Film Grain Synthesis with Debanding Feature

Andrey Norkin*

*Netflix
121 Albright Way
Los Gatos, CA 95008, USA
anorkin@netflix.com

***Abstract**:* Film grain synthesis is a powerful tool that can significantly reduce the bitrate of a grainy video. It is typically used with noise removal before the compression, which can make banding more pronounced in the compressed video. When the synthesized grain is added, the banding can still be visible, even at mid QPs. This article describes three algorithms that can be used with the AV1/AV2 film grain synthesis to reduce visibility of underlying bands in the re-noised video. These changes to the film grain synthesis algorithm are computationally inexpensive and improve the perceptual video quality when banding is present.

1. Introduction

Banding (or contouring) artifacts are often present in compressed video at a certain bitrate range. They are usually found in flat areas of the video with a slowly changing gradient. When higher frequencies are not kept in the compressed video, a gradient may be quantized to uniform bands of samples that have the same value. Even if the difference between the neighboring bands is small, the resulting bands can be observed during the video playback. These artifacts are present in many video codecs under certain conditions.

In some scenarios, these artifacts can be mitigated by increasing the internal and output codec bit depth, such as setting it equal to 10 bits for 8-bit content [1]. However, even when the internal bit depth of 10 is used, banding can still be visible in some HDR content or when playing the video on displays with the bit depth of 8.

Several approaches to address banding have been proposed in the literature. For example, in [2], a quantization-based approach in the encoder has been proposed, which keeps transform coefficients in banding prone areas. Another group of approaches applies debanding algorithms as postprocessing [3], [4]. In [5], a debanding filter can be applied as either a post- or in-loop filter. Typically, a debanding approach tries to find areas in the picture that can exhibit banding after compression. A detector for banding artifacts is often used, which identifies areas of the reconstructed picture that are likely to have banding. If banding is detected, the algorithm applies dithering to this area that helps to mitigate banding. For example, [5] uses a CAMBI metric [6] to decide whether debanding should be applied to a particular frame, which is signaled in the frame header. A normative approach to reduce contouring artifacts due to intra-prediction was proposed in HEVC [7]. Finally, [8] shows that banding in AVM/AV2 is caused by several coding tools and proposes ways to prevent banding by modifying them.

This paper investigates the scenario when a *film grain synthesis* (FGS) tool is used to decrease the video bitrate while preserving grain, with an example of AVM [9], the reference software for the Alliance of Open Media (AOMedia) AV2 video codec under

development. The paper proposes a solution to mitigate banding with a simple and low-cost modifications to the film grain synthesis algorithm.

In the following, Section 2 briefly describes the film grain synthesis tools in AV1 [10] and AV2 [9] codecs and Section 3 studies banding artifacts in compressed video when the film grain synthesis tool is used. Section 4 proposes modifications to the film grain synthesis tool that can reduce banding. Sections 5 and 6 introduce two variants of the proposed approach, and Section 7 introduces a version of the algorithm with the reduced decoder complexity. Section 8 describes the results and visual quality improvements of the approach. Finally, Section 9 concludes the paper.

2. Film grain synthesis in AV1 and AV2 video codecs

The film grain tool in AV1 [10], AFGS1 [11], and the emerging AV2 codec [9] is applied as follows. In the encoder, the input video is denoised, and the denoised video is compressed. The noise / grain residual is analyzed to estimate the film grain model parameters, which are sent in the bitstream. In particular, the grain pattern is modeled with an auto-regressive (AR) model. In the decoder, a film grain synthesis template consisting of 64×64 luma grain samples is generated based on the AR model, with its coefficients communicated to the decoder. The film grain template contains a grain pattern with zero mean [12].

Then, film grain synthesis is applied in 32×32 blocks in a raster scan order by extracting grain blocks from the grain template based on the pseudo-random offsets and adding the grain blocks to the picture. In the emerging AV2 video codec, the film grain synthesis can use either 32×32 or 16×16 block size, while using the same 64×64 template. The encoder can decide which block size is more appropriate for the current content and select the film grain block size on a frame basis by signaling the corresponding parameter in the film grain synthesis parameter set [9].

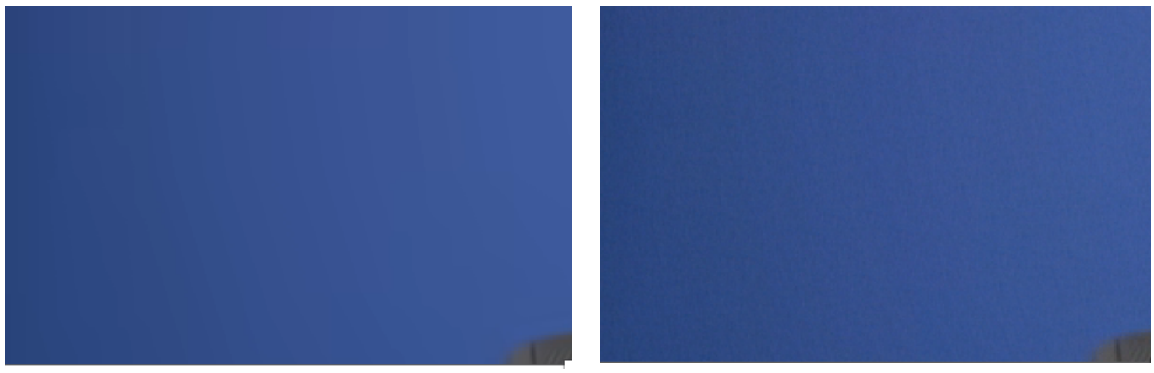
A pseudo-random generator based on a linear-feedback shift register (LFSR) with XOR is used to obtain the template offsets when extracting the 32×32 and 16×16 blocks. The details of the offset generation in AV2 are somewhat different from the AV1 and AFGS1 specifications and cover block sizes of 32×32 and 16×16; however, all three standards use an LFSR register with the same feedback polynomial.

Before the film grain is added to the output video samples, it is scaled according to the sample intensity/values. The scaling is based on a piece-wise linear function signaled in the film grain parameter set that uses the underlying sample values as the input, as shown in eq. (1). More details on the AV1 film grain synthesis can be found in [12].

3. Banding artifacts in video with applied FGS

One of the reasons banding appears in compressed video is that noise that was present in the original video is no longer in the decoded video. When using the film grain synthesis tool, noise and grain are removed from the source video before the encoding. Fig 1 shows a source video sequence “Johnny” from the AVM Common Test Conditions [13]. Fig. 2(a) shows the same video sequence compressed with AVM at QP 160 in the Random Access (RA) configuration. The denoising was applied to the source video as part of the FGS tool. One can clearly see grain/noise in the background area of the source video (Fig. 1), which

Figure 1: Source sequence Johnny (a part of the picture).



(a) before adding grain. (b) after adding grain.

Figure 2: Johnny, RA, encoded at base QP 160 with using FGS. Bands are still noticeable in (b).

is removed in the reconstructed video in Fig. 2 (a). Fig. 2 (b) shows the reconstructed picture after the FGS has been applied. The synthesized texture in the background of Fig. 2 (b) looks more like the source than in Fig. 2 (a), which indicates that the FGS may be a better way to handle the banding problem than a dithering approach. However, the synthesized grain does not completely conceal the banding artifacts in the background area, which are somewhat visible even in the still picture. These banding artifacts are usually more visible during the video playback since the grain is temporally uncorrelated, while the banding artifacts are often somewhat static. Also, when the film grain is applied right after the video reconstruction, e.g. as part of the video decoder, the debanding cannot be successfully applied afterwards since the synthesized grain would interfere with debanding algorithms.

4. Film grain synthesis modification to remove banding

The remaining banding described in Section 3 can be mitigated using the following approach. The reason that the bands are still visible after applying the synthesized grain (see Fig. 2) is that the average value of the added FGS noise is zero. The approach described in this paper adds an extra offset when adding a grain sample to the output video sample.

This offset is derived in a way that: (a) makes the average added grain locally non-zero mean and (b) is derived in a way that compensates the bands in the output video. This is illustrated in the following. A film grain sample is added to the output luma sample in AV1 or AV2 as follows (for 8-bit video):

$$Y'(x, y) = \text{clamp}(Y(x, y) + (f(Y(x,y)) * G(i , j) + 2^{scale-1}) >> scale), 0, 255), \quad (1)$$

where $Y(x, y)$ is the value of a luma sample with coordinates (x, y) before adding grain, $Y'(x, y)$ is the value of the luma sample after adding the grain, $f(Y)$ is the scaling function with Y sample value as input, $G(i , j)$ is the value of the film grain sample in the film grain template at coordinates (i, j), and *scale* is the scale/bit depth parameter to facilitate sub-integer precision. The proposed algorithm generates the luma sample values after applying the film grain as:

$$Y'(x, y) = \text{clamp}(Y(x, y) + (f(Y(x,y)) * G(i , j) + Off_d(x, y) + 2^{scale-1}) >> scale), 0, 255), \quad (2)$$

where $Off_d(x, y)$ is the debanding offset at the picture coordinates (x, y) generated to conceal the banding artifact. The rest of the paper describes different approaches/algorithms of deriving the offset $Off_d(x, y)$.

One of the advantages of the described approach is that the result often looks more like the original video than the result of typical debanding with dithering. Often, the source video has grain or noise, which is why banding is not observed in the source. The author observes that the reconstructed video with added grain (see Fig. 2(b)) often resembles the source video more than the result of debanding that uses dithering, since typical results of the dithering would often look smoother than the grain present in the source.

Note that the FGS with the proposed modification can perform debanding even when the film grain is not present in the source video. The FGS tool in AV1 applies white gaussian noise to the output video when no AR coefficients are signaled in the FGS parameter set. The strength of the noise can be chosen such that the noise is just enough to deband the video but is not well visible otherwise.

The proposed algorithms, described in this paper, are applied to the luma component. However, their extension to the chroma components is straightforward and may be warranted in the cases when there are banding artifacts in chroma.

5. Debanding offset derivation using gradients (Algorithm 1)

One approach to derive $Off_d(x, y)$ is to find horizontal and vertical gradients and derive the offsets as a function of the band width in the corresponding direction. The algorithm operates only on the previously processed samples (Fig. 4). First, previous horizontal and vertical values are stored for each sample which requires a line buffer the size of the picture width and a column buffer the height of one FGS block, i.e. 16 or 32 samples. The vertical and horizontal counters for the number of samples with the values belonging to the same band are initialized and incremented when applying the grain to a sample. The maximum value of the counter is equal to MAX_BAND, 32 in the experiments. If a sample value outside of the current band is encountered, the current counter *cur_count* is assigned to *prev_count*, and *cur_count* is reset to zero (see Fig. 4). When a sample with the absolute difference from the current band value higher than MAX_STEP is encountered, *cur_count*

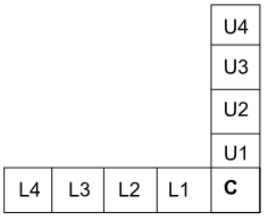


Figure 3: Previous samples, used in computations in Algorithm 1.

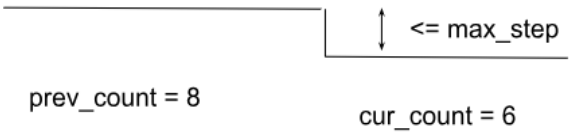


Figure 4: Previous and current band sample counters in Algorithm 1.

and *prev_count* are set to zero since this corresponds to a textured area or an edge encountered in the picture. The horizontal offset is found as

$$g_{hor} = 2^{scale} * (band_{left} - Y)) * \max((prev\_count - cur\_count - 1), 0) / prev\_count, \quad (3)$$

where g_{hor} is the horizontal offset at the current position, $band_{left}$ is the sample value of the band to the left of the current sample, Y is the current sample value, and 2^{scale} is a constant that facilitates fixed point sub-integer arithmetics and is the same as in (2). The vertical and horizontal offsets are averaged to produce the final offset $Off_d(x, y)$. If only one of the offsets g_{hor} and g_{ver} is not equal to zero, its value is used without averaging. To make the algorithm more robust to a single sample switched within the band, a condition may be added that the counter *prev_count* is only assigned the value of *cur_count* when the new *cur_count* value is higher than a SWITCH_DIST threshold, set to 1 in the current implementation.

One feature of Algorithm 1 is that it does not take future sample values into account. This enables application of this algorithm in the raster scan order without a lookahead, along with applying the FGS. However, the algorithm can make mistakes in the estimation of gradients in the case of bands of irregular size. To handle irregular bands, Algorithm 2 described in the next section may be more suitable.

6. Debanding offset derivation with box filter (Algorithm 2)

Another approach to derive $Off_d(x, y)$ is to use a rectangular-shaped box filter (Fig. 5). In this filter, all weights, such as $s1$, $s2$ in Fig. 5. are equal, which allows an efficient implementation with the integral image approach [14]. As known from the literature [14], the integral image is an image, in which each sample is a sum of the sample values to the left and top of the current sample in the original image. After the integral image is computed, the sum of the sample values in the box can be obtained as a $S_{box} = B_1 + B_4 - B_2 - B_3$, where B_i are the integral image values corresponding to the vertices of the box (see [14] for details of the integral image operations). To find offset $Off_d(x, y)$ values, Algorithm 2 operates on the FGS application blocks of 16×16 or 32×32. First, an integral image including the current FGS block and the area outside the block is created, as in Fig 6. Then the sum of the samples S_{box} within the box filter is found. The offset $Off_d(x, y)$ is found as:

$$Off_d(x, y) = \text{clamp}(2^{scale} * S_{box} / (box\_width * box\_height) - 2^{scale} * Y(x, y), -off_{max}, off_{max}) \quad (4)$$

where *box_width* and *box_height* are box filter width and height, respectively. The offset is clipped to the range $(-off_{max}, off_{max})$ so that the sample modifications are limited. In this paper, off_{max} is fixed to $2*2^{scale}$ for 8-bit output video. The offsets are applied to all samples in the current FGS block (see Fig 6).

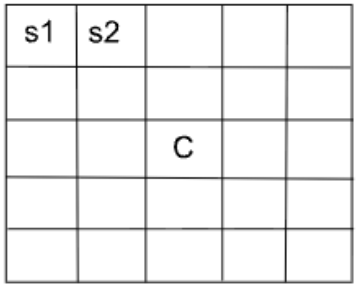


Figure 5: Rectangle / box-shaped filter for Algorithm 2.

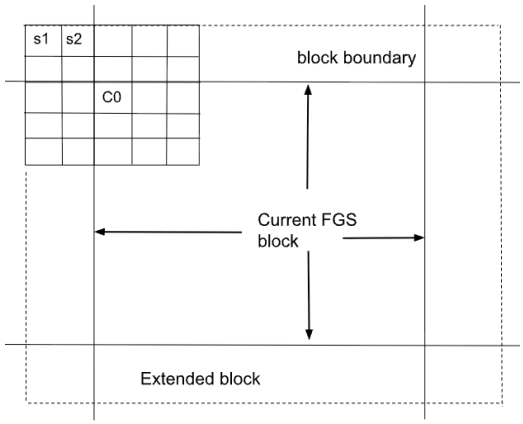


Figure 6: The application of filtering to the current FGS processing block.

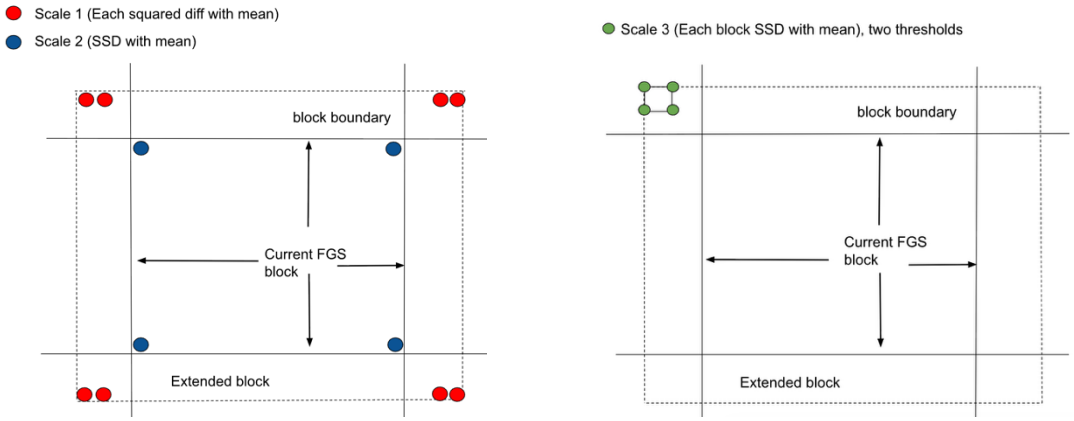
Some blocks, to which the FGS is applied, have details, and applying the box filter may result in strong low-pass filtering even when the offset is clipped. To make sure debanding is only applied where necessary, each 32×32 or 16×16 block is classified with respect to whether the debanding is applied to it, as described in the next sub-section.

6.1. Block classification for the box filter algorithm

The block classification can also use the integral image. From the integral image, the mean value of a block is easily obtained. Then, squared differences with the mean are computed for selected samples to classify the block as smooth or detailed. The debanding is only applied to smooth blocks. In the following, P is number of samples outside the current FGS block to provide support for the box filter, and *FGsize* if the FGS block size. The classification uses three scales (see Fig. 7). First, squared differences of four averages between two neighboring samples in the corners of a ($FGsize$+2P)×($FGsize$+2P) area and the mean of the area are calculated (Scale 1 in Fig. 7(a)), and each squared difference is compared to Threshold_1. Second, the *sum of squared differences* (SSD) for four samples in the corners of the current $FGsize \times FGsize$ block is calculated (Scale 2 in Fig. 7(a)) and compared to Threshold_2. Then, the SSDs of four corner samples in each 4×4 block are calculated (Scale 3 in Fig. 7(b)) and compared to Threshold_3. If the compared value for any block exceeds the threshold, the whole FGS block is classified as detailed. In addition, the block is classified as detailed if more than N comparisons at Scale 3 exceed Threshold_4, which is typically set below Threshold_3. Otherwise, the FGS block is classified as smooth.

6.2. Complexity of Algorithm 2

Block-wise integral image generation requires about 2 additions per sample (an overlap between blocks may need to be accounted for as well). The filtering takes 3 additions, 1 multiplication with a constant and 1 shift to get the offset value, 1 shift + 1 clamping to get the final offset value and 1 addition operation to add the obtained offset in (2), on top of the regular FGS operations. Overall, the worst-case debanding filtering operations per sample (when debanding is applied to all blocks in the picture) is *4 additions, 1 multiplication with a constant, 2 shifts and 1 clamp per sample*.



(a) Classification at Scales 1 and 2. (b) Classification at Scale 3.

Figure 7: Classification of a 16×16 or 32×32 FGS block in Algorithm 2.

Classification is done on a block basis and the estimate for a 16×16 block is:

- Scale 1: 7 adds., 1 shift, 4 subtractions, 4 mults., and 4 comparisons
- Scale 2: 3 adds., 1 shift, 4 subtractions, 4 mults., and 1 comparison
- Each Scale 3 block: 3 adds, 1 shift, 4 subtractions, 4 mults. and 3 comparisons.

Overall, *classification takes 1.8 additions, 0.25 shifts, 1 multiplication, and 0.75 comparisons per sample*. The per-sample operations count is lower than that of most in-loop filters in AV1 and AV2. However, the filter in Algorithm 2 can use 17×17 or a larger window size, which may require keeping 8 lines or more of luma samples in the line buffer. This memory can be reduced to lines at positions $P/2$ and P below the current block and positions $-P/2$ and $-P$ above the current block (see Fig. 8). The samples between those lines are obtained using the linear interpolation in the vertical direction.

7. Line-wise debanding with classification in encoder (Algorithm 3)

The AV1 and AV2 FGS algorithms can be applied to the output picture line-by-line even though the reference FGS algorithm is described in terms of the blocks. This may be useful when applying FGS in the display pipeline. Algorithm 2 requires block-by-block application since the classification is done on the block basis. Algorithm 3, described below is based on Algorithm 2 and enables implementation on a line-by-line basis. In Algorithm 3, the block classification is done in the encoder. In the experiments, the encoder uses the classification described in Sub-section 6.1. Then, a classification map is sent to the decoder using the segmentation feature of AV1/AV2, which assigns coding blocks to one of 8 (in AV1) or 16 (in AV2) segments. The segmentation map is signaled in the bitstream and has certain parameters, such as q_index, associated with each segment. It is proposed to add a segmentation feature SEG_DEBAND_BLOCK that indicates whether to apply debanding to the blocks associated with the segment. This feature takes values 0 (no debanding) and 1 (debanding). The decoder converts the segmentation map to the debanding map, assigning the corresponding values to 16×16 or 32×32 blocks (see Fig 9). Production AV1 and AV2 encoders would often use adaptive QP algorithms with the segmentation feature, and smooth areas prone to banding typically coincide with the lower QP segments determined by adaptive QP algorithms. Then, signaling the debanding map is cheap since it only requires signaling the SEG_DEBAND_BLOCK feature for each segment at the frame level (1 bit per segment at the frame level).

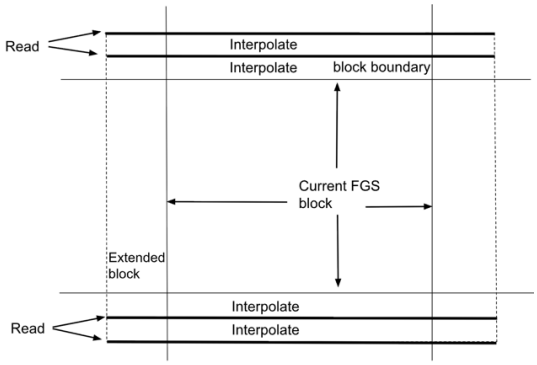


Figure 8: Reduced line buffers for Alg. 2. Only bold lines outside the block are used.

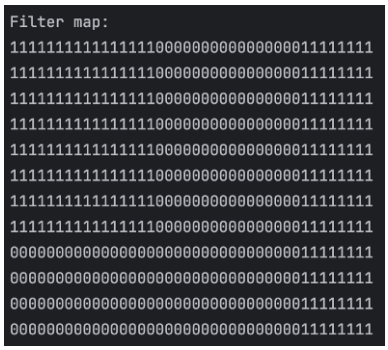


Figure 9: Example of filtering / debanding map.

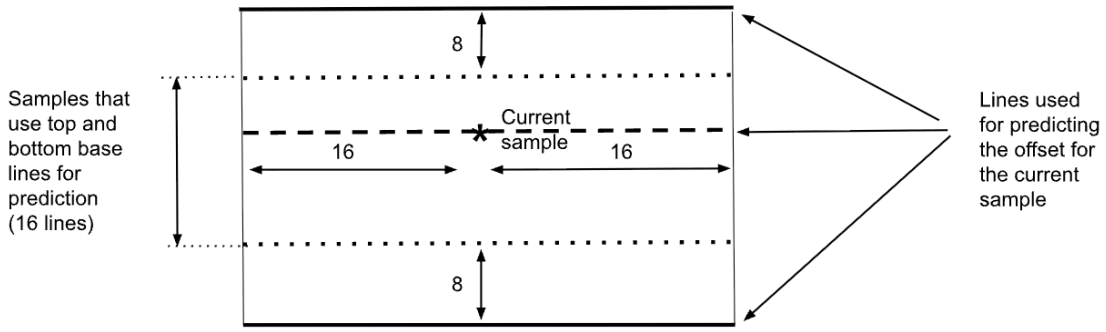


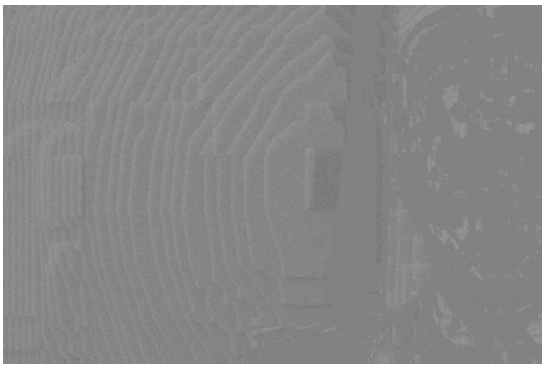
Figure 10: Offset derivation in Algorithm 3. Base lines are solid lines. Current line is dashed. Interpolation is done for samples between two dotted lines, which are horizontal FGS block boundaries.

7.1. Deriving debanding offset in Algorithm 3

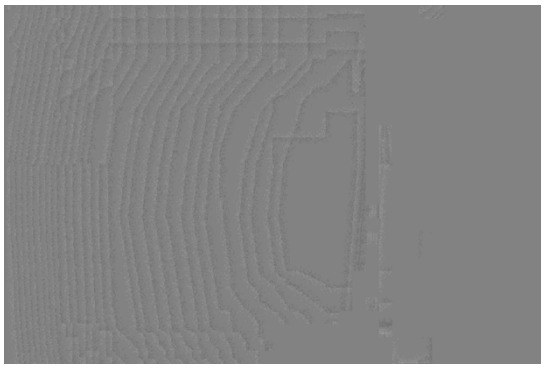
To allow the line-by-line implementation, the debanding offsets are based on linear interpolation between so called base lines (solid lines in Fig. 10) and the current line (dashed). Three vertically aligned 1-D sliding horizontal windows are used with top and bottom base lines and the current line. The windows are centered around the horizontal coordinate x of the current sample (x, y); and the sums of the sample values in these windows are denoted as S_{top}, S_{bot}, and S_{cur}, respectively. Then, the offset value is found as:

$$Off_d(x, y) = \text{clamp}(2^{scale} * ((n - pos_{\text{vert}}) * S_{\text{top}} + pos_{vert} * S_{\text{bot}} + n * S_{\text{cur}}) / \\ / (2 * n * win\_length) - 2^{scale} * Y(x, y), -\text{off}_{\max}, \text{off}_{\max}), \quad (5)$$

where *win_length* is the number of samples in each horizontal 1-D window centered around the current sample, n is the distance between the top and bottom base lines, and pos_{vert} is vertical distance of the current line from the top base line. One can consider (5) a optimization of equation (4) that only uses the current line and two base lines that need to be stored in the memory. Therefore, the line buffer is reduced to 2 base lines stored in the on-chip memory, and the algorithm can be applied line-by-line. Note that the same two base lines are used for deriving offsets for all samples between two dotted lines in Fig. 10, which correspond to the horizontal FGS block boundaries.



(a) Algorithm 1



(b) Algorithm 2

Figure 11: Sequence Johnny, QP185. Scaled difference between anchor and proposals.

(a) Anchor

(b) Algorithm 2

Figure 12: Sequence FountainSky, QP210, 8 bits, FGS applied.

8. Results and visual quality

The algorithm has been implemented on top of AVM anchor v10 with the FGS block size of 16×16. Fig. 11 shows the scaled difference between the anchor and results of Algorithms 1 and 2. The anchor uses the FGS tool, whereas the tests use the FGS with Algorithms 1 and 2. Both algorithms remove the banding artifacts from the background. Algorithm 1 adds offsets at one side of the band since it only uses causal information, while Algorithm 2 adds offsets on both sides of the band boundaries. Also, Algorithm 2 doesn't modify the face area due to the classification. Fig. 12 shows FountainSky sequence coded at base QP 210, with the banding present in the sky area. The banding is removed with Algorithm 2. (Algorithm 1 has difficulties to completely remove this type of banding since the bands are irregular in width and the step size). Fig. 13 shows rather extreme banding / blockingness in sequence Sparks encoded at QP 235. Algorithm 3 reduces banding. In general, it has been observed that the results of Algorithm 3 are similar to those of Algorithm 2. The pictures included in the paper represent content encoded at high QP and bit depth of 8; however, improvements have also been observed at lower QPs and bit depth of 10.

9. Conclusions

Banding artifacts are present in most modern video codecs. These artifacts can also be observed when the FGS algorithm is used. Three algorithms for debanding are proposed, which are based on modifications to the film grain synthesis feature in AV1 and AV2. The

(a) Anchor (b) Algorithm 3

Figure 13: Sequence Sparks, QP235, 8 bits, FGS applied.

algorithms are computationally inexpensive and can increase the output video quality when banding is present. Even though the approaches have been implemented for the AV2 FGS, the author believes that they are applicable to other FGS approaches.

References

[1] D. Hoang, V. L. Pham, H. Egilmez, L. Guo, Y. Zheng, G. Li, A. Tourapis, "Coding 8-bit Video using 10-bit", *AOMedia document* CWG-D088, July 31, 2023.

[2] N. Casali, M. Naccari, M. Mrak, and R. Leonardi, "Adaptive quantisation in HEVC for contouring artefacts removal in UHD content", *in Proc ICIP 2015*, Sept. 2015.

[3] Z. Tu, J. Lin, Y. Wang, B. Adsumilli, and A. C. Bovik, "Adaptive Debanding Filter", *IEEE Signal Processing Letters, vo.27*, pp. 1715-1719, 2020.

[4] S. Bhagavathy; J. Llach; J. Zhai, "Multi-Scale Probabilistic Dithering for Suppressing Banding Artifacts in Digital Images", *in Proc ICIP 2007*, San Antonio, TX, Oct. 2007

[5] J. Sole, M. Afonso, "A debanding algorithm for AV2", *in Proc. DCC 2023*, Snowbird, UT, USA, March 2023.

[6] J. Sole, M. Afonso, L. Krasula, Z. Li, and P. Tandon, "CAMBI, a banding artifact detector", *Tech Blog* https://netflixtechblog.com/cambi-a-banding-artifact-detector-96777ae12fe2

[7] T. K. Tan and Y. Suzuki, "Contouring artefact and solution," *JCT-VC document* JCTVC-K0139, JCT-VC, Shanghai, CN, Oct. 2012.

[8] A. Norkin, "Banding prevention for AVM video codec", *in Proc. IEEE Data Compression Conference (DCC)*, Snowbird, Utah, Mar. 2025.

[9] AOMedia Video Model (AVM): https://gitlab.com/AOMediaCodec/avm/.

[10] "AV1 Bitstream & Decoding Process Specification", https://aomediacodec.github.io/av1-spec/av1-spec.pdf

[11] "AOMedia Film Grain Synthesis 1 (AFGS1) specification", version 1.0.0, Jan. 17, 2024.

[12] A. Norkin and N. Birkbeck, "Film Grain Synthesis for AV1 Video Codec", *in Proc. IEEE Data Compression Conference (DCC)*, Snowbird, Utah, Mar. 2018.

[13] X. Zhao, Z. (R.) Lei, A. Norkin, T. Daede, A. Tourapis, "AOM Common Test Conditions v7.0", *AOMedia Document* CWG-E083o, Apr. 15, 2024.

[14] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features", *in Proc. IEEE CVPR* 2001, Dec. 2001.