Alliance for Open Media
Codec Working Group

Document: **CWG-B078[o]_v1**

Tool Description for AV1 and libaom

Date: October 4, 2021

Status: Output document

Purpose: Information

Author(s): Xin Zhao, Shan Liu, Adrian Grange, Andrey Norkin

Email(s): xinzzhao@tencent.com, shanl@tencent.com, agrange@google.com, anorkin@netflix.com

Source: Tencent, Google, Netflix

Abstract

This document provides a description of the main coding features in *libaom*, a software implementation of the AV1 standard specification. Both normative decoding processes and some key encoder algorithms are described in this document.

CONTENTS

1 Introduction

The framework of the Alliance for Open Media Video 1 (AV1) codec is based on a hybrid video coding structure that consists of a few major function blocks, such as prediction, transform, quantization, entropy coding, and loop filtering. Each function block processes the input data using a certain type of video coding technology, and its output is fed into another function block or taken as the final output of the video codec. These function blocks are connected following a specific design and work collaboratively to achieve substantial data compression. The function blocks included in AV1 reference codec *libaom* [1] are summarized as follows and described in detail in Section 3.

- Block partitioning
  - Coding block partitioning [2]
  - Transform block partitioning [2]
- Intra prediction
  - Directional intra prediction [2]
  - Non-directional intra prediction [2]
  - Recursive intra prediction [2]
  - Chroma from luma (CfL) prediction [3]
  - Intra prediction mode signalling
- Inter prediction
  - Reference frame system [2]
  - Spatial motion vector prediction
  - Temporal motion vector prediction
  - Dynamic motion vector prediction [4]
  - Inter prediction mode signalling
  - Translational motion compensation
  - Warped motion compensation [5]
  - Overlapped block motion compensation [6]
  - Compound inter prediction
  - Compound inter-intra prediction
- Transform coding
  - Core transforms
  - Transform selection and signalling
- Quantization
- Entropy coding
  - Multi-symbol arithmetic coding engine [8]
  - Coefficient coding [9]
- Loop filtering and post-processing
  - Deblocking filter
  - Constrained directional enhancement filter [10]
  - Loop restoration filter [11]
  - Frame super-resolution
  - Film grain synthesis [12]
- Screen content coding
  - Intra block copy [13]

- Palette mode
- Encoder content-type detection

The coding features described for each building block are all included in the *libaom* [1] implementation of AV1 codec.

In this document, syntax elements are written using `Courier` font, e.g., syntax element `base_q_idx`.

2 Abbreviations

For the purposes of this document, the following abbreviations apply:

ARF	alternate reference frame
AV1	AOMedia Video 1
BV	block vector
CDEF	constrained directional enhancement filter
CfL	chroma from luma
DRL	dynamic reference list
EOB	end of block
FIR	finite impulse response
IntraBC	intra block copy
LR	loop restoration
LRU	loop restoration unit
MV	motion vector
OBMC	overlapped block motion compensation
SGF	self-guided filter

3 Tool description

3.1 Block partitioning

3.1.1 Coding block partitioning

Coding blocks of different sizes are used in AV1. The largest coding blocks, called superblocks, have sizes of either 128×128 or 64×64, with the default size being 128×128. The size is signalled in the sequence header. The minimum coding block size is 4×4.

Superblocks can be further partitioned into smaller coding blocks. The partitioning strategy is signalled in the bitstream. Besides the no-partitioning mode, PARTITION_NONE, there are up to nine supported partitioning modes (see Figure 1). These include three 4-partition modes: PARTITION_SPLIT, PARTITION_VERT_4, and PARTITION_HORZ_4; four 3-partition (T-shaped) modes: PARTITION_HORZ_A, PARTITION_HORZ_B, PARTITION_VERT_A, and PARTITION_VERT_B; and two 2-partition modes: PARTITION_HORZ and PARTITION_VERT.

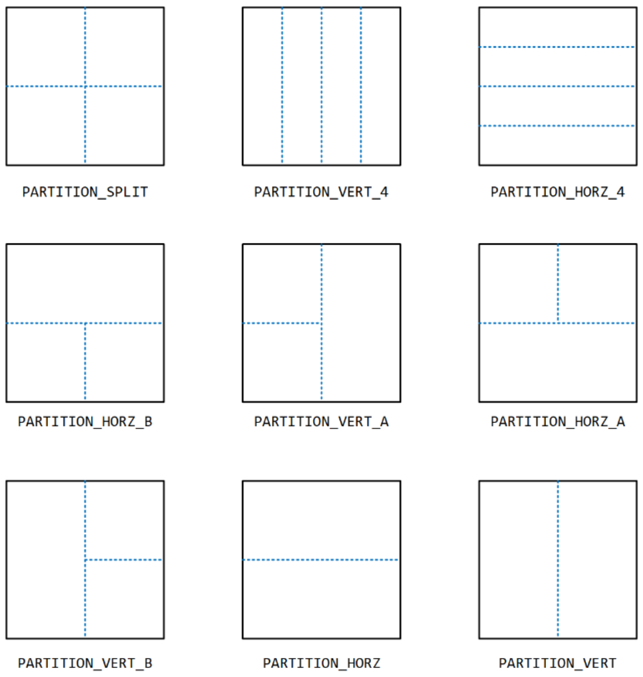

Figure 1: Coding block partitioning modes

Among all the partitioning modes, only PARTITION_SPLIT allows recursive partitioning; that is, the subpartitions can be further partitioned. For all other partitioning modes, the subpartitions cannot be further partitioned. Furthermore, PARTITION_VERT_4 and PARTITION_HORZ_4 modes are not allowed for 8×8 or 128×128 block sizes, and T-shaped partitioning modes are not allowed for 8×8 blocks.

3.1.2 Transform block partitioning

Both intra and inter coding blocks can be further partitioned into multiple transform blocks, with a partitioning depth of up to two levels. The transform block size is determined by the transform partitioning mode and the coding block size. The mapping from the transform size of the current depth to the transform size of the next depth is shown in Table 1.

Table 1: Transform partitioning size setting

Current depth		Next depth	
Enumerator	**Transform size**	**Enumerator**	**Transform size**
TX_4X4	4×4	TX_4X4	4×4
TX_8X8	8×8	TX_4X4	4×4
TX_16X16	16×16	TX_8X8	8×8
TX_32X32	32×32	TX_16X16	16×16
TX_64X64	64×64	TX_32X32	32×32
TX_4X8	4×8	TX_4X4	4×4
TX_8X4	8×4	TX_4X4	4×4
TX_8X16	8×16	TX_8X8	8×8
TX_16X8	16×8	TX_8X8	8×8

TX_16X32	16×32	TX_16X16	16×16
TX_32X16	32×16	TX_16X16	16×16
TX_32X64	32×64	TX_32X32	32×32
TX_64X32	64×32	TX_32X32	32×32
TX_4X16	4×16	TX_4X8	4×8
TX_16X4	16×4	TX_8X4	8×4
TX_8X32	8×32	TX_8X16	8×16
TX_32X8	32×8	TX_16X8	16×8
TX_16X64	16×64	TX_16X32	16×32
TX_64X16	64×16	TX_32X16	32×16

The transform types in Table 1 can be generalized as follows. For 1:1 square blocks, transform partitioning of the next level will create four 1:1 square transform blocks. For 1:2 or 2:1 non-square blocks, transform partitioning of the next level will create two 1:1 square transform blocks. Finally, for 1:4 or 4:1 non-square blocks, transform partitioning of the next level will create two 1:2 or 2:1 transform blocks, respectively.

For the luma colour component of intra coding blocks, all the transform blocks must have equal size. For example, for a 32×16 coding block, level 1 transform partitioning would create two 16×16 transform blocks, and level 2 transform partitioning would create eight 8×8 transform subblocks. An example of transform block partitioning for intra coding blocks is shown in Figure 2. The coding order of transform blocks is illustrated by arrows.

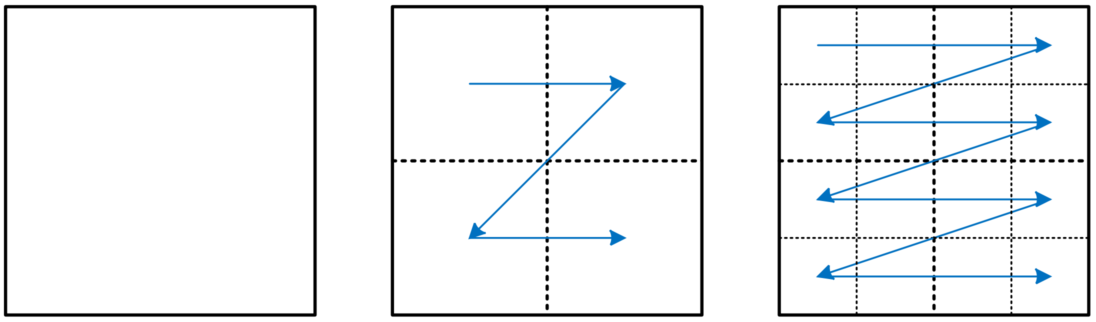

Figure 2: Example of transform partitioning for intra coding block

For the luma colour component of inter coding blocks, after level 1 transform partitioning, each subblock can be partitioned independently in level 2. Therefore, inter coding blocks can have variable transform block sizes, as shown in Figure 3.

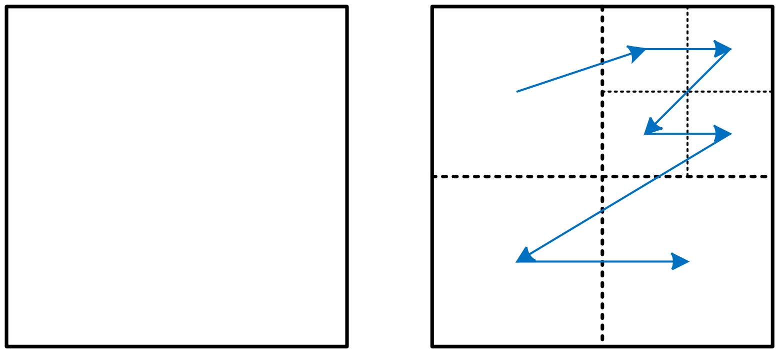

Figure 3: Example of transform partitioning for inter coding block

For the chroma colour components, the transform block size is the same as the corresponding chroma coding block size, except that when the chroma coding block width or heigh is greater than 32, the chroma transform block width or height is set equal to 32.

If the coding block size is smaller than or equal to 64×64, the above transform block partitioning scheme starts from the coding block size. However, if the coding block width *W* or height *H* is greater than 64 in terms of luma samples, it is first implicitly partitioned into multiples of min(*W*, 64)×min(*H*, 64) subblocks. No signalling is required for this step. Thereafter, starting from each min(*W*, 64)×min(*H*, 64) subblock, the transform block partitioning scheme described in this section applies, and the partitioning is explicitly signalled. For example, for a 128×64 coding block, the residual block is first implicitly partitioned into two 64×64 subblocks, then each 64×64 subblock can be further partitioned into smaller transform blocks, with the partitioning mode being explicitly signalled.

3.2 Intra prediction

3.2.1 Directional intra prediction

Directional intra prediction is used to model local textures using a set of edge directions. There are eight nominal directional intra prediction modes, each of which has an associated set of angle delta offsets indexed as integer values between −3 and +3, with the nominal angle situated at 0. The prediction direction is derived by adding the angle delta to the nominal intra angle. In total, there are 56 directional intra prediction modes. Figure 4 shows the eight nominal modes (solid arrows) with an example of the set of angle delta offsets around the D67_PRED nominal mode (dotted arrows) when the angle delta is nonzero.

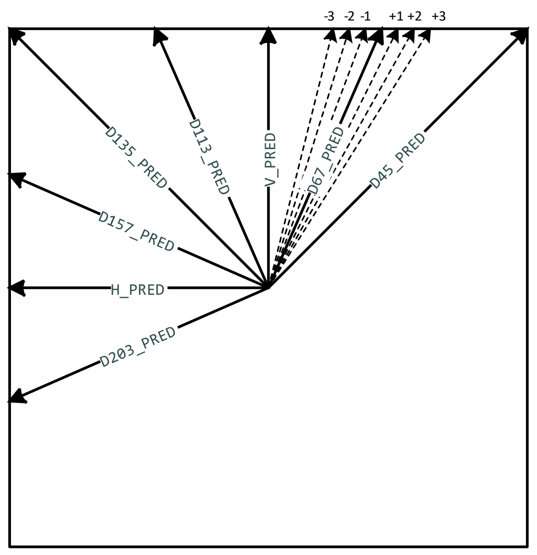

Figure 4: Directional intra prediction modes. The solid arrows indicate nominal intra prediction modes and the dashed arrows represent intra prediction modes with non-zero angle deltas.

Both the nominal mode and associated angle delta indices are signalled, with the nominal mode index signalled before the associated angle delta index. For small block sizes, i.e., 4×4, 4×8, and 8×4, the additional coding gain from extending the intra prediction angle precision is usually marginal; therefore, in such cases, only the nominal mode is used, with no angle delta applied or signalled.

3.2.2 Non-directional intra prediction

In addition to the directional intra prediction modes, there are five non-directional intra prediction modes, DC_PRED, SMOOTH, SMOOTH_H, SMOOTH_V, and Paeth, which are usually used for the prediction of smooth areas.

The DC_PRED mode generates prediction samples in the current block by averaging the reconstructed samples from the top and left neighbouring blocks.

The SMOOTH_V and SMOOTH_H modes generate prediction values using quadratic interpolation along the vertical and horizontal directions, respectively, while the SMOOTH mode generates prediction values using the average of the quadratic interpolation results along both directions. The samples used for quadratic interpolation include reconstructed samples from the top and left neighbouring reconstructed blocks and samples from the right and bottom boundaries that are estimated by the top and left reconstructed samples.

The Paeth prediction mode predicts each sample from its top (T), left (L), and top-left (TL) reference samples, as shown in Figure 5. Of these reference samples, the one with the value closest to the value of (T + L – TL) is selected as the prediction sample.

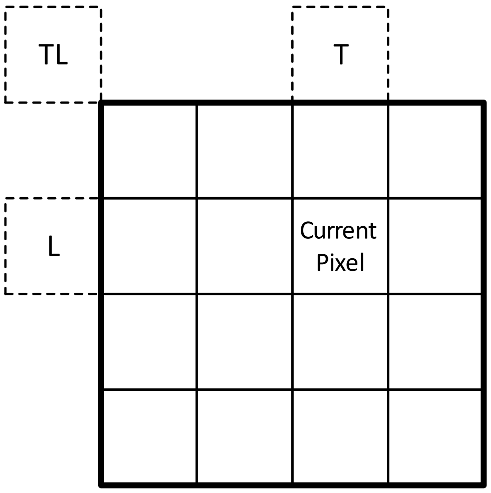

Figure 5: Reference samples used in Paeth prediction mode

3.2.3 Recursive intra prediction

Five recursive intra prediction modes are defined in AV1. Each mode specifies a set of eight 7-tap filters. Given the selected recursive intra prediction mode index (0,…,4), the current block is divided into multiples of 4×2 subblocks. For each 4×2 subblock, each sample is predicted by 7-tap interpolation using the seven neighbouring samples from the top and left blocks as inputs, as shown in Figure 6. Different filters are applied for samples located at different coordinates within each 4×2 subblock. This prediction process is performed on 4×2 subblocks one by one within a coding block, and the prediction samples generated for each 4×2 subblock can be used to predict the following 4×2 subblock. An example of recursive intra prediction is shown in Figure 6.

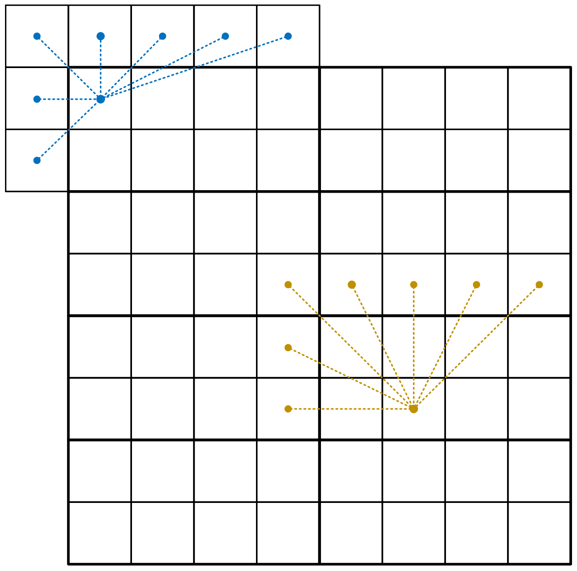

Figure 6: Illustration of recursive intra prediction mode

3.2.4 Chroma from luma prediction

Chroma from luma (CfL) is an intra prediction mode applied only to chroma coding blocks. The CfL prediction mode derives chroma prediction samples using their collocated reconstructed luma samples via a linear model. When the corresponding luma and chroma resolutions are different, e.g., for 4:2:0 and 4:2:2 chroma subsampling formats, the reconstructed luma samples need to be subsampled before feeding into the CfL mode. The prediction block is derived as the sum of the chroma DC contribution and the scaled luma AC contribution. The DC contribution of a block consists of the average value of the block, while the AC contribution is derived by removing the DC contribution from the block. In CfL mode, the model parameters, such as the scaling factor applied to the luma AC contribution, are calculated during the encoding process and signalled into the bitstream. A flowchart of the CfL prediction mode is presented in Figure 7.

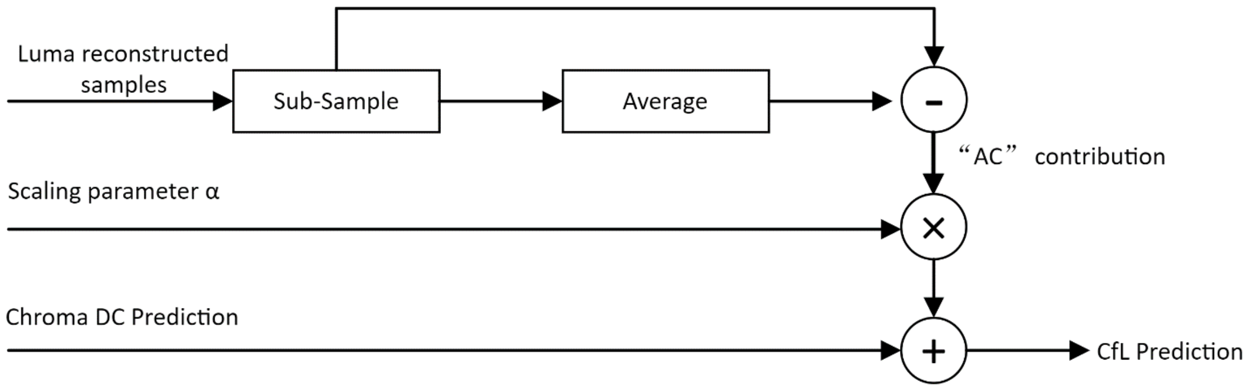

Figure 7: CfL prediction mode

3.2.5 Intra prediction mode signalling

For luma colour component, the intra prediction modes include the 56 directional intra prediction modes, 5 non-directional prediction modes, and 5 recursive filtering modes. The following process is used to signal the intra prediction mode for a coding block:

- Syntax `y_mode` is first signalled to indicate which of the eight nominal directional intra prediction modes or five non-directional intra prediction modes is to be applied.
- If the block size is greater than 8×8 and `y_mode` is indicating a directional intra prediction mode, i.e., 1,…,8, then syntax `angle_delta_y` is further signalled to indicate the angle delta index in the range of −3 to +3.
- Otherwise, if the luma prediction mode is DC_PRED and the maximum width and height of the coding block is less than or equal to 32, flag `use_filter_intra` is signalled to indicate whether a recursive intra prediction mode is to be applied.
- If `use_filter_intra` is signalled as 1, then `filter_intra_mode` is further signalled to indicate which of the five recursive intra prediction modes is to be applied.

Similarly, for chroma colour components, the intra prediction modes include the 56 directional intra prediction modes, 5 non-directional prediction modes, and CfL prediction mode. The following process is used to signal the intra prediction mode for a coding block:

- Syntax `uv_mode` is signalled to indicate which of the eight nominal directional intra prediction modes, five non-directional prediction modes, and CfL prediction mode is to be applied.
- If the block size is greater than 8×8 and `uv_mode` is indicating a directional intra prediction mode, then syntax `angle_delta_uv` is further signalled to indicate the angle delta index in the range of −3 to +3.
- Otherwise, if `uv_mode` is indicating the CfL mode as the intra prediction mode, a scaling parameter *α* is used, which is further signalled for Cb and Cr colour components.

The mapping between the values of `y_mode`, `uv_mode`, and intra prediction modes is listed in Table 2.

Table 2: Mapping between `y_mode`/`uv_mode` and intra prediction modes

`y_mode`	**`uv_mode`**	**Intra prediction mode**
0	0	DC_PRED
1	1	V_PRED
2	2	H_PRED
3	3	D45_PRED
4	4	D135_PRED
5	5	D113_PRED
6	6	D157_PRED
7	7	D203_PRED
8	8	D67_PRED
9	9	SMOOTH
10	10	SMOOTH_V
11	11	SMOOTH_H
12	12	Paeth
N/A	13	CfL

For signalling `y_mode`, different contexts are applied for coding blocks in intra and inter frames. In intra frames, the context for signalling `y_mode` is derived from the neighbouring luma intra prediction modes. In inter frames, the context for signalling `y_mode` is derived from the current coding block size. For signalling `uv_mode`, the context is derived using the collocated luma intra prediction mode.

3.3 Inter prediction

3.3.1 Reference frame system

A maximum of eight frames can be stored in the decoded frame buffer. To code each frame, up to two reference frames can be selected from the decoded frame buffer for use as reference frames for inter prediction. Single reference inter prediction uses only one frame as the reference frame. Compound inter prediction uses two reference frames, and generates a prediction by combining two prediction blocks from these reference frames.

Uni-directional and bi-directional compound prediction are special cases of compound inter prediction. Uni-directional compound prediction is when both the selected reference frames are either preceding or following the current frame in display order, whereas bi-directional compound prediction is when one reference frame precedes the current frame and the other follows the current frame in display order. For uni-directional compound prediction, the applicable reference frame combinations are limited to four combinations, whereas for bi-directional compound prediction, all 12 possible reference frame combinations are supported. After the coding is finished for one frame, the encoder decides which reference frame stored in the decoded frame buffer needs to be replaced, and this update is explicitly signalled in the bitstream.

The decoded frame buffer stores four types of reference frames, namely, 1) LAST frame: a frame that was displayed in the near past; 2) BWD frame: a frame that will be displayed in the future; 3) GOLDEN frame: a frame that was displayed in the distant past; and 4) alternate reference frame (ARF): a frame from either the past or the future. ARFs also have the option of not being displayed. ARFs can be synthesized by the encoder, for example, they can be generated by temporal filtering along the motion trajectories of multiple original frames.

3.3.2 Spatial motion vector prediction

Spatial motion vector (MV) predictors can be identified by utilizing spatial neighbouring blocks, including adjacent spatial neighbouring blocks, which are direct neighbours of the current block to the top and left sides, as well as non-adjacent spatial neighbouring blocks, which are close but not directly adjacent to the current block. An example of a set of spatial neighbouring blocks for a luma block is illustrated in Figure 8, wherein each spatial neighbouring block is an 8×8 block.

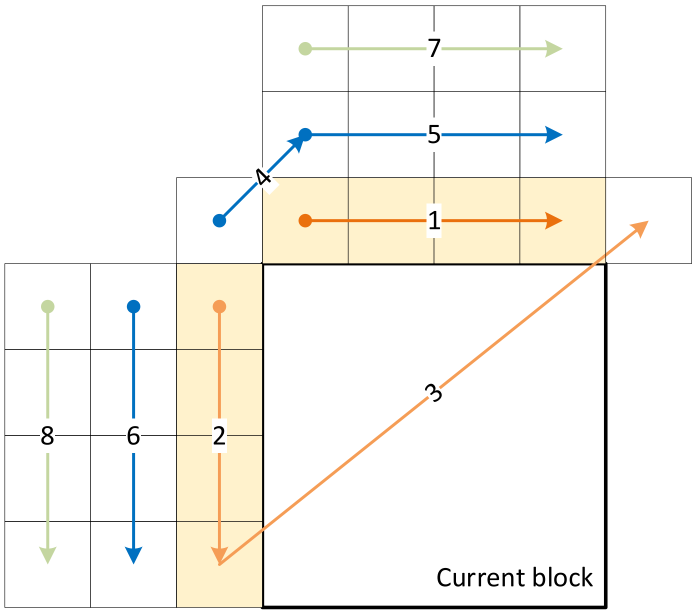

Figure 8: Spatial neighbouring motion for MV prediction

The spatial neighbouring blocks are examined to find one or multiple MVs that are associated with the same reference frame index as the current block. For the current block, the search order of spatial neighbouring 8×8 luma blocks is as indicated by the numbers 1–8 in Figure 8:

1. The top adjacent row is checked from left to right.

2. The left adjacent column is checked from top to bottom.
3. The top-right neighbouring block is checked.
4. The top-left block neighbouring block is checked.
5. The first top non-adjacent row is checked from left to right.
6. The first left non-adjacent column is checked from top to bottom.
7. The second top non-adjacent row is checked from left to right.
8. The second left non-adjacent column is checked from top to bottom.

In step 1, the bottom two 4×4 blocks of each 8×8 neighbouring block are checked. In step 2, the right two 4×4 blocks of each 8×8 neighbouring block are checked. In step 3, the bottom-left 4×4 block of the top-right 8×8 neighbouring block is checked. After checking the top-right neighbouring block, the temporal MV predictor (described in Section 3.3.3) is checked. Thereafter, in steps 4–8, the bottom-right 4×4 block of each 8×8 neighbouring block is checked.

In single reference inter prediction, the spatial MV predictor is generated by identifying the spatial neighbouring blocks that are predicted using the same single reference frame as the current block, and their associated MVs are used as the spatial MV predictor, as shown in Figure 9.

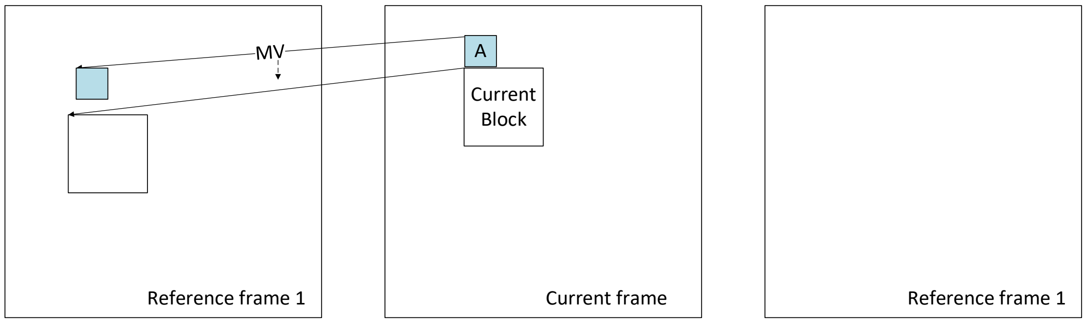

Figure 9: Spatial MV predictor generation when both the current block and a neighbouring block A are predicted using the same single reference frame

In compound prediction, when a spatial neighbouring block is predicted by the compound prediction mode using the same reference frames as the current block, the associated MVs can be used as the spatial MV predictor, as shown in Figure 10.

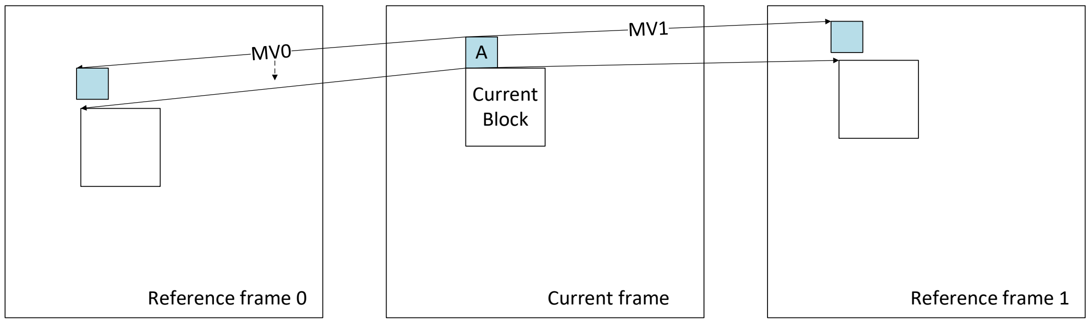

Figure 10: Spatial MV predictor generation when both the current block and a neighbouring block A are predicted using the compound prediction mode, where the reference frames of the neighbouring block are the same as those of the current block

For compound prediction, which utilizes two reference frames, non-adjacent spatial neighbours are not checked when deriving the MV predictor.

3.3.3 Temporal motion vector prediction

In addition to spatial neighbouring blocks, MV predictors known as temporal MV predictors can also be derived using collocated blocks of reference pictures. To generate temporal MV predictors, the MVs of reference frames are first stored with reference indices associated with the respective reference frames. Thereafter, for each 8×8 block of the current frame, the MVs of a reference frame that pass the 8×8 block are identified and stored with the reference frame index in a temporal MV buffer.

An example of this is shown in Figure 11. In this example, the MV of reference frame 1 (R1; right-hand side of Figure 11), i.e., MV_{ref}, points from R1 to a reference frame of R1 (left-hand side of Figure 11). In doing so, it passes through an 8×8 block of the current frame. MV_{ref} is stored in the temporal MV buffer associated with this 8×8 block. During the motion projection process for deriving the temporal MV predictor, the reference frames are scanned in a predefined order: LAST_FRAME, BWDREF_FRAME, ALTREF2_FRAME, ALTREF_FRAME, and LAST2_FRAME. The MVs from later reference frames (in scanning order) replace the previously identified MVs.

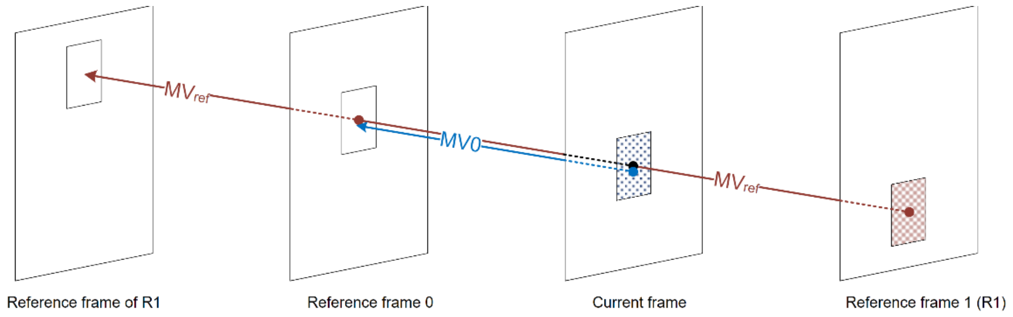

Figure 11: Motion field estimation by linear projection

Finally, given the predefined block coordinates, the associated MVs stored in the temporal MV buffer are identified and projected to derive a temporal MV predictor that points from the current block to its reference frame, e.g., MV0 in Figure 11. In Figure 12, the pre-defined block positions for deriving temporal MV predictors of a 16×16 block are shown. Up to seven blocks are checked to find valid temporal MV predictors. The temporal MV predictors are checked after the nearest spatial MV predictors but before the non-adjacent spatial MV predictors.

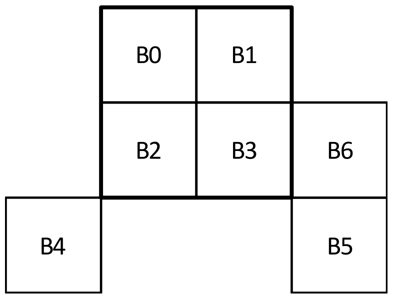

Figure 12: Block positions for deriving temporal MV predictors

For the derivation of MV predictors, all the spatial and temporal MV candidates are pooled, and each predictor is assigned a weighting that is determined during the scanning of the spatial and temporal neighbouring blocks. Based on the associated weightings, the candidates are sorted and ranked, and up to four candidates are identified and added to an MV predictor list. This list of

MV predictors is also referred to as a dynamic reference list (DRL), which is further used in dynamic MV prediction modes, as described in the next subsection.

3.3.4 Dynamic motion vector prediction

MVs can be predicted by either spatial neighbouring blocks in the current frame or temporal neighbouring blocks in a reference frame. A set of up to four MV predictors can be identified by checking all these blocks. For single reference inter prediction, there are separate lists of MVs for each reference frame. For compound inter prediction, the list of MV pairs for each reference pair is used to derive the MV predictors. In the bitstream for single reference prediction, a DRL index `drl_idx` is signalled to specify which of the following MV prediction modes is to be used (see also Figure 13):

- NEARESTMV: Always use the 0-indexed entry from the MV predictor list.
- NEARMV: use one of the 1, 2, or 3-indexed entries, a ternary DRL index is signalled to indicate which entry is used as the MV predictor.
- NEWMV: Use the 0-, 1-, or 2-indexed entry. A ternary DRL index is signalled to indicate which entry is to be used as the MV predictor. An MV difference (MVD) to the MV predictor is signalled.
- GLOBALMV: Use an MV based on frame-level global motion parameters as the MV predictor.

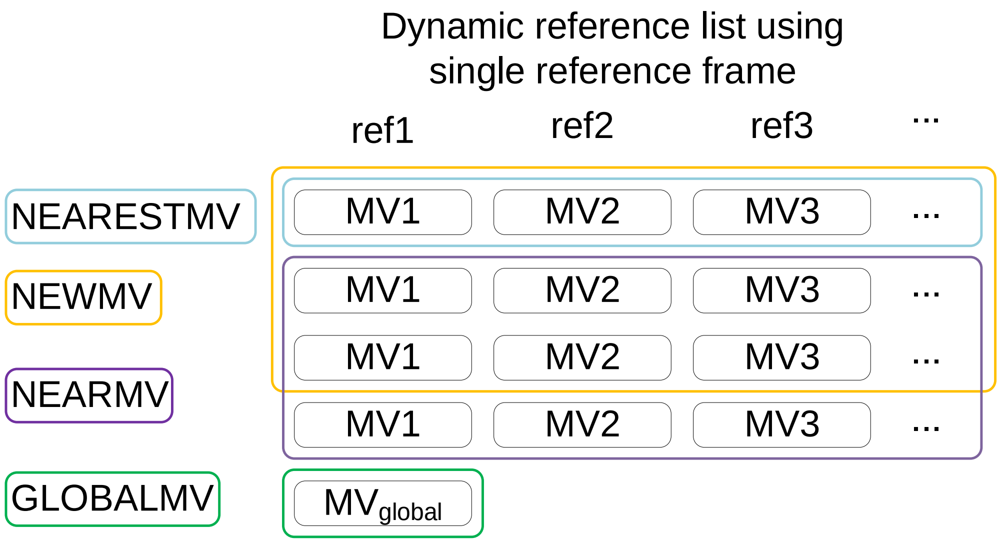

Figure 13: Motion vector prediction modes for single reference frame case

For compound inter prediction modes, the DRL index `drl_idx` is signalled to specify which of the following modes is to be used (see also Figure 14):

- NEAREST_NEARESTMV: Always use the 0-indexed MV pair from the list.
- NEAR_NEARMV: Use the 1-, 2-, or 3-indexed MV pair signalled by a ternary DRL index.
- NEAREST_NEWMV: Always use the 0-indexed MV pair from the list as the MV predictor. An MVD is signalled for the second MV.
- NEW_NEARESTMV: Always use the 0-indexed MV pair from the list as the MV predictor. An MVD is signalled for the first MV.

- NEAR_NEWMV: Use the 1-, 2-, or 3-indexed MV pair signalled by a ternary DRL index as an MV predictor. An MVD is signalled for the second MV.
- NEW_NEARMV: Use the 1-, 2-, or 3-indexed MV pair signalled by a ternary DRL index as an MV predictor. An MVD is signalled for the first MV.
- NEW_NEWMV: Use the 0-, 1-, or 2-indexed MV pair signalled by a ternary DRL index as an MV predictor. MVDs are signalled for both the first and second MVs.
- GLOBAL_GLOBALMV: Use MVs from each reference based on their frame-level global motion parameters.

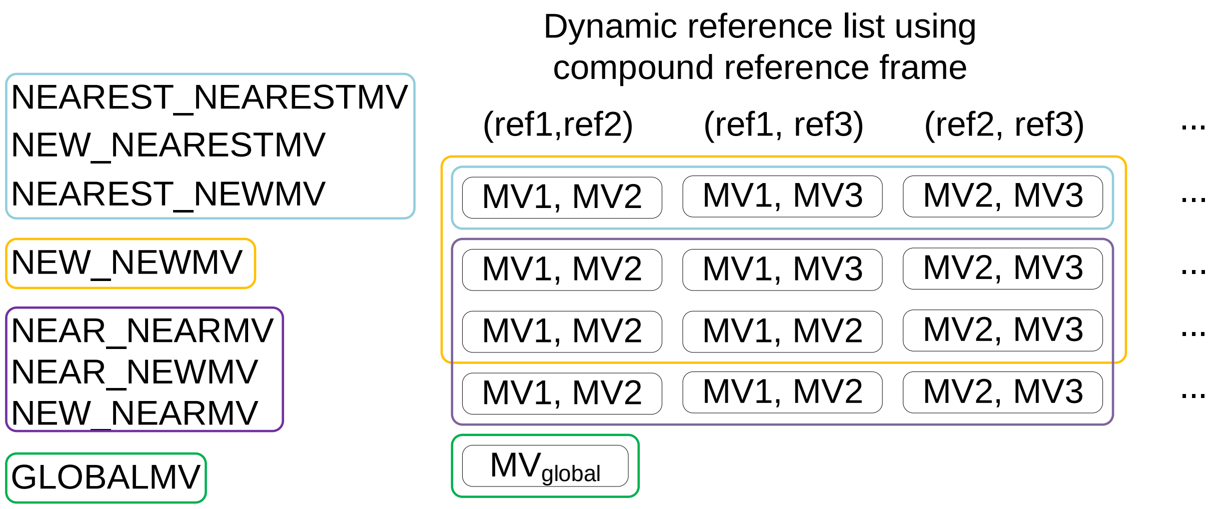

Figure 14: Motion vector prediction modes for compound prediction mode case

In all cases except the NEARESTMV and NEAREST_NEARESTMV modes, the DRL index is signalled to specify the exact MV or MV-pair to use as the MV predictor. However, the DRL index range can be either [0, 1, 2] or [1, 2, 3] in the reference lists depending on the MV prediction mode.

3.3.5 Inter prediction mode signalling

For signalling an inter prediction mode, the flag `skip_mode` is first signalled to determine whether SKIP mode applies to the current coding block. When SKIP mode is enabled, compound inter prediction is conducted using two reference frames and translational motion. The MVs and reference frames are all implicitly derived (taken as the first entry in the DRL) instead of being signalled. When SKIP mode is not enabled, the reference frame information is signalled.

When signalling the reference frame, the flag `comp_mode` is first signalled to indicate whether one or two reference frames is to be used for inter prediction. This flag is signalled only when the coding block width and height are both greater than or equal to eight. For coding blocks with a width or height of less than eight, only a single reference frame is used for inter prediction. When compound prediction is applied, `comp_ref_type` syntax is further signalled to indicate whether the compound prediction is uni-directional or bi-directional. Furthermore, when the uni-directional compound prediction mode is applied, one of the four predefined reference frame combinations is signalled. When the bi-directional compound prediction mode is applied, the reference frame indices for the forward and backward reference frames are signalled separately. For single reference predictions, just one of the seven possible reference frames is signalled.

After reference frame signalling, a YMode value is derived by multiple syntaxes to indicate which mode is to be applied for MV prediction and signalling. The mapping between YMode and the MV prediction is listed in Table 3. After the derivation of YMode, drl_mode syntax is further signalled to indicate which candidate in the DRL is to be used for MV prediction. After that, the MVD is further signalled when one of NEWMV, NEAREST_NEWMV, NEW_NEARESTMV, NEAR_NEWMV, NEW_NEARMV, or NEW_NEWMV is applied. When YMode is equal to NEW_NEARESTMV, NEW_NEARMV, or NEW_NEWMV, the MVD is signalled for the MV that is associated with the first reference frame. When YMode is equal to NEAREST_NEWMV, NEAR_NEWMV, or NEW_NEWMV, the MVD is signalled for the MV that is associated with the second reference frame.

Table 3: Mapping between YMode and MV prediction mode

YMode	**MV prediction mode**
0	NEARESTMV
1	NEARMV
2	GLOBALMV
3	NEWMV
4	NEAREST_NEARESTMV
5	NEAR_NEARMV
6	NEAREST_NEWMV
7	NEW_NEARESTMV
8	NEAR_NEWMV
9	NEW_NEARMV
10	GLOBAL_GLOBALMV
11	NEW_NEWMV

After signalling the MV prediction mode, the compound inter–intra mode (see Section 3.3.10) is further signalled. The compound inter–intra mode is signalled only when single reference frame inter prediction is applied, and when the block size is greater than or equal to 8×8 but smaller than or equal to 32×32. Signalling of the compound inter–intra mode includes a flag interintra that indicates whether the compound inter–intra mode is to be applied.

When the compound inter–intra mode is applied, an intra mode index interintra_mode is further signalled to indicate which of the DC_PRED, V_PRED, H_PRED, and SMOOTH modes is to be used to perform the intra prediction.

Thereafter, a flag wedge_interintra is further signalled to indicate whether the Wedge-based inter–intra prediction mode (see Section 3.3.10.2) is to be applied. When the Wedge-based inter–intra prediction mode is applied, wedge_index syntax is further signalled, which indicates the Wedge partition pattern that is to be applied in the Wedge-based inter–intra prediction.

After signalling the compound inter–intra mode, motion_mode is further signalled or implicitly derived to indicate which of the translational, overlapped block motion compensation (OBMC), and warped motion compensation modes is to be applied.

After signalling motion_mode, the compound prediction type is further signalled to indicate which compound prediction mode is to be applied. The compound prediction modes include compound

Wedge-based prediction, difference-modulated compound prediction, and distance-based compound prediction.

Finally, the interpolation filter index is signalled if applicable to indicate the selection of interpolation filter when translational motion compensation is applied.

3.3.6 Translational motion compensation

Translational motion models assume that all pixels within a coding block share the same single MV, as shown in Figure 15. The MV can be represented in 1/16-pixel accuracy; that is, the minimum granularity of the MV is 1/16 of a pixel. When deriving the prediction block using an MV pointing to a fractional sample location, interpolation is applied to generate the prediction block. If the MV is pointing to a fractional horizontal position, a row-wise horizontal interpolation filter is first applied to generate an intermediate block. Thereafter, if the MV is pointing to a fractional vertical position, a second column-wise vertical interpolation filter is applied using the intermediate block to generate the final prediction block.

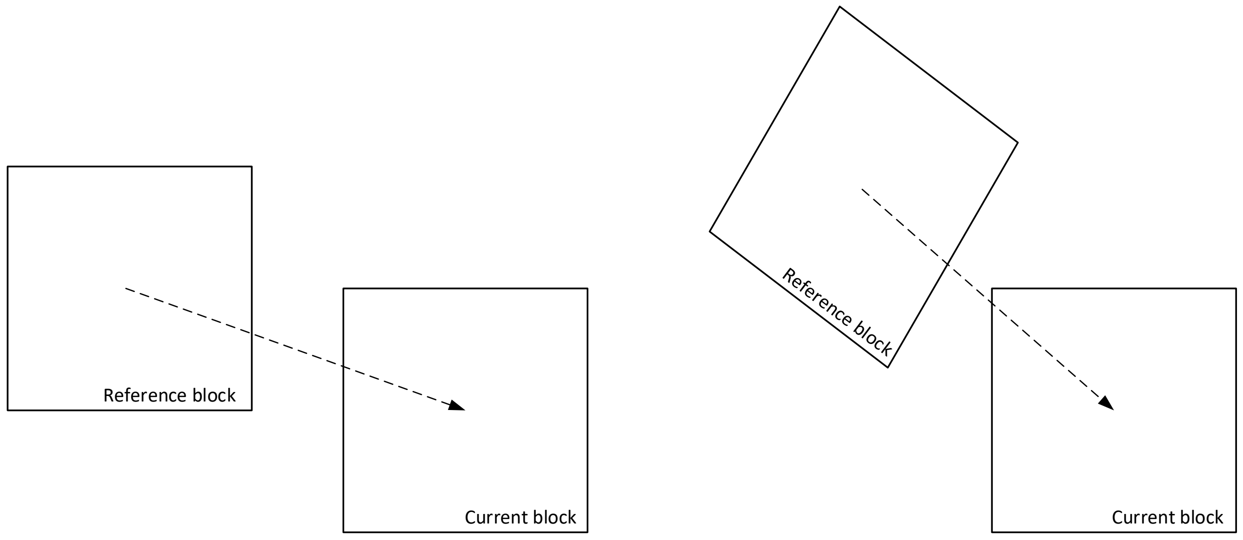

Figure 15: Examples of translational motion (left) and affine motion (right)

In the frame header, a flag is first signalled to indicate whether the interpolation filter can be switchable for the current frame. If the flag is signalled with a value indicating that the interpolation filter is not switchable, two bits are further signalled to indicate which of the four predefined finite impulse response (FIR) filters is to be applied. The four FIR filters include a 6-tap regular FIR filter (REGULAR; Table 4), a 6-tap FIR filter (SMOOTH; Table 5), an 8-tap FIR filter (SHARP; Table 6), and a 2-tap BILINEAR filter (Table 7).

Table 4: Filter coefficients of the REGULAR filter

Sample position	Tap 0	Tap 1	Tap 2	Tap 3	Tap 4	Tap 5	Tap 6	Tap 7
0	0	0	0	128	0	0	0	0
1/16	0	2	−6	126	8	−2	0	0
2/16	0	2	−10	122	18	−4	0	0
3/16	0	2	−12	116	28	−8	2	0
4/16	0	2	−14	110	38	−10	2	0

5/16	0	2	−14	102	48	−12	2	0
6/16	0	2	−16	94	58	−12	2	0
7/16	0	2	−14	84	66	−12	2	0
8/16	0	2	−14	76	76	−14	2	0
9/16	0	2	−12	66	84	−14	2	0
10/16	0	2	−12	58	94	−16	2	0
11/16	0	2	−12	48	102	−14	2	0
12/16	0	2	−10	38	110	−14	2	0
13/16	0	2	−8	28	116	−12	2	0
14/16	0	0	−4	18	122	−10	2	0
15/16	0	0	−2	8	126	−6	2	0

Table 5: Filter coefficients of the SMOOTH filter

Sample position	**Tap 0**	**Tap 1**	**Tap 2**	**Tap 3**	**Tap 4**	**Tap 5**	**Tap 6**	**Tap 7**
0	0	0	0	128	0	0	0	0
1/16	0	2	28	62	34	2	0	0
2/16	0	0	26	62	36	4	0	0
3/16	0	0	22	62	40	4	0	0
4/16	0	0	20	60	42	6	0	0
5/16	0	0	18	58	44	8	0	0
6/16	0	0	16	56	46	10	0	0
7/16	0	-2	16	54	48	12	0	0
8/16	0	−2	14	52	52	14	−2	0
9/16	0	0	12	48	54	16	−2	0
10/16	0	0	10	46	56	16	0	0
11/16	0	0	8	44	58	18	0	0
12/16	0	0	6	42	60	20	0	0
13/16	0	0	4	40	62	22	0	0
14/16	0	0	4	36	62	26	0	0
15/16	0	0	2	34	62	28	2	0

Table 6: Filter coefficients of the SHARP filter

Sample position	**Tap 0**	**Tap 1**	**Tap 2**	**Tap 3**	**Tap 4**	**Tap 5**	**Tap 6**	**Tap 7**
0	0	0	0	128	0	0	0	0
1/16	−2	2	−6	126	8	−2	2	0
2/16	−2	6	−12	124	16	−6	4	−2
3/16	−2	8	−18	120	26	−10	6	−2

4/16	−4	10	−22	116	38	−14	6	−2
5/16	−4	10	−22	108	48	−18	8	−2
6/16	−4	10	−24	100	60	−20	8	−2
7/16	−4	10	−24	90	70	−22	10	−2
8/16	−4	12	−24	80	80	−24	12	−4
9/16	−2	10	−22	70	90	−24	10	−4
10/16	−2	8	−20	60	100	−24	10	−4
11/16	−2	8	−18	48	108	−22	10	−4
12/16	−2	6	−14	38	116	−22	10	−4
13/16	−2	6	−10	26	120	−18	8	−2
14/16	−2	4	−6	16	124	−12	6	−2
15/16	0	2	−2	8	126	−6	2	−2

Table 7: Filter coefficients of the BILINEAR filter

Sample position	Tap 0	Tap 1	Tap 2	Tap 3	Tap 4	Tap 5	Tap 6	Tap 7
0	0	0	0	128	0	0	0	0
1/16	0	0	0	120	8	0	0	0
2/16	0	0	0	112	16	0	0	0
3/16	0	0	0	104	24	0	0	0
4/16	0	0	0	96	32	0	0	0
5/16	0	0	0	88	40	0	0	0
6/16	0	0	0	80	48	0	0	0
7/16	0	0	0	72	56	0	0	0
8/16	0	0	0	64	64	0	0	0
9/16	0	0	0	56	72	0	0	0
10/16	0	0	0	48	80	0	0	0
11/16	0	0	0	40	88	0	0	0
12/16	0	0	0	32	96	0	0	0
13/16	0	0	0	24	104	0	0	0
14/16	0	0	0	16	112	0	0	0
15/16	0	0	0	8	120	0	0	0

The interpolation filter can be switchable in both the horizontal and vertical directions of a coding block. If a dual filter is enabled in the sequence header, the interpolation filter selection can be signalled separately for the horizontal and vertical directions. Otherwise, only one interpolation filter selection can be signalled per coding block, which is shared for the horizontal and vertical directions. When the interpolation filter can be switchable, the candidate interpolation filters include three predefined FIR filters, i.e., REGULAR, SMOOTH, and SHARP. For smaller coding block sizes with dimensions of less than or equal to four along either the horizontal or vertical direction, the REGULAR and SMOOTH filters are replaced by another two 4-tap filters, as shown in Table 8, and the SHARP filter is no longer useable.

Table 8: Filter coefficients of the REGULAR and SMOOTH filters for dimensions of less than or equal to 4

Sample position	**REGULAR**				**SMOOTH**			
	Tap 0	**Tap 1**	**Tap 2**	**Tap 3**	**Tap 4**	**Tap 5**	**Tap 6**	**Tap 7**
0	0	128	0	0	0	128	0	0
1/16	−4	126	8	−2	30	62	34	2
2/16	−8	122	18	−4	26	62	36	4
3/16	−10	116	28	−6	22	62	40	4
4/16	−12	110	38	−8	20	60	42	6
5/16	−12	102	48	−10	18	58	44	8
6/16	−14	94	58	−10	16	56	46	10
7/16	−12	84	66	−10	14	54	48	12
8/16	−12	76	76	−12	12	52	52	12
9/16	−10	66	84	−12	12	48	54	14
10/16	−10	58	94	−14	10	46	56	16
11/16	−10	48	102	−12	8	44	58	18
12/16	−8	38	110	−12	6	42	60	20
13/16	−6	28	116	−10	4	40	62	22
14/16	−4	18	122	−8	4	36	62	26
15/16	−2	8	126	−4	2	34	62	30

3.3.7 Warped motion compensation

Unlike conventional motion compensation, which assumes a translational motion model between the reference and target block, warped motion utilizes an affine model, as shown in Figure 15.

The affine motion model can be represented by Equation (1):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{1}$$

where [*x*, *y*] are the coordinates of the original pixel and [*x*′, *y*′] are the warped coordinates of the reference block. It can be seen from Equation (1) that up to six parameters are needed to specify the warped motion: a_3 and b_3 specify a conventional translational MV; a_1 and b_2 specify the scaling along the MV; and a_2 and b_1 specify the rotation.

3.3.7.1 Global warped motion compensation

Global motion information is signalled for each inter reference frame, which includes the global motion type and a number of motion parameters. The global motion types and numbers of associated parameters are listed in Table 9.

Table 9: Global motion types with associated number of parameters

Global motion type	Number of parameters
Identity (zero motion)	0
Translational	2
Rotational	4
Zoom	4
General affine	6

After signalling the reference frame index, if global motion is selected, the global motion type and the parameters associated with the given reference frame are used for the current coding block.

3.3.7.2 Local warped motion compensation

For an inter coding block, local warped motion is allowed when the following conditions are met:

- The current block uses single reference prediction
- The width or height of the coding block is greater than or equal to eight samples
- At least one of the adjacent neighbouring blocks uses the same reference frame as the current block

If local warped motion is used for the current block, the affine model parameters are estimated by mean-squared minimization of the difference between the reference and modelled projections based on the MVs of the current block and its adjacent neighbouring blocks. To estimate the parameters of local warped motion, a projection sample pair of the center pixel in the neighbouring block and its corresponding pixel in the reference frame are collected if the neighbouring block uses the same reference frame as the current block. After that, three extra samples are created by shifting the center position by a quarter sample in one or two dimensions. These extra samples are also considered as projection sample pairs to ensure the stability of the model parameter estimation process.

The MVs of neighbouring blocks, which are used to derive the motion parameters, are referred to as motion samples. The motion samples are selected from neighbouring blocks that use the same reference frame as the current block. Note that the warped motion prediction mode is only enabled for blocks that use a single reference frame. This means that the motion samples can only be selected from blocks that are coded by uni-prediction (single reference frame). For example, in Figure 16, the MVs of neighbouring blocks B0, B1, and B2 are referred as MV0, MV1, and MV2, respectively. The current block is predicted using uni-prediction with reference frame Ref0. Neighbouring block B0 is predicted using compound prediction with reference frames Ref0 and Ref1; B1 is predicted using uni-prediction with reference frame Ref0; and B2 is predicted using compound prediction with reference frames Ref0 and Ref2. Since only B1 has the same reference frame as the current block, MV1 becomes a motion sample that can be used to derive the affine motion parameters of the current block, whereas MV0 and MV2 cannot be used.

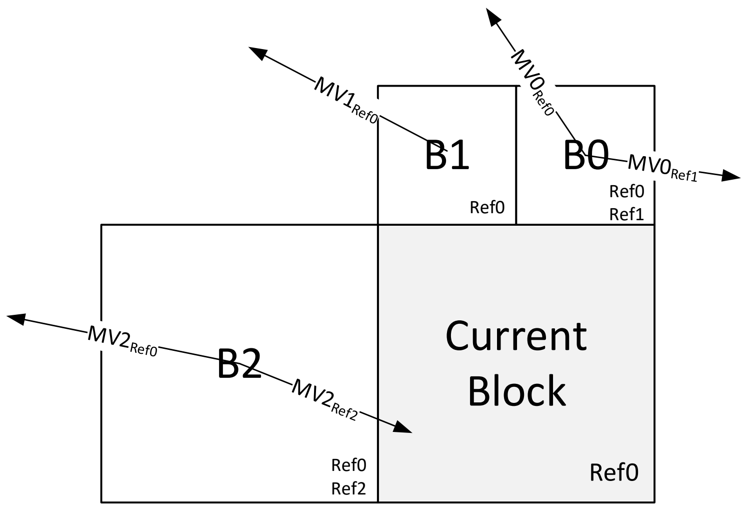

Figure 16: Example of motions samples used for deriving the model parameters of local warped motion prediction

3.3.8 Overlapped block motion compensation

For an inter coding block, overlapped block motion compensation (OBMC) is allowed when the following conditions are met:

- The current block uses single reference prediction
- The width or height of the coding block is greater than or equal to eight samples
- At least one of the neighbouring blocks is inter coded

When OBMC is applied to the current block, the initial inter prediction samples are first generated by using the MV associated with the current block. Then, the inter prediction samples for the current block and the inter prediction samples derived using MVs from the upper and left neighbouring blocks are blended to generate the final prediction samples. The maximum number of neighbouring MVs is limited by the size of the current block; up to four MVs from each of the upper and left blocks can be used in the OBMC process for the current block.

An example of the processing order of neighbouring blocks is shown in Figure 17, wherein the values marked in each block indicate the processing order of the MVs of the current and neighbouring blocks. Specifically, the MV of the current block is first applied to generate inter prediction samples $p_0(x, y)$. Then, the MV of block 1 is applied to generate prediction samples $p_1(x, y)$. After that, the prediction samples in the overlapping area between block 0 and block 1 (marked in grey in Figure 17) are given as a weighted average of $p_0(x, y)$ and $p_1(x, y)$. The MVs of blocks 2, 3, and 4 are further applied and blended in the same way.

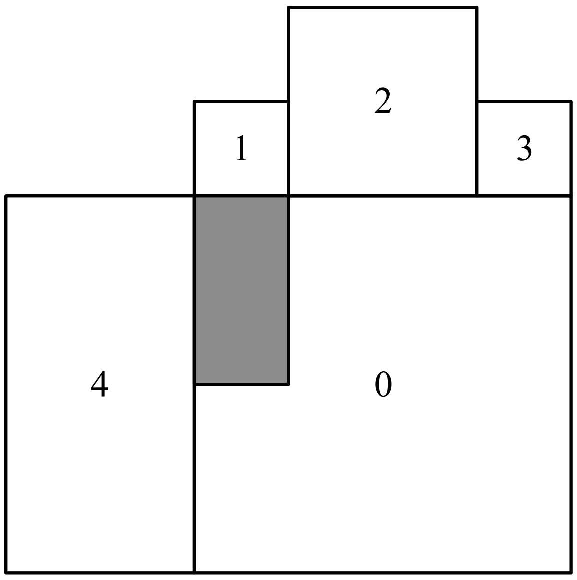

Figure 17: Example of neighbouring blocks used in OBMC process

3.3.9 Compound inter prediction

Inter prediction using two reference frames is called compound prediction, as formulated below.

$$P(x,y) = (w(x,y) \cdot P_0(x,y) + (64 - w(x,y)) \cdot P_1(x,y) + 32) \gg 6, \quad (2)$$

where $P_0(x, y)$ and $P_1(x, y)$ refer to the prediction samples from two reference frames for the current sample located at (x, y), $w(x, y)$ is the weighting factor applied to the prediction sample from the first reference frame, and $P(x, y)$ is the final compound prediction. Depending on the derivations of the weighting factor and prediction block, different compound motion prediction modes are defined, as described in the following subsections. For SKIP mode using two reference frames, $w(x, y)$ is always set to 32.

3.3.9.1 Compound Wedge-based prediction

In compound Wedge-based prediction mode, for each block size, a set of 16 predefined two-dimensional weighting arrays are hard coded. In each array, the weightings are arranged in such a way as to project to a predefined Wedge partitioning pattern. In each Wedge partitioning pattern, two Wedge partitions are specified along a certain edge direction and position. For samples located in one of the two Wedge partitions, the weightings are mostly set as 64. For samples located in the other Wedge partition, the weightings are mostly set as 0. Moreover, along the Wedge partitioning boundaries, the weightings are valued as 32.

In Wedge-based prediction mode, two syntaxes are predefined: `wedge_index`, which specifies the Wedge partitioning pattern index (ranging from 0 to 15); and `wedge_sign`, which specifies which of the two partitions is to be assigned the dominant weighting.

3.3.9.2 Difference-modulated compound prediction

In difference-modulated compound prediction mode, the weighting $w(x, y)$ is derived as follows:

$$w(x,y) = \begin{cases} 38 + \frac{|P_1(x,y) - P_2(x,y)|}{16}, & mask\_type = 0 \\ 64 - \left(38 + \frac{|P_1(x,y) - P_2(x,y)|}{16}\right), & mask\_type = 1 \end{cases}, \tag{3}$$

where `mask_type` is a signalled flag that indicates the assignment of weighting, and *w*(*x*, *y*) is further clipped in the range of [0, 64] before being applied in Equation (2) for deriving the final prediction block.

3.3.9.3 Distance-based compound prediction

In distance-based compound prediction mode, the weighting *w*(*x*, *y*) is dependent on the temporal distance between the current frame and the reference frame. Let d_0 and d_1 represent the distance from the current frame to two reference frames. If d_0 is less than d_1, the derivation of *w*(*x*, *y*) that is applied to $P_0(x, y)$ is described as follows:

$$w(x,y) = \begin{cases} 36, & 2d_0 < 3d_1 \\ 44, & 2d_0 < 5d_1 \\ 48, & 2d_0 < 7d_1 \\ 52, & 2d_0 \geq 7d_1 \end{cases}. \tag{4}$$

Otherwise, if d_0 is greater than d_1, the derivation is symmetric:

$$w(x,y) = \begin{cases} 64 - 36, & 2d_1 < 3d_0 \\ 64 - 44, & 2d_1 < 5d_0 \\ 64 - 48, & 2d_1 < 7d_0 \\ 64 - 52, & 2d_1 \geq 7d_0 \end{cases}. \tag{5}$$

A special case of the weighting assignment is used when *w*(*x*, *y*) is equal to 32, whereby equal weightings are given to $P_0(x, y)$ and $P_1(x, y)$.

3.3.10 Compound inter–intra prediction

In compound inter–intra prediction mode, the prediction block is derived as a combination of intra prediction and inter prediction. That is, $P_0(x, y)$ is an intra prediction sample and $P_1(x, y)$ is an inter prediction sample. The intra prediction block is derived using the DC_PRED, V_PRED, H_PRED, or SMOOTH prediction mode, and the inter prediction block is derived using single reference inter prediction with translational motion. Depending on how the weightings are derived for the intra and inter prediction samples, two compound inter–intra prediction modes can be applied, including regular inter–intra prediction and Wedge-based inter–intra prediction.

3.3.10.1 Regular inter–intra prediction

The different intra prediction modes use weightings that are derived differently. In regular inter–intra prediction, the weighting value decreases along the prediction direction. More specifically, the weighting applied to the intra prediction sample $P_0(x, y)$ is described as follows:

$$w(x,y) = \begin{cases} 32, & \text{DC\_PRED} \\ WeightLUT[x \cdot sizeScale], & \text{V\_PRED} \\ WeightLUT[y \cdot sizeScale], & \text{H\_PRED} \\ WeightLUT[\min(x,y) \cdot sizeScale], & \text{SMOOTH} \end{cases}, \tag{6}$$

where *sizeScale* is derived using the block width *W* and block height *H*:

$$sizeScale = 128/\max(W,\ H), \tag{7}$$

and WeightLUT is a hard-coded lookup table (Table 10).

Table 10: Lookup table used in compound inter–intra motion prediction

60	58	56	54	52	50	48	47	45	44	42	41	39	38	37	35	34	33	32
31	30	29	28	27	26	25	24	23	22	22	21	20	19	19	18	18	17	16
16	15	15	14	14	13	13	12	12	12	11	11	10	10	10	9	9	9	8
8	8	8	7	7	7	7	6	6	6	6	6	5	5	5	5	5	4	4
4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1					

3.3.10.2 Wedge-based inter–intra prediction

Compound inter–intra prediction can be also applied using a Wedge-based weighting scheme. More specifically, the prediction block is a combination of intra and inter prediction blocks, and the weightings are specified using the Wedge partitioning pattern specified by `wedge_index` (ranging from 0 to 15). The Wedge-based inter–intra motion prediction mode differs from the regular Wedge-based motion prediction mode described in 3.3.9.1, because the value of `wedge_sign`, which specifies the partition with the dominant weighting, is derived rather than signalled.

3.4 Transform coding

3.4.1 Core transforms

A separable two-dimensional (2D) transform process is applied to prediction residuals. For the forward transform, a one-directional (1D) vertical transform is first performed on each column of the input residual block, then a horizontal transform is performed on each row of the vertical transform output. For the backward transform, a 1D horizontal transform is first performed on each row of the input dequantized coefficient block, then a vertical transform is performed on each column of the horizontal transform output. The primary transforms include four different types of transforms: a) 4-, 8-, 16-, 32-, and 64-point discrete cosine transforms (DCT-2); b) 4-, 8-, and 16-point asymmetric discrete sine transforms (DST-4 and DST-7) and c) their flipped versions; and d) 4-, 8-, 16-, and 32-point identity transforms (IDTs). For the asymmetric DSTs (ADSTs), a transform size of 4-point refers to DST-7, otherwise, when the transform size is greater than 4-point, it refers to DST-4. The transform bases of the supported transform types are listed in Table 11.

Table 11: Transform basis functions for DCT-2, DST-4, DST-7, and IDT for *N*-point input

Transform Type	Basis function $T_i(j)$, with i, $j = 0,\ldots,(N-1)$
DCT-2	$T_i(j) = \omega_0 \cdot \sqrt{\frac{2}{N}} \cdot \cos\left(\frac{\pi \cdot i \cdot (2j+1)}{2N}\right)$

	where $\omega_0 = (i == 0)\ ?\ \sqrt{\frac{2}{N}} : 1$
DST-4	$T_i(j) = \sqrt{\frac{2}{N}} \cdot \sin\left(\frac{\pi \cdot (2i+1) \cdot (2j+1)}{4N}\right)$
DST-7	$T_i(j) = \sqrt{\frac{4}{2N+1}} \cdot \sin\left(\frac{\pi \cdot (2i+1) \cdot (j+1)}{2N+1}\right)$
IDT	$T_i(j) = (i == j)?\,1 : 0$

3.4.2 Transform selection and signalling

For the luma colour component, each transform block can select one pair of horizontal and vertical transforms given a predefined set of transform type candidates. This selection is explicitly signalled in the bitstream, unless max(*W*, *H*) is equal to 64, when the selection is not signalled. When the transform block has a width or height greater than or equal to 32 (max(*W*, *H*) is greater than or equal to 32), the set of transform type candidates is dependent on the prediction mode as per Table 12. Otherwise, when the transform block width and height are both smaller than 32 (max(*W*, *H*) is less than 32), the set of transform type candidates is dependent on the prediction mode as per Table 13.

Table 12: Transform type candidates for luma when max(*W*, *H*) is greater than or equal to 32

max(*W*, *H*)	**Intra**	**Inter**
32	DCT only	DCT only, IDTX
64	DCT only	DCT only

Table 13: Transform type candidates for luma when max(*W*, *H*) is less than 32

min(*W*, *H*)	**Intra**	**Inter**
4	DTT4, IDTX, 1DDCT	ALL16
8	DTT4, IDTX, 1DDCT	ALL16
16	DTT4, IDTX	DTT9, IDTX, 1DDCT

The sets of transform type candidates are defined in Table 14.

Table 14: Specification of transform set

Transform set	**Vertical transform**	**Horizontal transform**
DCT only	DCT	DCT
IDTX	IDT	IDT
1DDCT	DCT	IDT
	IDT	DCT
DTT4	ADST	ADST
	ADST	DCT
	DCT	ADST

	DCT	DCT
DTT9	DCT	DCT
	DCT	ADST
	DCT	Flipped ADST
	ADST	DCT
	ADST	ADST
	ADST	Flipped ADST
	Flipped ADST	DCT
	Flipped ADST	ADST
	Flipped ADST	Flipped ADST
ALL16	DCT	DCT
	DCT	ADST
	DCT	Flipped ADST
	DCT	IDT
	ADST	DCT
	ADST	ADST
	ADST	Flipped ADST
	ADST	IDT
	Flipped ADST	DCT
	Flipped ADST	ADST
	Flipped ADST	Flipped ADST
	Flipped ADST	IDT
	IDT	DCT
	IDT	ADST
	IDT	Flipped ADST
	IDT	IDT

For chroma colour components, the transform type selection process is performed implicitly. For intra prediction residuals, the transform type is selected according to the intra prediction mode as specified in Table 15. For inter prediction residuals, the transform type is selected according to the transform type selection of the collocated luma block. Therefore, for chroma colour components, there is no transform type signalling in the bitstream.

Table 15: Transform type selection for chroma colour component intra prediction residuals

Intra prediction mode	Vertical transform	Horizontal transform
DC_PRED	DCT	DCT
V_PRED	ADST	DCT
H_PRED	DCT	ADST
D45_PRED	DCT	DCT
D135_PRED	ADST	ADST
D113_PRED	ADST	DCT
D157_PRED	DCT	ADST
D203_PRED	DCT	ADST
D67_PRED	ADST	DCT

SMOOTH	ADST	ADST
SMOOTH_V	ADST	DCT
SMOOTH_H	DCT	ADST
Paeth	ADST	ADST

The computational cost of large size (e.g., 64-point) transforms is further reduced by zeroing out all the coefficients, except in the following two cases:

- The top-left 32×32 quadrant for 64×64/64×32/32×64 (DCT, DCT) transform combinations.
- The left 32×16 area and top 16×32 area for 64×16 and 16×64 (DCT, DCT) transform combinations, respectively.

Both DCT-2 and ADST (DST-4 or DST-7) are implemented using a butterfly structure [7], which includes multiple stages of butterfly operations. Each butterfly operation is calculated in parallel, and the different stages are cascaded in sequential order.

3.5 Quantization

Quantization is applied to the transform coefficients at the encoder. The quantized transform coefficients are then further entropy coded and signalled into the bitstream. At the decoder, dequantization is applied to the coefficients parsed from the bitstream, and the dequantized coefficients are fed into the inverse transform process to derive residual samples. Different quantization step sizes may be applied for DC and AC transform coefficients, as well as for luma and chroma transform coefficients. The process is as follows:

- To specify the quantization step size, a `base_q_idx` syntax element is first signalled in the frame header. `base_q_idx` is an 8-bit fixed length code specifying the quantization step size for luma AC coefficients, and has a valid value range of [0, 255].
- After that, the delta value relative to `base_q_idx` for luma DC coefficients (indicated as DeltaQYDc) is further signalled.
- If there are multiple colour planes, then a flag `diff_uv_delta` is signalled to indicate whether different quantization index values should be applied to the Cb and Cr colour components.
- If `diff_uv_delta` is signalled as 0, then only the delta values relative to `base_q_idx` for chroma DC coefficients (indicated as DeltaQUDc) and AC coefficients (indicated as DeltaQUAc) are signalled.
- Otherwise, the delta values relative to `base_q_idx` for the Cb and Cr DC coefficients (indicated as DeltaQUDc and DeltaQVDc) and Cb and Cr AC coefficients (indicated as DeltaQUAc and DeltaQVAc) are signalled.

The above decoded DeltaQYDc, DeltaQUAc, DeltaQUDc, DeltaQVAc, and DeltaQVDc values are added to `base_q_idx` to derive the quantization indices. These quantization indices are further mapped to quantization step size according to the appropriate lookup table. The mapping from quantization index to quantization step size for 8-, 10-, and 12-bit internal bit depths is specified by a 3×256 lookup table Dc_Qlookup for DC coefficients and by a 3×256 lookup table Ac_Qlookup for AC coefficients. The mappings between quantization index, e.g., `base_q_idx`, to quantization step size are shown in Figure 18 and Figure 19 for DC and AC transform coefficients, respectively.

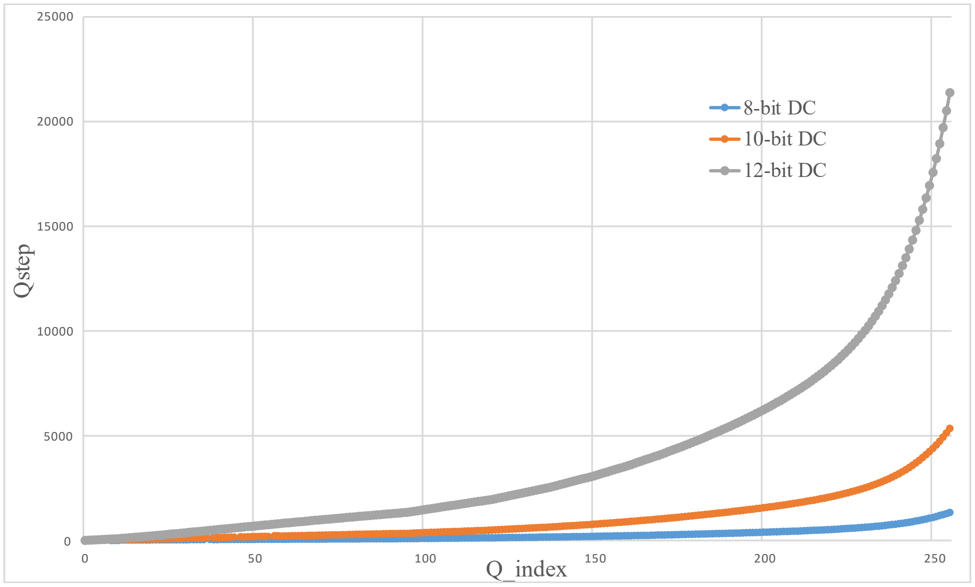

Figure 18: Quantization step size of DC coefficients for different internal bit depths

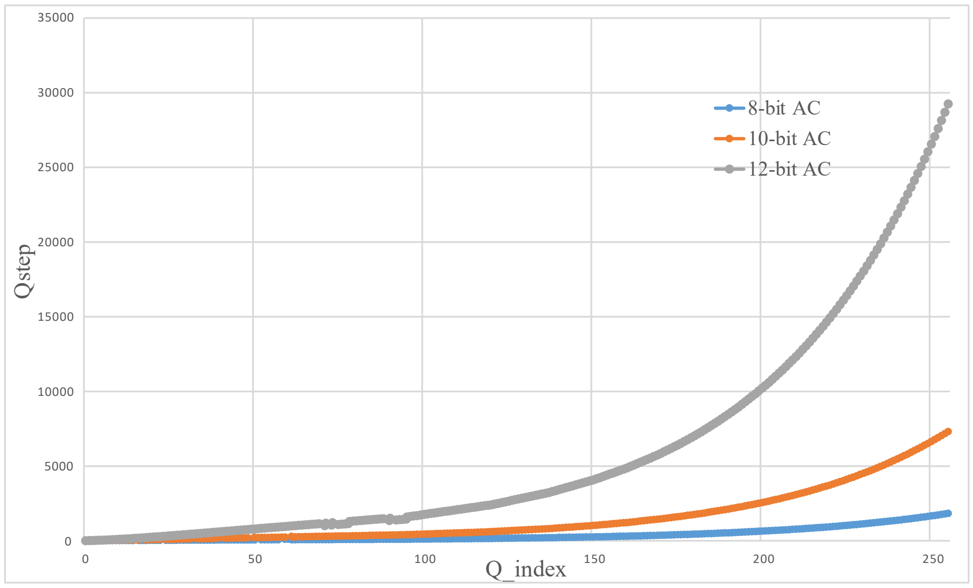

Figure 19: Quantization step size of AC coefficients for different internal bit depths

Given a quantization step size Q_{step}, the input quantized coefficients are further dequantized using the following formula:

$$F = sign \times ((f \times Q_{\text{step}}) \% 0xFFFFFF)/deNorm, \tag{8}$$

where *f* is the input quantized coefficient, *F* is the output dequantized coefficient, and *deNorm* is a constant derived from the transform block area size, as indicated by Table 16.

Table 16: Selection of *deNorm* for different transform block area sizes

deNorm	**Transform block area size**

1	< 512 samples
2	512 or 1024 samples
4	> 1024 samples

When the quantization index is 0, quantization is performed using a quantization step size of 1, which is a lossless coding mode.

3.6 Entropy coding

3.6.1 Multisymbol arithmetic coding engine

An *M*-ary arithmetic coding engine is applied for entropy coding of syntax elements. Each syntax element is associated with an alphabet of *M* elements, where *M* can be any integer value between 2 and 16. The input to the encoding is an *M*-ary symbol, and a context consists of a set of *M* probabilities. The probability is updated after coding/parsing each syntax, and the probabilities are represented as 15-bit cumulative distribution functions (CDFs). The cumulative distribution functions are arrays of *M* 15-bit integers as follows:

$$C = \left[c_0, c_1, \ldots, c_{(M-2)}, 2^{15}\right], \tag{9}$$

where c_n/32768 is the probability of the symbol being less than or equal to *n*. The probability update is performed using the following equations:

$$\begin{cases} c_m = c_m \cdot (1-\alpha) & m \in [0, symbol) \\ c_m = c_m + \alpha \cdot (1 - c_m) & m \in [symbol, M-1) \end{cases}, \tag{10}$$

where α is the probability update rate that adapts based on the number of times the symbol has been decoded (up to a maximum of 32) and *m* is the element index in the CDF. This adaptation of α allows faster probability updates at the beginning of coding/parsing the syntax elements. The *M*-ary arithmetic coding process follows the conventional arithmetic coding engine design; however, only the upper 9 bits are input to the arithmetic encoder/decoder.

3.6.2 Coefficient coding

For each transform block, the coefficient coding starts with coding an `all_zero` flag, which indicates whether the transform block has only zero residuals. This flag is followed by signalling the primary transform kernel type and the end-of-block (EOB) position, in cases where the `all_zero` flag is signalled as 0. Thereafter, the coefficient values are coded with multiple level maps together with sign values. The level maps are coded as three level planes, namely, lower-level, middle-level, and higher-level planes, and the sign is coded as a separate plane. The lower-, middle-, and higher-level planes correspond to different ranges of coefficient magnitudes (0–2, 3–14, 15 and above, respectively).

The three level planes are coded as follows. After the EOB value is coded, the lower- and middle-level planes are coded together in backward scanning order (where the scanning order refers to a zig-zag scan applied on the transform coefficients of the current transform block). Then, the sign plane and higher-level plane are coded together in forward scan order. Thereafter, the remainder (coefficient magnitude minus 14) is entropy coded using Exp-Golomb code.

The context model applied to the lower-level plane depends on 1) the primary transform directions, namely: bi-directional, horizontal, and vertical; 2) the transform block size; and 3) up to five neighbouring transform coefficients in the transform domain. The middle-level plane uses a similar context model, but the number of context neighbour coefficients is reduced from 5 to 2. The higher-level plane is coded by Exp-Golomb code without using context modelling. For the sign plane, with the exception of the DC sign, which is coded using the DC signs from neighbouring transform units, sign values of all other coefficients are coded directly without context modelling.

3.7 Loop filtering and post-processing

3.7.1 Deblocking filter

In the loop filtering pipeline, a deblocking filter is first utilized to reduce blocking artifacts at the transform block boundaries caused by quantization. For luma colour component, 4-, 8-, and 14-tap predefined filters can be applied as the deblocking filter. For chroma colour components, 4- and 6-tap predefined filters can be applied as the deblocking filter. The selection of filter length is determined by the minimum transform block sizes that are applied on both sides of the transform block boundary. Specifically, if a transform block has a width and height of greater than 16 on both sides, the 14-tap filter is applied on the luma colour component. An illustration of a deblocking filter using a 14-tap filter is shown in Figure 20.

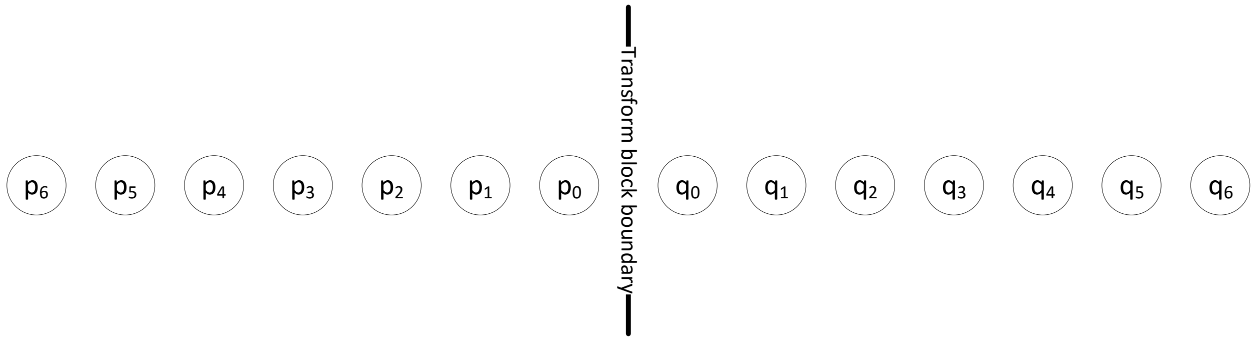

Figure 20: Illustration of a deblocking filter applied on a vertical transform block boundary using a 14-tap filter

The filters applied in the deblocking process are low-pass filters. To avoid over-smoothing of an actual edge in the texture, a boundary condition is checked when deblocking. That is, for a boundary sample that shows high local variance, the presence of an actual edge is recognized and no deblocking filter is applied. The boundary sample is identified as high variance when the following conditions are met:

- $|p_1 - p_0| > T_0$
- $|q_1 - q_0| > T_0$
- $2|p_0 - q_0| + |p_1 - q_1|/2 > T_1$

where p_0, p_1, q_0, and q_1 indicate the reconstruction samples located on the left (or top) and right (or bottom) sides of the transform block boundary. If the filter length is 8 or 14, two additional samples are checked to determine if high variance is present, indicated by the following conditions:

- $|p_3 - p_2| > T_0$
- $|q_3 - q_2| > T_0$

In the above conditions, T_0 and T_1 are threshold values that can be decided on a superblock basis. Greater threshold values indicate a higher chance of applying the deblocking filter.

3.7.2 Constrained directional enhancement filter

3.7.2.1 Edge direction estimation

Constrained directional enhancement filters (CDEFs) perform edge direction searching at the 8×8 block-level. In CDEFs, there are eight edge directions in total, as illustrated in Figure 21.

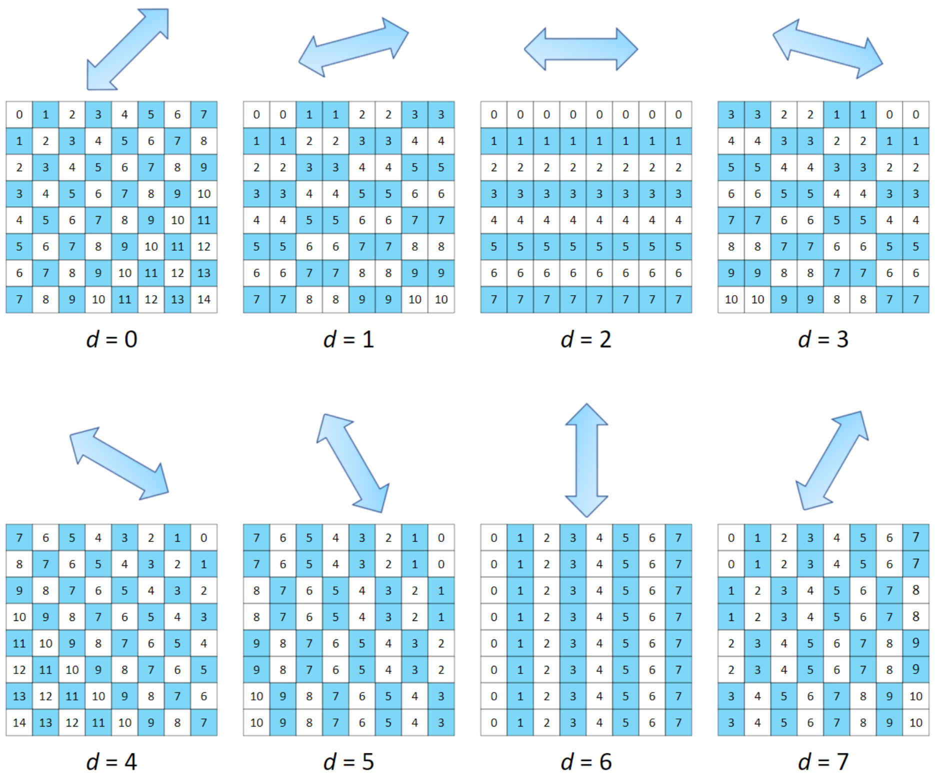

Figure 21: Line number k for pixels following direction $d = 0,\ldots,7$ in an 8×8 block

The optimal edge direction d_{opt} is found by maximizing the following term:

$$d_{\text{opt}} = \max_{d} S_d, \tag{11}$$

where

$$s_d = \sum_k \frac{1}{N_{d,k}} \left(\sum_{p \in P_{d,k}} x_p\right)^2; \tag{12}$$

x_p is the value of pixel p; $P_{d,k}$ is the set of pixels in line k following direction d; and $N_{d,k}$ is the cardinality of $P_{d,k}$.

3.7.2.2 Directional filter

The CDEF filtering process consists of a primary and secondary filter. The primary filter processes reconstruction samples along the edge direction (as shown in Figure 22), while the secondary filter processes reconstruction samples along a direction 45-degrees from the edge direction (as shown in Figure 23). In the primary filter, for even strengths, *a* is set equal to 2 and *b* is set equal to 4, for odd strengths, *a* is set equal to 3 and *b* is set equal to 3.

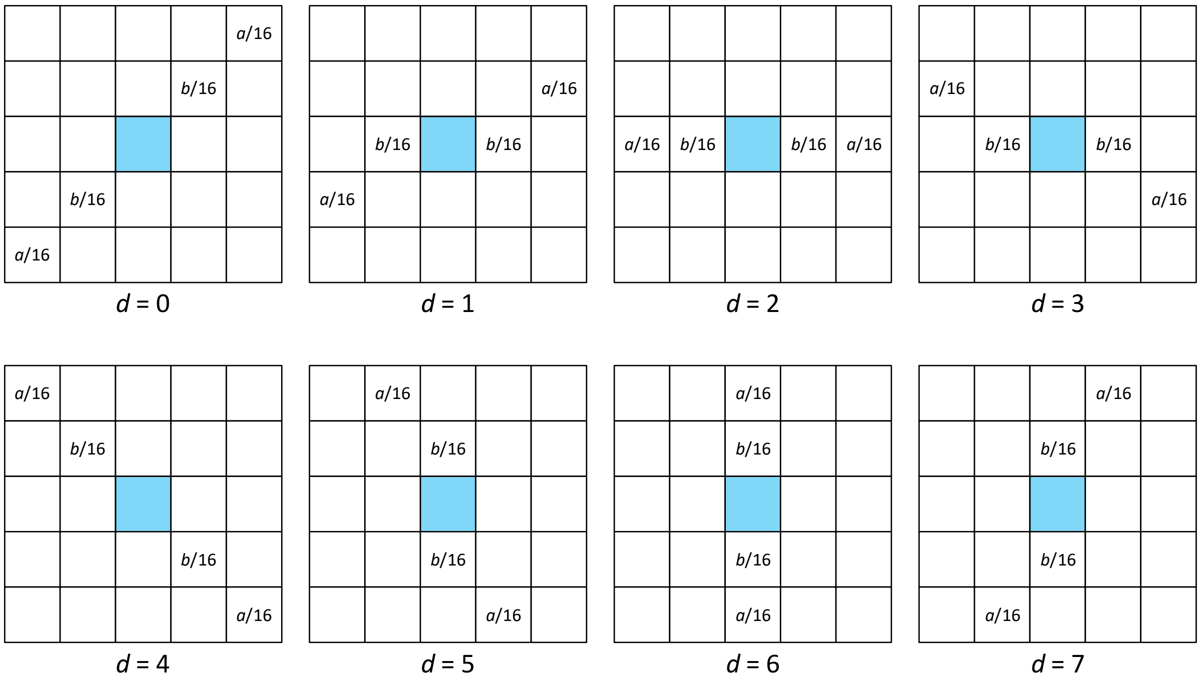

Figure 22: Primary filter taps following edge direction. The pixel to be filtered is coloured

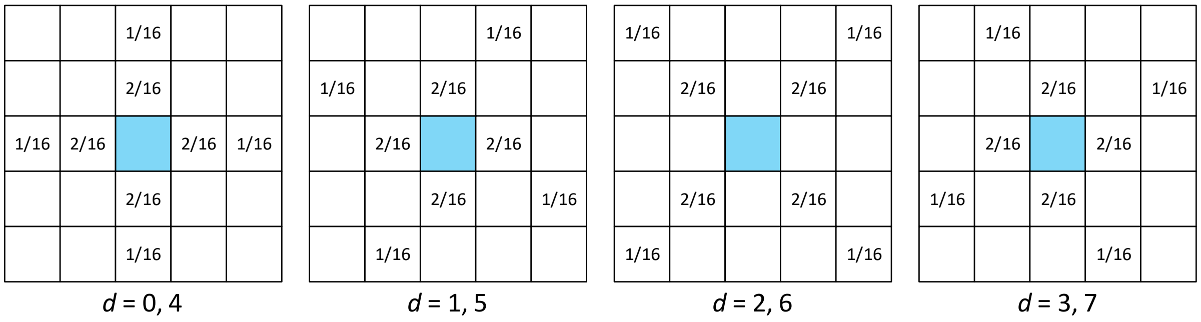

Figure 23: Secondary filter taps. The pixel to be filtered is coloured

CDEFs can be described by the following equation:

$$y_{i,j} = x_{i,j} + round\left(\sum_{m,n} w^p f(x_{m,x} - x_{i,j}, S^p, D) + \sum_{m,n} w^s f(x_{m,x} - x_{i,j}, S^s, D)\right) \quad (13)$$

where $x_{i,j}$ and $y_{i,j}$ are the input and output reconstructed sample values of the CDEF filter, respectively; superscripts *p* and *s* denote the primary and secondary filters, respectively; *w* is a weighting factor applied between the primary and secondary filter taps; *f*(*d*,*S*,*D*) is a nonlinear filtering function; *S* is filter strength; and *D* is a damping parameter. For 8-bit content, S^p ranges from 0 to 15, and S^s can be 0, 1, 2, or 4. *D* ranges from 3 to 6 for luma colour component and from 2 to 4 for chroma colour components.

CDEF is a nonlinear filtering process that prevents excessive blurring when a filtering process is applied across an edge. It is achieved by ignoring pixels that are too different from the current pixel that is to be filtered. When the difference between the current pixel and its neighbouring pixel *d* is within a given threshold, *f*(*d*,*S*,*D*) is set equal to *d*; otherwise, *f*(*d*,*S*,*D*) is set equal to 0. Specifically, the strength *S* determines the maximum allowable difference, and damping *D* determines the sample positions that ignore the filter tap.

3.7.3 Loop restoration filter

Loop restoration (LR) filters are applied in units of 64×64, 128×128, or 256×256 blocks, and each unit is called a loop restoration unit (LRU). An LR filter may be applied on an LRU subject to one of three options: 1) when applying a Wiener filter, 2) when applying a self-guided filter (SGF), and 3) when not applying any filter. LR is the final stage in the loop filtering pipeline. The filter coefficients for different filter taps are signalled in different bit depths.

3.7.3.1 Separable symmetric Wiener filter

In LR filtering, a 7×7 separable filter is used in a Wiener filter, which includes a 7-tap vertical filter and a 7-tap horizontal filter. Filtering of the reconstruction samples of a block is performed by applying the vertical and horizontal filters sequentially. After applying the vertical and horizontal filters, the final filtered reconstruction samples are generated. The filter coefficients are derived by an encoder and signalled into the bitstream. Since the Wiener filter is a symmetric filter, only three coefficients need to be signalled for a 7-tap filter, with the three mirrored coefficients derived as the same values. An example of a symmetric 7-tap Wiener filter is shown in Figure 24.

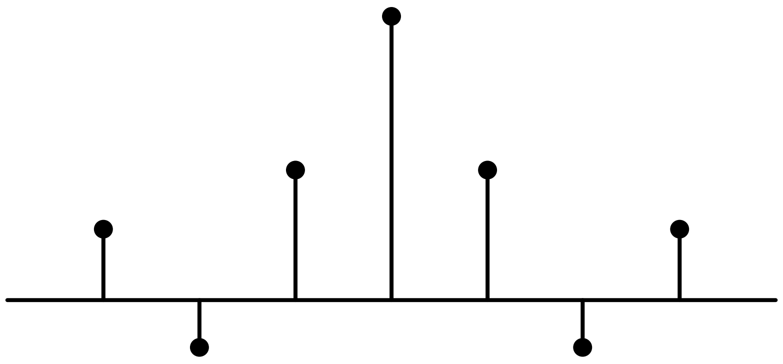

Figure 24: Example of a symmetric 7-tap Wiener filter

3.7.3.2 Self-guided filter

Self-guided filters (SGFs) are designed to obtain two initial filtered frames, X_1 and X_2. The final restoration X_r is obtained as a combination of the degraded samples and the difference between the degraded sample and the coarse restoration as follows:

$$X_r = X + \alpha(X_1 - X) + \beta(X_2 - X) \tag{14}$$

At the encoder side, α and β are computed using least mean squares:

$$\alpha, \beta^T = (A^T A)^{-1} A^T b \tag{15}$$

where $A = \{X_1 - X, X_2 - X\}$; $b = y - x$; and y is a vector recording original samples.

X_1 and X_2 are obtained using guided filtering, and the filter is controlled by a radius parameter r and a noise parameter e, where greater values of r imply higher spatial variance, and greater values of e imply higher range variance. X_1 and X_2 can be described by $\{r_1, e_1\}$ and $\{r_2, e_2\}$, respectively. To apply the filter, the encoder needs to signal the values of r_1, e_1, r_2, e_2, α, and β into the bitstream.

The steps to derive the initial filtered frames, i.e., X_1 and X_2, are as follows:

- A pair of (r, e) is given, which is signalled for the generation of X_1 and X_2 separately.
- The mean value μ and variance σ^2 of input samples in a $(2r + 1) \times (2r + 1)$ window are calculated, wherein the window is centered at the pixel to be filtered.
- For each sample, the filtered sample is computed using $x' = \frac{\sigma^2}{\sigma^2+e}x + \frac{e}{\sigma^2+e}\mu$.
- With the filtered samples x', for each sample of an LRU, X_1 and X_2 are generated using the signalled (r_1, e_1) and (r_2, e_2), respectively.

The generated X_1 and X_2 are used as input for the SGF. SGF is applied using Equation (14) and the signalled (α, β).

3.7.4 Frame super-resolution

To improve the perceptual quality of decoded pictures, a super-resolution process can be applied at low bitrates. The super-resolution process can be performed on a per-frame basis. First, at the encoder side, the source video is downscaled as a nonnormative procedure. Second, the downscaled video is encoded, followed by deblocking and CDEF filtering processes. Third, a linear upscaling process is applied as a normative procedure to convert the encoded video back to its original resolution. Finally, an LR filter is applied to further recover some lost details of the upscaled picture. The last two steps make up the super-resolving process; the deblocking and CDEF filters are applied at lower spatial resolution on the decoder side, after which the frames are passed through the super-resolution process. To reduce complexity regarding the use of line buffers in hardware implementation, the upscaling and downscaling processes are applied to the horizontal dimension only.

3.7.5 Film grain synthesis

The perceived quality of motion picture and TV content is often enhanced by the presence of film grain, but film grain is generally difficult to compress due to its high degree of randomness. However, film grain noise may be synthesized, to a high degree of perceptual accuracy, by a low-order parameterized model that may be encoded more efficiently than the noise field itself. AV1 includes a film grain synthesis tool, using a representation based on an autoregressive (AR) model, which enables the modelled noise to be synthesized and added back to the decoded denoised video signal at the decoder. The coding gain of the denoised video is higher while the visual quality of the final reconstructed video is well preserved.

At the encoder, the film grain noise is removed from the input video by a denoising process, and the resulting denoised signal is encoded into the bitstream. The parameters of the noise model are then estimated in smooth areas of the denoised video, identified by analyzing its structure and intensity using an edge detector. The noise model parameters are also encoded into the bitstream. At the decoder, the noise model parameters are used to synthesize the film grain noise, which is then added to the decoded denoised video to produce the final reconstructed video signal.

Several model parameters are signalled in the bitstream, including a lag value that defines the number of AR model coefficients, the values of AR model coefficients, sets of points that define a number of piece-wise linear scaling functions, and parameters for mapping the strength of the film grain noise in the chroma components. These parameters are quantized and signalled as integers, which include 64 bytes for the scaling function and 74 bytes for the AR coefficients.

The film grain noise is modelled using a 2D causal autoregressive process with pseudo-Gaussian noise input. For each of the two chroma components, there is an additional AR coefficient to model the correlation with the collocated luma sample.

The film grain noise synthesis for the luma component is formulated as follows:

$$Y' = Y + f(Y) \cdot G_L \tag{16}$$

where *Y*´ is the output luma sample with synthesized film grain noise, *Y* is the reconstructed luma sample before adding film grain noise, G_L is the luma film grain noise sample, and *f*(*Y*) is a piece-wise linear scaling function that is signalled in the bitstream.

For chroma component *C* (Cb or Cr), the film grain noise synthesis is formulated as follows:

$$C' = C + f(u) \cdot G_C \tag{17}$$

$$u = b_C \cdot C + d_C \cdot Y_{\text{av}} \cdot h \tag{18}$$

where *C*´ is the output chroma sample with synthesized film grain noise, *C* is the reconstructed chroma sample before adding film grain noise, G_C is the chroma film grain noise sample, *f*(*u*) is the scaling function, b_C, d_C and *h* are parameters used to model the dependency between the film grain noise strength and sample intensity, and Y_{av} is the average of the collocated luma sample values.

The synthesized film grain noise is added to the output pictures before sending them to the display. However, the synthesized film grain noise is not added to the reference pictures, because its random nature may reduce subsequent inter-prediction efficiency.

When applying film grain noise synthesis, an autoregressive process is applied in a raster scan in order to generate one 64×64 luma, and two 32×32 chroma film grain noise templates. In the synthesis process, 32×32 luma and 16×16 chroma film grain noise blocks are taken from these templates at pseudo-random positions and added to the reconstructed samples. The 32×32 luma and 16×16 chroma film grain noise blocks are processed by applying sample-wise film grain noise scaling using the scaling functions described above and parameters contained in the bitstream. There is also an option to apply film grain noise blocks in an overlapping manner to reduce potential discontinuities at the film grain noise block boundaries.

3.8 Screen content coding

To improve the compression performance of screen-captured content, AV1 incorporates several coding tools, such as intra block copy (IntraBC) to handle repeated patterns in a screen picture, and palette mode to handle screen blocks with a limited number of colours.

3.8.1 Intra block copy

IntraBC [13]uses a vector to indicate a prediction block in the same picture as the current block. This vector is called a block vector (BV). BVs can be signalled in the bitstream and the precision for representing a BV is integer-point. The prediction process in IntraBC mode is similar to the prediction process in inter-picture prediction mode, with the main difference being that, in IntraBC, a predictor block is formed from the current picture *before* applying loop filters, while in inter-picture prediction, the prediction block is formed from the reconstructed samples of a previously coded picture after applying loop filters. Therefore, IntraBC may be considered “motion compensation” within the current picture, essentially using the BV as an MV.

When coding the current block, a flag indicating whether IntraBC is to be used is first signalled. If the flag indicates that IntraBC is to be used for coding the current block, then the BV difference is calculated by subtracting the value of the predicted BV from that of the current BV, and classified into one of the following four types: 1) the horizontal and vertical components are both zero; 2) the horizontal component is nonzero and the vertical component is zero; 3) the horizontal component is zero and the vertical component is nonzero; and 4) the horizontal and vertical components are both nonzero. The BV type information is signalled, followed by the BV difference value.

While IntraBC is exceptionally effective for coding screen content, it introduces some challenges for hardware design. To facilitate hardware design, the following adjustments are adopted.

- When IntraBC is used, the loop filters are disabled, including deblocking, CDEF, and LR filters. By doing this, a second picture buffer dedicated to enabling IntraBC is not needed.
- To facilitate parallel decoding, the allowed prediction area is restricted. Specifically, if the top-left pixel coordinate of a superblock is (x_0, y_0), IntraBC prediction is available at pixel position (x, y) only if the value of the vertical coordinate y is less than y_0 and the value of the horizontal coordinate x is less $x_0 + 2(y_0 - y)$.
- Due to hardware write-back delays, the immediate reconstructed area may not be accessible by IntraBC prediction, which may contain one or more superblocks. Hence, the allowed IntraBC prediction area is further restricted as follows: if the top-left pixel coordinate of a superblock is (x_0, y_0), IntraBC prediction is available at pixel position (x, y) only if the value of the vertical coordinate y is less than y_0 and the value of the horizontal coordinate x is less than $x_0 + 2(y_0 - y) - D$, where D denotes the number of pixels in the horizontal direction from the left side of the current block. In AV1, D is set equal to twice the width of a superblock, i.e., 256 pixels. The IntraBC prediction area is shown in Figure 25.

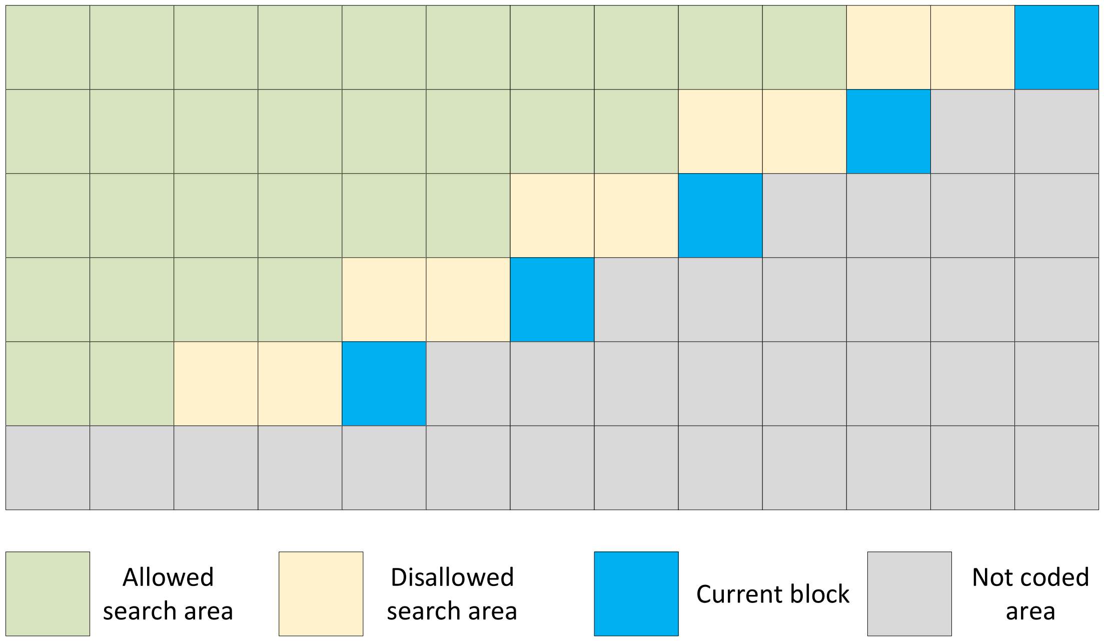

Figure 25: The prediction area for IntraBC mode, each block is a superblock

3.8.2 Palette mode

Palette mode may be applied when the current block is intra coded using the DC_PRED prediction mode. Palette mode may be applied for both luma and chroma blocks and can only be applied when the block size is greater than or equal to 8×8, and when both the width and height are less than or equal to 64.

Several syntax elements are signalled when palette mode is enabled, including a flag `has_palette_y` that indicates whether palette mode is to be applied to the current coding block; a syntax `palette_size_y_minus_2` that specifies the palette size; and a flag `use_palette_color_cache_y` that indicates whether the colour index is inherited for each entry of the palette. If the number of inherited palette entries is less than the signalled palette size, the remaining colour indices are explicitly signalled. For chroma colour components, `has_palette_uv` and `palette_size_uv_minus_2` are shared between two chroma colour components, but the colour indices in the palette are signalled for Cb and Cr separately.

The selected palette indices of a palette mode coded block are signalled and coded in a diagonal scan order, as shown in Figure 26. The scan follows a diagonal direction that starts from top-right and ends at the bottom-left. After all indices along a diagonal line are coded, the pointer moves to the top-right sample of the next diagonal line. The first index of the current Palette coded block is first coded using a separate syntax, `color_index_map_y`, and the remaining indices are coded using their top, left, and top-left neighbouring indices as context for entropy coding.

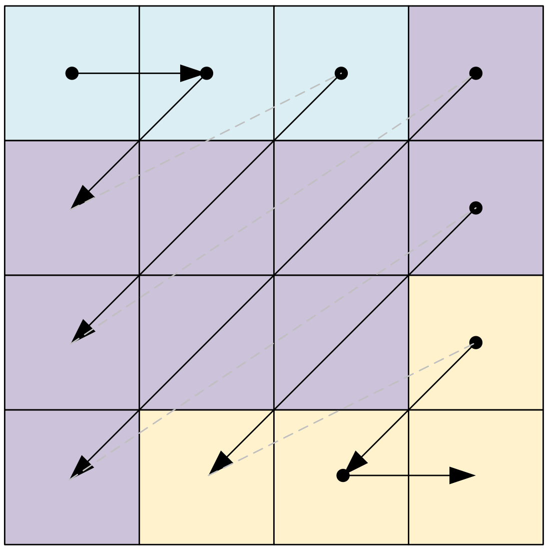
Figure 26: Wavefront coding order of palette tokens

3.8.3 Encoder content-type detection

Frame-level content-type detection may be enabled and applied before encoding each frame. This content-type detection process analyzes the characteristics of the current input frame and suggests whether it likely contains screen content. Based on this suggestion, screen content coding tools such as IntraBC and palette mode may be turned on for coding the current frame. The content-type detection process is described as follows, wherein the frame width and height are denoted as FrameWidth and FrameHeight, respectively.

Two counters, namely counter1 and counter2, are maintained in this content-type detection process. For each 16×16 luma block of the current frame, if there are only 2, 3, or 4 different luma values, then counter1 is first incremented by 1, and the variance of this 16×16 luma block is calculated. If the variance is greater than a predefined threshold value, then counter2 is incremented by 1. After all 16×16 luma blocks of the current frame are processed, counter1 is first compared with FrameWidth×FrameHeight/2560 to determine whether palette mode should be enabled for encoding the current frame, and counter2 is then compared with FrameWidth×FrameHeight/3072 to determine whether IntraBC should be enabled for encoding the current frame.

4 References

[1] https://aomedia.googlesource.com/aom/

[2] J. Han et al., "A Technical Overview of AV1," Proc. IEEE, 2021, vol. 109, pp. 1435-1462.

[3] L. Trudeau, N. Egge and D. Barr, "Predicting Chroma from Luma in AV1," 2018 Data Compression Conference, 2018, pp. 374-382.

[4] J. Han, Y. Xu and J. Bankoski, "A Dynamic Motion Vector Referencing Scheme for Video Coding," 2016 IEEE International Conference on Image Processing (ICIP), 2016, pp. 2032-2036.

[5] S. Parker et al., “Global and Locally Adaptive Warped Motion Compensation in Video Compression,” 2017 IEEE International Conference on Image Processing (ICIP), 2017, pp. 275-279.

[6] Y. Chen and D. Mukherjee, “Variable Block-Size Overlapped Block Motion Compensation in the Next Generation Open-Source Video Codec,” 2017 IEEE International Conference on Image Processing (ICIP), 2017, pp. 938-942.

[7] S. Parker et al., “On Transform Coding Tools Under Development for VP10,” Proc. SPIE 9971, Applications of Digital Image Processing XXXIX, 997119, Oct. 2016.

[8] J.-M. Valin et al., “Daala: Building a Next-Generation Video Codec from Unconventional Technology,” in Proc. IEEE 18th Int. Workshop Multimedia Signal Process. (MMSP), Sep. 2016, pp. 1–6.

[9] J. Han, C. Chiang and Y. Xu, “A Level-Map Approach to Transform Coefficient Coding,” 2017 IEEE International Conference on Image Processing (ICIP), 2017, pp. 3245-3249.

[10] S. Midtskogen and J. Valin, “The AV1 Constrained Directional Enhancement Filter (CDEF),” 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2018, pp. 1193-1197.

[11] D. Mukherjee et al., “A Switchable Loop-Restoration with Side-Information Framework for the Emerging AV1 Video Codec,” 2017 IEEE International Conference on Image Processing (ICIP), 2017, pp. 265-269.

[12] A. Norkin and N. Birkbeck, “Film Grain Synthesis for AV1 Video Codec,” 2018 Data Compression Conference, 2018, pp. 3-12.

[13] J. Li et al., “Intra Block Copy for Screen Content in the Emerging AV1 Video Codec,” 2018 Data Compression Conference, 2018, pp. 355-364.