|  |  |
| --- | --- |
| **Joint Collaborative Team on Video Coding (JCT-VC)**  **of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11**  39th Meeting: by teleconference, 18–24 April 2020 | Document: JCTVC-AM0026-v2 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **Alternative film grain characteristics SEI message** | | |
| *Status:* | Input document to JCT-VC | | |
| *Purpose:* | Proposal | | |
| *Author(s) or Contact(s):* | Andrey Norkin | Tel: Email: | [anorkin@netflix.com](mailto:anorkin@netflix.com) |
| *Source:* | Netflix | | |

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

# Abstract

This document is similar to the contribution JVET-R0384 submitted to JVET. It proposes an alternative film grain characteristics SEI message. The syntax for the proposed SEI message is compatible with that of the mandatory film grain post-processing algorithm in the AV1 video specification. Adopting the proposed SEI in HEVC (and possibly H.264/AVC) would enable re-use of the post-processing modules that may already be on the device, thus improving the compression efficiency on movie and TV content with little if any additional costs for the manufacturers. Applications such as transcoding between different codecs could also be better supported. The document proposes the alternative film grain characteristics SEI message syntax and semantics (including the film grain synthesis process) and explains the film grain synthesis algorithm. Version 2 of the document adds implementation of the algorithm in HM and example of decoded pictures with added film grain.

# Introduction

At the Brussels JVET meeting, several documents proposed to adopt the film grain synthesis algorithm that is used in the AV1 video specification. In particular, JVET-Q0424 [3] proposed mandatory film grain synthesis support in HEVC with signalling the film grain parameters in the APS and including film grain synthesis algorithm in the normative scope of VC. In JCTVC-AL0022 [6], the film grain synthesis from AV1 was proposed to be included in HEVC as an SEI message, along with including it in VVC and H.264/AVC standards as well. Finally, JVET-Q0533 argued that the exact set of the parameters from the AV1 film grain synthesis algorithm should be used to enable re-use of the post-processing hardware modules between two codecs.

The SEI message proposed in this document is compatible with the film grain synthesis SEI in AV1. Re-using AV1 film grain post-processing could improve the coding efficiency on the movie and TV content without additional burden for many implementations since HEVC and AV1 implementations could then re-use the same post-processing block. This document explains the AV1 film grain synthesis algorithm and proposes the syntax and semantics necessary to invoke the AV1 film grain synthesis algorithm.

As mentioned earlier, AV1 [1] specifies a film grain synthesis algorithm, which is applied as post-processing before outputting a reconstructed picture. This film grain synthesis algorithm is mandatory for implementation in AV1 decoding systems. The film grain is usually implemented as part of the post-processing pipeline, and implementations of the AV1 film grain synthesis algorithm should therefore be readily available at many manufacturers of the video decoding systems. Re-using the AV1 film grain synthesis algorithm in HEVC would therefore not incur significant implementation costs.

This contribution proposes for the HEVC video codec to use the same parameters and the reference film grain synthesis algorithm as the AV1 video codec to enable re-use of this post-processing block among the codecs implementations. The signaling and grain synthesis algorithm have been implemented in VTM8.0 (as part of JVET-R0384 [7]), implementation in JM will follow.

# Motivation for film grain synthesis

The film grain is present in much of the movie and TV content. Often, the film grain is part of the creative intent and as such needs to be preserved when encoding the video.

Since film grain is a realization of a random process (with some degree of correlation in the spatial domain) it is difficult to compress it efficiently. Randomness makes prediction difficult, motion estimation less precise, and the prediction residual noisy. It often requires prohibitively high bitrates to reconstruct film grain with sufficient quality. Moreover, some encoding tools, such as a de-ringing filter may suppress the film grain.

The film grain SEI messages exist in H.264 and HEVC. These SEI messages, however, being optional in H.264/AVC and HEVC, are not supported by most decoders. The encoders cannot rely on support of these SEI messages to provide more efficient compression since if the film grain synthesis algorithm is not implemented by the decoding system, the reconstructed video may have sub-par quality.

# Film grain modeling/synthesis framework

The framework for film grain estimation and synthesis is shown in the figure below.

Macintosh HD:Users:anorkin:work:StandardizationMeetings:AOM:f2f_August_2017:GrainSynthesis:Presentation:Flowchart.emf

**Figure 1**. Film grain estimation and synthesis.

As one could see from Figure 1, film grain is removed from the video before encoding, its parameters are estimated based on the noisy and de-noised versions of the video sequence, and sent along with the compressed video bitstream. After decoding, film grain is synthesized and added to the reconstructed video frames.

The details of the algorithm are as follows. Synthesized noise is added separately to Y, Cb, and Cr color components. The noise is estimated in smooth areas of the picture. Then, the noise standard deviation is modeled using piece-wise linear function for each of Y, Cb, and Cr components.

In luma, grain strength is modulated with a function of luminance as shown below:

*Y’ = Y + f (Y) \* GrainL,* (1)

where Y’ is the resulting signal after grain application, Y is the reconstructed signal (without the film grain), and *GrainL* is the grain sample. *f(Y)* is the piece-wise linear function that scales with the noise intensity depending on the value of the luma component.

Chroma modeling for noise is done as follows. For chroma component (e.g. Cb), the noise is modulated using the following formula:

*Cb’ = Cb + f( z ) \* GrainCb,* (2)

z = Cb \* mCb + Y \* mY,Cb + OffsetCb,  (3)

Parameters mCb , mY,Cb, and OffsetCb are used to model the chroma grain dependency on luma values.

Cr is calculated similarly. Here, the value of *Y* is the average value corresponding to values of Y at co-located positions as the chroma sample.

# Modeling of film grain pattern

The film grain pattern is modeled with an auto-regression process. This auto-regression model is used to synthesize the template with film grain at the decoder. Let *G*(*x*, *y*) be a zero-mean film grain sample at the current position. For lag *L* = 2, the grain sample *G* is calculated as follows (see Fig. 3):

*G*(*x*, *y*) = *a*0 *G*(*x* − 2, *y* − 2) + *a*1 *G*(*x* − 1, *y* − 2) + *a*2 *G*(*x*, *y* − 2) +… + *z,* (4)

where *a*0… *an* are AR-coefficients, *G*(*x + k*, *y + m*) are previous film grain sample values in the causal neighborhood, and *z* is the unit-variance Gaussian noise. The Gaussian variable *z* can be obtained from a predefined set stored at the decoder side. In the case of chroma components, there is one more coefficient to capture correlation with luma grain.

In Fig. 2, the value of *lag* (ar\_coeff\_lag) is equal to 2. The value of lag is signaled in the bitstream and taking values from 0 to 3.

Macintosh HD:Users:anorkin:work:StandardizationMeetings:AOM:f2f_August_2017:GrainSynthesis:Presentation:NewNeighborhood.emf

**Figure 2.** Sample of film grain (green square with cross) with the neighborhood used for noise modeling.

# Film grain reconstruction

Using AR coefficients and a Gaussian sequence, AR process is applied to a 64x64 template, called the film grain template. With a pseudo-random generator, the decoder generates offsets for a 32 x 32 block (+ 2 additional lines and rows when optional overlap is used) inside the 64x64 template. The 32x32 block of the film grain is scaled with intensities of reconstructed samples, added to the reconstructed component (luma or chroma), and the clipping is applied. The block sizes for chroma components are twice smaller for 4:2:0.

The pseudo-random generator is a shift-back linear-feedback shift register (LFSR) based on XOR of the length of 16 bits. The XOR-ed values are at positions 16, 14, 13 and 11, which corresponds to the feedback polynomial of *x*16 + *x*15 + *x*13 + *x*4 + 1. The offset is generated using 8 least significant bits on the register, 4 bits for y offset and 4 bits for x offset (generating offsets from 0 to 15, with luma offsets being in multiples of 2 of these numbers).

The pseudo-random generator is initialized in the beginning of each row of 32x32 blocks to allow parallel row processing. The generator is initialized based on a random\_seed element.

Film grain application process can use optional overlap between the noise blocks, which helps to reduce possible block artifacts in case of larger grain sizes. In this case, luma block sizes are 34 x 34 with the two last rows and two last columns overlapping with the blocks to the left and to the right. The size of chroma blocks is 16x16 when no overlap is used and 17 x 17 with 1 sample overlap when 4:2:0 subsampling is used.

The scaling for a color component can use a look-up table (LUT) with the 256 8-bit values for 8-bit video and 1024 8-bit values for the 10-bit video. The table is filled based on the points of the piece-wise linear scaling function in the beginning of noise signaling for each frame.

Then the sample-wise operations for adding noise to the samples of 32 x 32 block are applied as follows:

Y´(*x*, *y*) = clip3( Y(*x*, *y*) + ((*GL*(*x* + *sx*, y + *sy*) \* *f* (Y) + 2shift−1 ) >> shift ), a, b), (5)

where a and b define the range, x and y are coordinates inside the block, *sx* and *sy* are shift values inside the block, and parameter shift controls scaling of the film grain.

## Overlap between grain blocks

The operation used for overlapped samples over horizontal block boundaries in luma is:

*Gcur*(*x*, 0) = (27 \* *Gup*(*x*, 32) + 17 \* *Gcur*(*x*, 0) + 16) >> 5, (6)

*Gcur*(*x*, 1) = (17 \* *Gup*(*x*, 33) + 27 \* *Gcur*(*x*, 1) + 16) >> 5, (7)

where *Gcur*(*x*, 0) are samples of row 0 of the current block and *Gup*(*x*, 32) denotes samples of row 32 of the upper block.

The overlap operation between chroma blocks is applied as follows:

*Gcur*(*x*, 0) = (23\* *Gtop*(*x*, 16) + 22 \* *Gcur*(*x*, 0) + 16) >> 5 (8)

More details on AV1 film grain synthesis algorithm can be found in [2].

# Alternative film grain characteristics SEI message syntax

The alternative film grain characteristics SEI message uses the following parameters, that can be used to synthesize film grain, as described in the semantics section of the proposal. The syntax for the proposed SEI message is provided below.

|  |  |
| --- | --- |
| alternative\_film\_grain\_characteristics\_SEI( ) { | **Descriptor** |
| **apply\_film\_grain\_flag** | u(1) |
| if (apply\_film\_grain\_flag) { |  |
| **film\_grain\_seed** | u(16) |
| **update\_film\_grain\_model\_flag** | u(1) |
| if (update\_film\_grain\_model\_flag) { |  |
| **num\_y\_points** | u(4) |
| for ( i = 0; i < num\_y\_points; i++ ) { |  |
| **point\_y\_value**[ i ] | u(8) |
| **point\_y\_scaling**[ i ] | u(8) |
| } |  |
| **chroma\_scaling\_from\_luma** | u(1) |
| if ( chromaGrainScalingSignaling ) { |  |
| **num\_cb\_points** | u(4) |
| for ( i = 0; i < num\_cb\_points; i++ ) { |  |
| **point\_cb\_value**[ i ] | u(8) |
| **point\_cb\_scaling**[ i ] | u(8) |
| } |  |
| **num\_cr\_points** | u(4) |
| for ( i = 0; i < num\_cr\_points; i++ ) { |  |
| **point\_cr\_value**[ i ] | u(8) |
| **point\_cr\_scaling**[ i ] | u(8) |
| } |  |
| } |  |
| **grain\_scaling\_minus8** | u(2) |
| **ar\_coeff\_lag** | u(2) |
| if ( num\_y\_points ) { |  |
| for ( i = 0; i < numPosLuma; i++ ) |  |
| **ar\_coeffs\_y\_plus128**[ i ] | u(8) |
| } |  |
| if ( chroma\_scaling\_from\_luma || num\_cb\_points ) { |  |
| for ( i = 0; i < numPosChroma; i++ ) |  |
| **ar\_coeffs\_cb\_plus128**[ i ] | u(8) |
| } |  |
| if ( chroma\_scaling\_from\_luma || num\_cr\_points ) { |  |
| for ( i = 0; i < numPosChroma; i++ ) |  |
| **ar\_coeffs\_cr\_plus128**[ i ] | u(8) |
| } |  |
| **ar\_coeff\_shift\_minus6** | u(2) |
| **grain\_scale\_shift** | u(2) |
| if ( num\_cb\_points ) { |  |
| **cb\_mult** | u(8) |
| **cb\_luma\_mult** | u(8) |
| **cb\_offset** | u(9) |
| } |  |
| if ( num\_cr\_points ) { |  |
| **cr\_mult** | u(8) |
| **cr\_luma\_mult** | u(8) |
| **cr\_offset** | u(9) |
| } |  |
| **grain\_blocks\_overlap\_flag** | u(1) |
| **clip\_to\_restricted\_range** | u(1) |
| **film\_grain\_model\_persistence\_flag** | u(1) |
| } |  |
| } |  |
| } |  |

# Alternative film grain characteristics SEI message semantics

This SEI message provides the decoder with an alternative model for parameterized film grain synthesis. The film grain process is applied to output pictures and not to reference pictures.

NOTE 1 – An encoder may use the alternative film grain characteristics SEI message to characterize film grain that was present in the original source video material and was removed by pre-processing filtering techniques.

**apply\_film\_grain\_flag** equal to 1 specifies that the film grain should be added to the reconstructed picture. apply\_film\_grain\_flag equal to 0 specifies that the film grain should not be added. If apply\_film\_grain\_flag is equal to 1, the film grain synthesis process is applied to the resonstructed picture samples as follows.

First, the arrays scalingLut[cIdx] are initialized (one for each component) and the grain templates grainTemplate[cIdx] are generated. It follows by generating the film grain planes grPlanes[cIdx], which are added to the reconstructed output picture recPicture as shown below.

NOTE 1 – Although the specification defines the generation process of grPlanes[cIdx] before adding grain samples to the reconstructed picture samples, the actual implementation does not need to do that. Instead, grPlanes[cIdx] blocks can be generated just in time, before adding them to the reconstructed picture.

First, film grain is added to the chroma components recPictureCb and recPictureCr of the reconstructed picture if indicated by num\_cb\_points, num\_cr\_points and chroma\_scaling\_from\_luma syntax elements.

for ( y = 0; y < picHeight >> subsY; y++ ) {  
 for ( x = 0; x < picWidth >> subsX; x++ ) {  
 lumaX = x << subsX  
 lumaY = y << subsY  
 lumaNextX = min( lumaX + 1, picWidth − 1 )  
 if ( subsX )  
 avLuma = (recPictureL[ lumaY ][ lumaX ] + recPictureL[ lumaY ][ lumaNextX ]) >> 1  
 else  
 avLuma = recPictureL[ lumaY ][ lumaX ]  
 if ( num\_cb\_points > 0 || chroma\_scaling\_from\_luma ) {  
 if ( chroma\_scaling\_from\_luma ) {  
 scInp = avLuma  
 } else {  
 scInp = Clip1(( (avLuma \* ( cb\_luma\_mult − 128 ) + recPictureCb[ y ][ x ] \*  
 ( cb\_mult − 128 ) ) >> 6 ) + ( (cb\_offset − 256 ) << (BitDepth − 8)))  
 }  
 grSamp = (scaleLut(1, grPlanes[ 1 ][ y ][ x ]) \* grSamp + 2scalingShift−1) >>scalingShift  
 recPictureCb[ y ][ x ] = Clip3(maxVal[1], maxVal[1], recPictureCb[ y ][ x ] + grSamp)  
 }  
 if ( num\_cr\_points > 0 || chroma\_scaling\_from\_luma) {  
 if ( chroma\_scaling\_from\_luma ) {  
 scInp = avLuma  
 } else {  
 scInp=Clip1(((avLuma \* (cr\_luma\_mult − 128) +recPictureCr[y][x]\*  
 (cr\_mult−128 ))>>6) + ( (cr\_offset − 256 ) << (BitDepth − 8)))  
 }  
 grSamp = (scaleLut ( 2, scInp) \* grPlanes[ 2 ][ y ][ x ] + 2scalingShift−1) >>scalingShift  
 recPictureCr[ y ][ x ] = Clip3(maxVal[2], maxVal[2], recPictureCr [ y ][ x ]+ grSamp)  
 }  
 }  
}  
Then, if num\_y\_points is equal to 1, the film grain is added to the luma component recPictureL of the reconstructed picture.

for ( y = 0; y < h ; y++ ) {  
 for ( x = 0; x < w ; x++ ) {  
 grSamp = ( scaleLut( 0, recPictureL[y][x]) \* grPlanes[ 0 ][ y ][ x ] + 2scalingShift−1) >> scalingShift  
 if ( num\_y\_points > 0 )   
 recPictureL[ y ][ x ] = Clip3(maxVal[0], maxVal[0], recPictureL[y][x]+ grSamp)  
 }  
}

**film\_grain\_seed** specifies the starting value for the pseudo-random number generator used in film grain synthesis. In particular, film\_grain\_seed contains the value used to initialize a 16-bit randomGenerator variable. randomGenerator is used to generate pseudo random numbers that are used in the film grain synthesis process with the function getRandNum.

getRandNum() has an input value nBits and the output value randResult as follows:

rVal = randRegister  
 newBit = ((rVal >> 0) ^ (rVal >> 1) ^ (rVal >> 3) ^ (rVal >> 12)) & 1  
 rVal = (rVal >> 1) | (newBit << 15)  
 randResult = (rVal >> (16 − nBits)) & ((1 << nBits) − 1)  
 randRegister = rVal

gaussSample is a look-up table that contains 2048 Gaussian distributed samples with zero mean and standard deviation of 512. The sample values are in the range [-2048, 2047].

**update\_film\_grain\_model\_flag** equal to 1 specifies that the alternative film grain SEI message contains a new set of parameters for the film grain synthesis algorithm. update\_film\_grain\_model\_flag equal to 0 specifies that the previously signalled film grain model for the current layer should be used for the picture associated with the SEI message. It is a requirement of the bitstream conformance that if apply\_film\_grain\_flag is equal to 1 and update\_film\_grain\_model\_flag is equal to 0, there should exist a persisting film grain model previously signaled for this layer in the current coded video sequence.

**num\_y\_points** specifies the number of points for the piece-wise linear scaling function of the luma component. It is a requirement of bitstream conformance that num\_y\_points is less than or equal to 14.

**point\_y\_value**[ i ] specifies the luma value for the i-th point of the piecewise linear scaling function for the luma component.

If i is greater than 0, it is a requirement of bitstream conformance that point\_y\_value[ i ] is greater than point\_y\_value[ i − 1], which ensures x coordinates are signaled in the ascending order.

NOTE: – The values of point\_y\_value[ i ]are from 0 to 255. In 10-bit content, these values correspond to luma values divided by 4. In 12-bit content, these values correspond to luma values divided by 16.

**point\_y\_scaling**[ i ] specifies the scaling value for the i-th point of the piecewise linear scaling function for luma component.

For a color plane cIdx, the following initialization procedure is envoked to initialize the scalingLut[cIdx].

If cIdx is equal to 0, numPoints is equal to num\_y\_points, pointValue[ i ] is equal to point\_y\_value [ i ] and pointScaling[ i ] is equal to point\_y\_scaling [ i ].

If cIdx is equal to 1, numPoints is equal to num\_cb\_points, pointValue[ i ] is equal to point\_cb\_value [ i ] and pointScaling[ i ] is equal to point\_cb\_scaling [ i ].

If cIdx is equal to 2, numPoints is equal to num\_cr\_points, pointValue[ i ] is equal to point\_cr\_value [ i ] and pointScaling[ i ] is equal to point\_cr\_scaling [ i ].

The scalingLut[cIdx] are initialized as follows.

if ( numPoints == 0 ) {  
 for ( x = 0; x < 256; x++ )   
 scalingLut[cIdx][ x ] = 0  
} else {  
 for ( x = 0; x < pointValue[ 0 ]; x++ )   
 scalingLut[cIdx][ x ] = pointScaling[ 0 ]  
 for ( i = 0; i < numPoints − 1; i++ ) {  
 dY = pointScaling[ i + 1 ] − pointScaling[ i ]  
 dX = pointValue[ i + 1 ] − pointValue[ i ]  
 stepY = dY \* ( ( 65536 + (dX >> 1) ) / dX )  
 for ( x = 0; x < dX; x++ )  
 scalingLut[cIdx][ pointValue[ i ] + x ] = pointScaling[ i ] + ( ( x \* stepY + 32768 ) >> 16 )  
 }  
 for ( x = pointValue[ numPoints − 1 ]; x < 256; x++ )   
 scalingLut[cIdx][ x ] = pointScaling[numPoints − 1]  
}

To obtain values of the scaling function, the following procedure is invoked with the color plane cIdx and the input value pointVal as inputs. The output is the value of the scaling function pointScal.

scaleLut(cIdx, pointVal) {  
 x = pointVal >> (BitDepth – 8)  
 rem = pointVal % x  
 if ( BitDepth == 8 || x == 255)  
 pointScal = ScalingLut[cIdx][ x ]  
 else  
 pointScal = scalingLut[cIdx][ x ] +   
 (scalingLut[cIdx][ x ] − scalingLut[cIdx][ x + 1 ]) \* rem + 2BitDepth–8-1) >> (BitDepth – 8)  
}

**chroma\_scaling\_from\_luma** specifies that the chroma scaling function is inferred from the luma scaling function.

If any of the following conditions is true, chromaGrainScalingSignaling is set to 0, otherwise chromaGrainScalingSignaling is set to 1:

* chroma\_scaling\_from\_luma is equal to 1
* chroma\_format\_idc is equal to 0
* chroma\_format\_idc is equal to 1 and num\_y\_points is equal to 0
* chroma\_format\_idc is equal to 2 and num\_y\_points is equal to 0

To obtain tables scalingLut[1] and scalingLut[2], when chroma\_scaling\_from\_luma is equal to 1, their values are copied from scalingLut[0].

**num\_cb\_points** specifies the number of points for the piece-wise linear scaling function of the Cb component. It is a requirement of bitstream conformance that num\_cb\_points is less than or equal to 10.

NOTE: – When chroma\_scaling\_from\_luma is equal to 1, chroma scaling also needs to support up to 14 points since it is allowed for num\_y\_points to take values up to 14.

**point\_cb\_value**[ i ] specifies the x coordinate for the i-th point of the piece-wise linear scaling function for Cb component. The values are signaled on the scale of 0..255.

For values of i greater than 0, it is a requirement of bitstream conformance that point\_cb\_value[ i ] is greater than point\_cb\_value[ i − 1].

**point\_cb\_scaling**[ i ] specifies the function value for the i-th point of the piecewise linear scaling function for Cb component.

**num\_cr\_points** specifies the number of points for the piece-wise linear scaling function of the Cr component. It is a requirement of bitstream conformance that num\_cr\_points is less than or equal to 10.

When chroma\_format\_idc is equal to 1 and num\_cb\_points is equal to 0, it is a requirement of bitstream conformance that num\_cr\_points is equal to 0.

When chroma\_format\_idc is equal to 1 and num\_cb\_points is not equal to 0, it is a requirement of bitstream conformance that num\_cr\_points is not equal to 0.

NOTE: – This requirement ensures that for 4:2:0 chroma subsampling, film grain noise will be applied to both chroma components, or to neither. There is no restriction for 4:2:2 or 4:4:4 chroma subsampling.

**point\_cr\_value**[ i ] specifies the x coordinate for the i-th point of the piece-wise linear scaling function for Cr component. The values are signaled on the scale of 0..255.

For values of greater than 0, it is a requirement of bitstream conformance that point\_cr\_value[ i ] is greater than point\_cr\_value[ i − 1].

**point\_cr\_scaling**[ i ] specifies the scaling value for the i-th point of the piecewise linear scaling function for Cr component.

**grain\_scaling\_minus8** plus 8 specifies the shift applied to the values of the grain and determines the range and quantization step of the standard deviation of film grain. The value of grain\_scaling\_minus8 shall be in the range of 0..3, inclusive. scalingShift is set to grain\_scaling\_minus\_8 + 8

**ar\_coeff\_lag** specifies the number of auto-regressive coefficients for luma and chroma. The variables numPosLuma and numPosChroma are derived as follows:

numPosLuma = 2 \* ar\_coeff\_lag \* ( ar\_coeff\_lag + 1 )

numPosChroma = numPosLuma + 1

**ar\_coeffs\_y\_plus128**[ i ] minus 128 specifies the value of the i-th auto-regressive coefficient used for modeling the luma component of the film grain. aRCoeffsY [ i ] = ar\_coeffs\_y\_plus128[ i ] −128

**ar\_coeffs\_cb\_plus128**[ i ] minus 128 specifies the value of the i-th auto-regressive coefficient used for modeling the Cb component of the film grain. aRCoeffsCb [ i ] = ar\_coeffs\_cb\_plus128[ i ] −128

**ar\_coeffs\_cr\_plus128**[ i ] minus 128 specifies the value of the i-th auto-regressive coefficient used for modeling the Cr component of the film grain. aRCoeffsCr [ i ] = ar\_coeffs\_cr\_plus128[ i ] −128

**ar\_coeff\_shift\_minus6** specifies the range of auto-regressive coefficients. aRCoeffShift is set equal to ar\_coeff\_shift\_minus\_6 + 6

NOTE: – Values of 0, 1, 2, and 3 correspond to the auto-regressive coefficients ranges of [−2, 2), [−1, 1), [−0.5, 0.5), and [−0.25, 0.25) respectively.

**grain\_scale\_shift** specifies how much the Gaussian random numbers should be scaled down during the grain synthesis process. scaleShift is set to 12 - BitDepth + grain\_scale\_shift.

The film grain templates are generated as follows. The variable cIdx is the color component index.

If cIdx is equal to 0 (luma component), nCurrSw = 82, nCurrSh = 73, subsY=0, subsX = 0

If cIdx is equal to 1 or cIdx is equal to 2

- if chroma\_format\_idc is equal to 1, nCurrSw = 44, nCurrSh = 38, subsY=1, subsX = 1

- if chroma\_format\_idc is equal to 2, nCurrSw = 44, nCurrSh = 73, subsY=1, subsX = 0

- if chroma\_format\_idc is equal to 2, nCurrSw = 82, nCurrSh = 73, subsY=0, subsX = 0

If cIdx is equal to 1, randRegister is set to film\_grain\_seed ^ 0xb524

If cIdx is equal to 2, randRegister is set to film\_grain\_seed ^ 0x49d8

for ( y = 0; y < nCurrSh; y++ ) {  
 for ( x = 0; x < nCurrSw; x++ ) {  
 if ( num\_y\_points > 0 )   
 grainTemplate[cIdx][y][x]=(gaussSample[getRandNum(11)]+2scaleShift-1)>> scaleShift

The luma grain template is generated by the process below.

for ( y = 3; y < nCurrSh; y++ ) {  
 for ( x = 3; x < nCurrSw − 3; x++ ) {  
 pos = 0, s = 0  
 for ( idxY = −ar\_coeff\_lag; idxY <= 0; idxY ++ )   
 for (idxX = −ar\_coeff\_lag; idxX <= ar\_coeff\_lag && (idxY!=0 || idxX != 0); idxX ++ )   
 s += grainTemplate[cIdx][ [ y + idxY][ x + idxX] \* aRCoeffsY [ pos++ ]  
 grainTemplate [0][y][x] = Clip3(−2BitDepth-1, 2BitDepth-1−1, grainTemplate[0][y][x] + (( s + 2aRCoeffShift-1) >> aRCoeffShift )))  
 }  
}

After generating the luma template, the chroma grain templates are generated by the process below. cIdx is set equal to 1 and 2. aRCoeffsCh [ i ] is equal to aRCoeffsCb [ i ] when cIdx is equal to 1 and to aRCoeffsCr [ i ] when cIdx is equal to 2.

for ( y = 3; y < nCurrSh; y++ ) {  
 for ( x = 3; x < nCurrSw − 3; x++ ) {  
 s = 0, pos = 0  
 for (idxY = −ar\_coeff\_lag; idxY <= 0; idxY ++ ) {  
 for (idxX = −ar\_coeff\_lag; idxX <= ar\_coeff\_lag && (idxY!=0 || idxX != 0); idxX ++ ) {  
 s = s + grainTemplate[cIdx][ y + idxY][ x + idxX] \* aRCoeffsCh [ pos++ ]  
 }  
 }  
 if ( num\_y\_points > 0 ) {  
 lumaAv = 0  
 lumaX = ( (x − 3) << subsX ) + 3  
 lumaY = ( (y − 3) << subsY ) + 3  
 for ( i = 0; i <= subsY; i++ )  
 for ( j = 0; j <= subsX; j++ )  
 lumaAv += grainTemplate [0] [ lumaY + i ][ lumaX + j ]  
 s = s + (( lumaAv + 2 subsX + subsY -1) >> (subsX + subsY) ) \* aRCoeffsCh [ pos++ ]  
 }  
 grainTemplate[cIdx][y][x] = Clip3((−2BitDepth-1, 2BitDepth-1−1, grainTemplate [cIdx][y][x] + (( s + 2aRCoeffShift-1) >> aRCoeffShift )))   
 }  
}

**cb\_mult** specifies a multiplier for the Cb component used in derivation of the input index to the Cb component scaling function.

**cb\_luma\_mult** specifies a multiplier for the average luma component used in derivation of the input index to the Cb component scaling function.

**cb\_offset** specifies an offset used in derivation of the input index to the Cb component scaling function.

**cr\_mult** specifies a multiplier for the Cr component used in derivation of the input index to the Cr component scaling function.

**cr\_luma\_mult** specifies a multiplier for the average luma component used in derivation of the input index to the Cr component scaling function.

**cr\_offset** specifies an offset used in derivation of the input index to the Cr component scaling function.

**grain\_blocks\_overlap\_flag** equal to 1 specifies that the blending between film grain blocks is applied. **g**rain\_blocks\_overlap\_flag equal to 0 specifies that the blending between film grain blocks is not applied.

First, the blending is performed over the vertical grain block boundaries and then over the horizontal grain block boundaries. For each of the color component, if subsX is equal to 0, a two-sample wide blending is applied to the vertical grain block boundaries on the 66 samples height. If subsX is equal to 1, a one-sample blending is applied to the horizontal grain boundaries of the 33 sample height.

The result of the blending is grPlanes[cIdx] array that contains the grain samples before scaling. The blending process is as follows.

rowNum = 0  
for ( y = 0; y < (picHeight + 1) >> 1; y += 16 ) {  
 randRegister = film\_grain\_seed  
 randRegister ^= ((rowNum \* 37 + 178) & 255) << 8  
 randRegister ^= ((rowNum \* 173 + 105) & 255)  
 for ( x = 0; x < (Width + 1) >> 1; x += 16 ) {  
 rand = getRandNum( 8 )  
 offsetX = rand >> 4  
 offsetY = rand & 15  
 for ( cIdx = 0; cIdx < 3; cIdx ++ ) {  
 cSubsX = (cIdx > 0) ? subsX : 0  
 cSubsY = (cIdx > 0) ? subsY : 0  
 planeOffsetX = cSubsX? 6 + offsetX : 9 + offsetX \* 2  
 planeOffsetY = cSubsY? 6 + offsetY : 9 + offsetY \* 2  
 for ( i = 0; i < 34 >> cSubsY; i++ )   
 for ( j = 0; j < 34 >> cSubsX; j++ ) {  
 grSCur[cIdx][ i ][ j ] = grainTemplate [cIdx] [ lumaY + i ][ lumaX + j ]  
 if (grain\_blocks\_overlap\_flag)  
 blendingOperation(grSCur[cIdx])  
 grPlanes[cIdx][y+i][x+j] = grSCur[cIdx][ i ][ j ]  
 }  
 }  
 }  
 rowNum++  
}

If grain\_blocks\_overlap\_flag is equal to 0, a blending operation blendingOperation(grSCur[cIdx]) is applied to the block boundaries as follows.

blendingOperation(grSCur[cIdx]):

If subsX is equal to 0, a two-sample blending operation is applied to the vertical grain block boundaries. If subsX is equal to 1, a one-sample blending operation is applied to the vertical grain block boundaries.

If subsY is equal to 0, a two-sample blending operation is applied to the horizontal grain boundaries. If subsY is equal to 1, a one-sample blending operation is applied to the horizontal grain boundaries.

The blending operation for samples of the first two columns of grain samples in the block is as follows, where grSCur is the array of grain samples corresponding to the current block, and grSLeft is the array of grain samples corresponding to the grain block on the left. yCurr and xCurr are the coordinates of the first (0, 0) sample of the current grain block. i = 0,..,33.

grSCur[cIdx][yCurr+i][0] = (27\*grSLeft[cIdx][yCurr+i][32] + 17\*grSCur[cIdx][yCurr+i][0]+16) >> 5

grSCur[cIdx][yCurr+i][1] = (17\*grSLeft [cIdx][yCurr+i][33] + 27\*grSCur[cIdx][yCurr+i][1]+16) >> 5

The blending operation for the first of rows of a grain block below the horizontal grain block boundary is performed as follows, i = 0,..,33. grSAbove is the array of grain samples corresponding to the grain block above the current block.

grSCur[cIdx][0][xCurr+i] = (27\*grSAbove[cIdx][32][xCurr+i] + 17\*grSCur[cIdx][0][xCurr+i]+16) >> 5

grSCur[cIdx][1][xCurr+i] = (17\*grSAbove[cIdx][33][xCurr+i] + 27\*grSCur[cIdx][1][xCurr+i]+16) >> 5

The one-sample blending operation is applied to the first row of the grain block for horizontal boundaries and the first colums on the block for vertical boundaries. The blending operation is as follows, where grSCur is the array of grain samples corresponding to the current block, and grSPrev is the array of grain samples corresponding to the previous grain block. yCurr and xCurr are the coordinates of the first (0, 0) sample of the current grain block. In the following, i = 0,...16.

grSCur[cIdx][yCurr+i][0] = (23\*grSLeft[cIdx][yCurr+i][16] + 22\*grSCur[cIdx][yCurr+i][0]+16) >> 5

The one-sample blending operation for the horizontal grain block boundary is as follows, i = 0,...16.

grSCur[cIdx][0][yCurr+i] = (23\*grSAbove[cIdx][16][yCurr+i] + 22\*grSCur[cIdx][0][yCurr+i] +16) >> 5

**clip\_to\_restricted\_range** equal to 1 specifies that clipping to the restricted (studio) range is applied to the sample values after adding the film grain. clip\_to\_restricted\_range equal to 0 specifies that clipping to the full range shall be applied to the sample values after adding the film grain.

After adding film grain to the reconstructed picture, a clipping operation is applied to the output samples. The minimum and maximum values minVal[cIdx] and maxVal[cIdx] for the clipping operation are derived as follows. If clip\_to\_restricted\_range is equal to zero, minVal[cIdx] is equal to 0, and maxVal[cIdx] is equal to 2BitDepth – 1.

If clip\_to\_restricted\_range is equal to zero, minVal[cIdx] is equal to 16\*2BitDepth–8 and maxVal[0] is equal to 235\*2BitDepth–8.

- If matrix\_coeffs is equal to 0, maxVal[1] and maxVal[2] are equal to 235\*2BitDepth–8

- otherwise, maxVal[1] and maxVal[2] are equal to 240\*2BitDepth–8

**film\_grain\_model\_persistence\_flag** specifies the persistence of the alternative film grain characteristics SEI message model for the current layer.

film\_grain\_model\_persistence\_flag equal to 0 specifies that the model specified in alternative film grain characteristics SEI message applies to the current decoded picture only.

Let picA be the current picture. film\_grain\_model\_persistence\_flag equal to 1 specifies that the alternative film grain characteristics SEI message model persists for the current layer in output order until any of the following conditions are true:

– A new CLVS of the current layer begins.

– The bitstream ends.

– A picture picB in the current layer in an access unit containing an alternative film grain characteristics SEI message that is applicable to the current layer is output for which PicOrderCnt( picB ) is greater than PicOrderCnt( picA ), where PicOrderCnt( picB ) and PicOrderCnt( picA ) are the PicOrderCntVal values of picB and picA, respectively, immediately after the invocation of the decoding process for picture order count for picB.

# Alternative film grain synthesis implementation

The new film grain synthesis algorithm is described in Section 7, in the semantics section. An example implementation of the proposed alternative film grain characteristics SEI message has been done in HM (it does not include film grain estimation in the encoder). The implementation is on top of HM master, git hash 2b19c875a3 (patch with signalling of the SEI is attached with the contribution).

Some examples of adding film grain noise to HM encoded sequences are shown below.

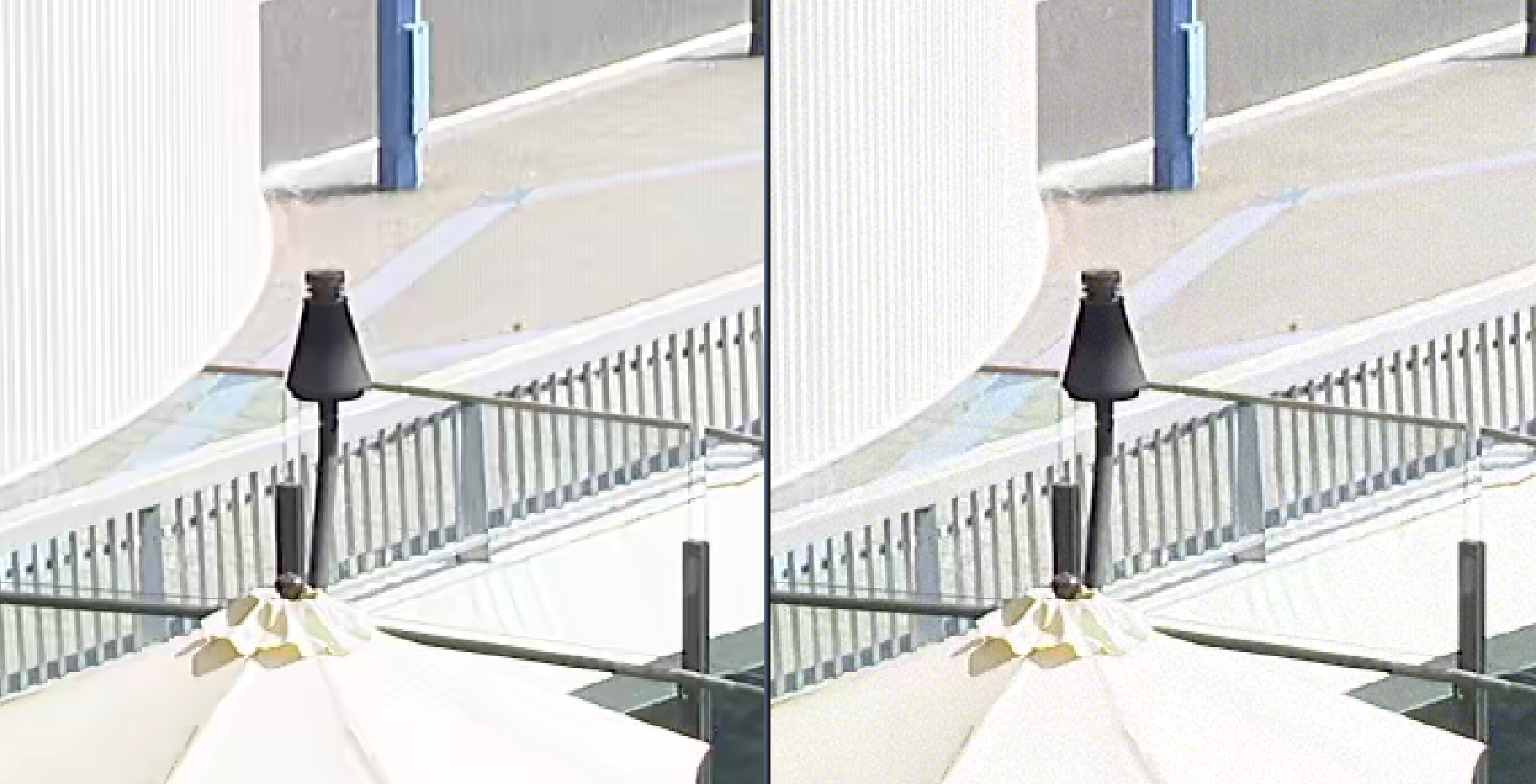


Fig. 3. BQ Terrace RA, QP32, decoded (left), decoded with film grain post-processing (right)



Fig. 4. BQ Terrace RA, QP32, decoded (left), decoded with film grain post-processing (right)



Fig. 5. BQ Terrace RA, QP37, decoded (left), decoded with film grain post-processing (right)

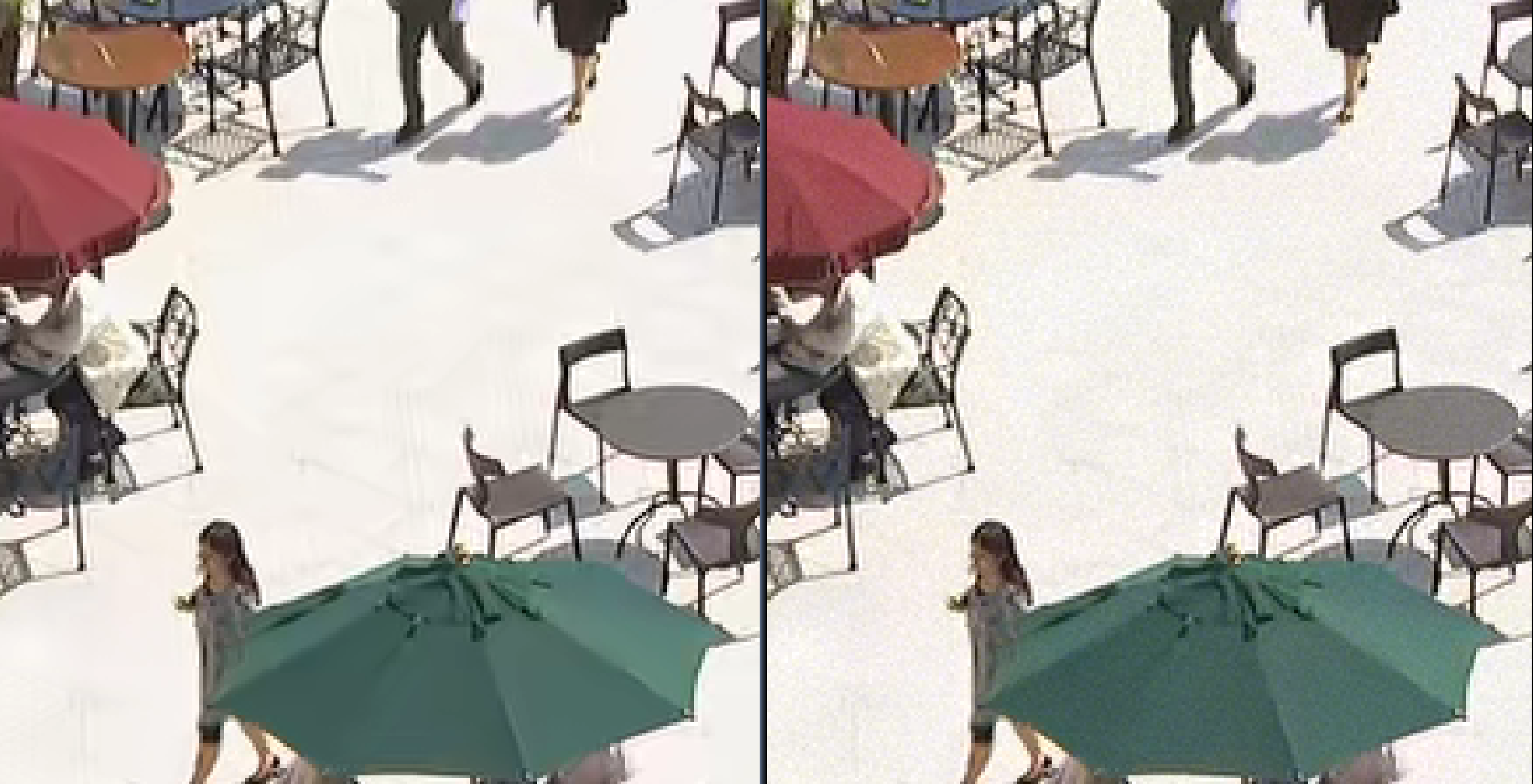


Fig. 6. BQ Square RA, QP37, decoded (left), decoded with film grain post-processing (right)

# Conclusions

This document proposes an Alternative film grain characteristics SEI in HEVC (and JVET-R0384 has been submitted to support the same SEI message in VVC). The proposed SEI is compatible with the AV1 film grain synthesis algorithm. It is argued that if the proposed SEI is adopted in HEVC, products implementing both HEVC and AV1 specifications can re-use the AV1 synthesis algorithm implementations, thus there is no implementation overhead while improving compression efficiency on movie and TV content as well as facilitating transcoding. Support of the alternative film grain characteristics SEI message would provide advantage for movie and TV content on platforms that implement both HEVC and AV1 specifications.

# References

[1] “AV1 Bitstream & Decoding Process Specification”, Version 1.0.0 with Errata 1, Jan. 8, 2019, https://aomediacodec.github.io/av1-spec/av1-spec.pdf

[2] A. Norkin and N. Birkbeck, “Film Grain Synthesis for AV1 Video Codec”, in Proc. IEEE Data Compression Sonference (DCC), Snowbird, Utah, Mar. 2018.

[3] R. Sjöberg, D. Saffar, M. Pettersson, M. Damghanian, “Mandatory film grain”, JVET-Q0424, Brussels, Jan. 2020.

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

[5] A. Norkin, "Film Grain Synthesis Support", JVET-Q0533, Brussels, January 2020.

[6] A. M. Tourapis, K. Rapaka, D. Singer, K. Kolarov, "Film Grain Synthesis Support in AVC and HEVC", JCTVC-AL0022, Brussels, January 2020.

[7] A. Norkin, “Alternative film grain characteristics SEI message”, JVET-R0384, Teleconference, April 2020.

# Patent rights declaration(s)

**Netflix, 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).**