|  |  |
| --- | --- |
| **Joint Collaborative Team on Video Coding (JCT-VC)**  **of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11**  8th Meeting: San José, CA, USA, 1–10 February, 2012 | Document: JCTVC-H0315 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **Non-CE2: Separate RQT structure for Y, U and V components** | | |
| *Status:* | Input Document to JCT-VC | | |
| *Purpose:* | Proposal | | |
|  |  | | |
| *Author(s) or Contact(s):* | \*Jicheng An, \*Xin Zhao, \*Xun Guo, #Shawmin Lei  \*North Building 10F, Raycom Infotech Park Tower C, No. 2 Kexueyuan South Rd., Haidian District, Beijing, China 100190  #No. 1, Dusing Rd. 1, Hsinchu Science Park, Hsinchu, Taiwan 30078 | Email: | {jicheng.an, x.zhao, xun.guo, shawmin.lei}@mediatek.com | |
| *Source:* | MediaTek Inc. | | |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Abstract

This contribution proposes a method using separate residue quad tree (RQT) structure for Y, U and V components. In HM5.0, the splitting transform unit (TU) information, signaled at TU level, is shared by Y, U and V components. In the proposed method, the splitting information of Y, U and V is signaled separately. Meanwhile, the maximum RQT depth values in sequence parameter set (SPS) are also separated for luma and chroma. It is reported that, by setting the proposed maximum chroma RQT depth as one, 1%~5% BD-rate reduction is achieved for chroma. It is also reported that, more BD-rate reduction is achieved for chroma by setting the maximum chroma RQT depth larger than one. The run time is reported to be similar to HM5.0.

# Introduction

In HM5.0, the RQT structure for Y, U, and V components is shared to be the same. In TU level, a syntax element **split\_transform\_flag** is used to specify whether a TU is split into four sub-TUs for all Y, U, and V components. However, since chroma components generally have smoother textures than luma, it may hurt the coding performance if the same luma RQT structure is applied for chroma. To improve the chroma performance, several contributions [1][2][3] were proposed to set the chroma RQT depth in an implicit manner, which is generally less than the corresponding luma RQT depth.

# Proposed method

In order to make the chroma TU splitting more efficient and flexible, this contribution proposes to separate the RQT structure for Y, U and V components. In specific, transform split flag is signaled separately for Y, U, and V components.

## Separate RQT structure for Y, U, and V components

In HM5.0, as shown in Fig. 1, only one **split\_transform\_flag** is signaled at each TU depth for all three components. And in this proposal, as shown in Fig.2, separate **split\_transform\_Y\_flag, split\_transform\_U\_flag and split\_transform\_V\_flag** are transmitted at each TU depth for Y, U and V, respectively.

Fig. 1 TU split in HM5.0

Fig. 2 TU split in proposed method

In HM5.0, the maximum RQT depth is signaled in SPS and shared between luma and chroma. In this proposal, the maximum RQT depth for luma and chroma are separately signaled in SPS. Table 1 shows comparisons of the related syntax definitions between HM5.0 and proposed method.

Table 1. Maximum RQT depth signaling in SPS for HM5.0 and proposed method

|  |  |
| --- | --- |
| **HM5.0** | **Proposed** |
| **max\_transform\_hierarchy\_depth\_inter** | **max\_transform\_hierarchy\_depth\_inter\_luma** |
| **max\_transform\_hierarchy\_depth\_inter\_chroma** |
| **max\_transform\_hierarchy\_depth\_intra** | **max\_transform\_hierarchy\_depth\_intra\_luma** |
| **max\_transform\_hierarchy\_depth\_intra\_chroma** |

## TU Split flag and cbf flag coding

Since different RQT structure from HM5.0 is proposed in this contribution, the coding scheme of TU split and cbf flag need to be changed accordingly. As shown in Fig. 3, in HM5.0, the TU split and cbf flags are recursively coded. Furthermore, for luma, the cbf is coded only at leaf TU level, and for chroma, the cbf is coded from root to leaf TU level. In the proposed method, the similar cbf coding process is maintained, but separate signaling of TU split is applied for Y, U, and V, as shown in Fig. 4 and Fig. 5. Note that TrDepth and blkIdx in the figures represent the current transform depth and block index, respectively.



Fig. 3 TU split and cbf flag coding in HM5.0



Fig. 4. TU split and cbf coding for Y component in proposed method



Fig. 5. TU split and cbf coding for U or V component in proposed method

## Transform coefficients coding for Y, U, and V components

In HM5.0, the transform coefficients for Y, U, and V components are coded interleaved at leaf TU level as shown in Fig 6. In the proposed method, this interleaving is changed straightforwardly as shown in Fig. 7. Note that currTrDepth and blkIdx in the figures represent the current transform depth and block index, respectively. The TrDepth in Fig. 6 represents the final transform depth of a leaf TU for all Y, U, V components, and the TrDepthY, TrDepthU, and TrDepthV in Fig. 7 represent the final transform depth of a leaf TU for Y, U, and V components respectively.



Fig. 6 Interleave coding for Y, U, and V coefficients in HM5.0



Fig. 7 Interleave coding for Y, U, and V coefficients in the proposed method

# Experimental results

The proposed method has been implemented on top of HM5.0. The anchor is HM5.0 with common test condition setting [4]. In the proposed method, different maximum chroma RQT depth values, i.e., 1, 2 and 3, for both inter (**max\_transform\_hierarchy\_depth\_inter\_chroma**)and intra (**max\_transform\_hierarchy\_depth\_intra\_chroma**) are tested. And the maximum RQT depth for luma is always set to 3, which is applied for both luma and chroma in the HM5.0 anchor.

Table 2 Performance of maximum chroma RQT depth = 1

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **All Intra HE** | | | **All Intra LC** | | | **All Intra HE-10** | | |
|  | Y | U | V | Y | U | V | Y | U | V |
| Class A (8bit) | -0.02% | -2.43% | -3.41% | 0.01% | -1.51% | -2.42% |  |  |  |
| Class B | -0.11% | -3.09% | -3.34% | 0.02% | -2.53% | -3.24% |  |  |  |
| Class C | -0.05% | -1.27% | -1.47% | 0.02% | -1.20% | -1.46% |  |  |  |
| Class D | -0.02% | -0.75% | -0.81% | 0.01% | -0.72% | -0.70% |  |  |  |
| Class E | 0.02% | -5.67% | -4.94% | 0.05% | -5.00% | -4.26% |  |  |  |
| **Overall** | -0.04% | -2.52% | -2.64% | 0.02% | -2.13% | -2.36% |  |  |  |
|  | -0.05% | -2.54% | -2.64% | 0.02% | -2.16% | -2.38% |  |  |  |
| Class F | 0.03% | -1.70% | -1.88% | 0.02% | -1.02% | -1.04% |  |  |  |
| Enc Time[%] | 100.6% | | | 100.6% | | |  | | |
| Dec Time[%] | 100.7% | | | 99.8% | | |  | | |
|  |  |  |  |  |  |  |  |  |  |
|  | **Random Access HE** | | | **Random Access LC** | | | **Random Access HE-10** | | |
|  | Y | U | V | Y | U | V | Y | U | V |
| Class A (8bit) | -0.10% | -1.98% | -3.25% | -0.01% | -1.41% | -2.59% | 0.01% | -7.25% | -7.54% |
| Class B | -0.08% | -2.67% | -2.93% | -0.02% | -2.40% | -3.12% | -0.03% | -2.97% | -3.23% |
| Class C | -0.03% | -0.87% | -0.97% | 0.01% | -0.66% | -0.77% |  |  |  |
| Class D | -0.03% | -0.09% | -0.03% | -0.01% | -0.03% | -0.42% |  |  |  |
| Class E |  |  |  |  |  |  |  |  |  |
| **Overall** | -0.05% | -1.41% | -1.68% | -0.01% | -1.17% | -1.70% | -0.01% | -4.87% | -5.14% |
|  | -0.05% | -1.43% | -1.68% | -0.01% | -1.19% | -1.74% | -0.01% | -4.86% | -5.08% |
| Class F | -0.06% | -1.51% | -1.72% | 0.06% | -1.24% | -0.89% |  |  |  |
| Enc Time[%] | 96.8% | | | 98.2% | | | 96.85% | | |
| Dec Time[%] | 101.3% | | | 101.5% | | | 100.67% | | |
|  |  |  |  |  |  |  |  |  |  |
|  | **Low delay B HE** | | | **Low delay B LC** | | | **Low delay B HE-10** | | |
|  | Y | U | V | Y | U | V | Y | U | V |
| Class A |  |  |  |  |  |  |  |  |  |
| Class B | -0.09% | -1.68% | -1.59% | -0.10% | -1.90% | -2.25% |  |  |  |
| Class C | -0.06% | -0.18% | -0.42% | -0.04% | -0.05% | -0.50% |  |  |  |
| Class D | -0.10% | 1.20% | 0.48% | -0.07% | 1.43% | -0.12% |  |  |  |
| Class E | -0.09% | -5.63% | -4.76% | 0.06% | -5.67% | -5.11% |  |  |  |
| **Overall** | -0.08% | -1.32% | -1.37% | -0.05% | -1.31% | -1.82% |  |  |  |
|  | -0.08% | -1.26% | -1.37% | -0.05% | -1.36% | -1.83% |  |  |  |
| Class F | -0.20% | -1.51% | -1.65% | 0.13% | -1.76% | -1.62% |  |  |  |
| Enc Time[%] | 97.5% | | | 99.1% | | |  | | |
| Dec Time[%] | 101.5% | | | 100.9% | | |  | | |

Table 3 Performance of maximum chroma RQT depth = 2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **All Intra HE** | | | **All Intra LC** | | | **All Intra HE-10** | | |
|  | Y | U | V | Y | U | V | Y | U | V |
| Class A (8bit) | 0.16% | -3.42% | -4.58% | 0.00% | -1.62% | -2.52% |  |  |  |
| Class B | 0.06% | -4.23% | -4.53% | 0.00% | -2.62% | -3.44% |  |  |  |
| Class C | 0.04% | -1.87% | -2.18% | 0.01% | -1.21% | -1.55% |  |  |  |
| Class D | 0.04% | -1.05% | -1.29% | 0.01% | -0.70% | -0.78% |  |  |  |
| Class E | 0.16% | -6.11% | -5.39% | 0.10% | -4.83% | -4.17% |  |  |  |
| **Overall** | 0.08% | -3.22% | -3.44% | 0.02% | -2.14% | -2.45% |  |  |  |
|  | 0.07% | -3.22% | -3.43% | 0.02% | -2.16% | -2.48% |  |  |  |
| Class F | 0.09% | -1.84% | -2.11% | 0.03% | -0.97% | -1.00% |  |  |  |
| Enc Time[%] | 103.3% | | | 103.8% | | |  | | |
| Dec Time[%] | 100.7% | | | 100.1% | | |  | | |
|  |  |  |  |  |  |  |  |  |  |
|  | **Random Access HE** | | | **Random Access LC** | | | **Random Access HE-10** | | |
|  | Y | U | V | Y | U | V | Y | U | V |
| Class A (8bit) | -0.11% | -2.52% | -3.84% | -0.01% | -1.96% | -3.09% | 0.03% | -10.40% | -9.93% |
| Class B | -0.11% | -2.93% | -3.16% | -0.01% | -3.21% | -3.92% | -0.07% | -3.35% | -3.60% |
| Class C | -0.10% | -0.92% | -1.12% | -0.03% | -1.26% | -1.52% |  |  |  |
| Class D | -0.08% | -0.01% | -0.24% | -0.04% | -0.45% | -0.84% |  |  |  |
| Class E |  |  |  |  |  |  |  |  |  |
| **Overall** | -0.10% | -1.56% | -1.93% | -0.03% | -1.79% | -2.34% | -0.03% | -6.49% | -6.41% |
|  | -0.10% | -1.60% | -1.92% | -0.03% | -1.80% | -2.42% | -0.02% | -6.46% | -6.34% |
| Class F | 0.00% | -1.69% | -1.73% | 0.04% | -1.41% | -1.05% |  |  |  |
| Enc Time[%] | 98.7% | | | 99.3% | | | 98.72% | | |
| Dec Time[%] | 101.3% | | | 101.1% | | | 100.73% | | |
|  |  |  |  |  |  |  |  |  |  |
|  | **Low delay B HE** | | | **Low delay B LC** | | | **Low delay B HE-10** | | |
|  | Y | U | V | Y | U | V | Y | U | V |
| Class A |  |  |  |  |  |  |  |  |  |
| Class B | -0.22% | -0.08% | 0.28% | -0.09% | -2.63% | -3.42% |  |  |  |
| Class C | -0.20% | 0.54% | 0.77% | -0.05% | -1.01% | -1.34% |  |  |  |
| Class D | -0.18% | 2.76% | 2.33% | -0.10% | 0.23% | -1.12% |  |  |  |
| Class E | -0.06% | -2.71% | -3.44% | 0.02% | -7.45% | -6.06% |  |  |  |
| **Overall** | -0.18% | 0.29% | 0.22% | -0.06% | -2.42% | -2.82% |  |  |  |
|  | -0.18% | 0.33% | 0.15% | -0.06% | -2.43% | -2.83% |  |  |  |
| Class F | -0.31% | -1.47% | -1.95% | 0.09% | -1.86% | -2.15% |  |  |  |
| Enc Time[%] | 99.0% | | | 100.2% | | |  | | |
| Dec Time[%] | 101.3% | | | 100.9% | | |  | | |

Table 4 Performance of maximum chroma RQT depth = 3

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **All Intra HE** | | | **All Intra LC** | | | **All Intra HE-10** | | |
|  | Y | U | V | Y | U | V | Y | U | V |
| Class A (8bit) | 0.19% | -3.55% | -4.71% | 0.00% | -1.66% | -2.56% |  |  |  |
| Class B | 0.08% | -4.44% | -4.72% | 0.00% | -2.66% | -3.49% |  |  |  |
| Class C | 0.05% | -1.90% | -2.25% | 0.01% | -1.21% | -1.56% |  |  |  |
| Class D | 0.05% | -1.12% | -1.30% | 0.02% | -0.71% | -0.77% |  |  |  |
| Class E | 0.15% | -6.15% | -5.39% | 0.10% | -4.83% | -4.18% |  |  |  |
| **Overall** | 0.09% | -3.33% | -3.52% | 0.02% | -2.15% | -2.47% |  |  |  |
|  | 0.09% | -3.32% | -3.51% | 0.02% | -2.18% | -2.50% |  |  |  |
| Class F | 0.10% | -1.91% | -2.16% | 0.03% | -0.88% | -1.01% |  |  |  |
| Enc Time[%] | 104.5% | | | 105.1% | | |  | | |
| Dec Time[%] | 100.8% | | | 100.1% | | |  | | |
|  |  |  |  |  |  |  |  |  |  |
|  | **Random Access HE** | | | **Random Access LC** | | | **Random Access HE-10** | | |
|  | Y | U | V | Y | U | V | Y | U | V |
| Class A (8bit) | -0.09% | -2.80% | -3.78% | 0.01% | -2.04% | -3.24% | 0.09% | -12.42% | -12.40% |
| Class B | -0.10% | -3.31% | -3.44% | 0.00% | -3.34% | -4.06% | -0.06% | -3.56% | -3.77% |
| Class C | -0.12% | -1.00% | -1.32% | 0.03% | -1.31% | -1.63% |  |  |  |
| Class D | -0.08% | -0.10% | -0.28% | -0.03% | -0.51% | -0.95% |  |  |  |
| Class E |  |  |  |  |  |  |  |  |  |
| **Overall** | -0.10% | -1.77% | -2.08% | 0.00% | -1.87% | -2.47% | 0.01% | -7.50% | -7.60% |
|  | -0.10% | -1.79% | -2.07% | 0.00% | -1.89% | -2.53% | 0.01% | -7.45% | -7.51% |
| Class F | 0.00% | -1.73% | -1.84% | 0.02% | -1.44% | -1.16% |  |  |  |
| Enc Time[%] | 100.0% | | | 100.2% | | | 100.0% | | |
| Dec Time[%] | 101.2% | | | 100.6% | | | 100.4% | | |
|  |  |  |  |  |  |  |  |  |  |
|  | **Low delay B HE** | | | **Low delay B LC** | | | **Low delay B HE-10** | | |
|  | Y | U | V | Y | U | V | Y | U | V |
| Class A |  |  |  |  |  |  |  |  |  |
| Class B | -0.18% | -0.25% | -0.09% | -0.07% | -2.89% | -3.18% |  |  |  |
| Class C | -0.17% | 0.24% | 0.58% | -0.02% | -0.97% | -1.42% |  |  |  |
| Class D | -0.19% | 2.44% | 1.53% | -0.08% | 0.54% | -1.12% |  |  |  |
| Class E | -0.13% | -1.79% | -2.84% | 0.02% | -8.41% | -4.61% |  |  |  |
| **Overall** | -0.17% | 0.26% | -0.03% | -0.04% | -2.59% | -2.50% |  |  |  |
|  | -0.17% | 0.31% | -0.07% | -0.04% | -2.59% | -2.48% |  |  |  |
| Class F | -0.17% | -1.50% | -1.45% | 0.13% | -2.25% | -1.69% |  |  |  |
| Enc Time[%] | 100.1% | | | 101.0% | | |  | | |
| Dec Time[%] | 101.3% | | | 100.8% | | |  | | |

From Table 1, it is reported that when maximum chroma RQT depth is set to 1, 1%~5% chroma BD-rate gain and encoding complexity reduction are achieved. When larger maximum chroma RQT depth values are applied, as shown in Table 2 and Table 3, more chroma BD-rate reduction is achieved for most cases. The reported results validate the conclusion that separate settings of RQT depth for Y, U and V can improve the coding performance, especially for some special sequences (such as NebutaFestival, SteamLocomotiveTrain (chroma gain larger than 20%), and so on).

# Conclusion

In this proposal, a method using separate RQT depth for Y, U, and V components is proposed. Since Y, U, and V components generally present different statistics in residual coding, it is proposed to apply separate RQT structure signaling for Y, U and V, respectively. Experimental results demonstrate that, by explicitly signaling the TU split flag for Y, U, and V components separately, further chroma BD-rate reduction is achieved.

# References

1. X. Zhao, X. Guo, X. Li, and S. Lei, “Residue Quad Tree Depth for Chroma in Intra coding,” JCTVC-G283, Geneva, CH, Nov., 2011.
2. A. Minezawa, K. Sugimoto, and S. Sekiguchi, “Improvement to chroma TU specification”, JCTVC-G442, Geneva, CH, Nov., 2011.
3. L. Guo, X. Wang, and M. Karczewicz, “Evaluation of Limiting Chroma Transform Depth in RQT on HM4.0”, JCTVC-G980, Geneva, CH, Nov., 2011.
4. F. Bossen, “Common test conditions and software reference configurations”, JCTVC-G1200, Geneva, CH, Nov., 2011.

# Patent rights declaration(s)

**MediaTek Inc. may have current or pending patent rights relating to the technology described in this contribution and, conditioned on reciprocity, is prepared to grant licenses under reasonable and non-discriminatory terms as necessary for implementation of the resulting ITU-T Recommendation | ISO/IEC International Standard (per box 2 of the ITU-T/ITU-R/ISO/IEC patent statement and licensing declaration form).**

# Appendix: proposed working draft modification

### Coding unit syntax

|  |  |
| --- | --- |
| coding\_unit( x0, y0, log2CUSize ) { | Descriptor |
| if( slice\_type != I ) |  |
| **skip\_flag[** x0 **][** y0 **]** | ae(v) |
| if( skip\_flag[ x0 ][ y0 ] ) |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| else if( slice\_type != I | | log2CUSize = = Log2MinCUSize ) { |  |
| **pred\_type** | ae(v) |
| x1 = x0 + ( ( 1 << log2CUSize ) >> 1 ) |  |
| y1 = y0 + ( ( 1 << log2CUSize ) >> 1 ) |  |
| x2 = x1 − ( ( 1 << log2CUSize ) >> 2 ) |  |
| y2 = y1 − ( ( 1 << log2CUSize ) >> 2 ) |  |
| x3 = x1 + ( ( 1 << log2CUSize ) >> 2 ) |  |
| y3 = y1 + ( ( 1 << log2CUSize ) >> 2 ) |  |
| if( PartMode == PART\_2Nx2N ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| } else if( PartMode == PART\_2NxN ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x0, y1 , log2CUSize ) |  |
| } else if( PartMode == PART\_Nx2N ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x1, y0 , log2CUSize ) |  |
| } else if( PartMode == PART\_2NxnU ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x0, y2 , log2CUSize ) |  |
| } else if( PartMode == PART\_2NxnD ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x0, y3 , log2CUSize ) |  |
| } else if( PartMode == PART\_nLx2N ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x2, y0 , log2CUSize ) |  |
| } else if( PartMode == PART\_nRx2N ) { |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x3, y0 , log2CUSize ) |  |
| } else { /\* PART\_NxN \*/ |  |
| prediction\_unit( x0, y0 , log2CUSize ) |  |
| prediction\_unit( x1, y0 , log2CUSize ) |  |
| prediction\_unit( x0, y1 , log2CUSize ) |  |
| prediction\_unit( x1, y1 , log2CUSize ) |  |
| } |  |
| if( !pcm\_flag ) { |  |
| if(PredMode != MODE\_INTRA && !(PartMode==PART\_2Nx2N && merge\_flag[x0][y0])) { |  |
| **no\_residual\_data\_flag** | ae(v) |
| **}** |  |
| if( !no\_residual\_data\_flag ) { |  |
| transform\_tree( x0, y0, log2CUSize, log2CUSize, 0, 0, 1 ) |  |
| transform\_tree( x0, y0, log2CUSize, log2CUSize, 0, 0, 2 ) |  |
| transform\_tree( x0, y0, log2CUSize, log2CUSize, 0, 0, 0 ) |  |
| transform\_coeff( x0, y0, log2CUSize, log2CUSize, 0, 0 ) |  |
| transform\_coeff( x0, y0, log2CUSize, log2CUSize, 0, 1 ) |  |
| transform\_coeff( x0, y0, log2CUSize, log2CUSize, 0, 2 ) |  |
| } |  |
| } |  |
| } |  |
| } |  |

### Transform tree syntax

|  |  |
| --- | --- |
| transform\_tree( x0, y0, log2TrafoWidth, log2TrafoHeight, trafoDepth, blkIdx, cIdx ) { | Descriptor |
| log2TrafoWidth = cIdx!=0 ? log2TrafoWidth>>1: log2TrafoWidth |  |
| log2TrafoHeight = cIdx!=0 ? log2TrafoHeight >>1: log2TrafoHeight |  |
| log2TrafoSize = ( log2TrafoWidth + log2TrafoHeight ) >> 1 |  |
| xBase = x0 − ( x0 & ( 1 << log2TrafoWidth ) ) |  |
| yBase = y0 − ( y0 & ( 1 << log2TrafoHeight ) ) |  |
| If (cIdx == 0) { |  |
| maxDepth = ( PredMode = = MODE\_INTRA ?  max\_transform\_hierarchy\_depth\_intra\_luma + IntraSplitFlag :  max\_transform\_hierarchy\_depth\_inter\_luma ) |  |
| if( log2TrafoSize <= Log2MaxTrafoSize &&  log2TrafoSize > Log2MinTrafoSize &&  trafoDepth < maxDepth && !(intraSplitFlag && trafoDepth = = 0) ) { |  |
| ImplicitSplitFlag = (maxDepth= =0 && PartMode != PART\_2Nx2N) |  |
| If (!ImplicitSplitFlag) { |  |
| **split\_transform\_flag**[cIdx][ x0 ][ y0 ][ trafoDepth ] | ae(v) |
| } |  |
| } |  |
| If(split\_transform\_flag[cIdx][ x0 ][ y0 ][ trafoDepth ]) { |  |
| if( InterTUSplitDirection = = 2 ) { |  |
| x1 = x0 + ( ( 1 << log2TrafoWidth ) >> 1 ) |  |
| y1 = y0 |  |
| x2 = x0 |  |
| y2 = y0 + ( ( 1 << log2TrafoHeight ) >> 1 ) |  |
| x3 = x1 |  |
| y3 = y2 |  |
| } else { |  |
| x1 = x0 + ( ( 1 << log2TrafoWidth ) >> 2 ) \* InterTUSplitDirection |  |
| y1 = y0 + ( ( 1 << log2TrafoHeight) >> 2 ) \* ( 1 − InterTUSplitDirection ) |  |
| x2 = x1 + ( ( 1 << log2TrafoWidth ) >> 2 ) \* InterTUSplitDirection |  |
| y2 = y1 + ( ( 1 << log2TrafoHeight) >> 2 ) \* ( 1 − InterTUSplitDirection ) |  |
| x3 = x2 + ( ( 1 << log2TrafoWidth ) >> 2 ) \* InterTUSplitDirection |  |
| y3 = y2 + ( ( 1 << log2TrafoHeight) >> 2 ) \* ( 1 − InterTUSplitDirection ) |  |
| log2TrafoHeight = log2TrafoHeight + 2 \* InterTUSplitDirection − 1 |  |
| log2TrafoWidth = log2TrafoWidth − 2 \* InterTUSplitDirection + 1 |  |
| } |  |
| transform\_tree( x0, y0, log2TrafoWidth − 1, log2TrafoHeight − 1, trafoDepth + 1, 0, 0 ) |  |
| transform\_tree( x1, y1, log2TrafoWidth − 1, log2TrafoHeight − 1, trafoDepth + 1, 1, 0 ) |  |
| transform\_tree( x2, y2, log2TrafoWidth − 1, log2TrafoHeight − 1, trafoDepth + 1, 2, 0 ) |  |
| transform\_tree( x3, y3, log2TrafoWidth − 1, log2TrafoHeight − 1, trafoDepth + 1, 3, 0 ) |  |
| } |  |
| Else { |  |
| if( PredMode = = MODE\_INTRA | | trafoDepth != 0 | |  cbf\_cb[ x0 ][ y0 ][ trafoDepth ] | |  cbf\_cr[ x0 ][ y0 ][ trafoDepth ] ) { |  |
| readCbf = true |  |
| if( blkIdx = = 3 && PredMode != MODE\_INTRA ) |  |
| readCbf = cbf\_luma[ xBase ][ yBase ][ trafoDepth ] | |   cbf\_luma[ xBase + ( 1 << log2TrafoWidth ) ][ yBase ][ trafoDepth ] | |   cbf\_luma[ xBase ][ yBase + ( 1 << log2TrafoHeight ) ][ trafoDepth ] |  |
| if ( !readCbf && (log2TrafoSize < Log2MaxTrafoSize ||  (trafoDepth= =1 && !cbf\_cb[ x0 ][ y0 ][ trafoDepth ]  && !cbf\_cb[ x0 ][ y0 ][ trafoDepth ])) |  |
| cbf\_luma[ x0 ][ y0 ][ trafoDepth ] = 1 |  |
| else |  |
| **cbf\_luma**[ x0 ][ y0 ][ trafoDepth ] | ae(v) |
| } |  |
| } |  |
| } | ae(v) |
| Else{ |  |
| maxDepth = ( PredMode = = MODE\_INTRA ?  max\_transform\_hierarchy\_depth\_intra\_chroma + IntraSplitFlag :  max\_transform\_hierarchy\_depth\_inter\_chroma ) |  |
| if( log2TrafoSize <= Log2MaxTrafoSize){ |  |
| firstChromaCbf = ( log2TrafoSize = = Log2MaxTrafoSize | |  trafoDepth = = 0 ? 1 : 0 ) |  |
| if( firstChromaCbf | | cbf\_cb[ xBase ][ yBase ][ trafoDepth − 1 ] ) { |  |
| readCbf = true |  |
| if( blkIdx = = 3 && log2TrafoSize < Log2MaxTrafoSize ) |  |
| readCbf = cbf\_cb[ xBase ][ yBase ][ trafoDepth ] | |   cbf\_cb[ xBase + ( 1 << log2TrafoWidth ) ][ yBase ][ trafoDepth ] | |   cbf\_cb[ xBase ][ yBase + ( 1 << log2TrafoHeight ) ][ trafoDepth ] |  |
| if ( !readCbf ) |  |
| cIdx= =1 ? cbf\_cb[ x0 ][ y0 ][ trafoDepth ] = 1 :  cbf\_cr[ x0 ][ y0 ][ trafoDepth ] = 1 |  |
| else |  |
| cIdx= =1 ? **cbf\_cb**[ x0 ][ y0 ][ trafoDepth ] ： **cbf\_cr**[ x0 ][ y0 ][ trafoDepth ] | ae(v) |
| } |  |
| cbf = cIdx= =1 ? cbf\_cb[ x0 ][ y0 ][ trafoDepth ] :  cbf\_cr[ x0 ][ y0 ][ trafoDepth ] |  |
| If(cbf || PredMode = = MODE\_INTRA){ |  |
| If( log2TrafoSize > Log2MinTrafoSize &&  trafoDepth < maxDepth && !(intraSplitFlag && trafoDepth = = 0) ) { |  |
| ImplicitSplitFlag = (maxDepth= =0 && PartMode != PART\_2Nx2N) |  |
| If (!ImplicitSplitFlag) { |  |
| **split\_transform\_flag**[cIdx][ x0 ][ y0 ][ trafoDepth ] | ae(v) |
| } |  |
| } |  |
| Else{ |  |
| split\_transform\_flag[cIdx][ x0 ][ y0 ][ trafoDepth ] = 0 |  |
| } |  |
| } |  |
| If(split\_transform\_flag[cIdx][ x0 ][ y0 ][ trafoDepth ]){ |  |
| if( InterTUSplitDirection = = 2 ) { |  |
| x1 = x0 + ( ( 1 << log2TrafoWidth ) >> 1 ) |  |
| y1 = y0 |  |
| x2 = x0 |  |
| y2 = y0 + ( ( 1 << log2TrafoHeight ) >> 1 ) |  |
| x3 = x1 |  |
| y3 = y2 |  |
| } else { |  |
| x1 = x0 + ( ( 1 << log2TrafoWidth ) >> 2 ) \* InterTUSplitDirection |  |
| y1 = y0 + ( ( 1 << log2TrafoHeight) >> 2 ) \* ( 1 − InterTUSplitDirection ) |  |
| x2 = x1 + ( ( 1 << log2TrafoWidth ) >> 2 ) \* InterTUSplitDirection |  |
| y2 = y1 + ( ( 1 << log2TrafoHeight) >> 2 ) \* ( 1 − InterTUSplitDirection ) |  |
| x3 = x2 + ( ( 1 << log2TrafoWidth ) >> 2 ) \* InterTUSplitDirection |  |
| y3 = y2 + ( ( 1 << log2TrafoHeight) >> 2 ) \* ( 1 − InterTUSplitDirection ) |  |
| log2TrafoHeight = log2TrafoHeight + 2 \* InterTUSplitDirection − 1 |  |
| log2TrafoWidth = log2TrafoWidth − 2 \* InterTUSplitDirection + 1 |  |
| } |  |
| transform\_tree( x0, y0, log2TrafoWidth − 1, log2TrafoHeight – 1  , trafoDepth + 1, 0, cIdx ) |  |
| transform\_tree( x1, y1, log2TrafoWidth − 1, log2TrafoHeight – 1  , trafoDepth + 1, 1, cIdx ) |  |
| transform\_tree( x2, y2, log2TrafoWidth − 1, log2TrafoHeight – 1  , trafoDepth + 1, 2, cIdx ) |  |
| transform\_tree( x3, y3, log2TrafoWidth − 1, log2TrafoHeight – 1  , trafoDepth + 1, 3, cIdx ) |  |
| } |  |
| } |  |
| } |  |

**split\_transform\_flag**[cIdx][ x0 ][ y0 ][ trafoDepth ] specifies whether a block is split into four blocks with smaller horizontal or vertical size for the purpose of transform coding. The array index cIdx specifies an indicator for the colour component; it is equal to 0 for luma, equal to 1 for Cb, and equal to 2 for Cr. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered block relative to the top-left luma sample of the picture. The array index trafoDepth specifies the current subdivision level of a coding unit into blocks for the purpose of transform coding. trafoDepth is equal to 0 for blocks that correspond to coding units.

When split\_transform\_flag[ x0 ][ y0 ][ trafoDepth ] is not present, it is inferred as follows:

* If log2TrafoSize is greater than Log2MaxTrafoSize or intraSplitFlag is equal to 1, the value of split\_transform\_flag[cIdx] [ x0 ][ y0 ][ trafoDepth ] is inferred to be equal to 1.
* Otherwise (log2TrafoSize is less than or equal to Log2MaxTrafoSize and intraSplitFlag is equal to 0), the value of split\_transform\_flag[cIdx] [ x0 ][ y0 ][ trafoDepth ] is inferred to be equal to 0.



### Deblocking filter process

A conditional filtering process shall be performed on a treeblock basis after the completion of the picture construction process prior to deblocking filter process for the entire decoded picture (as specified in subclauses XXX and YYY) [Ed.: (WJ) those subclauses seem not defined yet], with all treeblocks in a picture processed in order of increasing treeblock addresses.

Each treeblock is processed on a coding unit basis with the same order as decoding process. For each coding unit, vertical edges are filtered first, starting with the edge on the left-hand side of the coding unit proceeding through the edges towards the right-hand side of the coding unit in their geometrical order. Then, the horizontal edges are filtered starting with the edge on the top of the coding unit proceeding through the edges towards the bottom of the coding unit in their geometrical order.

Sample values above of the current coding unit that may have already been modified by the filtering of horizontal edges of deblocking filter process operation on previous coding unit shall be used as inputs to the deblocking filter process on the current coding unit and may be further modified during the filtering of the current coding unit. Sample values to the left of the current coding unit shall be used as inputs to the deblocking filter process on the current coding unit and may be further modified during the filtering of the current coding unit. Sample values to the left of the current coding unit may be modified by the filtering of vertical edge and may be further modified by the filtering of horizontal edges.

Sample values modified during filtering of vertical edges are used as input for the filtering of the horizontal edges. For sample values modified by both filtering of horizontal edges and filtering of vertical edges, filtering of horizontal edges is applied after filtering of vertical edges.

The deblocking filter process shall be applied to all prediction unit edges and transform unit edges of a picture, except edges at the boundary of the picture, any edges for which the deblocking filter process is disabled by disable\_deblocking\_filter\_flag and any edges coinside with slice boundaries when loop\_filter\_across\_slice\_flag is equal to 0. For the transform units and prediction units with edges smaller than 8 samples in either vertical or horizontal direction, only the edges lying on the 8x8 sample grid are filtered.

When disable\_deblocking\_filter\_flag is not equal to 1, the deblocking filter process is invoked as the following ordered steps for each coding unit with the same order as decoding process.

1. The coding unit size nS is set equal to 1 << log2CUSize.
2. The variables FilterInternalEdgesFlag, FilterLeftCuEdgeFlag and FilterTopCuEdgeFlag are derived as follows.

* The variable FilterInternalEdges is set equal to 1.
* If the left boundary of current coding unit is the left boundary of the picture or if the left boundary of current coding unit is the left boundary of the slice and loop\_filter\_across\_slice\_flag is equal to 0, the variable FilterLeftCuEdgeFlag is set equal to 0, otherwise set equal to 1.
* If the top boundary of current coding unit is the top boundary of the picture or if the top boundary of current coding unit is the top boundary of the slice and loop\_filter\_across\_slice\_flag is equal to 0, the variable FilterTopCuEdgeFlag is set equal to 0, otherwise set equal to 1.

1. All elements of two-dimensional array of size (nS)x(nS), horEdgeFlags and verEdgeFlags, horEdgeCbFlags, verEdgeCbFlags, horEdgeCrFlags, and verEdgeCrFlags are initialized to zero.
2. The derivation process of transform unit boundary specified in subclause are invoked with the luma location ( xB, yB ) set equal to ( 0, 0 ), the transform unit width log2TrafoWidth set equal to log2CUSize, the transform unit height log2TrafoHeight set equal to log2CUSize and the variable trafoDepth set equal to 0 as the inputs and the modified horEdgeFlags and verEdgeFlags as outputs.
3. The derivation process of chroma transform unit boundary specified in subclause are invoked with the chroma location ( xB, yB ) set equal to ( 0, 0 ), the transform unit width log2TrafoWidth set equal to log2CUSize-1, the transform unit height log2TrafoHeight set equal to log2CUSize-1 and the variable trafoDepth set equal to 0 as the inputs and the modified horEdgeCbFlags, verEdgeCbFlags, horEdgeCrFlags, and verEdgeCrFlags as outputs.
4. The derivation process of prediction unit boundary specified in subclause are invoked with the coding unit size log2CUSize and the prediction partition mode PartMode as inputs, and the modified horEdgeFlags and verEdgeFlags as outputs.
5. The derivation process of the boundary filtering strength specified in subclause 8.6.1.4 is invoked with the luma location ( xC, yC ), the coding unit size log2CUSize, horEdgeFlags and verEdgeFlags as inputs and an array of size (2)x(nS)x(nS), bS as output.
6. The derivation process of the chroma boundary filtering strength specified in subclause is invoked with the chroma location ( xC/2, yC/2 ), the chroma coding unit size log2CUSize-1, horEdgeCbFlags, verEdgeCbFlags, horEdgeCrFlags, and verEdgeCrFlags as inputs and two arraies of size (2)x(nS)x(nS), bSCb and bSCr as output.
7. The filtering process for coding unit specified in subclause 8.6.1.6 are invoked with the luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture, the coding unit size log2CUSize, and the array bS, bSCb, and bSCr as inputs and the modified reconstructued picture as output.

#### Derivation process of chroma transform unit boundary

Input of this process are:

– a chroma location ( xB, yB ) specifying the top-left luma sample of the current block relative to the top‑left chroma sample of the current coding unit,

– a variable log2TrafoWidth specifying the width of the current chroma block,

– a variable log2TrafoHeight specifying the height of the current chroma block,

– a variable trafoDepth.

Outputs of this process are:

– two-dimensional arrays of (nS)x(nS), horEdgeCbFlags and verEdgeCbFlags,

– two-dimensional arrays of (nS)x(nS), horEdgeCrFlags and verEdgeCrFlags.

Depending on split\_transform\_flag[1][ xB ][ yB ][ trafoDepth ], the following applies:

– If split\_transform\_flag[1][ xB ][ yB ][ trafoDepth ] is equal to 1, the following ordered steps apply:

1. The variables xB1, yB1, xB2, yB2, xB3 and yB3 are derived as follows.
   * If InterTUSplitDirection is equal to 2, the following applies.
   * The variable xB1 is set equal to xB + ( ( 1 << log2TrafoWidth ) >> 1 ).
   * The variable yB1 is set equal to yB.
   * The variable xB2 is set equal to xB.
   * The variable yB1 is set equal to yB + ( ( 1 << log2TrafoHeight ) >> 1 ).
   * The variable xB3 is set equal to xB1.
   * The variable yB3 is set equal to yB2.
   * The variable log2TrafoWidth1 is set equal to log2TrafoWidth − 1.
   * The variable log2TrafoHeight1 is set equal to log2TrafoHeight − 1.
   * Otherwise (InterTUSplitDirection is equal to 0 or 1), the following applies.
   * The variable xB1 is set equal to xB + ((1 << (log2TrafoWidth)) >> 2) \* InterTUSplitDirection.
   * The variable yB1 is set equal to yB + ((1 << (log2TrafoHeight)) >> 2) \* (1 − InterTUSplitDirection).
   * The variable xB2 is set equal to xB1 + ((1 << (log2TrafoWidth)) >> 2) \* InterTUSplitDirection.
   * The variable yB2 is set equal to yB1 + ((1 << (log2TrafoHeight)) >> 2) \* (1 − InterTUSplitDirection).
   * The variable xB3 is set equal to xB2 + ((1 << (log2TrafoWidth)) >> 2) \* InterTUSplitDirection.
   * The variable yB3 is set equal to yB2 + ((1 << (log2TrafoHeight)) >> 2) \* (1 − InterTUSplitDirection).
   * The variable log2TrafoWidth1 is set equal to (log2TrafoWidth − 2) \* InterTUSplitDirection.
   * The variable log2TrafoHeight1 is set equal to (log2TrafoHeight − 2) \* (1 − InterTUSplitDirection).
2. The deriviation process of transform unit boundary as specified in this subclause is invoked with the chroma location ( xB, yB ), the variable log2TrafoWidth set equal to log2TrafoWidth1, the variable log2TrafoHeight set equal to log2TrafoHeight1 and the variable trafoDepth1 set equal to trafoDepth + 1 as inputs and the outputs are the modified versions of two arrays, horEdgeCbFlags and verEdgeCbFlags.
3. The deriviation process of transform unit boundary as specified in this subclause is invoked with the chroma location ( xB1, yB1 ), the variable log2TrafoWidth set equal to log2TrafoSizeWidth1, the variable log2TrafoHeight set equal to log2TrafoHeight1 and the variable trafoDepth1 set equal to trafoDepth + 1 as inputs and the outputs are the modified versions of two arrays, horEdgeCbFlags and verEdgeCbFlags.
4. The deriviation process of transform unit boundary as specified in this subclause is invoked with the chroma location ( xB2, yB2 ), the variable log2TrafoWidth set equal to log2TrafoSizeWidth1, the variable log2TrafoHeight set equal to log2TrafoHeight1 and the variable trafoDepth1 set equal to trafoDepth + 1 as inputs and the outputs are the modified versions of two arrays, horEdgeCbFlags and verEdgeCbFlags.
5. The deriviation process of transform unit boundary as specified in this subclause is invoked with the chroma location ( xB3, yB3 ), the variable log2TrafoWidth1 set equal to log2TrafoSizeWidth1, the variable log2TrafoHeight set equal to log2TrafoHeight1 and the variable trafoDepth1 set equal to trafoDepth + 1 as inputs and the outputs are the modified versions of two arrays, horEdgeCbFlags and verEdgeCbFlags.

– Otherwise (split\_transform\_flag[1][ xB ][ yB ][ trafoDepth ] is equal to 0), the following applies:

* If yB is equal to zero, horEdgeCbFlags[ xB + k ][ yB ] is set equal to FilterTopCuEdgeFlag, otherwise horEdgeCbFlags[ xB + k ][ yB ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2TrafoWidth ) – 1.
* If xB is equal to zero, verEdgeCbFlags[ xB ][ yB + k ] is set equal to FilterLeftCuEdgeFlag, otherwise verEdgeCbFlags[ xB ][ yB + k ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2TrafoHeight ) – 1.

Depending on split\_transform\_flag[2][ xB ][ yB ][ trafoDepth ], the following applies:

– If split\_transform\_flag[2][ xB ][ yB ][ trafoDepth ] is equal to 1, the following ordered steps apply:

1. The variables xB1, yB1, xB2, yB2, xB3 and yB3 are derived as follows.
   * If InterTUSplitDirection is equal to 2, the following applies.
   * The variable xB1 is set equal to xB + ( ( 1 << log2TrafoWidth ) >> 1 ).
   * The variable yB1 is set equal to yB.
   * The variable xB2 is set equal to xB.
   * The variable yB1 is set equal to yB + ( ( 1 << log2TrafoHeight ) >> 1 ).
   * The variable xB3 is set equal to xB1.
   * The variable yB3 is set equal to yB2.
   * The variable log2TrafoWidth1 is set equal to log2TrafoWidth − 1.
   * The variable log2TrafoHeight1 is set equal to log2TrafoHeight − 1.
   * Otherwise (InterTUSplitDirection is equal to 0 or 1), the following applies.
   * The variable xB1 is set equal to xB + ((1 << (log2TrafoWidth)) >> 2) \* InterTUSplitDirection.
   * The variable yB1 is set equal to yB + ((1 << (log2TrafoHeight)) >> 2) \* (1 − InterTUSplitDirection).
   * The variable xB2 is set equal to xB1 + ((1 << (log2TrafoWidth)) >> 2) \* InterTUSplitDirection.
   * The variable yB2 is set equal to yB1 + ((1 << (log2TrafoHeight)) >> 2) \* (1 − InterTUSplitDirection).
   * The variable xB3 is set equal to xB2 + ((1 << (log2TrafoWidth)) >> 2) \* InterTUSplitDirection.
   * The variable yB3 is set equal to yB2 + ((1 << (log2TrafoHeight)) >> 2) \* (1 − InterTUSplitDirection).
   * The variable log2TrafoWidth1 is set equal to (log2TrafoWidth − 2) \* InterTUSplitDirection.
   * The variable log2TrafoHeight1 is set equal to (log2TrafoHeight − 2) \* (1 − InterTUSplitDirection).
2. The deriviation process of transform unit boundary as specified in this subclause is invoked with the chroma location ( xB, yB ), the variable log2TrafoWidth set equal to log2TrafoWidth1, the variable log2TrafoHeight set equal to log2TrafoHeight1 and the variable trafoDepth1 set equal to trafoDepth + 1 as inputs and the outputs are the modified versions of two arrays, horEdgeCrFlags and verEdgeCrFlags.
3. The deriviation process of transform unit boundary as specified in this subclause is invoked with the chroma location ( xB1, yB1 ), the variable log2TrafoWidth set equal to log2TrafoSizeWidth1, the variable log2TrafoHeight set equal to log2TrafoHeight1 and the variable trafoDepth1 set equal to trafoDepth + 1 as inputs and the outputs are the modified versions of two arrays, horEdgeCrFlags and verEdgeCrFlags.
4. The deriviation process of transform unit boundary as specified in this subclause is invoked with the chroma location ( xB2, yB2 ), the variable log2TrafoWidth set equal to log2TrafoSizeWidth1, the variable log2TrafoHeight set equal to log2TrafoHeight1 and the variable trafoDepth1 set equal to trafoDepth + 1 as inputs and the outputs are the modified versions of two arrays, horEdgeCrFlags and verEdgeCrFlags.
5. The deriviation process of transform unit boundary as specified in this subclause is invoked with the chroma location ( xB3, yB3 ), the variable log2TrafoWidth1 set equal to log2TrafoSizeWidth1, the variable log2TrafoHeight set equal to log2TrafoHeight1 and the variable trafoDepth1 set equal to trafoDepth + 1 as inputs and the outputs are the modified versions of two arrays, horEdgeCrFlags and verEdgeCrFlags.

– Otherwise (split\_transform\_flag[2][ xB ][ yB ][ trafoDepth ] is equal to 0), the following applies:

* If yB is equal to zero, horEdgeCrFlags[ xB + k ][ yB ] is set equal to FilterTopCuEdgeFlag, otherwise horEdgeCrFlags[ xB + k ][ yB ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2TrafoWidth ) – 1.
* If xB is equal to zero, verEdgeCrFlags[ xB ][ yB + k ] is set equal to FilterLeftCuEdgeFlag, otherwise verEdgeCrFlags[ xB ][ yB + k ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2TrafoHeight ) – 1.

#### Derivation process of chroma boundary filtering strength

Inputs of this process are:

– a chroma location ( xC, yC ) specifying the top-left chroma sample of the current coding unit relative to the top-left chroma sample of the current picture,

– a variable log2CUSize specifying the size of the current chroma coding unit,

– two-dimensional arrays of size (nS)x(nS), horEdgeCbFlags, verEdgeCbFlags, horEdgeCrFlags and verEdgeCrFlags.

Output of this process is two arraies of size (2)x(nS)x(nS), bSCb and bSCr specifying the boundary filtering strength for Cb and Cr respectively.

Let ( xEk, yEj ) with k = 0..nE-1 and j = 0..nE-1 specify a set of edge sample locations where nE is set equal to ( ( 1 << log2CUSize ) >> 2 ), xE0 = 0, yE0 = 0, xEk+1 = xEk + 4 and yEj+1 = yEj + 4.

For ( xEk, yEj ) with k = 0..nE-1 and j = 0..nE-1, the following applies.

* If horEdgeCbFlags[ xEk ][ yEj ] is equal to 1,
* Set sample p0 = recPicture[ xC + xEk ][ yC + yEj – 1 ] and q0 = recPicture[ xC + xEk ][ yC + yEj ].
* The variable filterDir is set equal to 1.
* Otherwise, if verEdgeCbFlags[ xEk ][ yEj ] is equal to 1,
* Set sample p0 = recPicture[ xC + xEk – 1 ][ yC + yEj ] and q0 = recPicture[ xC + xEk ][ yC + yEj ].
* The variable filterDir is set equal to 0.
* If horEdgeCbFlags[ xEk ][ yEj ] or horEdgeCbFlags[ xEk ][ yEj ] is equal to 1, depending on the value of filterDir, the variable bSCb[ filterDir ][ xEk ][ yEj ] is derived as follows.
* if the following condition is true, the variable bSCb[ filterDir ][ xEk ][ yEj ] is set equal to 3.
* The sample p0 or q0 is in a coding unit coded with intra prediction mode
* Otherwise, the variable bSCb[ filterDir ][ xEk ][ yEj ] is set equal to 0.
* If horEdgeCrFlags[ xEk ][ yEj ] is equal to 1,
* Set sample p0 = recPicture[ xC + xEk ][ yC + yEj – 1 ] and q0 = recPicture[ xC + xEk ][ yC + yEj ].
* The variable filterDir is set equal to 1.
* Otherwise, if verEdgeCrFlags[ xEk ][ yEj ] is equal to 1,
* Set sample p0 = recPicture[ xC + xEk – 1 ][ yC + yEj ] and q0 = recPicture[ xC + xEk ][ yC + yEj ].
* The variable filterDir is set equal to 0.
* If horEdgeCrFlags[ xEk ][ yEj ] or horEdgeCrFlags[ xEk ][ yEj ] is equal to 1, depending on the value of filterDir, the variable bSCr[ filterDir ][ xEk ][ yEj ] is derived as follows.
* if the following condition is true, the variable bSCr[ filterDir ][ xEk ][ yEj ] is set equal to 3.
* The sample p0 or q0 is in a coding unit coded with intra prediction mode
* Otherwise, the variable bSCr[ filterDir ][ xEk ][ yEj ] is set equal to 0.

#### Filtering process for coding unit

Inputs of this process are:

– a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,

– a variable log2CUSize specifying the coding unit size,

– Three arraies bS, bSCb, bSCr specifying the boundary filtering strength for luma, Cb, and Cr respectively.

Output of this process is:

– modified reconstruction of the picture.

The filtering process for luma edges in the current coding unit consists of the following ordered steps:

1. The variable nD is set equal to 1 << ( log2CUSize – 3 ).
2. All elements of the three-dimensional array of size (2)x(nD)x(nD), dEdge are initialized to zero.
3. All elements of the three-dimensional array of size (2)x(nD)x(1<<log2CUSize), dSample are initialized to zero.
4. All elements of the three-dimensional array of size (2)x(nD)x(nD), bStrength ae initialized to zero.
5. For xDk set equal to xC+( k << 3 ), k=0..nD – 1, the following applies:

* For yDm set equal to yC+( m << 3 ), m=0..nD – 1, the following ordered steps apply:
  + 1. Boundary filtering strength bSVer is derived as follows:

bSVer = Max( bS[ 0 ][ xDk ][ yDm + i ] ) for i = 0..7 (8‑428)

* + 1. bStrength[1][k][m] is set equal to bSVer.
    2. Decision process for luma block edge in subclause 8.6.1.5.1 is invoked with the luma location of the coding unit ( xC, yC ), the luma location of the block ( xDk, yDm ), a variable verticalEdgeFlag set equal to 1, and the boundary filtering strength bSVer as inputs and the decision dEdge[1][k][m] and an array dS of size (8) as outputs.
    3. dSample[1][k][(m<<3)+i] is set equal to dS[i] for i=0..7.
    4. Boundary filtering strength bSHor is derived as follows:

bSHor = Max( bS[ 1 ][ xDk + i ][ yDm ] ) for i = 0..7 (8‑428)

* + 1. bStrength[0][k][m] is set equal to bSHor.
    2. Decision process for luma block edge in subclause 8.6.1.5.1 is invoked with the luma location of the coding unit ( xC, yC ), the luma location of the block ( xDk, yDm ), a variable verticalEdgeFlag set equal to 0, the boundary filtering strength bSHor as inputs, the decision dEdge[0][k][m] and an array dS of size (8) as outputs.
    3. dSample[0][m][(k<<3)+i] is set equal to dS[i] for i=0..7.

1. For xDk set equal to xC+( k << 3 ), k=0..nD - 1, the following applies:

* For yDm set equal to yB+( m << 3 ), m=0..nD – 1, the following ordered steps apply:
  + 1. dS[i] is set equal to dSample[1][k][(m<<3)+i] for i=0..7.
    2. Filtering process for luma block edge in subclause 8.6.1.5.2 is invoked with the luma location of the coding unit ( xC, yC ), the luma location of the block ( xDk, yDm ), a variable verticalEdgeFlag set equal to 1, the boundary filtering strength bStrength[1][k][m], the decision dEdge[1][k][m], and the array of size (8), dS as inputs and the modified luma picture buffer as outputs.

1. For yDm set equal to yC+( m << 3 ), m=0..nD - 1, the following applies:

* For xDk set equal to xC+( k << 3 ), k=0..nD – 1, the following ordered steps apply:
  + 1. If xDk is equal to 0, the parameter xPOS is set equal to 1. If xDk is equal to xB+( ( nD – 1) << 3 ) xPOS is set equal to 2. Otherwise xPOS is set to 0.
    2. dS[i] is set equal to dSample[0][m][ (k << 3) + i ] for i = 0..7.
    3. Filtering process for luma block edge in subclause 8.6.1.4.2 is invoked with the luma location of the coding unit ( xC, yC ), the luma location of the block ( xDk, yDm ), a variable verticalEdgeFlag set equal to 0, the boundary filtering strength bStrength[0][k][m], the decision dEdge[0][k][m], and the array of size (8), dS, xPOS, dSL[m][], dEL[m], bSL[m], and tCL[m], as inputs and the modified luma picture buffer as output.
* The elements of the two dimensional array of size (3)x(nD), dSL are set as follows. dSL[m][0], dSL[m][1], and dSL[m][2], are set equal to dS[5], dS[6] and dS[7].
* The elements of the array of size (nD), dEL are set as follows. dEL[m] is set equal to dEdge[0][k][m].
* The elements of the array of size (nD), bSL are set as follows. bSL[m] is set equal to bStrength[0][k][m].
* The elements of the array of size (nD), tC are set as follows. tCL[m] is set equal to tc.

The filtering process for chroma edges in the current coding unit consists of the following ordered steps:

1. The variable nD is set equal to 1 << ( Max( log2CUSize, 4 ) – 4 ).
2. For xDk set equal to ( xC / 2 )+( k << 3 ), k=0..nD – 1, the following applies:

* For yDm set equal to ( yC / 2)+( m << 2 ), m=0..nD\*2 – 1, the following ordered steps apply:

1. Boundary filtering strength bSVer is derived as follows:

bSCbVer = bSCb[ 0 ][ xDk\*2 ][ yDm\*2 ] (8‑430)

bSCrVer = bSCr[ 0 ][ xDk\*2 ][ yDm\*2 ] (8‑430)

1. Filtering process for chroma block edge in subclause 8.6.1.5.3 is invoked with the chroma location of the coding unit ( xC/2, yC/2 ), the chroma location of the block ( xDk, yDm ), a variable verticalEdgeFlag set equal to 1, a chroma component index cIdx set equal to 1, and the boundary filtering strength bSCbVer as inputs and the modified chroma picture buffer as output.
2. Filtering process for chroma block edge in subclause 8.6.1.5.3 is invoked with the chroma location of the coding unit ( xC/2, yC/2 ), the chroma location of the block ( xDk, yDm ), a variable verticalEdgeFlag set equal to 1, a chroma component index cIdx set equal to 2, and the boundary filtering strength bSCrVer as inputs and the modified chroma picture buffer as output.
3. For yDm set equal to ( yC / 2 )+( m << 3 ), m=0..nD – 1, the following applies:

* For xDk set equal to ( xC / 2 )+( k << 2 ), k=0..nD\*2 – 1, the following ordered steps apply:

1. If xDk is equal to 0, the parameter xPOS is set equal to 1. If xDk is equal to xB+( ( nD\*2 – 1) << 2 ) xPOS is set equal to 2. Otherwise xPOS is set to 0.
2. Boundary filtering strength bSHor is derived as follows:

bSCbHor = bSCb[ 1 ][ xDk\*2 ][ yDm\*2 ] (8‑431)

bSCrHor = bSCr[ 1 ][ xDk\*2 ][ yDm\*2 ] (8‑431)

1. Filtering process for chroma block edge in subclause 8.6.1.4.3 is invoked with the chroma location of the coding unit ( xC/2, yC/2 ), the chroma location of the block ( xDk, yDm ), a variable verticalEdgeFlag set equal to 0, a chroma component index cIdx set equal to 1 and the boundary filtering strength bSCbHor, xPOS, bSL[m] and tCL[m] as inputs and the modified chroma picture buffer as output.
2. Filtering process for chroma block edge in subclause 8.6.1.4.3 is invoked with the chroma location of the coding unit ( xC/2, yC/2 ), the chroma location of the block ( xDk, yDm ), a variable verticalEdgeFlag set equal to 0, a chroma component index cIdx set equal to 2 and the boundary filtering strength bSCrHor, xPOS, bSL[m] and tCL[m] as inputs and the modified chroma picture buffer as output.