Bilateral Filtering for Video Coding

Per Wennersten #1, Jacob Ström #2, Ying Wang *3, Kenneth Andersson #4 Rickard Sjöberg #5 Jack Enhorn #5

Ericsson Research, Sweden
1 per.wennersten@ericsson.com
2 jacob.strom@ericsson.com
4 kenneth.r.andersson@ericsson.com
5 rickard.sjoberg@ericsson.com
6 jack.enhorn@ericsson.com

* *Skype Sweden AB, Sweden*
3 yingwan@microsoft.com

***Abstract*—This paper proposes the use of a bilateral filter as a coding tool for video compression. The filter is applied after transform and reconstruction, and the filtered result is used both for output as well as for spatial and temporal prediction. The implementation is based on a look-up table (LUT), making it fast enough to give a reasonable trade-off between complexity and compression efficiency. By varying the center filter coefficient and avoiding storing zero LUT entries, it is possible to reduce the size of the LUT to 2202 bytes. It is also demonstrated that the filter can be implemented without divisions, which is important for full custom ASIC implementations. The method has been implemented and tested according to the common test conditions in JEM version 5.0.1. For still images, or intra frames, we report a 0.4% bitrate reduction with a complexity increase of 6% in the encoder and 5% in the decoder. For video, we report a 0.5% bitrate reduction with a complexity increase of 3% in the encoder and 0% in the decoder.**

***Index Terms*—video coding compression bilateral filtering**

I. Background

A. Deringing

Quantizing in the transform domain instead of in the spatial domain is a well-known technique for better preserving information in images and video, and has been used in video coding standards since H.261 [1]. However, it is also well-known that this technique may produce ringing artifacts around edges.

Several methods exist to combat these problems. For JPEG images, Nosratinia suggests applying shifted transforms and averaging the result [2]. This works because the true signal tends to be preserved in each shifted version, whereas the artifacts will be different each time. This method produces good results, but is quite slow, especially for video.

For video compression, there are also several methods to combat ringing. WMV 9 [3] includes a deringing post-process filter which classifies samples as edge or non-edge, and filters them based on this classification. C.Y. Tsai et al. [4] propose an adaptive loop filter, which filters frames within the coding loop, but has no explicit de-ringing focus.

IEEE VCIP'17, Dec. 10-13, 2017, St Petersburg, FL, U.S.A.
978-1-5386-0452-5/17/$31.00

Sample adaptive offset is an in-loop filter introduced by C. M. Fu et al. [5] and included in HEVC [6] which addresses ringing. Briefly, it works by classifying samples into categories including local maximum and local minimum, and then signaling offsets to be added to the samples in each category.

B. Joint Exploration Model — JEM

JEM is an experimental codec being developed by JVET, which is based on HEVC. It includes a version of adaptive loop filtering, as well as sample adaptive offset. In this paper, version 5.0.1 has been used as a reference.

C. Bilateral filtering

Bilateral filtering is a filtering technique first proposed by Tomasi and Manduchi [7]. The core concept is a filter kernel where the contribution of each sample depends not only on the spatial distance to the sample being filtered, but also on the difference in intensity between the samples.

A sample located at position (i, j), will be filtered using (among others) its neighboring sample (k, l). The weight $w(i, j, k, l)$ is the weight assigned to sample (k, l) for filtering the sample (i, j), and it is defined as

$$w(i,j,k,l) = e^{-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2} - \frac{(I(i,j)-I(k,l))^2}{2\sigma_r^2}}. \quad (1)$$

$I(i, j)$ and $I(k, l)$ are the intensity value of samples (i, j) and (k, l) respectively. The strength of the bilateral filter is controlled by σ_d (spatial strength) and σ_r (intensity strength). Samples located closer to the sample to be filtered, and samples having smaller intensity difference to the sample to be filtered, will have larger weights than samples further away and with a larger intensity difference. The output filtered sample value $I_F(i, j)$ is calculated as:

$$I_F(i,j) = \frac{\sum_{k,l} I(k,l) * w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)} \quad (2)$$

II. Proposed method

This paper proposes to reduce ringing artifacts within a video encoder and decoder by using a bilateral filter. A small, low-complex bilateral filter is applied on reconstructed

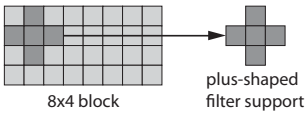

Fig. 1. Example of an 8x4 block and the shape of the filter for the sample located at (1,1).

samples of the luminance channel after the inverse transform has been performed and the result has been combined with the predicted sample values. Like the method of Nosratinia [2], the idea is that strong structures, which are expected to have survived the quantization and therefore represent the real signal, are preserved by the filtering, while weak structures, which are likely to be caused by ringing, are surpressed.

For complexity reasons we only use the four closest neighboring samples for filtering, giving a plus-shaped filter as shown in Figure 1. Although this filter is very small, it may still reduce ringing artifacts as shown in Figure 2. An original

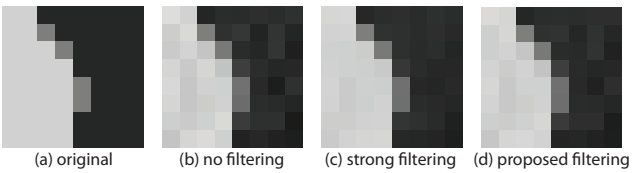

Fig. 2. (a) original block, (b) after compression/decompression at QP 32 (18.1 dB), (c) after strong bilateral filtering (21.4 dB), (d) after suggested bilateral filtering (19.2 dB).

block (a) is transformed, quantized with quantization parameter (QP) 32 and inverse transformed, giving rise to ringing (b). After filtering with a strong plus-shaped bilateral filter, much of this ringing is removed (c). Setting the filter strength is a trade off—a strong filter such as in (c) may remove ringing artifacts almost completely, but may also remove part of the signal. This paper therefore proposes a weaker filtering (d), which reduces ringing artifacts only slightly, but on the other hand affects the signal less.

The plus-shaped filter means that Equation 2 only contains five weights; the weight for the center sample, w_C, the weight for the sample above (w_A), below (w_B), left (w_L) and right (w_R). We can therefore simplify Equation 2 to

$$I_F = \frac{w_C I_C + w_A I_A + w_B I_B + w_L I_L + w_R I_R}{w_C + w_A + w_B + w_L + w_R}. \quad (3)$$

Weights for samples outside the transform block are set to zero. The weight w_C equals 1 since the center sample has no difference in neither position nor intensity. For the other weights, $(i-k)^2 + (j-l)^2$ is always 1, which means that

$$w = e^{-\frac{1}{2\sigma_d^2} - \frac{|\Delta I|^2}{2\sigma_r^2}}, \quad (4)$$

where ΔI is the difference in intensity to the center sample. In our implementation, σ_d is set based on the width and height of the transform unit, since smaller blocks typically contain more detail and therefore benefit from stronger filtering. We use

$$\sigma_d = p - \frac{\min(width, height, 16)}{40}, \quad (5)$$

where $p = 0.92$ for intra predicted blocks and $p = 0.72$ for inter predicted block. The motivation for this difference in p is that inter predicted blocks refer to previous frames where samples have already been through the bilateral filter at least once, so a weaker filter is used to avoid overfiltering. We set σ_r based on the QP used for the current block

$$\sigma_r = \max(\frac{QP - 17}{2}, 0.01). \quad (6)$$

The motivation for using QP to control the filter strength is that a high QP (i.e., a low bit rate) will give a lot of ringing/quantization artifacts, justifying stronger filtering. At higher bit rates, there are less ringing artifacts to correct, and the filtering is weakened with lower QP values until it is turned off completely for $QP < 18$. Also, the filter is turned off for blocks that do not have any non-zero transform coefficients and for inter predicted blocks of size 16×16 and larger.

This bilateral filter is applied to each transform block directly after adding the reconstructed residual values to the predicted values for the block, in both the encoder and the decoder. As a result of this, subsequent intra-coded blocks can predict from the sample values that have been filtered with the bilateral filter. It also means that the filter is applied before deblocking, sample adaptive offset filtering and adaptive loop filtering. In our implementation, the bilateral filter operation is also included in the rate-distortion decisions in the encoder, in order to select the modes which are best after filtering.

III. Computational Complexity

A. Look-up Table Based Implementation

A brute force implementation would calculate the four weights w_A, w_B, w_L and w_R using Equation 4 and then get the filtered sample using Equation 3. However, in such a case, the calculation of the weights becomes a bottleneck, particularly the four exponential functions.

In order to reduce the number of calculations, a look-up-table (LUT) can be used, storing all possible outcomes of Equation 4 in a three-dimensional array $w = \mathrm{LUT}(\sigma_d, \sigma_r, |\Delta I|)$. Since σ_d can take five different values, σ_r can take 34 different values, and $|\Delta I|$ can take 1023 different values assuming 10 bit luma data, the entire LUT can become over 170k values. Thus, while such a LUT may solve the computational complexity issue, its storage requirements may be too high for some implementations. Part of the contribution of this paper is therefore techniques devoted to reducing the size of this LUT.

Our first attempt of reduction has to do with the variable σ_d, which can take five different values according to Equation 5; 0.52, 0.72 and 0.82 for intra predicted blocks and 0.52 and 0.62 for inter predicted blocks (only two values since the filter is turned off for 16×16). Assume that we have stored the weights for just one of these, for instance 0.82,

$$w = e^{-\frac{1}{2(0.82)^2} - \frac{|\Delta I|^2}{2\sigma_r^2}}. \quad (7)$$

Now assume that we instead want the weight associated with another value of σ_d, such as 0.52. This new weight equals

$$w_{new} = e^{-\frac{1}{2(0.52)^2} - \frac{|\Delta I|^2}{2\sigma_r^2}} \tag{8}$$

$$= (e^{-\frac{1}{2(0.52)^2}})(e^{-\frac{|\Delta I|^2}{2\sigma_r^2}}), \tag{9}$$

and thus w_{new} can be written as

$$w_{new} = \frac{e^{-\frac{1}{2(0.82)^2}}}{e^{-\frac{1}{2(0.82)^2}}}(e^{-\frac{1}{2(0.52)^2}})(e^{-\frac{|\Delta I|^2}{2\sigma_r^2}}) \tag{10}$$

$$= \frac{e^{-\frac{1}{2(0.52)^2}}}{e^{-\frac{1}{2(0.82)^2}}}(e^{-\frac{1}{2(0.82)^2}})(e^{-\frac{|\Delta I|^2}{2\sigma_r^2}}), \tag{11}$$

$$= sw, \tag{12}$$

where $s = \frac{e^{-\frac{1}{2(0.52)^2}}}{e^{-\frac{1}{2(0.82)^2}}}$. Thus we can calculate a weight for another σ_d, such as 0.52, by taking the weight associated with $\sigma_d = 0.82$ and multiplying it by a constant s. By always using $\sigma_d = 0.82$ in the LUT, it can instead be made two-dimensional, $w = \mathrm{LUT}(\sigma_r, |\Delta I|)$, reducing it by a factor of five. The filtered value then becomes

$$I_F = \frac{I_C + sw_A I_A + sw_B I_B + sw_L I_L + sw_R I_R}{1 + sw_A + sw_B + sw_L + sw_R}. \tag{13}$$

To reduce the number of multiplications, what is actually used is the equivalent formula

$$I_F = \frac{s^{-1} I_C + w_A I_A + w_B I_B + w_L I_L + w_R I_R}{s^{-1} + w_A + w_B + w_L + w_R}, \tag{14}$$

which is equal to Equation 3 with the center weight value changed from 1.0 to s^{-1}. In our fixed point implementation, the value 65 is used to represent 1.0, making the largest LUT value 31, so five bits are enough for storage. For a 16×16 intra block, $\sigma_d = 0.92 - \frac{16}{40} = 0.52$, and the value for the center weight thus becomes $s^{-1} = 65 \frac{e^{-\frac{1}{2(0.82)^2}}}{e^{-\frac{1}{2(0.52)^2}}} = 196.35$, which is rounded to 196. The other center weights are calculated similarly, and are shown in Table I.

TABLE I
CENTER WEIGHT VALUE (s^{-1})

Block Type	min(width, height, 16)		
	4	8	16
intra	65	81	196
inter	113	196	not used

Since Equation 4 is monotoneously decreasing in $|\Delta I|$, if $\mathrm{LUT}(\sigma_r, |\Delta I|)$ is quantized to zero in five bits for a certain $|\Delta I_{limit}|$, we know that $\mathrm{LUT}(\sigma_r, x) = 0$ for all $x \geq |\Delta I_{limit}|$. Thus, if we access the LUT using $\mathrm{LUT}(\sigma_r, \min(\Delta I, \Delta I_{limit}))$ we will get the same result as $\mathrm{LUT}(\sigma_r, \Delta I)$, and the values $\mathrm{LUT}(\sigma_r, \Delta I_{limit} + 1) \ldots, \mathrm{LUT}(\sigma_r, 1023)$ (which are all zero) will never be accessed. By keeping track of ΔI_{limit} for every σ_r using a small table, we do not need to store these zero values. This dramatically reduces the storage demands since the average number of stored values per σ_r goes down from 1023 to 102. Therefore $\lceil 102 * 34 * 5/8 \rceil = 2168$ bytes are sufficient to store the LUT entries, where 34 is the number of possible values of σ_r. Including 34 bytes for the table to store ΔI_{limit}, the total number of bytes used for the LUT becomes 2202.

When filtering a sample $I(i,j)$, the absolute difference $|\Delta I| = |I(i,j) - I(i+1,j)|$ is used for the LUT look-up of the right weight w_R. When filtering the next sample, $I(i+1,j)$, the same absolute difference $|\Delta I| = |I(i+1,j) - I(i,j)|$ is used again, this time to calculate the left weight w_L. Since σ_r does not change inside a block, weight w_R for sample $I(i,j)$ will be the same as weight w_L for sample $I(i+1,j)$. By reusing the weight from the left and above sample, it is possible to lower the number of LUT lookups to two lookup operations per sample.

B. Hardware Friendly Option

The division in Equation 14 can be implemented efficiently on a CPU using an integer division instruction. Since these instructions round down, half of the denominator is added to the nominator before the division. However, for hardware implementations, an integer division may be expensive in terms of silicon area. In these cases, rather than dividing by n, it may be better to multiply by $2^k \frac{1}{n}$, which can be stored in a LUT called divLUT(n), and then right shift k steps. To preserve accuracy, k needs to be larger for larger values of n, and how much larger is stored in another LUT called shiftLUT(n). The maximum size of the nominator will determine how many bits each value of divLUT will have to be. Given 10-bit luma samples, the largest value of $w_A I_A$ is equal to $32 \times 1023 = 32736$, and the same goes for the other terms, making the largest possible nominator in Equation 14 quite high. By instead rewriting Equation 14 as

$$I_F = I_C + \frac{w_A \Delta I_A + w_B \Delta I_B + w_L \Delta I_L + w_R \Delta I_R}{s^{-1} + w_A + w_B + w_L + w_R}, \tag{15}$$

the largest possible nominator becomes much smaller. This is due to the fact that ΔI and w cannot be large simultaneously; if the intensity difference is big, the weight will be small. In our implementation, $w\Delta I \leq 1300$, and the maximum nominator size is therefore 5200. This means that the largest number stored in divLUT is 2^{14}, requiring 15 bits. Since the value 0 is never used in divLUT, this value can be used to represent 2^{14}, lowering the bit count to 14 bits per value. The shift value to use varies between 14 and 25 meaning that we need to store 4-bit values ($k - 14$) in shiftLUT. The number of entries in divLUT and shiftLUT is given by the maximum denominator n, which is $196 + 4 * 31 = 320$. Given that the denominator is always at least $s^{-1} \geq 65$, only the last $320 - 64 = 256$ values need to be stored, requiring $256 * (14 + 4)/8 = 576$ bytes of storage. In summary, $2202 + 576 = 2778$ bytes of LUT storage is sufficient for an efficient hardware implementation. If care is taken when doing rounding, it is possible to get the division-free implementation to match Equation 14 bit-exactly. Hence for CPU implementations, it may be easier to use the integer division instruction, while ASIC implementations can use the division-free version, maintaining compatibility.

IV. Results

The proposed filter was implemented in JEM 5.0.1, and tested according to the JVET common test conditions [8]. The main indicator of compression performance is the average Bjontegaard delta-rate [9] computed over the 21 sequences defined in the test conditions. Complexity is measured as run time. The results for still image coding (intra) and standard random access video coding (inter) are displayed in Table II. Figure 3 shows a number of examples of compressed material

TABLE II
Performance of proposed algorithm

	BD-rate	Encoder complexity	Decoder complexity
intra	-0.4%	+6%	+5%
inter	-0.5%	+3%	+0%

using first the anchor and then using the same compression but with the proposed bilateral filter enabled. The anchor is JEM 5.0.1, which includes sample adaptive offset and adaptive loop filtering, so any supression of ringing artifacts are on top of these. As can be seen in the figures, the amount of ringing is notacibly less for the bilateral filter. Note that the encoder will make slightly different mode choices when compressing the sequence with the proposed method, which affects the result. This means that if a block is sharper in one configuration (say, the bilateral version), it may be due to the fact that the encoder has chosen a more expensive way to encode the block than for the anchor. In another block, more bits may be spent on the anchor version, increasing quality there. On average, however, quality is improved for the bilateral filter since it decreases BD-rate. As can be seen in Figure 3, ringing is reduced under 'ria' in 'Adriano' and around 'Co' in 'Coletta'. Also, the dark-brown line of ringing situated four pixels above the yellow area in Basketballdrill is surpressed. The common test conditions [8] also specify a class of sequences which include screen content and computer generated content, which is not included in the regular test. For this class, the bilateral filter provides a substantially higher gain; the BD-rate is reduced by 1.8% for intra and 1.2% for inter. The third column of Figure 3 shows an example from the Chinaspeed sequence of this class, where considerable surpression of ringing around the black lines is visible. Note that unlike some other codecs, JEM does not include any specialized tools to improve coding of this type of content, and the gain from bilateral filtering for synthetic content such as Chinaspeed could be reduced if such tools were included.

V. Conclusion

This paper proposes the use a of a bilateral filter as a coding tool for video coding. The filtering is applied directly after the inverse transformed residual has been added to the prediction, and can therefore be used both for spatial and temporal prediction of subsequent blocks. A LUT based implementation is used to lower computational complexity, and part of the contribution of the paper is how to construct this LUT in order to make it small. It is also demonstrated how the filter can be implemented without division for hardware-friendlyness. The BD rate is reduced by 0.5% for video data, while increasing the encoding time by 3% and the decoding time by 0%.

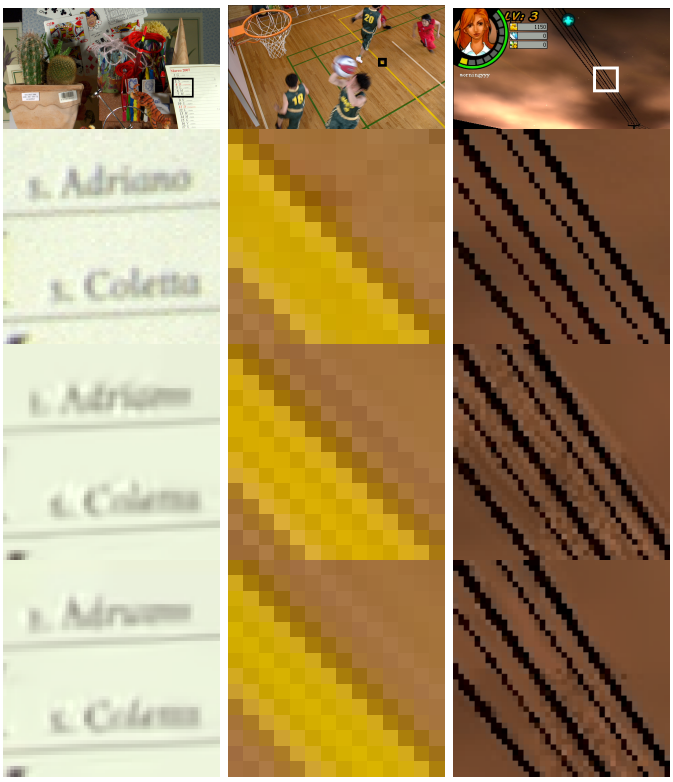

Fig. 3. First row: Original frames (or part thereof); Cactus, Basketballdrill, Chinaspeed. Second row: Original zoom-in. Third row: Anchor. Fourth row: Using the proposed bilateral filtering.

References

[1] *Video codec for audiovisual services at p x 384 kbit/s - Recommendation H.261 (11/88)*, ITU-T Std., 1997.

[2] A. Nosratinia, "Enhancement of jpeg-compressed images by re-application of jpeg," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 27, pp. 69–79, Feb. 2001.

[3] S. Srinivasan, P. J. Hsu, T. Holcomb, K. Mukerjee, S. L. Regunathan, B. Lin, J. Liang, M.-C. Lee, and J. Ribas-Corbera, "Windows media video 9: overview and applications," *Signal Processing: Image Communication*, vol. 19, no. 9, pp. 851 – 875, 2004.

[4] C. Y. Tsai, C. Y. Chen, T. Yamakage, I. S. Chong, Y. W. Huang, C. M. Fu, T. Itoh, T. Watanabe, T. Chujoh, M. Karczewicz, and S. M. Lei, "Adaptive loop filtering for video coding," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 6, pp. 934–945, Dec 2013.

[5] C. M. Fu, E. Alshina, A. Alshin, Y.-W. Huang, C.-Y. Chen, C.-Y. Tsai, C.-W. Hsu, S.-M. Lei, J.-H. Park, and W.-J. Han, "Sample adaptive offset in the hevc standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 1755–1764, Dec. 2012.

[6] *ISO/IEC 23008-2: High Efficiency Video Coding - Recommendation H.265 (11/88)*, ITU-T Std., 2013.

[7] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Sixth International Conference on Computer Vision*, Bombay, India, Jan. 1998, pp. 839–846.

[8] K. Suehring and X. Li. (2016) Jvet common test conditions and software reference configurations. [Online]. Available: http://phenix.int-evry.fr/jvet/doc_end_user/current_document.php?id=2617

[9] G. Bjøntegaard, "Calculation of average psnr differences between rd curves," in *document VCEG-M33, 13th VCEG Meeting, Apr. 2001.*, Austin, Texas, USA, Apr. 2001.