hrd_params_present_flag equal to 1 specifies that the syntax structure **general_hrd_parameters()** and other HRD parameters are present in the VPS RBSP syntax structure.

vps_general_hrd_params_present_flag equal to 0 specifies that the syntax structure **general_hrd_parameters()** and other HRD parameters are not present in the VPS RBSP syntax structure.

When not present, the value of vps_general_hrd_params_present_flag is inferred to be equal to 0.

vps_sublayer_cpb_params_present_flag equal to 1 specifies that the *i*-th **ols_hrd_parameters()** syntax structure in the VPS contains HRD parameters for the sublayer representations with TemporalId in the range of 0 to **hrd_max_tid[i]**, inclusive. **vps_sublayer_cpb_params_present_flag** equal to 0 specifies that the *i*-th **ols_hrd_parameters()** syntax structure in the VPS contains HRD parameters for the sublayer representation with TemporalId equal to **hrd_max_tid[i]** only. When **vps_max_sublayers_minus1** is equal to 0, the value of **vps_sublayer_cpb_params_present_flag** is inferred to be equal to 0.

When **vps_sublayer_cpb_params_present_flag** is equal to 0, the HRD parameters for the sublayer representations with TemporalId in the range of 0 to **hrd_max_tid[i] - 1**, inclusive, are inferred to be the same as that for the sublayer representation with TemporalId equal to **hrd_max_tid[i]**. These include the HRD parameters starting from the **fixed_pic_rate_general_flag[i]** syntax element till the **sublayer_hrd_parameters(i)** syntax structure immediately under the condition "if(**general_vcl_hrd_params_present_flag**)" in the **ols_hrd_parameters** syntax structure.

num_ols_hrd_params_minus1 plus 1 specifies the number of **ols_hrd_parameters()** syntax structures present in the **general_hrd_parameters()** syntax structure **when vps_general_hrd_params_present_flag is equal to 1**. The value of **num_ols_hrd_params_minus1** shall be in the range of 0 to 63, inclusive. *When TotalNumOlss is equal to 1, the value of num_ols_hrd_params_minus1 is inferred to be equal to 0.*

hrd_max_tid[i] specifies the TemporalId of the highest sublayer representation for which the HRD parameters are contained in the *i*-th **ols_hrd_parameters()** syntax structure. The value of **hrd_max_tid[i]** shall be in the range of 0 to **vps_max_sublayers_minus1**, inclusive. When **vps_max_sublayers_minus1** is equal to 0, the value of **hrd_max_tid[i]** is inferred to be equal to 0. When **vps_max_sublayers_minus1** is greater than 0 and **vps_all_layers_same_num_sublayers_flag** is equal to 1, the value of **hrd_max_tid[i]** is inferred to be equal to **vps_max_sublayers_minus1**.

ols_hrd_idx[i] specifies the index, **to the list of ols_hrd_parameters() syntax structures in the VPS**, of the **ols_hrd_parameters()** syntax structure that applies to the *i*-th OLS **when NumLayersInOls[i] is greater than 1**. When present, the value of **ols_hrd_idx[i]** shall be in the range of 0 to **num_ols_hrd_params_minus1**, inclusive. When **NumLayersInOls[i] is greater than 1 and num_ols_hrd_params_minus1 is equal to 0**, the value of **ols_hrd_idx[i]** is inferred to be equal to 0.

vps_extension_flag equal to 0 specifies that no **vps_extension_data_flag** syntax elements are present in the VPS RBSP syntax structure. **vps_extension_flag** equal to 1 specifies that there are **vps_extension_data_flag** syntax elements present in the VPS RBSP syntax structure.

vps_extension_data_flag may have any value. Its presence and value do not affect decoder conformance to profiles specified in this version of this Specification. Decoders conforming to this version of this Specification shall ignore all **vps_extension_data_flag** syntax elements.

6.1.2. SPS semantics

7.4.3.3 Sequence parameter set RBSP semantics

An SPS RBSP shall be available to the decoding process prior to it being referenced, included in at least one AU with TemporalId equal to 0 or provided through external means.

All SPS NAL units with a particular value of **sps_seq_parameter_set_id** in a CVS shall have the same content.

sps_decoding_parameter_set_id, when greater than 0, specifies the value of **dps_decoding_parameter_set_id** for the DPS referred to by the SPS. When **sps_decoding_parameter_set_id** is equal to 0, the SPS does not refer to a DPS and no DPS is referred to when decoding each CLVS referring to the SPS. The value of **sps_decoding_parameter_set_id** shall be the same in all SPSs that are referred to by coded pictures in a bitstream.

sps_video_parameter_set_id, when greater than 0, specifies the value of **vps_video_parameter_set_id** for the VPS referred to by the SPS. **The value of **sps_video_parameter_set_id** shall be the same in all SPSs that are referred to by coded pictures in a CVS and that have **sps_video_parameter_set_id** greater than 0.**

When **sps_video_parameter_set_id** is equal to 0, the following applies:

- The SPS does not refer to a VPS.
- No VPS is referred to when decoding each CLVS referring to the SPS.
- The value of **vps_max_layers_minus1** is inferred to be equal to 0.
- *The CVS shall contain only one layer (i.e., all VCL NAL unit in the CVS shall have the same value of **nuh_layer_id**).*
- The value of **GeneralLayerIdx[nuh_layer_id]** is inferred to be equal to 0.
- The value of **vps_independent_layer_flag[GeneralLayerIdx[nuh_layer_id]]** is inferred to be equal to 1.

When **vps_independent_layer_flag[GeneralLayerIdx[nuh_layer_id]]** is equal to 1, the SPS referred to by a CLVS with a particular **nuh_layer_id** value **nuhLayerId** shall have **nuh_layer_id** equal to **nuhLayerId**.

6.1.3. Slice header semantics

7.4.8.1 General slice header semantics

...

slice_type specifies the coding type of the slice according to Table 9.

Table 9 – Name association to slice_type

slice_type	Name of slice_type
0	B (B slice)
1	P (P slice)
2	I (I slice)

When nal_unit_type is in the range of IDR_W_RADL to CRA_NUT, inclusive, **and vps independent layer flag[GeneralLayerIdx[nuh_layer_id]] is equal to 1**, slice_type shall be equal to 2.

[0044] FIG. 1 is a block diagram showing an example video processing system 1900 in which various techniques disclosed herein may be implemented. Various implementations may include some or all of the components of the system 1900. The system 1900 may include input 1902 for receiving video content. The video content may be received in a raw or uncompressed format, e.g., 8 or 10 bit multi-component pixel values, or may be in a compressed or encoded format. The input 1902 may represent a network interface, a peripheral bus interface, or a storage interface. Examples of network interface include wired interfaces such as Ethernet, passive optical network (PON), etc. and wireless interfaces such as Wi-Fi or cellular interfaces.

[0045] The system 1900 may include a coding component 1904 that may implement the various coding or encoding methods described in the present document. The coding component 1904 may reduce the average bitrate of video from the input 1902 to the output of the coding component 1904 to produce a coded representation of the video. The coding techniques are therefore sometimes called video compression or video transcoding techniques. The output of the coding component 1904 may be either stored, or transmitted via a communication connected, as represented by the component 1906. The stored or communicated bitstream (or coded) representation of the video received at the input 1902 may be used by the component 1908 for generating pixel values or displayable video that is sent to a display interface 1910. The process of generating user-viewable video from the bitstream is sometimes called video decompression. Furthermore, while certain video processing operations are referred to as “coding” operations or tools, it will be appreciated that the coding tools or operations are used at an encoder and corresponding decoding tools or operations that reverse the results of the coding will be performed by a decoder.

[0046] Examples of a peripheral bus interface or a display interface may include universal serial bus (USB) or high definition multimedia interface (HDMI) or Displayport, and so on. Examples of storage interfaces include SATA (serial advanced technology attachment), PCI, IDE interface, and the like. The techniques described in the present document may be embodied in various electronic devices such as mobile phones, laptops, smartphones or other devices that are capable of performing digital data processing and/or video display.

[0047] FIG. 2 is a block diagram of a video processing apparatus 3600. The apparatus 3600 may be used to implement one or more of the methods described herein. The apparatus 3600 may be embodied in a smartphone, tablet, computer, Internet of Things (IoT) receiver, and so on. The apparatus 3600 may include one or more processors 3602, one or more memories 3604 and video processing hardware 3606. The processor(s) 3602 may be configured to implement one or more methods described in the present document. The memory (memories) 3604 may be used for storing data and code used for implementing the methods and techniques described herein. The video processing hardware 3606 may be used to implement, in hardware circuitry, some techniques described in the present document. In some embodiments, the hardware 3606 may be partially or fully located within the processor(s) 3602, e.g., a graphics processor.

[0048] FIG. 4 is a block diagram that illustrates an example video coding system 100 that may utilize the techniques of this disclosure.

[0049] As shown in FIG. 4, video coding system 100 may include a source device 110 and a destination device 120. Source device 110 generates encoded video data which may be referred to as a video encoding device. Destination device 120 may decode the encoded video data generated by source device 110 which may be referred to as a video decoding device.

[0050] Source device 110 may include a video source 112, a video encoder 114, and an input/output (I/O) interface 116.

[0051] Video source 112 may include a source such as a video capture device, an interface to receive video data from a video content provider, and/or a computer graphics system for generating video data, or a combination of such sources. The video data may comprise one or more pictures. Video encoder 114 encodes the video data from video source 112 to generate a bitstream. The bitstream may include a sequence of bits that form a coded representation of the video data. The bitstream may include coded pictures and associated data. The coded picture is a coded representation of a picture. The associated data may include sequence parameter sets, picture

parameter sets, and other syntax structures. I/O interface 116 may include a modulator/demodulator (modem) and/or a transmitter. The encoded video data may be transmitted directly to destination device 120 via I/O interface 116 through network 130a. The encoded video data may also be stored onto a storage medium/server 130b for access by destination device 120.

[0052] Destination device 120 may include an I/O interface 126, a video decoder 124, and a display device 122.

[0053] I/O interface 126 may include a receiver and/or a modem. I/O interface 126 may acquire encoded video data from the source device 110 or the storage medium/ server 130b. Video decoder 124 may decode the encoded video data. Display device 122 may display the decoded video data to a user. Display device 122 may be integrated with the destination device 120, or may be external to destination device 120 which be configured to interface with an external display device.

[0054] Video encoder 114 and video decoder 124 may operate according to a video compression standard, such as the High Efficiency Video Coding (HEVC) standard, Versatile Video Coding (VVC) standard and other current and/or further standards.

[0055] FIG. 5 is a block diagram illustrating an example of video encoder 200, which may be video encoder 114 in the system 100 illustrated in FIG. 4.

[0056] Video encoder 200 may be configured to perform any or all of the techniques of this disclosure. In the example of FIG. 5, video encoder 200 includes a plurality of functional components. The techniques described in this disclosure may be shared among the various components of video encoder 200. In some examples, a processor may be configured to perform any or all of the techniques described in this disclosure.

[0057] The functional components of video encoder 200 may include a partition unit 201, a predication unit 202 which may include a mode select unit 203, a motion estimation unit 204, a motion compensation unit 205 and an intra prediction unit 206, a residual generation unit 207, a transform unit 208, a quantization unit 209, an inverse quantization unit 210, an inverse transform unit 211, a reconstruction unit 212, a buffer 213, and an entropy encoding unit 214.

[0058] In other examples, video encoder 200 may include more, fewer, or different functional components. In an example, predication unit 202 may include an intra block copy (IBC) unit. The IBC unit may perform predication in an IBC mode in which at least one reference picture is a picture where the current video block is located.

[0059] Furthermore, some components, such as motion estimation unit 204 and motion compensation unit 205 may be highly integrated, but are represented in the example of FIG. 5 separately for purposes of explanation.

[0060] Partition unit 201 may partition a picture into one or more video blocks. Video encoder 200 and video decoder 300 may support various video block sizes.

[0061] Mode select unit 203 may select one of the coding modes, intra or inter, e.g., based on error results, and provide the resulting intra- or inter-coded block to a residual generation unit 207 to generate residual block data and to a reconstruction unit 212 to reconstruct the encoded block for use as a reference picture. In some example, Mode select unit 203 may select a combination of intra and inter predication (CIIP) mode in which the predication is based on an inter predication signal and an intra predication signal. Mode select unit 203 may also select a resolution for a motion vector (e.g., a sub-pixel or integer pixel precision) for the block in the case of inter-predication.

[0062] To perform inter prediction on a current video block, motion estimation unit 204 may generate motion information for the current video block by comparing one or more reference frames from buffer 213 to the current video block. Motion compensation unit 205 may determine a predicted video block for the current video block based on the motion information and decoded samples of pictures from buffer 213 other than the picture associated with the current video block.

[0063] Motion estimation unit 204 and motion compensation unit 205 may perform different operations for a current video block, for example, depending on whether the current video block is in an I slice, a P slice, or a B slice.

[0064] In some examples, motion estimation unit 204 may perform uni-directional prediction for the current video block, and motion estimation unit 204 may search reference pictures of list 0 or list 1 for a reference video block for the current video block. Motion estimation unit 204 may then generate a reference index that indicates the reference picture in list 0 or list 1 that contains the reference video block and a motion vector that indicates a spatial displacement between the current video block and the reference video block. Motion estimation unit 204 may output the reference index, a prediction direction indicator, and the motion vector as the motion information of the current video block. Motion compensation unit 205 may generate the predicted video block of the current block based on the reference video block indicated by the motion information of the current video block.

[0065] In other examples, motion estimation unit 204 may perform bi-directional prediction for the current video block, motion estimation unit 204 may search the reference pictures in list 0 for a reference video block for the current video block and may also search the reference pictures in list 1 for another reference video block for the current video block. Motion estimation unit 204 may then generate reference indexes that indicate the reference pictures in list 0 and list 1 containing the reference video blocks and motion vectors that indicate spatial displacements between the reference video blocks and the current video block. Motion estimation unit 204 may output the reference indexes and the motion vectors of the current video block as the motion information of the current video block. Motion compensation unit 205 may generate the predicted video block of the current video block based on the reference video blocks indicated by the motion information of the current video block.

[0066] In some examples, motion estimation unit 204 may output a full set of motion information for decoding processing of a decoder.

[0067] In some examples, motion estimation unit 204 may do not output a full set of motion information for the current video. Rather, motion estimation unit 204 may signal the motion information of the current video block with reference to the motion information of another video block. For example, motion estimation unit 204 may determine that the motion information of the current video block is sufficiently similar to the motion information of a neighboring video block.

[0068] In one example, motion estimation unit 204 may indicate, in a syntax structure associated with the current video block, a value that indicates to the video decoder 300 that the current video block has the same motion information as the another video block.

[0069] In another example, motion estimation unit 204 may identify, in a syntax structure associated with the current video block, another video block and a motion vector difference (MVD). The motion vector difference indicates a difference between the motion vector of the current video block and the motion vector of the indicated video block. The video decoder 300 may use the motion vector of the indicated video block and the motion vector difference to determine the motion vector of the current video block.

[0070] As discussed above, video encoder 200 may predictively signal the motion vector. Two examples of predictive signaling techniques that may be implemented by video encoder 200 include advanced motion vector predication (AMVP) and merge mode signaling.

[0071] Intra prediction unit 206 may perform intra prediction on the current video block. When intra prediction unit 206 performs intra prediction on the current video block, intra prediction unit 206 may generate prediction data for the current video block based on decoded samples of other video blocks in the same picture. The prediction data for the current video block may include a predicted video block and various syntax elements.

[0072] Residual generation unit 207 may generate residual data for the current video block by subtracting (e.g., indicated by the minus sign) the predicted video block(s) of the current video block from the current video block. The residual data of the current video block may include residual video blocks that correspond to different sample components of the samples in the current video block.

[0073] In other examples, there may be no residual data for the current video block for the current video block, for example in a skip mode, and residual generation unit 207 may not perform the subtracting operation.

[0074] Transform processing unit 208 may generate one or more transform coefficient video blocks for the current video block by applying one or more transforms to a residual video block associated with the current video block.

[0075] After transform processing unit 208 generates a transform coefficient video block associated with the current video block, quantization unit 209 may quantize the transform coefficient video block associated with the current video block based on one or more quantization parameter (QP) values associated with the current video block.

[0076] Inverse quantization unit 210 and inverse transform unit 211 may apply inverse quantization and inverse transforms to the transform coefficient video block, respectively, to reconstruct a residual video block from the transform coefficient video block. Reconstruction unit 212 may add the reconstructed residual video block to corresponding samples from one or more predicted video blocks generated by the prediction unit 202 to produce a reconstructed video block associated with the current block for storage in the buffer 213.

[0077] After reconstruction unit 212 reconstructs the video block, loop filtering operation may be performed reduce video blocking artifacts in the video block.

[0078] Entropy encoding unit 214 may receive data from other functional components of the video encoder 200. When entropy encoding unit 214 receives the data, entropy encoding unit 214

may perform one or more entropy encoding operations to generate entropy encoded data and output a bitstream that includes the entropy encoded data.

[0079] FIG. 6 is a block diagram illustrating an example of video decoder 300 which may be video decoder 114 in the system 100 illustrated in FIG. 4.

[0080] The video decoder 300 may be configured to perform any or all of the techniques of this disclosure. In the example of FIG. 5, the video decoder 300 includes a plurality of functional components. The techniques described in this disclosure may be shared among the various components of the video decoder 300. In some examples, a processor may be configured to perform any or all of the techniques described in this disclosure.

[0081] In the example of FIG. 6, video decoder 300 includes an entropy decoding unit 301, a motion compensation unit 302, an intra prediction unit 303, an inverse quantization unit 304, an inverse transformation unit 305, and a reconstruction unit 306 and a buffer 307. Video decoder 300 may, in some examples, perform a decoding pass generally reciprocal to the encoding pass described with respect to video encoder 200 (FIG. 5).

[0082] Entropy decoding unit 301 may retrieve an encoded bitstream. The encoded bitstream may include entropy coded video data (e.g., encoded blocks of video data). Entropy decoding unit 301 may decode the entropy coded video data, and from the entropy decoded video data, motion compensation unit 302 may determine motion information including motion vectors, motion vector precision, reference picture list indexes, and other motion information. Motion compensation unit 302 may, for example, determine such information by performing the AMVP and merge mode.

[0083] Motion compensation unit 302 may produce motion compensated blocks, possibly performing interpolation based on interpolation filters. Identifiers for interpolation filters to be used with sub-pixel precision may be included in the syntax elements.

[0084] Motion compensation unit 302 may use interpolation filters as used by video encoder 20 during encoding of the video block to calculate interpolated values for sub-integer pixels of a reference block. Motion compensation unit 302 may determine the interpolation filters used by video encoder 200 according to received syntax information and use the interpolation filters to produce predictive blocks.

[0085] Motion compensation unit 302 may use some of the syntax information to determine sizes of blocks used to encode frame(s) and/or slice(s) of the encoded video sequence, partition

information that describes how each macroblock of a picture of the encoded video sequence is partitioned, modes indicating how each partition is encoded, one or more reference frames (and reference frame lists) for each inter-encoded block, and other information to decode the encoded video sequence.

[0086] Intra prediction unit 303 may use intra prediction modes for example received in the bitstream to form a prediction block from spatially adjacent blocks. Inverse quantization unit 303 inverse quantizes, i.e., de-quantizes, the quantized video block coefficients provided in the bitstream and decoded by entropy decoding unit 301. Inverse transform unit 303 applies an inverse transform.

[0087] Reconstruction unit 306 may sum the residual blocks with the corresponding prediction blocks generated by motion compensation unit 202 or intra-prediction unit 303 to form decoded blocks. If desired, a deblocking filter may also be applied to filter the decoded blocks in order to remove blockiness artifacts. The decoded video blocks are then stored in buffer 307, which provides reference blocks for subsequent motion compensation/intra prediction and also produces decoded video for presentation on a display device.

[0088] A listing of solutions preferred by some embodiments is provided next.

[0089] The following solutions show example embodiments of techniques discussed in the previous section (e.g., item 1).

[0090] 1. A method of video processing (e.g., method 600 in FIG. 3), comprising: performing a conversion between a video comprising one or more scalable video layers and a coded representation of the video, wherein the coded representation conforms to a format rule, wherein the format rule specifies that a value of a field indicative of a slice type of a slice is set to indicate a type of intra slice in case that a corresponding network abstraction layer unit type is in a predetermined range and that a corresponding video layer flag indicates that a video layer corresponding to the slice does not use inter-layer prediction.

[0091] 2. The method of solution 1, wherein the slice type of the slice is set to 2.

[0092] 3. The method of solution 1, wherein the pre-determined range is from IDR_W_RADL to CRA_NUT, both inclusive.

[0093] 4. A method of video processing, comprising: performing a conversion between a video comprising one or more scalable video layers and a coded representation of the video, wherein the coded representation conforms to a format rule, wherein the format rule specifies that an indication

for an index to a syntax structure describing a profile and tier level of the one or more scalable video layers is inferred to be equal to 0 in case the index is excluded from the coded representation.

[0094] 5. The method of solution4, wherein the indication comprises an `ols_ptl_idx[ i ]` syntax element, where *i* is an integer.

[0095] 6. A method of video processing, comprising: performing a conversion between a video comprising one or more scalable video layers and a coded representation of the video, wherein the coded representation contains a plurality of output layer sets (OLSs), wherein each OLS is a set of layers in the coded representation with the set of layers specified to be output, wherein the coded representation conforms to a format rule, wherein the format rule specifies that a video parameter set of the coded representation is permitted to have repetitive information regarding a profile, a tier and a level of an OLS in case that the OLS comprises a single layer.

[0096] 7. The method of solution6, wherein the format rule specifies to signal a number field indicative of a total number of the set of information regarding a profile, a tier and a level for the plurality of OLSs, where the total number is at least one.

[0097] 8. The method of any of solutions 1-7, wherein the performing the conversion comprising encoding the video to generate the coded representation.

[0098] 9. The method of any of solutions 1-7, wherein the performing the conversion comprises parsing and decoding the coded representation to generate the video.

[0099] 10. A video decoding apparatus comprising a processor configured to implement a method recited in one or more of solutions 1 to 9.

[00100] 11. A video encoding apparatus comprising a processor configured to implement a method recited in one or more of solutions 1 to 9.

[00101] 12. A computer program product having computer code stored thereon, the code, when executed by a processor, causes the processor to implement a method recited in any of solutions 1 to 9.

[00102] 13. A method, apparatus or system described in the present document.

[00103] With respect to FIGS. 7A to 7I, the following listed solutions may be preferably implemented in some embodiments.

[00104] For example, the following solutions may be implemented according to item 1 in previous section.

[00105] 1. A method of video processing (e.g., method 710 depicted in FIG. 7A), comprising performing (712) a conversion between a video comprising one or more scalable video layers and a bitstream of the video, wherein the video comprises one or more video pictures comprising one or more slices; wherein the bitstream conforms to a format rule, wherein the format rule specifies that a value of a field indicative of a slice type of a slice is set to indicate a type of intra slice in case that a corresponding network abstraction layer unit type is in a predetermined range and that a corresponding video layer flag indicates that a video layer corresponding to the slice does not use inter-layer prediction.

[00106] 2. The method of solution 1, wherein the slice type of the slice is set to 2.

[00107] 3. The method of solution 1, wherein the pre-determined range is from IDR_W_RADL to CRA_NUT, both inclusive, wherein IDR_W_RADL indicates a slice having an instantaneous decoding refresh type and wherein CRA_NUT indicates a slice having a clean random access type.

[00108] 4. The method of any of solutions 1-3, wherein the corresponding video layer flag corresponds to `vps_independent_layer_flag[ GeneralLayerIdx[ nuh_layer_id ]]`, wherein `nuh_layer_id` corresponds to an identifier of the video layer, `GeneralLayerIdx[]` is an index of the video layer and `vps_independent_layer_flag` corresponds to the video layer flag.

[00109] 5. The method of any of solutions 1-4, wherein the performing the conversion comprising encoding the video into the bitstream.

[00110] 6. The method of any of solutions 1-4, wherein the performing the conversion comprises parsing and decoding the video from the bitstream.

[00111] 7. A method for storing bitstream of a video, comprising: generating a bitstream from the video comprising one or more scalable video layers; and storing the bitstream in a non-transitory computer-readable recording medium; wherein the video comprises one or more video pictures comprising one or more slices; wherein the bitstream conforms to a format rule, wherein the format rule specifies that a value of a field indicative of a slice type of a slice is set to indicate a type of I slice in case that a corresponding network abstraction layer unit type is in a predetermined range and that a corresponding video layer flag indicates that a video layer corresponding to the slice does not use inter-layer prediction.

[00112] For example, the following solutions may be implemented according to item 5 in previous section.

[00113] 1. A method of video processing(e.g., method 720 depicted in FIG. 7B), comprising: performing (722) a conversion between a video comprising a plurality of video layers and a bitstream of the video, wherein the bitstream comprises a plurality of output layer sets (OLSs), each comprising one or more of the plurality of scalable video layers, and the bitstream conforms to a format rule, wherein the format rule specifies that, for an OLS having a single layer, a profile-tier-level (PTL) syntax structure that indicates a profile, a tier and a level for the OLS is included in a video parameter set for the bitstream, and the PTL syntax structure for the OLS is also included in a sequence parameter set coded in the bitstream.

[00114] 2. The method of solution 1, wherein the format rule specifies that a syntax element is included in the video parameter set indicative of a number of PTL syntax structures coded in the video parameter set.

[00115] 3. The method of solution 2, wherein the number of PTL syntax structures coded in the video parameter set is equal to a value of the syntax element plus one.

[00116] 4. The method of solution 2, wherein the number of PTL syntax structures coded in the video parameter set is equal to a value of the syntax element.

[00117] 5. The method of solution 4, wherein the format rule specifies that the syntax element is coded in the video parameter set, during encoding, or parsed from the video parameter set, during decoding, when it is determined that one or more of the plurality of OLSs contain more than one video layer.

[00118] 6. The method of solution 4, wherein the format rule specifies that the syntax element is coded in the video parameter set, during encoding, or parsed from the video parameter set, during decoding, when it is determined that a number of video layers is greater than 0 and one or more of the video layers uses inter-layer prediction.

[00119] 7. The method of any of solutions 1 to 6, wherein the format rule further specified that, for an OLS having a single layer, a decoded picture buffer size parameter for the OLS is included in a video parameter set for the bitstream, and the decoded picture buffer size parameter for the OLS is also included in a sequence parameter set coded in the bitstream.

[00120] 8. The method of any of solutions 1 to 6, wherein the format rule further specified that, for an OLS having a single layer, parameter information of the decoded picture buffer for the OLS is included in a video parameter set for the bitstream, and the parameter information of the decoded picture buffer for the OLS is also included in a sequence parameter set coded in the bitstream.

[00121] For example, the following solutions may be implemented according to item 6 in previous section.

[00122] 9. A method of video processing (e.g., method 730 depicted in FIG. 7C), comprising: performing (732) a conversion between a video comprising a plurality of video layers and a bitstream of the video, wherein the bitstream comprises a plurality of output layer sets (OLSs), each of which comprises one or more of the plurality of video layers, and the bitstream conforms to a format rule, wherein the format rule specifies a relationship between occurrence of a number of profile-tier-level (PTL) syntax structures in a video parameter set for the bitstream and a byte alignment syntax field in the video parameter set; wherein each PTL syntax structure indicates a profile, a tier and a level of one or more of the plurality of OLSs.

[00123] 10. The method of solution 9, wherein the format rule specifies that, at least one PTL syntax structure is included in the video parameter set, and due to inclusion of the at least one PTL syntax structure, one or more instances of the byte alignment syntax field are included in the video parameter set.

[00124] 11. The method of solution 10, wherein the byte alignment syntax field is one bit.

[00125] 12. The method of solution 11, wherein the value of each of the one or more instances of byte alignment syntax field has a value 0.

[00126] For example, the following solutions may be implemented according to item 2 in previous section.

[00127] 13. A method of video processing (e.g., method 740 depicted in FIG. 7D), comprising: performing (742) a conversion between a video comprising a plurality of scalable video layers and a bitstream of the video, wherein the bitstream comprises a plurality of output layer sets (OLSs), each comprising one or more of the plurality of scalable video layers, and the bitstream conforms to a format rule, wherein the format rule specifies that: during encoding, a syntax element indicative of an index to a syntax structure describing a profile, a tier and a level of one or more of the plurality of OLSs is excluded from a video parameter set for the bitstream in case that a value of the index is zero, or, during decoding, the value is inferred to be zero in case that the syntax element is not present in the bitstream.

[00128] 14. The method of solution 13, wherein the index is to a syntax structure that indicates a profile, a tier and a level of at least one of the OLSs.

[00129] 15. A method for storing bitstream of a video, comprising: generating a bitstream from the video comprising a plurality of video layers, and storing the bitstream in a non-transitory computer-readable recording medium; wherein the bitstream comprises a plurality of output layer sets (OLSs), each comprising one or more of the plurality of video layers, and the bitstream conforms to a format rule, wherein the format rule specifies that, for an OLS having a single layer, a profile-tier-level (PTL) syntax structure that indicates a profile, a tier and a level for the OLS is included in a video parameter set for the bitstream, and the PTL syntax structure for the OLS is also included in a sequence parameter set coded in the bitstream.

[00130] 16. The method of any of solutions 1-14, wherein the performing the conversion comprises encoding the video into the bitstream; and the method further comprises storing the bitstream in a non-transitory computer-readable recording medium.

[00131] For example, the following solutions may be implemented according to item 3 in previous section.

[00132] 1. A method of video processing (e.g., method 750 depicted in FIG. 7E), comprising: performing (752) a conversion between a video comprising a plurality of video layers and a bitstream of the video, wherein the bitstream comprises a plurality of output layer sets (OLSs), each of which comprises one or more of the plurality of video layers, and the bitstream conforms to a format rule, wherein the format rule specifies that, for a layer i , where i is an integer, the bitstream includes a set of first syntax element indicative of a first variable indicating whether the layer i is included in at least one of the plurality of OLSs.

[00133] 2. The method of solution 1, wherein the format rule specifies that, in case that, the first variable for the layer i is equal to zero, meaning that the layer i is not included in any of the plurality of OLSs, the bitstream excludes a second set of syntax element indicative of decoded picture buffer parameters for the layer i .

[00134] 3. The method of any of solutions 1-2, wherein the format rule further specifies that the bitstream includes a third set of syntax element that indicative of a second variable indicating whether the layer i is used as a reference layer for at least one of the plurality of video layers, and wherein the format rule disallows the first variable and the second variable to have zero values.

[00135] 4. The method of solution 3, wherein, wherein the format rule disallows values of the first variable and the second variable are both equal to 0 which indicates that no layer that is neither a direct reference layer of any other layer nor an output layer of at least one OLS.

[00136] 5. The method of any of solutions 1-4, wherein the first variable is a one-bit flag, denoted as LayerUsedAsOutputLayerFlag.

[00137] 6. The method of solution 5, wherein the first variable is determined based on iteratively checking, for each of the plurality of video layers, a value of a third variable indicative of a relationship between a number of layers included in the plurality of OLSs.

[00138] 7. The method of solution 6, wherein the third variable indicative of a relationship between number of layers included in the plurality of OLSs is permitted to have a value 0, 1, or 2.

[00139] 8. The method of any of solutions 1-7, wherein the performing the conversion comprising encoding the video into the bitstream; and the method further comprises storing the bitstream in a non-transitory computer-readable recording medium.

[00140] 9. A method for storing bitstream of a video, comprising: generating a bitstream from the video comprising a plurality of video layers, and storing the bitstream in a non-transitory computer-readable recording medium; wherein the bitstream comprises a plurality of output layer sets (OLSs), each of which comprises one or more of the plurality of video layers, and the bitstream conforms to a format rule, wherein the format rule specifies that, for a layer i , where i is an integer, the bitstream includes a set of first syntax element indicative of a first variable indicating whether the layer i is included in at least one of the plurality of OLSs.

[00141] For example, the following solutions may be implemented according to item 4 in previous section.

[00142] 1. A method of video processing (e.g., method 760 depicted in FIG. 7F), comprising: performing (762) a conversion between a video and a bitstream of the video, wherein the bitstream includes one or more output layer sets each comprising one or more video layers; wherein the bitstream conforms to a format rule, wherein the format rule specifies that a number of decoded picture buffer parameter syntax structures included in a video parameter set for the bitstream is equal to: zero, in a case that each output layer set includes a single video layer; or one plus a value of a syntax element, in a case that each output layer set includes a single layer is not true.

[00143] 2. The method of solution 1, wherein the syntax element corresponds to a `vps_num_dpb_params_minus1` syntax element.

[00144] 3. The method of any of solutions 1-2, wherein the format rule specifies that, in case that another syntax element indicative of whether a same size is used for indicating decoded picture buffer syntax structures for video layers that are included and not included in the one or more

output layer sets is absent in the video parameter set, a value of the another syntax element is inferred to be equal to 1.

[00145] 4. The method of any of solutions 1-3, wherein the performing the conversion comprises encoding the video into the bitstream; and the method further comprises storing the bitstream in a non-transitory computer-readable recording medium.

[00146] 5. A method for storing bitstream of a video, comprising: generating a bitstream from the video; and storing the bitstream in a non-transitory computer-readable recording medium; wherein the bitstream includes one or more output layer sets each comprising one or more video layers; wherein the bitstream conforms to a format rule, wherein the format rule specifies that a number of decoded picture buffer parameter syntax structures included in a video parameter set for the bitstream is equal to: zero, in a case that each output layer set includes a single video layer; or one plus a value of a syntax element, in a case that each output layer set includes a single layer is not true.

[00147] For example, the following solutions may be implemented according to item 7 in previous section.

[00148] 1. A method of video processing (e.g., method 770 depicted in FIG. 7G), comprising: performing (772) a conversion between a video and a bitstream of the video, wherein the bitstream includes a coded video sequence (CVS) comprising one or more coded video pictures of one or more video layers; and wherein the bitstream conforms to a format rule that specifies that one or more sequence parameter sets (SPS) indicative of conversion parameters that are referred to by the one or more coded pictures of the CVS have a same reference video parameter set (VPS) identifier indicative of a referenced VPS.

[00149] 2. The method of solution 1, wherein the format rule further specifies that the same reference VPS identifier has a value greater than 0.

[00150] 3. The method of any of solutions 1-2 wherein the format rule further specifies that a value zero for an SPS identifier is used in response to and only when the CVS comprises a single video layer.

[00151] 4. The method of any of solutions 1-3, wherein the performing the conversion comprises encoding the video into the bitstream; and the method further comprises storing the bitstream in a non-transitory computer-readable recording medium.

[00152] 5. A method for storing bitstream of a video, comprising: generating a bitstream from the video, and storing the bitstream in a non-transitory computer-readable recording medium; wherein the video comprises one or more video pictures comprising one or more slices; wherein the bitstream includes a coded video sequence (CVS) comprising one or more coded video pictures of one or more video layers; and wherein the bitstream conforms to a format rule that specifies that one or more sequence parameter sets (SPS) indicative of conversion parameters that are referred to by the one or more coded pictures of the CVS have a same reference video parameter set (VPS) identifier indicative of a referenced VPS.

[00153] For example, the following solutions may be implemented according to item 8 in previous section.

[00154] 1. A method of video processing (e.g., method 780 depicted in FIG. 7H), comprising: performing (782) a conversion between a video and a bitstream of the video, wherein the bitstream comprises one or more output layer sets (OLSs) each comprising one or more video layers, wherein the bitstream conforms to a format rule; wherein the format rule specifies whether or how a first syntax element indicating whether a first syntax structure descriptive of parameters of a hypothetical reference decoder (HRD) used for the conversion is included in a video parameter set (VPS) of the bitstream.

[00155] 2. The method of solution 1, wherein the first syntax structure comprises a set of general HRD parameters.

[00156] 3. The method of solutions 1- 2, wherein the format rule specifies that the first syntax element is omitted from the VPS and is inferred to have a zero value when not present in the VPS due to each of the one or more video layers being included in an OLS of the one or more OLSs, and wherein each of the one or more OLSs comprises a single video layer

[00157] 4. The method of solutions 1- 2, wherein the format rule specifies that the first syntax element has a zero value in the VPS when present in the VPS due to each of the one or more video layers being included in an OLS of the one or more OLSs, and wherein each of the one or more OLSs comprises a single video layer.

[00158] 5. The method of any of solutions 1-4, wherein the format rule further specifies whether or how a second syntax element indicative of a number of syntax structures descriptive of OLS-specific HRD parameters is included in the VPS.

[00159] 6. The method of solution 5, wherein the format rule further specifies that the second syntax element is included in the VPS when the first syntax element has a value 1, regardless of whether the total number of OLSs in the one or more OLSs is greater than 1.

[00160] For example, the following solutions may be implemented according to items 9, 10 in previous section.

[00161] 7. A method of video processing (e.g., method 790 depicted in FIG. 7I), comprising: performing (792) a conversion between a video and a bitstream of the video, wherein the bitstream comprises one or more output layer sets (OLSs) each comprising one or more video layers, wherein the bitstream conforms to a format rule; wherein the format rule specifies whether or how a first syntax structure descriptive of general hypothetical reference decoder (HRD) parameters and a number of second syntax structures descriptive of OLS-specific HRD parameters are included in a video parameter set (VPS) for the bitstream.

[00162] 8. The method of solution 7, wherein the format rule specifies to omit the first syntax structure from the VPS in case that no second syntax structure is included in the VPS.

[00163] 9. The method of solution 7, wherein the format rule excludes inclusion of first syntax structure and the second syntax structures in the VPS for an OLS comprising only one video layer, and wherein the format rule permits inclusion of the first syntax structure and the second syntax structures an OLS comprising only one video layer.

[00164] 10. A method for storing bitstream of a video, comprising: generating a bitstream from the video; storing the bitstream in a non-transitory computer-readable recording medium; wherein the bitstream comprises one or more output layer sets (OLSs) each comprising one or more video layers, wherein the bitstream conforms to a format rule; wherein the format rule specifies whether or how a first syntax element indicating whether a first syntax structure descriptive of parameters of a hypothetical reference decoder (HRD) used for the conversion is included in a video parameter set (VPS) of the bitstream.

[00165] The above-listed solutions may further include:

[00166] In some embodiments, in the above-listed solutions, the performing the conversion comprising encoding the video into the bitstream.

[00167] In some embodiments, in the above-listed solutions, the performing the conversion comprises parsing and decoding the video from the bitstream.

[00168] In some embodiments, in the above-listed solutions, the performing the conversion comprises encoding the video into the bitstream; and the method further comprises storing the bitstream in a non-transitory computer-readable recording medium.

[00169] In some embodiments, a video decoding apparatus includes a processor configured to implement a method recited in one or more of above-listed solutions.

[00170] In some embodiments, a video encoding apparatus includes a processor configured to implement a method recited in one or more of above-listed solutions.

[00171] In some embodiments, a non-transitory computer-readable storage medium may store instructions that cause a processor to implement a method recited in one or more of above-listed solutions.

[00172] In some embodiments, a non-transitory computer-readable recording medium storing a bitstream of a video which is generated by a method recited in one or more of above-listed solutions.

[00173] In some embodiments, the above-described encoding methods may be implemented by an apparatus and the apparatus may further write a bitstream generated by implementing the method to a computer-readable medium.

[00174] The disclosed and other solutions, examples, embodiments, modules and the functional operations described in this document can be implemented in digital electronic circuitry, or in computer software, firmware, or hardware, including the structures disclosed in this document and their structural equivalents, or in combinations of one or more of them. The disclosed and other embodiments can be implemented as one or more computer program products, i.e., one or more modules of computer program instructions encoded on a computer readable medium for execution by, or to control the operation of, data processing apparatus. The computer readable medium can be a machine-readable storage device, a machine-readable storage substrate, a memory device, a composition of matter effecting a machine-readable propagated signal, or a combination of one or more them. The term “data processing apparatus” encompasses all apparatus, devices, and machines for processing data, including by way of example a programmable processor, a computer, or multiple processors or computers. The apparatus can include, in addition to hardware, code that creates an execution environment for the computer program in question, e.g., code that constitutes processor firmware, a protocol stack, a database management system, an operating system, or a combination of one or more of them. A propagated signal is an artificially generated signal, e.g.,

a machine-generated electrical, optical, or electromagnetic signal, that is generated to encode information for transmission to suitable receiver apparatus.

[00175] A computer program (also known as a program, software, software application, script, or code) can be written in any form of programming language, including compiled or interpreted languages, and it can be deployed in any form, including as a stand-alone program or as a module, component, subroutine, or other unit suitable for use in a computing environment. A computer program does not necessarily correspond to a file in a file system. A program can be stored in a portion of a file that holds other programs or data (e.g., one or more scripts stored in a markup language document), in a single file dedicated to the program in question, or in multiple coordinated files (e.g., files that store one or more modules, sub programs, or portions of code). A computer program can be deployed to be executed on one computer or on multiple computers that are located at one site or distributed across multiple sites and interconnected by a communication network.

[00176] The processes and logic flows described in this document can be performed by one or more programmable processors executing one or more computer programs to perform functions by operating on input data and generating output. The processes and logic flows can also be performed by, and apparatus can also be implemented as, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application specific integrated circuit).

[00177] Processors suitable for the execution of a computer program include, by way of example, both general and special purpose microprocessors, and any one or more processors of any kind of digital computer. Generally, a processor will receive instructions and data from a read only memory or a random-access memory or both. The essential elements of a computer are a processor for performing instructions and one or more memory devices for storing instructions and data. Generally, a computer will also include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto optical disks, or optical disks. However, a computer need not have such devices. Computer readable media suitable for storing computer program instructions and data include all forms of non-volatile memory, media and memory devices, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks; magneto optical disks; and CD ROM and DVD-ROM disks. The

processor and the memory can be supplemented by, or incorporated in, special purpose logic circuitry.

[00178] While this patent document contains many specifics, these should not be construed as limitations on the scope of any subject matter or of what may be claimed, but rather as descriptions of features that may be specific to particular embodiments of particular techniques. Certain features that are described in this patent document in the context of separate embodiments can also be implemented in combination in a single embodiment. Conversely, various features that are described in the context of a single embodiment can also be implemented in multiple embodiments separately or in any suitable subcombination. Moreover, although features may be described above as acting in certain combinations and even initially claimed as such, one or more features from a claimed combination can in some cases be excised from the combination, and the claimed combination may be directed to a subcombination or variation of a subcombination.

[00179] Similarly, while operations are depicted in the drawings in a particular order, this should not be understood as requiring that such operations be performed in the particular order shown or in sequential order, or that all illustrated operations be performed, to achieve desirable results. Moreover, the separation of various system components in the embodiments described in this patent document should not be understood as requiring such separation in all embodiments.

[00180] Only a few implementations and examples are described and other implementations, enhancements and variations can be made based on what is described and illustrated in this patent document.