SIP (2020), vol. 9, e6, page 1 of 15 © The Authors, 2020.

doi:10.1017/ATSIP.2020.2

ORIGINAL PAPER

An Overview of Coding Tools in AV1: the First Video Codec from the Alliance for Open Media

YUE CHEN,[1] DEBARGHA MUKHERJEE,[1] JINGNING HAN,[1] ADRIAN GRANGE,[1] YAOWU XU,[1] SARAH PARKER,[1] CHENG CHEN,[1] HUI SU,[1] URVANG JOSHI,[1] CHING-HAN CHIANG,[1] YUNQING WANG,[1] PAUL WILKINS,[1] JIM BANKOSKI,[1] LUC TRUDEAU,[2] NATHAN EGGE,[2] JEAN-MARC VALIN,[3*] THOMAS DAVIES,[4] STEINAR MIDTSKOGEN,[4] ANDREY NORKIN,[5] PETER DE RIVAZ[6] AND ZOE LIU[7**]

In 2018, the Alliance for Open Media (AOMedia) finalized its first video compression format AV1, which is jointly developed by the industry consortium of leading video technology companies. The main goal of AV1 is to provide an open source and royalty-free video coding format that substantially outperforms state-of-the-art codecs available on the market in compression efficiency while remaining practical decoding complexity as well as being optimized for hardware feasibility and scalability on modern devices. To give detailed insights into how the targeted performance and feasibility is realized, this paper provides a technical overview of key coding techniques in AV1. Besides, the coding performance gains are validated by video compression tests performed with the libaom AV1 encoder against the libvpx VP9 encoder. Preliminary comparison with two leading HEVC encoders, x265 and HM, and the reference software of VVC is also conducted on AOM's common test set and an open 4k set.

Keywords: Video compression, Open-source video coding, AV1, Alliance for Open Media

Received 11 July 2019; Revised 19 December 2019

I. INTRODUCTION

Over the last decade, web-based video applications have become prevalent, with modern devices and network infrastructure driving rapid growth in the consumption of high-resolution, high-quality content. Therefore predominant bandwidth consumers such as video-on-demand(VoD), live streaming, and conversational video, along with emerging new applications including virtual reality and cloud gaming, that critically rely on high resolution and low latency, are imposing severe challenges on delivery infrastructure and hence creating an even stronger need for high efficiency video compression technology.

It is widely acknowledged that the success of web applications is enabled by open, fast iterating, and freely implementable foundation technologies, for example, HTML, web browsers (Firefox, Chrome, etc.), and operating systems like Android. Therefore, in an effort to create an open video format at par with the leading commercial choices, in mid 2013, Google launched and deployed the VP9 video codec [1]. As a codec for production usage, libvpx-VP9 considerably outperforms x264 [2], a popular open source encoder for the most widely used format H.264/AVC [3], while is also a strong competitor to x265, the open source encoder of the state-of-the-art royalty-bearing format H.265/HEVC [4] codec on HD content [5]. Thereby after the release YouTube progressively pushes through VP9 adoption, as of now a good amount of YouTube content will be streamed in VP9 format when it is possible at client side.

However, as the demand for high efficiency video applications rose and diversified, it soon became imperative to continue the advances in compression performance, as well as to incorporate designs facilitating efficient streaming in scenarios beyond typical VoD. To that end, in late 2015, a few leading video-on-demand providers, along with firms in semiconductor and web browser industry, co-founded the Alliance for Open Media (AOMedia) [6], which is now a consortium of more than 40 leading hi-tech companies, to work jointly toward a next-generation open video coding format called AV1.

The focus of AV1 development includes, but is not limited to achieving: consistent high-quality real-time video

[1]Google, USA
[2]Mozilla, USA
[3]Amazon, USA
[4]Cisco, UK and Norway
[5]Netflix, USA
[6]Argon Design, UK
[7]Visionular, USA
*Work performed while with Mozilla.
**Work performed while with Google.

Corresponding authors:
Y. Chen and D. Mukherjee
E-mails: yuec@google.com and debargha@google.com

delivery, scalability to modern devices at various bandwidths, tractable computational footprint, optimization for hardware, and flexibility for both commercial and non-commercial content. With VP9 tools and enhancements as the foundation, the codec was first initialized by solidifying promising tools from the separate VP10, Daala, and Thor codecs that founding members have worked on. Coding tools and features were proposed, tested, discussed, and iterated in AOMedia's codec, hardware, and testing workgroups, at the same time would be reviewed to work toward the goal of royalty-free. The AV1 video compression format has been finalized in the end of 2018, incorporating a variety of new compression tools, high-level syntax, and parallelization features designed for specific use cases.

As has been the case for other open-source projects, the development of AV1 reference software – *libaom*, is conducted openly in a public repository. The reference encoder and decoder can be built from downloaded source code [7] by following the guide [8]. Since the soft freeze of coding tools in early 2018, *libaom* has made significant progress in terms of productization-readiness, by a radical acceleration mainly achieved via machine learning powered search space pruning and early termination, and extra bitrate savings via adaptive quantization and improved future reference frame generation/structure. Besides *libaom*, now there are other AV1 decoder and encoder implementations available or to be released soon designed for various goals and usage cases. Royalty-free AV1 decoder *dav1d* by VideoLAN and FFmpeg community [9], targeting for smooth playback with low CPU utilization, now has been enabled by default in Firefox desktop version and will potentially also be included in Chrome soon. In the meantime, encoders for a variety of productization purposes are equally important to AV1's ecosystem, for example, *SVT-AV1* (by Intel and Netflix), *rav1e* (by Mozilla and Xiph.org), and *Aurora* (by Visionular) AV1 encoders are emerging AV1 encoding solutions focusing on one or more specific goals like high-performance, real-time encoding, on-demand content, immerse/interactive content, perceptual quality, etc.

Due to the increasing interest in AV1 performance, many efforts [10–17] have been made to conduct performance comparison between libaom-AV1 and other mainstream encoders of other formats. Among other work, the conclusions are different, and some of them have not listed key libaom encoder configurations. To address community's concerns on the contradictory conclusions, we would like to point out some issues that can be spotted in the provided information. References [15–17] claims that libaom-AV1 performs better than x265 under PSNR metric. Reference [16] presents over −30% BDRates for HD (≥720p) content and −17% overall BDRate, [15] shows less gain of 15% possibly because of the quality loss introduced by multi-thread AV1 encoding, and the most recent work [17] mentions that AV1 has −36% and −24% BDRates over x265-placebo and HM on the JVET test sets. Note that the results in this paper are mostly inline with [17] with similar performance gaps reported for a similar configuration of the codecs but with different test tests. Other work [10–14] claims much worse AV1 performance. We would like to suggest solutions to a few common issues of libaom configuration in the comparison between libaom, HEVC encoders, and VTM (the reference software of the upcoming format VVC [18]). Firstly, when random access mode is used for HEVC and VVC codecs, the counterpart mode of libaom is two-pass coding with non-zero (19 is recommended) lag-in-frames, rather than 1-pass mode and zero lag-in-frames which will disable all bi-directional prediction tools as in Refs. [10–12]. Secondly, as constant quality mode is usually used for HEVC and VVC codecs in the comparison, to match the setting, libaom needs to choose such mode as well by specifying end-usage = q and passing in the QP (cq-level), rather than either using libaom in vbr mode to match the rates produced by other codecs [13], or manually restraining libaom's QP by setting min-q and max-q [14]. Thirdly, in tests enforcing a 1s intra period, as the default configurations of HM and VTM use open-loop GOP structure, to match it, forward key frames need to be manually enabled for libaom by specifying enable-fwd-kf = 1, otherwise [10–12,14], libaom encodes in closed gop structure losing the benefit of cross gop referencing at every intra frame. Need to mention that we appreciate all those work, which greatly encourages community discussions and pushes AOM developers to make better documentation on how to use AV1 encoders.

In this paper, we present the core coding tools in AV1 that contribute to the majority of the 30% reduction in average bitrate compared with the most performant libvpx VP9 encoder at the same quality. Preliminary compression performance comparison between the *libaom* AV1 encoder and four popular encoders, including the *libvpx* VP9 encoder, two widely recognized HEVC encoders – *x265* and *HM* reference software, as well as *VTM*, are operated on test sets (AOM's common test set and an open 4k set) covering various resolution and content types under conditions that approximate common VoD encoding configurations.

II. AV1 CODING TECHNIQUES

A) Coding block partition

VP9 uses a four-way partition tree starting from the 64 × 64 level down to 4 × 4 level, with some additional restrictions for blocks below 8 × 8 where within an 8x8 block all the sub-blocks should have the same reference frame, as shown in the top half of Fig. 1, so as to ensure the chroma component can be processed in a minimum of 4 × 4 block unit. Note that partitions designated as "R" refer to as recursive in that the same partition tree is repeated at a lower scale until we reach the lowest 4 × 4 level.

AV1 increases the largest coding block unit to 128 × 128 and expands the partition tree to support 10 possible outcomes to further include 4:1/1:4 rectangular coding block sizes. Similar to VP9 only the square block is allowed for further subdivision. In addition, AV1 adds more flexibility to sub-8 × 8 coding blocks by allowing each unit has their own

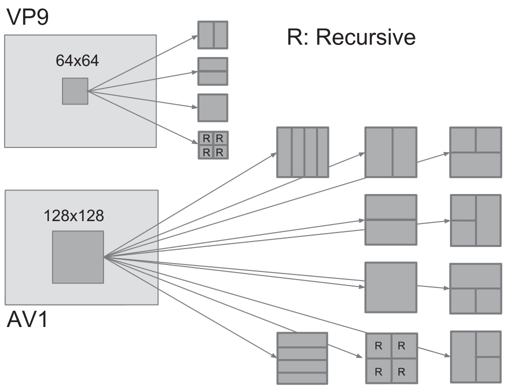

Fig. 1. Partition tree in VP9 and AV1 [32].

inter/intra mode and reference frame choice. To support such flexibility, it allows the use of 2×2 inter prediction for chroma component, while retaining the minimum transform size as 4×4.

B) Intra prediction

VP9 supports 10 intra prediction modes, including eight directional modes corresponding to angles from 45 to 207 degrees, and two non-directional predictors: DC and true motion (TM) mode. In AV1, the potential of an intra coder is further explored in various ways: the granularity of directional extrapolation is upgraded, non-directional predictors are enriched by taking into account gradients and evolving correlations, coherence of luma and chroma signals is exploited, and tools are developed particularly for artificial content.

1) Enhanced directional intra prediction

To exploit more varieties of spatial redundancy in directional textures, in AV1, directional intra modes are extended to an angle set with finer granularity for blocks larger than 8×8. The original eight angles are made nominal angles, based on which fine angle variations in a step size of 3 degrees are introduced, i.e. the prediction angle is presented by a nominal intra angle plus an angle delta, which is -3×3 multiples of the step size. To implement directional prediction modes in AV1 via a generic way, the 48 extension modes are realized by a unified directional predictor that links each pixel to a reference sub-pixel location in the edge and interpolates the reference pixel by a 2-tap bilinear filter. In total, there are 56 directional intra modes supported in AV1.

Another enhancement for directional intra prediction in AV1 is that, a low-pass filter is applied to the reference pixel values before they are used to predict the target block. The filter strength is pre-defined based on the prediction angle and block size.

2) New non-directional smooth intra predictors

VP9 has two non-directional intra prediction modes: DC_PRED and TM_PRED. AV1 expands on this by adding three new prediction modes: SMOOTH_PRED, SMOOTH_V_PRED, and SMOOTH_H_PRED. Also a fourth new prediction mode PAETH_PRED [19] replaces the existing mode TM_PRED. The new modes work as follows:

- *SMOOTH_PRED:* Useful for predicting blocks that have a smooth gradient. It works as follows: estimate the pixels on the rightmost column with the value of the last pixel in the top row, and estimate the pixels in the last row of the current block using the last pixel in the left column. Then calculate the rest of the pixels by an average of quadratic interpolation in vertical and horizontal directions, based on distance of the pixel from the predicted pixels.
- *SMOOTH_V_PRED:* Similar to SMOOTH_PRED, but uses quadratic interpolation only in the vertical direction.
- *SMOOTH_H_PRED:* Similar to SMOOTH_PRED, but uses quadratic interpolation only in the horizontal direction.
- *PAETH_PRED:* Calculate $base = left + top - top\_left$. Then predict this pixel as left, top, or top-left pixel depending on which of them is closest to "base". The idea is: (i) if the estimated gradient is larger in horizontal direction, then we predict the pixel from "top"; (ii) if it is larger in vertical direction, then we predict pixel from "left"; otherwise (iii) if the two are the same, we predict the pixel from "top-left".

3) Recursive-filtering-based intra predictor

To capture decaying spatial correlation with references on the edges, FILTER_INTRA modes are designed for luma blocks by viewing blocks as 2-D non-separable Markov models. Five filter intra modes are pre-designed for AV1, each represented by a set of eight 7-tap filters reflecting correlation between pixels in a 4×2 patch and the seven neighbors adjacent to the patch (e.g. $p_0 - p_6$ for the blue patch in Fig. 2). An intra block can pick one filter intra mode, and be predicted in batches of 4×2 patches. Each patch is predicted via the selected set of 7-tap filters weighting the neighbors differently at the 8 pixel locations. For those 4×2 units not fully attached to references on block boundary, e.g. the green patch in Fig. 2, predicted values of the immediate neighbors are used as the reference, meaning prediction is computed recursively among the 4×2 patches so as to combine more edge pixels at remote locations.

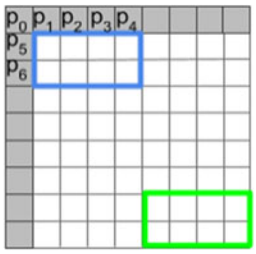

Fig. 2. Recursive-filter-based intra predictor.

4) Chroma predicted from Luma

Chroma from Luma (CfL) is a chroma-only intra predictor that models chroma pixels as a linear function of coincident reconstructed luma pixels. The predicted chroma pixels are obtained by adding the DC prediction to the scaled AC contribution. The DC prediction is computed using DC intra prediction mode on neighboring reconstructed chroma pixels. This is sufficient for most chroma content and has fast and mature implementations.

The scaled AC contribution is the result of multiplying the zero-mean-subsampled coincident reconstructed luma pixels by a scaling factor signaled in the bitstream. The subsampling step and average subtraction are combined to reduce the number of operations and also reduces rounding error.A scaling factors is signaled for each chroma plane but they are jointly-coded. Signaling scaling factors reduces decoder complexity and yields more precise RD-optimal predictions. Refer to Fig. 3 and [20] for more information.

5) Color palette as a predictor

Sometimes, especially for artificial videos like screen capture and games, blocks can be approximated by a small number of unique colors. Therefore, AV1 introduces the palette mode to the intra coder as a general extra coding tool. The palette predictor for each plane of a block is specified by (i) a color palette, with 2–8 colors, and (ii) color indices for all pixels in the block. The number of base colors determines the trade-off between fidelity and compactness. The base colors of a block are transmitted in the bitstream by referencing to the base colors of neighboring blocks. The base colors that are not present in the neighboring blocks' palettes are then delta-encoded. The color indices are entropy coded pixel-by-pixel, using neighborhood-based contexts. The luma and chroma channels can decide whether to use the palette mode independently. For the luma channel, each entry in the palette is a scalar value; for the chroma channels, each entry in the palate is a two-dimensional tuple. After the prediction of a block is established with the palette mode, transform coding and quantization is applied to the residue block, just like the other intra prediction modes.

6) Intra block copy

AV1 allows its intra coder to refer back to previously reconstructed blocks in the same frame, in a manner similar to how inter coder refers to blocks from previous frames. It can be very beneficial for screen content videos which typically contain repeated textures, patterns, and characters in the same frame. Specifically, a new prediction mode named IntraBC is introduced, and will copy a reconstructed block in the current frame as prediction. The location of the reference block is specified by a displacement vector in a way similar to motion vector compression in motion compensation. Displacement vectors are in whole pixels for the luma plane, and may refer to half-pel positions on corresponding chrominance planes, where bilinear filtering is applied for sub-pel interpolation.

The IntraBC mode is only available for keyframes or intra-only frames. It can be turned on and off by a frame-level flag. The IntraBC mode cannot refer to pixels outsize of current tile. To facilitate hardware implementations, there are certain additional constraints on the reference areas. For example, there is a 256-horizontal-pixel-delay between current superblock and the most recent superblock that IntraBC may refer to. Another constraint is that when the IntraBC mode is enabled for the current frame, all the in-loop filters, including deblocking filters, loop-restoration filters, and the CDEF filters, must be turned off. Despite all these constraints, the IntraBC mode still brings significant compression improvement for screen content videos.

C) Inter prediction

Motion compensation is an essential module in video coding. In VP9, up to two references, amongst up to three candidate reference frames, are allowed, then the predictor either operates a block-based translational motion compensation, or averages two of such predictions if two references are signaled. AV1 has a more powerful inter coder, which largely extends the pool of reference frames and motion vectors, breaks the limitation of block-based translational prediction, also enhances compound prediction by using highly adaptable weighting algorithms as well as sources.

1) Extended reference frames

AV1 extends the number of references for each frame from 3 to 7. In addition to VP9's LAST(nearest past) frame, GOLDEN(distant past) frame and ALTREF(temporal filtered future) frame, we add two near past frames (LAST2 and LAST3) and two future frames (BWDREF and ALTREF2) [21]. Figure 4 demonstrates the multi-layer structure of a golden-frame group, in which an adaptive number of frames share the same GOLDEN and ALTREF frames. BWDREF is a look-ahead frame directly coded without applying temporal filtering, thus more applicable

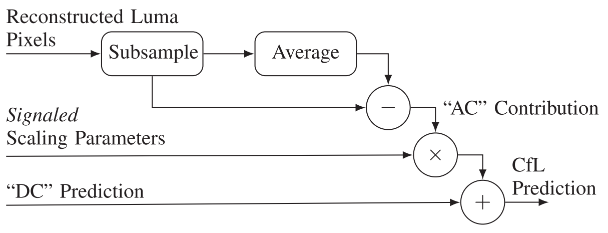

Fig. 3. Outline of the operations required to build the proposed CfL prediction [20].

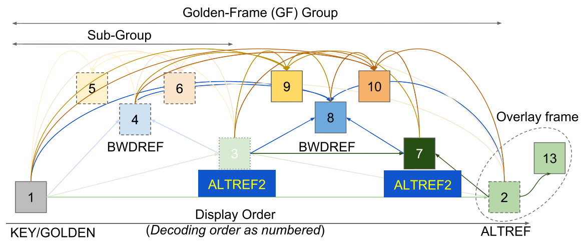

Fig. 4. Example of multi-layer structure of a golden-frame group [32].

as a backward reference in a relatively shorter distance. ALTREF2 serves as an intermediate filtered future reference between GOLDEN and ALTREF. All the new references can be picked by a single prediction mode or be combined into a pair to form a compound mode. AV1 provides an abundant set of reference frame pairs, providing both bi-directional compound prediction and uni-directional compound prediction, thus can encode a variety of videos with dynamic temporal correlation characteristics in a more adaptive and optimal way.

2) Dynamic spatial and temporal motion vector referencing

Efficient motion vector (MV) coding is crucial to a video codec because it takes a large portion of the rate cost for inter frames. To that end, AV1 incorporates a sophisticated MV reference selection scheme to obtain good MV references for a given block by searching both spatial and temporal candidates. AV1 not only searches a wider spatial neighborhood than VP9 to construct a spatial candidate pool, but also utilizes a motion field estimation mechanism to generate temporal candidates. The motion field estimation process works in three stages: motion vector buffering, motion trajectory creation, and motion vector projection. First, for each coded frame, we store its reference frame indices and the associated motion vectors. This information will be referenced by next coding frame to generate its motion field. The motion field estimation examines the motion trajectories, e.g. MV_{Ref2} in Fig. 5 pointing a block in one reference frame $Ref2$ to another reference frame $Ref0_{Ref2}$. It searches through the collocated 128×128 area to find all motion trajectories in 8×8 block resolution that pass each 64×64 processing unit. Next, at the coding block level, once the reference frame(s) have been determined, motion vector candidates are derived by linearly project passing motion trajectories onto the desired reference frames, e.g. converting MV_{Ref2} in Fig. 5 to MV_0 or MV_1. Once all spatial and temporal candidates have been aggregated in the pool, they are sorted, merged, and ranked to obtain up to four final candidates [22, 23]. The scoring scheme relies on calculating a likelihood of the current block having a particular MV as a candidate. To code an MV, AV1 signals the index of a selected reference MV from the list, followed by encoding the motion vector difference if needed.

3) Overlapped block motion compensation (OBMC)

OBMC can largely decrease prediction errors near block edges by smoothly combining predictions created from

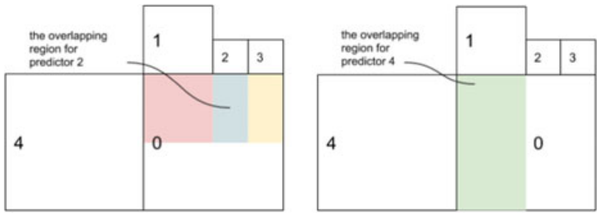
the overlapping region for predictor 2
1
2
3
4
0
the overlapping region for predictor 4
1
2
3
4
0

Fig. 6. Overlapping regions defined for AV1 OBMC.

adjacent motion vectors. In AV1, a two-sided causal overlapping algorithm is designed to make OBMC easily fit in the advanced partitioning framework [24]. It progressively combines the block-based prediction with secondary inter predictors in the above edge and then in the left, by applying predefined 1-D filters in vertical and horizontal directions. The secondary predictors only operate in restricted overlapping regions in top/left halves of the current block, so that they do not tangle with each other on the same side (see Fig. 6). AV1 OBMC is only enabled for blocks using a single reference frame, and only works with the first predictor of any neighbor with two reference frames, therefore the worst-case memory bandwidth is the same as what is demanded by a traditional compound predictor.

4) Warped motion compensation

Warped motion models are explored in AV1 through two affine prediction modes, global and local warped motion compensation [25]. The global motion tool is meant to handle camera motion, and allows frame-level signaling of an affine model between a frame and each reference. The local warped motion tool aims to capture varying local motion implicitly, using minimal overhead. Here, the model parameters are derived at the block level from 2D motion vectors signaled within the causal neighborhood. Both coding tools compete with translational modes at the block level, and are selected only if there is an advantage in RD cost. Additionally, affine models are limited to small degrees so that they can be implemented efficiently in SIMD and hardware through a consecutive horizontal and vertical shear (Fig. 7), each using an 8-tap interpolation filter at 1/64 pixel precision.

5) Advanced compound prediction

A collection of new compound prediction tools is developed for AV1 to make its inter coder more versatile. In this section, any compound prediction operation can be generalized for a pixel (i,j) as: $p_f(i,j) = m(i,j)p_1(i,j) + (1 - m(i,j))p_2(i,j)$, where p_1 and p_2 are two predictors, and p_f is the final joint prediction, with the weighting coefficients

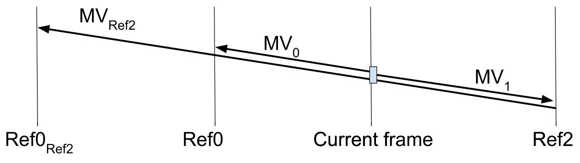

Fig. 5. Motion field estimation [32].

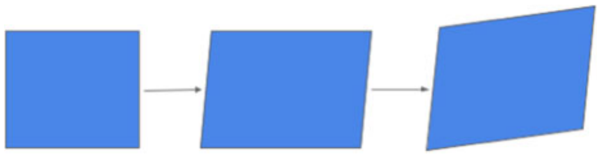

Fig. 7. Affine warping in two shears [32].

$m(i,j)$ in [0, 1] that are designed for different use cases and can be easily generated from predefined tables [26].

- *Compound wedge prediction*: Boundaries of moving objects are often difficult to be approximated by on-grid block partitions. The solution in AV1 is to predefine a codebook of 16 possible wedge partitions and to signal the wedge index in the bitstream when a coding unit chooses to be further partitioned in such a way. The 16-ary shape codebooks containing partition orientations that are either horizontal, vertical, or oblique with slopes ± 2 or ± 0.5, are designed for both square and rectangular blocks as shown in Fig. 8. To mitigate spurious high-frequency components, which often are produced by directly juxtaposing two predictors, soft-cliff-shaped 2-D wedge masks are employed to smooth the edges around the intended partition, i.e. $m(i,j)$ is close to 0.5 around the edges, and gradually transforms into binary weights at either end.
- *Difference-modulated masked prediction*: In many cases, regions in one predictor will contain useful content that is not present in the second predictor. For example, one predictor might include information that was previously occluded by a moving object. In these instances, it is useful to allow some regions of the final prediction to come more heavily from one predictor than the other. AV1 provides the option to use a non-uniform mask where the pixel difference between p_1 and p_2 is used to modulate mask weights over some base value. The mask is generated by: $m(i,j) = b + a|p_1(i,j) - p_2(i,j)|$ where b controls how strongly one predictor is weighted over the other within the differing regions and a scaling factor a ensures a smooth modulation.
- *Frame distance-based compound prediction*: Besides non-uniform weights, AV1 also utilizes a modified uniform weighting scheme by accounting for frame distances. Frame distance is defined as the absolute difference between timestamps of two frames. Intuitively if one reference frame is right next to a current frame and the other frame located more distant from current frame, it is expected that the reference block from the first frame has higher correlation with the current block and hence should be weighted more than the other. Let d_1 and d_2 ($d_1 \geq d_2$) represent distances from current frame to reference frames, from which p_1 and p_2 are computed. w_1 and w_2 are weights derived from d_1 and d_2. The most natural scheme is that weights are proportional to the reciprocal of frame distances, i.e. $w_1/w_2 = d_2/d_1$. However, a close observation reveals that the compound prediction should carry two major functionalities: exploiting the temporal correlation in the video signal and canceling the quantization noise from the reconstructed reference frames. The linear scheme does not take quantization noise into consideration. In a hierarchical coding structure with multiple reference frames, the relative distances of two reference frames from a current frame could differ substantially. A linear model would make the weight assigned to the block from a more distant frame too small to neutralize the quantization noise. On the other hand, the traditional average weighting, while not always tracking the temporal correlation closely, in general performs well to reduce the quantization noise. To balance these two factors, AV1 employs a modified weighting scheme derived based on the frame distances to give slightly more weight toward the distant predictor. The codebook is experimentally obtained and fixed in AV1:

$$p = \text{round}(w_1 * p_1 + w_2 * p_2 + 8) \gg 4$$

$$(w_1, w_2) = \begin{cases} (9,7), & \text{if } 2d_2 < 3d_1 \\ (11,5), & \text{if } 2d_2 < 5d_1 \\ (12,4), & \text{if } 2d_2 < 7d_1 \\ (13,3), & \text{if } 2d_2 \geq 7d_1 \end{cases} \quad (1)$$

The advantage of such quantized weighting coefficient scheme is that it effectively embeds certain attenuation of the quantization noise, especially when d_1 and d_2 are far apart. As a complementary mode to the traditional average mode, AV1 adopts a hybrid scheme, where the codec could switch between the frame distance-based mode and the averaging mode, at coding block level, based on the encoder's rate-distortion optimization decision.
- *Compound inter-intra prediction*: Compound inter-intra prediction modes, which combine intra prediction p_1 and single-reference inter prediction p_2, are developed to handle areas with emerging new content and old objects mixed. For the intra part, fourfrequently-used intra modes are supported. The mask $m(i,j)$ incorporates two types of smoothing functions: (i) smooth wedge masks similar to what is designed for wedge inter-inter modes, (ii) mode-dependent masks that weight p_1 in a decaying pattern oriented by the primary direction of the intra mode.

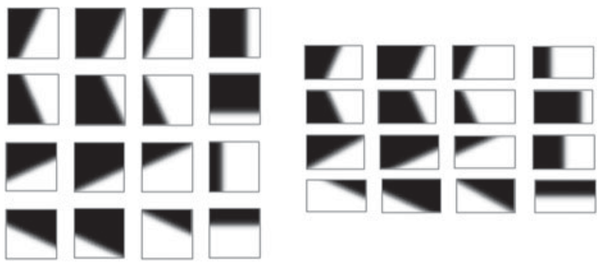

Fig. 8. Wedge codebooks for square and rectangular blocks [32].

D) Transform coding

1) Transform block partition

Instead of enforcing fixed transform unit sizes as in VP9, AV1 allows luma inter coding blocks to be partitioned into transform units of multiple sizes that can be represented by a recursive partition going down by up to two levels. To incorporate AV1's extended coding block partitions, we support square, 2:1/1:2, and 4:1/1:4 transform sizes from 4×4 to 64×64. For chroma blocks, only the largest possible transform units are allowed.

2) Extended transform Kernels

A richer set of transform kernels is defined for intra and inter blocks in AV1. The full 2-D kernel set is generated from horizontal/vertical combinations of four 1-D transform types, yielding 16 total kernel options [27]. The 1-D transform types are DCT and ADST, which have been used in VP9, flipADST, which applies ADST in reverse order, and the identity transform, which skips transform coding in order to preserve sharp edges. In practice, several of these kernels give similar results at large block sizes, allowing the gradual reduction of possible kernel types as transform size increases.

E) Entropy coding

1) Multi-symbol entropy coding

VP9 used a tree-based boolean non-adaptive binary arithmetic encoder to encode all syntax elements. AV1 moves to using a symbol-to-symbol adaptive multi-symbol arithmetic coder. Each syntax element in AV1 is a member of a specific alphabet of N elements, and a context consists of a set of N probabilities together with a small count to facilitate fast early adaptation. The probabilities are stored as 15-bit cumulative distribution functions (CDFs). The higher precision than a binary arithmetic encoderenables tracking probabilities of less common elements of an alphabet accurately. Probabilities are adapted by simple recursive scaling, with an update factor based on the alphabet size. Since the symbol bitrate is dominated by encoding coefficients, motion vectors, and prediction modes, all of which use alphabets larger than 2, this design in effect achieves more than a factor of 2 reduction in throughput for typical coding scenarios over pure binary arithmetic coding.

In hardware, the complexity is dominated by throughput and size of the core multiplier that rescales the arithmetic coding state interval. The higher precision required for tracking probabilities is not actually required for coding. This allows reducing the multiplier size substantially by rounding from 16×15 bits to an 8×9 bit multiplier. This rounding is facilitated by enforcing a minimum interval size, which in turn allows a simplified probability update in which values may become zero. In software, the operation count is more important than complexity, and reducing throughput and simplifying updates correspondingly reduces fixed overheads of each coding/decoding operation.

2) Level map coefficient coding

In VP9, the coding engine processes each quantized transform coefficient sequentially following the scan order. The probability model used for each coefficient is contexted on the previously coded coefficient levels, its frequency band, transform block sizes, etc. To properly capture the coefficient distribution in the vast cardinality space, AV1 alters to a level map design for sizeable transform coefficient modeling and compression [28]. It builds on the observation that the lower coefficient levels typically account for the major rate cost.

For each transform unit, AV1 coefficient coder starts with coding a skip sign, which will be followed by the transform kernel type and the ending position of all non-zero coefficients when the transform coding is not skipped. Then coefficients are broken down into sign plane and three level planes, where the sign plane cover the signs of coefficients and the three level planes correspond to different ranges of coefficient magnitudes. After the ending position is coded, the lower level and middle level planes are coded together in reverse scan order. Then the sign plane and higher level plane are coded together in forward scan order. The lower level plane corresponds to the range of 0–2, the middle level plane takes care of the range of 3–14, and the higher level plane covers the range of 15 and above.

Such separation allows one to assign a rich context model to the lower level plane, which accounts the transform directions: bi-directional, horizontal, and vertical; transform size; and up to five neighbor coefficients for improved compression efficiency, at the modest context model size. The middle level plane uses a context model similar to the lower level plane with number of context neighbor coefficients being reduced from 5 to 2. The higher level plane is coded by Exp-Golomb code without using context model. In the sign plane, except that the DC sign is coded using its neighbor transform units's dc signs as context information, other sign bits are coded directly without using context model.

F) In-loop filtering tools and post-processing

AV1 allows several in-loop filtering tools to be applied successively to a decoded frame. The first stage is the deblocking filter which is roughly the same as the one used in VP9 with minor changes. The longest filter is reduced to a 13-tap one from 15-taps in VP9. Further there is now more flexibility in signaling separate filtering levels horizontally and vertically for luma and for each chroma plane, as well as the ability to change levels from superblock to superblock. Other filtering tools in AV1 are described below.

1) Constrained directional enhancement filter (CDEF)

CDEF is a detail-preserving deringing filter designed to be applied after deblocking. It works by estimating the direction of edges and patterns and then applying a non-separable, non-linear low-pass directional filter of size 5×5 with 12 non-zero weights. To avoid signaling the directions, the decoder estimates the directions using a normative fast search algorithm. A full description of the decoding process can be found in Ref. [29].

- *Direction estimation:* The image to be filtered is divided into blocks of 8×8 pixels, which is large enough for reliable direction estimation. For each direction d, a line number k is assigned to each pixel as shown in Fig. 9 and the pixel average for line k is determined. The optimal direction is found by minimizing the square error calculated as the sum of squared differences between individual pixel values and the average for the corresponding lines.

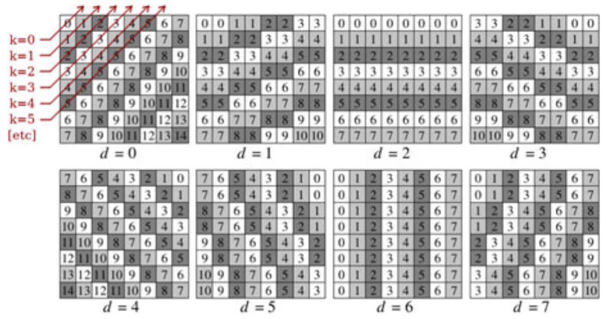

Fig. 9. Line number k following direction 0 to 7 in an 8×8 block [29].

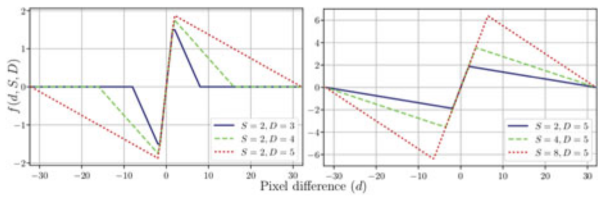

Fig. 10. The CDEF constraint function [29].

- *Non-linear low-pass filter:* The non-linear low-pass filter is designed to remove coding artifacts without blurring edges. This is achieved by selecting taps based on the identified direction and selecting filter strengths along and across the direction independently. The filter can be expressed as

$$y(i,j) = \text{round} \times \left(x(i,j) + g\left(\sum_{m,n \in R} w_{m,n} f\left(x\left(m,n\right) - x(i,j)\right) \right) \right) \tag{2}$$

  where R contains pixels in the neighborhood of $x(i,j)$ with the non-zero coefficients $w_{m,n}$, f and g are non-linear functions described below.Function $g(\cdot)$ ensures that the modification does not exceed the greatest difference between x and any $x(m,n)$. The function f constrains the pixel difference to be filtered by taking as arguments the difference d, a strength S, and a damping value D (see Fig. 10). The strength S controls the maximum difference allowed minus a ramp-down controlled by D.To allow control over the strength of the filtering along and across the identified direction, S is allowed to differ for different filter taps. Therefore for each direction, we define *primary taps* and *secondary taps* which have associated strengths, mapping to different sets of weights $w_{m,n}$. The strength values can be changed at 64×64 resolution. Coding blocks with no prediction residual ("skip" blocks) are not filtered. The damping D is signaled at the frame level.

2) Loop restoration filters

AV1 adds a set of tools for application in-loop after CDEF, that are selected in a mutually exclusive manner in units of what is called the loop-restoration unit (LRU) of selectable size 64×64, 128×128, or 256×256. Specifically, for each LRU, AV1 allows selecting between one of two filters [30] as follows.

- *Separable symmetric normalized Wiener filter*: Pixels are filtered with a 7×7 separable Wiener filter, the coefficients of which are signaled in the bit-stream. Because of the normalization and symmetry constraints, only three parameters need to be sent for each horizontal/vertical filter. The encoder makes a smart optimization to decide the right filter taps to use, but the decoder simply applies the filter taps as received from the bit-stream.
- *Dual self-guided filter*: For each LRU, the decoder first applies two cheap integerized self-guided filters of support size 3×3 and 5×5 respectively with noise parameters signaled in the bitstream. (Note self-guided means the guide image is the same as the image to be filtered.) Next, the outputs from the two filters, r_1 and r_2, are combined with weights (α, β) also signaled in the bit-stream to obtain the final restored LRU as $x + \alpha(r_1 - x) + \beta(r_2 - x)$, where x is the original degraded LRU. Even though r_1 and r_2 may not necessarily be good by themselves, an appropriate choice of weights on the encoder side can make the final combined version much closer to the undegraded source.

3) Frame super-resolution

It is common practice among video streaming services to adaptively switch the frame resolution based on the current bandwidth. For example, when the available bandwidth is low, a service may send lower resolution frames and then upscale them to the display device resolution. However, such a scaling occurs outside of the video codec as of now.

The motivation behind the new Frame Super-resolution framework in AV1 is to make this scaling process more effective by making it a part of the codec itself. This coding mode allows the frame to be coded at lower spatial resolution and then super-resolved normatively in-loop to full resolution before updating the reference buffers. Later, these super-resolved reference buffers can be used to predict subsequent frames, even if they are at a different resolution, thanks to AV1's scaled prediction capability.

Super-resolution is almost always observed to be much better than upscaling lower resolution frames outside the codec in objective metrics. In addition, at very low bit-rates, it is sometimes observed to be better than full resolution too in terms of perceptual metrics. Furthermore, it provides an extra dimension to the encoder for rate and quality control.

While there's plenty of research in this area, most super-resolution methods in the image processing literature are far too complex for in-loop operation in a video codec. In AV1, to keep operations computationally tractable, the super-resolving process is decomposed into linear upscaling followed by applying the loop restoration tool at higher spatial resolution. Specifically, the Wiener filter is particularly good at super-resolving and recovering lost high frequencies. The only additional normative operation is then a linear upscaling prior to use of loop restoration. Further, in order to enable a cost-effective hardware implementation with no

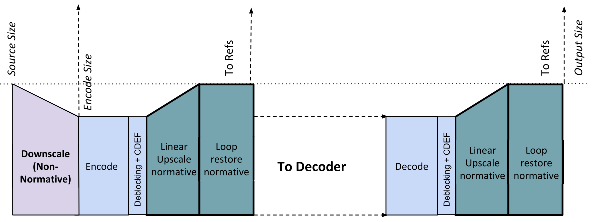

Fig. 11. In-loop filtering pipeline with optional super-resolution [32].

overheads in line-buffers, the upscaling/downscaling is constrained to operate only horizontally. Figure 11 depicts the overall architecture of the in-loop filtering pipeline when using frame super-resolution, where CDEF operates on the coded (lower) resolution, but loop restoration operates after the linear upscaler has expanded the image horizontally to resolve part of the higher frequencies lost. The downscaling factor is constrained to be in the range 15/16 to 8/16 (half).

4) Film grain synthesis

Film grain synthesis in AV1 is normative post-processing applied outside of the encoding/decoding loop [31]. Film grain, abundant in TV and movie content, is often part of the creative intent and needs to be preserved while encoding. Its random nature makes it difficult to compress with traditional coding tools. Instead, the grain is removed from the content before compression, its parameters are estimated and sent in the AV1 bitstream. The decoder synthesizes the grain based on the received parameters and adds it to the reconstructed video (see Fig. 12 for details).

The grain is modeled as an autoregressive (AR) process with up to 24 AR coefficients for luma and 25 for each chroma component (one more coefficient to capture possible correlation between the chroma and luma grain), which allows to support a wide range of different noise patterns. The AR process is used to generate 64×64 luma and 32×32 chroma grain templates (assuming 4:2:0 chroma subsampling). The 32×32 luma grain patches are then taken from pseudo-random positions in the template and applied to the video. To mitigate possible block artifacts from applying 32×32 patches separately to each block, an optional overlap operation can be applied to the noise samples before adding them to the reconstructed picture.

The tool supports flexible modeling of the relationship between the film grain strength and the signal intensity as follows:

$$Y' = Y + f(Y)G_L,$$

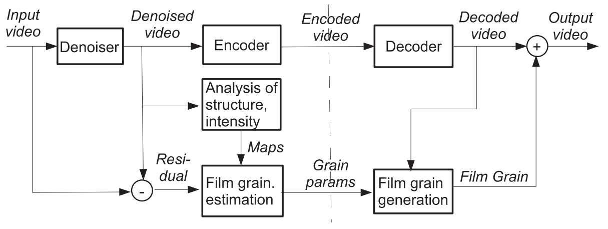

Fig. 12. Film grain estimation and synthesis framework [31].

where Y' is the resulting luma re-noised with film grain, Y is the reconstructed value of luma before adding film grain, and G_L is the luma film grain sample. Here, $f(Y)$ is a piecewise linear function that scales film grain depending on the luma component value. This piecewise linear function is signaled to the decoder and can be implemented as a precomputed look-up table (LUT) that is initialized before running the grain synthesis for the current frame. For a chroma component (e.g. Cb), the noise is modulated using the following formula to facilitate grain intensity modeling when the film grain strength in chroma depends on the luma component:

$$Cb' = Cb + f(u)G_{Cb},$$
$$u = b_{Cb}Cb + d_{Cb}Y_{av} + h_{Cb},$$

where u is an index in the LUT that corresponds to a Cb component scaling function, and parameters b_{Cb}, d_{Cb}, and h_{Cb} are signaled to the decoder.

A set of film grain parameters can take up to approximately 145 bytes. Each frame can either receive a new set of grain parameters or re-use parameters from one of the previously decoded frames if those are available.

For grainy content, film grain synthesis significantly reduces the bitrate necessary to reconstruct the grain (up to 50% bitrate savings can be found on sequences with heavy grain or noise). This tool is not used in the comparison in section 3 since it does not generally improve objective quality metrics because of the mismatch in positions of individual grains. More details on the film grain synthesis tool in AV1 can be found in Ref. [31].

G) Tiles and multi-threading

1) AV1 tiles

AV1 supports independent tiles consisting of multiple superblocks, and tiles can be encoded and decoded in arbitrary orders. Defined by encoding parameters, the tiles can be uniform(i.e. tiles have the same size) or non-uniform(i.e. tiles can have different size). Independent tile support provides the coding flexibility, so that the encoder and decoder could process tiles in parallel, and thus get much faster.

In *libaom* codebase, multi-threading (MT) has been implemented in both encoder and decoder, which includes tile-based MT and row-based MT. While using of tiles is permitted, tile-based MT gives a significant speedup. While none or few tiles are used, row-based MT allows the thread to encode and decode one single superblock row, and gives a further boost to the speed. Using four tiles and four threads in a 720p video coding, the encoder speedup is about 3×, and the decoder speedup is about 2.5×.

2) Large-scale tiles

With the increasing popularity of virtual reality (VR) applications, for the first time, AV1 provides a solution to make real-time VR applications feasible. The large-scale tile tool allows the decoder to extract only an interesting section in a frame without the need to decompress the entire frame.

This remarkably reduces the decoder complexity, and is extremely useful for real-time applications, such as light fields, which renders a single section of the frame following the viewer's head movement.

III. PERFORMANCE EVALUATION

After the short version of AV1 overview paper [32] drafted in mid 2018, the libaom AV1 encoder has evolved considerably to the good sides in terms of both compression performance and encoding complexity after the decoder syntax was finalized. Therefore, here the performance is re-validated using recent versions of AV1, in comparison with VP9, x265, HM, and VTM. We compare the coding performance obtained with the *libaom* AV1 encoder ([7] Aug 28, 2019 version, ddd5666c) on AOMedia's main test set *objective-1-fast* [33], against those achieved by the *libvpx* VP9 encoder ([34] Aug 28, 2019 version, bbb7f55), *x265* (June 18, 2019, v3.1), *HM* (version 16.17), and *VTM* (version 6.0). The *objective-1-fast* test set includes YUV420 8-bit videos of various resolution and types: 12 normal 1080p clips, four 1080p screen content clips, seven 720p clips, and seven 360p clips, all having 60 frames. In addition, due to the increasing interest in UHD content streaming, *sjtu4k* [35], a widely used set of 15 4k videos is also included in the evaluation, the YUV420 8-bit version of the *sjtu4k* set is used by the test discussed below.

Because the main focus of this paper is on compression technology of the AV1 format rather than the encoder implementation, and on AV1's progress over VP9 rather than extensively evaluating popular codecs on the market, we would like to mention that due to space and time limit, the performance evaluation in this manuscript is preliminary: inevitably not comprehensive due to the usage of the AOM test set as well as only for VoD scenario. Also it is difficult to achieve "complete fairness" due to the very different nature of codec designs. Therefore, comparisons conducted by other organizations on either large-scale production sets and configurations or other open test conditions are more than welcomed.

High compression quality modes of the five codecs (libvpx, x265, libaom, HM, and VTM) are performed on *objective-1-fast* and *sjtu4k*. Detailed configurations of all the codecs are enumerated in Table 1, in which we list the download links for HM and VTM's base configuration files and the extra configurations used that overwrite some of the default options. The codecs encode 60 frames of the test videos using 8-bit internal bit-depth, with only the first frame coded as a key frame. The quality parameters in Table 1 are chosen to make the five codecs produce videos in similar PSNR ranges to conduct meaningful BDRate computation.

Overall, the recommended highest quality modes are used. Both *libaom* and *libvpx* perform in 2-pass mode using constant quality(CQ) rate control at the slowest preset speed level *–cpu-used=0*. For other encoders, *x265* encodes in Constant Rate Factor (CRF) mode at the *placebo* (slowest) preset encoding speed, while for *HM* and *VTM* we use the recommended configuration files (download links [36,37] at the official repositories) for the *random access* mode. Note that the first pass of libaom and libvpx 2-pass mode simply conducts stats collections, consuming negligible time, rather than actual encodings, so in VoD scenario, it is fair to be compared with 1-pass HEVC or VVC encoding. Also need to mention that although *–cpu-used=0* is the slowest preset mode, a lot of pruning and early termination features are involved to achieve bearable complexities rather than conducting exhaustive search.

Table 1. Encoder configurations.

AV1, VP9	
–cpu-used=0	the best quality mode
–end-usage=q	the constant quality mode
–cq-level=*	*: the QP set is [20, 32, 43, 55, 63]
–frame-parallel=0 –threads=1	single threaded coding
–tile-columns=0	single tiled coding
–passes=2	2-pass coding
–kf-min-dist=1000	
–kf-max-dist=1000	only the 1st frame is a key frame
–lag-in-frames=19	support up to 19 frames in lookahead
-auto-alt-ref=6	(VP9 only) enable multi-layer GOP
-b 8	(default) 8-bit internal bit-depth
x265	
–preset placebo	the best quality mode
–crf *	*: the rate factor set is [15, 20, 25, 30, 35]
–frame-threads 1	single threaded coding
–no-wpp	no parallel CTU processing
–min-keyint 1000	
–keyint 1000	only the 1st frame is a key frame
–no-scene-cut	disable key frames triggered by scene cuts
–tune psnr	tune the quality in favor of PSNR scores
HM	
encoder_randomaccess_main.cfg	[36]
–QP=*	*: the QP set is [17, 22, 27, 32, 37]
–IntraPeriod=-1	only the 1st frame is a key frame
–DecodingRefreshType=2	closed loop GOP
VTM	
encoder_randomaccess_vtm.cfg	[37]
–QP=*	*: the QP set is [17, 22, 27, 32, 37]
–IntraPeriod=-1	only the 1st frame is a key frame
–DecodingRefreshType=2	closed loop GOP
–InternalBitDepth=8	8 bit internal bit-depth as other codecs

Regarding the Group of Picture (GOP) structures, libvpx, libaom, HM, and VTM use GOPs of up to 16 frames, either dynamic or fixed. Both *libaom* and *libvpx* adaptively allocate GOPs with a maximum of 16 frames, along with a lookahead buffer of 19 frames to properly determine the frame grouping. For HM and VTM, because GOP structure needs to be defined by the configuration file, as recommended [36,37], fixed GOPs of 16 frames are used. Although the range of GOP sizes is unclear in x265's placebo mode, we go with its preset configurations with up to 60 frames in the lookahead since the encoder is fast.

Table 2. BDRate(%) of libvpx-vp9, x265, HM, and VTM in comparison with libaom AV1 encoder on the objective-1-fast set and the sjtu4k set.

	Encoder			
Set	libvpx	x265	HM	VTM
	Avg-PSNR BDRate			
360p(o-1-f)	37.64	50.28	35.67	−4.02
720p(o-1-f)	43.25	54.08	38.97	−3.30
1080p(o-1-f)	38.98	45.71	30.92	−8.22
screen(o-1-f)	68.78	48.34	47.87	2.03
o-1-f*	39.77	49.19	34.37	−5.76
o-1-f	43.64	49.08	36.17	−4.72
sjtu4k	43.45	43.01	33.47	−1.04
	PSNR-Y BDRate			
360p(o-1-f)	38.46	46.35	32.55	−5.84
720p(o-1-f)	43.32	52.21	36.76	−5.53
1080p(o-1-f)	37.67	40.98	28.22	−9.34
screen(o-1-f)	66.40	45.41	43.92	1.21
o-1-f*	39.40	45.45	31.68	−7.37
o-1-f	43.00	45.44	33.32	−6.23
sjtu4k	42.56	38.98	30.74	−2.87
	PSNR-Cb BDRate			
360p(o-1-f)	30.81	77.54	55.15	8.73
720p(o-1-f)	48.02	89.54	66.38	11.12
1080p(o-1-f)	50.69	92.05	58.87	2.85
screen(o-1-f)	122.25	121.87	105.80	15.06
o-1-f*	44.62	87.46	59.89	6.66
o-1-f	54.97	92.05	66.01	7.78
sjtu4k	50.95	72.74	53.90	10.66
	PSNR-Cr BDRate			
360p(o-1-f)	48.66	91.85	65.10	17.86
720p(o-1-f)	49.02	90.71	68.22	11.23
1080p(o-1-f)	50.10	91.29	53.95	−0.30
screen(o-1-f)	90.89	85.70	79.45	7.40
o-1-f*	49.42	91.28	60.79	7.69
o-1-f	54.95	90.54	63.28	7.65
sjtu4k	47.23	68.50	50.60	10.48

The difference of coding performance is demonstrated in BDRates [38] under the average PSNR, PSNR-Y, PSNR-Cb, and PSNR-Cr metrics. Specifically the average PSNR accounts for the distortion at all the luma and chroma pixels (the ratio of Y/Cb/Cr pixels is 4:1:1 in an YUV420 video) in each frame then averages the PSNR numbers from all frames.

Table 2 shows the BDRates calculated for libvpx, x265, HM, and VTM, using libaom as the fixed anchor and AvgPSNR/PSNR-Y/PSNR-Cb/PSNR-Cr as the quality metric. A negative BDRate means using less bits to achieve the same quality score. The BDRates computed on subsets at different resolution are listed. In addition to the average on the whole *objective-1-fast* set (see "o-1-f"), we also evaluate the performance on only the 26 natural content clips (see "o-1-f*") of *objective-1-fast* by excluding the results on the four screen content clips, because HM and VTM may have separate extension or configuration specialized for screen content. The results in Table 2 validate that libaom has achieved considerable gains over libvpx, which is considered as the core component of the initial foundation of libaom's development, at all resolution, on

Table 3. Mutual Avg-PSNR BDRates(%) between libvpx-vp9, x265, libaom-AV1, HM, VTM on the objective-1-fast set.

	Anchor				
Codec	libvpx	x265	libaom	HM	VTM
libvpx	0	−2.63	43.67	6.91	53.34
x265	3.77	0	48.77	9.97	57.11
libaom	−29.46	−31.70	0	−25.41	6.41
HM	−5.87	−8.55	36.17	0	42.90
VTM	−33.83	−35.51	−4.72	−29.48	0

Table 4. Mutual Avg-PSNR BDRates(%) between libvpx-vp9, x265, libaom-AV1, HM, VTM on the sjtu4k set.

	Anchor				
Codec	libvpx	x265	libaom	HM	VTM
libvpx	0	2.12	44.73	8.64	47.05
x265	−0.81	0	43.01	7.53	46.77
libaom	−29.79	−29.50	0	−24.79	1.32
HM	−6.76	−6.70	33.47	0	36.22
VTM	−31.00	−31.31	−1.04	−26.33	0

both natural and screen content subsets, under all the listed PSNR metrics. The difference between BDRates under luma PSNR and chroma PSNR, for example, 43.00% PSNR-Y BDRate and 54.97% PSNR-Cb BDRate when evaluating libvpx against libaom on *objective-1-fast*, demonstrates that AV1 has made good progress in compressing the chroma component. Although BDRates under PSNR-Cb and PSNR-Cr cannot be accounted as comprehensive data points, need to mention that the disparity between BDRates calculated from luma and chroma channels is even bigger when comparing x265, HM and VTM against libaom, very likely due to more focus on Y components' quality in the design of these encoders.

To make the data more informative, we also provide BDRates for all possible pairs of encoders in Tables 3 and 4, on *objective-1-fast* and *sjtu4k* respectively. The reason for providing the data is that the absolute values of BDRates differ a lot if we switch the "anchor" codec and the "tested" codec when two encoders' performance has big difference. When assessing advances in compression performance, we usually use the less performant encoder as the anchor. Compared against libvpx, libaom has achieved substantial coding gains of −29.46% on *objective-1-fast* and −29.79% on *sjtu4k*. Overall, in our tests, among the five codecs, libvpx and x265, close with each other in coding efficiency, are the least performant. HM performs consistently better than x265 and libvpx by 6.76% to 8.69%, while is outperformed by libaom by around 24%. VTM achieves consistent coding gains over libaom by a margin of 4.72%.

We also measure the encoding and decoding complexity in Table 5 by the normalized encoding time and the normalized decoding time using libaom as the anchor. Note that x265 does not provide independent decoder, so there are no data for x265 decoding time. We can see that libaom speed 0 encoding is 20× slower than libvpx, and the decoder uses 4× time. Among all the encoders in the above mentioned

Table 5. Encoding and decoding complexity of libvpx-vp9, x265, HM, VTM using libaom-AV1 as the baseline.

	Codec				
Usage	libvpx	x265	libaom	HM	VTM
Encoding	0.048	0.125	1	0.152	2.342
Decoding	0.240	–	1	0.337	2.091

encoding configurations, libvpx is the fastest one, x265 is slightly faster than HM, while VTM is around 2.3× slower than libaom. Similar ranking can be observed when considering the decoding complexity.

IV. CONCLUSION

This paper provides a comprehensive technical overview of core coding tools in the state-of-the-art open video format AV1 developed by the Alliance for Open Media. Key features in the video codec's essential modules, including partitioning, prediction, transform coding, entropy coding, etc., are presented to demonstrate how AV1 distinguishes from VP9 (one of its anchor codecs) and leverages the new techniques into significant compression gains.

In addition, comparisons are performed among five encoders: libaom-AV1, libvpx-VP9, x265, HM, and VTM, under their corresponding high compression performance modes on AOM's test set and a third-party 4k set. The BDRate of libaom in comparison with libvpx-VP9 validates that the AV1 coding format has reached its original goal by about 30% bitrate savings over VP9. When comparing libaom with two HEVC encoders, considerable gains (24% and up) are also observed, while libaom is consistently outperformed by VTM.

ACKNOWLEDGMENTS

We would like to express deep gratitude to all AOMedia members and individual contributors for their ongoing contribution to AV1 development and deployment. Due to space limitations, we only list authors who participated in drafting this manuscript.

REFERENCES

1 Mukherjee D. *et al.*: The latest open-source video codec VP9 - an overview and preliminary results, in *Picture Coding Symp. (PCS)*, December 2013.

2 x264. https://www.videolan.org/developers/x264.html.

3 Wiegand T.; Sullivan G.J.; Bjontegaard G.; Luthra A.: Overview of the H.264/AVC video coding standard. *IEEE Trans. Circuits. Syst. Video. Technol.*, **13** (7) (2003), 560–576.

4 Sullivan G.J.; Ohm J.; Han W.; Wiegand T.: Overview of the high efficiency video coding (HEVC) standard. *IEEE Trans. Circuits. Syst. Video. Technol.*, **22** (12) (2012), 1649–1668.

5 MSU video codecs comparison. https://www.compression.ru/video/codec_comparison/hevc_2018/#hq_report.

6 Alliance for open media. http://aomedia.org.

7 libaom repository. https://aomedia.googlesource.com/aom/.

8 libaom build guide. https://aomedia.googlesource.com/aom/#get-the-code.

9 dav1d is an av1 decoder. https://code.videolan.org/videolan/dav1d.

10 Testing av1 and vvc. https://www.bbc.co.uk/rd/blog/2019-05-av1-codec-streaming-processing-hevc-vvc.

11 An overview of recent video coding developments in mpeg and aomedia. https://www.ibc.org/story.aspx?storyCode=3303.

12 Akyazi P.; Ebrahimi T.: Comparison of compression efficiency between hevc/h.265, vp9 and av1 based on subjective quality assessments, in *10th Int. Conf. on Quality of Multimedia Experience (QoMEX)*, 2018.

13 Laude T.; Adhisantoso Y.; Voges J.; Munderloh M.; Ostermann J.: A comparison of jem and av1 with hevc: Coding tools, coding efficiency and complexity, in *Picture Coding Symp. (PCS)*, 2018.

14 Nguyen T.; Marpe D.: Future video coding technologies: A performance evaluation of av1, jem, vp9, and hm, in *Picture Coding Symp. (PCS)*, 2018.

15 Guo L.; Cock J.; Aaron A.: Compression performance comparison of x264, x265, libvpx and aomenc for on-demand adaptive streaming applications, in *Picture Coding Symp. (PCS)*, 2018.

16 Best video codec: An evaluation of av1, avc, hevc and vp9. https://bitmovin.com/av1-multi-codec-dash-dataset/.

17 Laude T.; Adhisantoso Y.G.; Voges J.; Munderloh M.; Ostermann J.: A comprehensive video codec comparison. *APSIPA Transactions on Signal and Information Processing*, **8**, (2019), e30.

18 Versatile video coding. https://jvet.hhi.fraunhofer.de/.

19 Paeth filter. https://www.w3.org/TR/PNG-Filters.html.

20 Trudeau L.N.; Egge N.E.; Barr D.: Predicting chroma from luma in AV1, in *Data Compression Conf.*, March 2018.

21 Lin W. *et al.*: Efficient AV1 video coding using a multi-layer framework, in *Data Compression Conf.*, March 2018.

22 Han J.; Xu Y.; Bankoski J.: A dynamic motion vector referencing scheme for video coding, in *IEEE Int. Conf. on Image Processing*, September 2016.

23 Han J.; Feng J.; Teng Y.; Xu Y.; Bankoski J.: A motion vector entropy coding scheme based on motion field referencing for video compression, in *IEEE Int. Conf. on Image Processing*, October 2018.

24 Chen Y.; Mukherjee D.: Variable block-size overlapped block motion compensation in the next generation open-source video codec, in *IEEE Int. Conf. on Image Processing*, September 2017.

25 Parker S.; Chen Y.; Mukherjee D.: Global and locally adaptive warped motion compensation in video compression, in *IEEE Int. Conf. on Image Processing*, September 2017.

26 Joshi U. *et al.*: Novel inter and intra prediction tools under consideration for the emerging AV1 video codec, in *Proc. SPIE, Applications of Digital Image Processing XL*, 2017.

27 Parker S. *et al.*: On transform coding tools under development for VP10, in *Proc. SPIE, Applications of Digital Image Processing XXXIX*, 2016.

28 Han J.; Chiang C.-H.; Xu Y.: A level map approach to transform coefficient coding, in *IEEE Int. Conf. on Image Processing*, 2017.

29 Midtskogen S.; Valin J.-M.: The AV1 constrained directional enhancement filter (CDEF), in *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, April 2018.

30 Mukherjee D.; Li S.; Chen Y.; Anis A.; Parker S.; Bankoski J.: A switchable loop-restoration with side-information framework for the emerging AV1 video codec, in *IEEE Int. Conf. on Image Processing*, September 2017.

31 Norkin A.; Birkbeck N.: Film grain synthesis for AV1 video codec, in *Data Compression Conf.*, March 2018.

32 Chen Y. *et al.*: An overview of core coding tools in the av1 video codec, in *Picture Coding Symp. (PCS)*, 2018.

33 objective-1-fast test set. https://people.xiph.org/~tdaede/sets/objective-1-fast/.

34 libvpx repository. https://chromium.googlesource.com/webm/libvpx.

35 Sjtu 4k video sequences. http://medialab.sjtu.edu.cn/web4k/index.html.

36 encoder_randomaccess_main.cfg. https://hevc.hhi.fraunhofer.de/trac/hevc/browser/tags/HM-16.17/cfg/encoder_randomaccess_main.cfg.

37 encoder_randomaccess_vtm.cfg. https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM/blob/master/cfg/encoder_randomaccess_vtm.cfg.

38 Bjontegaard G.: Improvements of the BD-PSNR model, in *VCEG-AI11 ITU-T SG16 Q.6 Document*, July 2008.

Dr. Yue Chen is with the video compression team at Google. She is an active contributor to open-source video coding technology, including AV1 and VP9. Prior to joining Google, she received the B.S. degree in Electronic Engineering from Tsinghua University in 2011, and the M.S. and Ph.D. degrees in Electrical and Computer Engineering from the University of California Santa Barbara in 2013 and 2016, respectively. She holds many patents in the field of video compression. Her research interests include video/image compression and processing.

Dr. Debargha Mukherjee received his M.S./Ph.D. degrees in ECE from University of California Santa Barbara in 1999. Thereafter, through 2009 he was with Hewlett Packard Laboratories, conducting research on video/image coding and processing. Since 2010 he has been with Google Inc., where he is currently a Principal Engineer involved with open-source video codec research and development, notably VP9 and AV1. Prior to that he was responsible for video quality control and 2D-3D conversion on YouTube. Debargha has authored/co-authored more than 100 papers on various signal processing topics, and holds more than 60 US patents, with many more pending. He has delivered many workshops, keynotes and talks on Google's royalty-free line of codecs since 2012, and more recently on the AV1 video codec from the Alliance for Open Media (AOM). He has served as Associate Editors of the IEEE Trans. on Circuits and Systems for Video Technology and IEEE Trans. on Image Processing. He is also a member of the IEEE Image, Video, and Multidimensional Signal Processing Technical Committee (IVMSP TC).

Jingning Han received the B.S. degree in Electrical Engineering from Tsinghua University in 2007, and the M.S. and Ph.D. degrees in Electrical and Computer Engineering from University of California Santa Barbara in 2008 and 2012, respectively. His research interests include video coding and computer architecture. Dr. Han was a recipient of the Outstanding Teaching Assistant Awards in 2010 and 2011, the Dissertation Fellowship in 2012, both from the Department of Electrical and Engineering at University of California Santa Barbara. He was a recipient of the Best Student Paper Award at the IEEE International Conference on Multimedia and Expo in 2012. Dr. Han received the IEEE Signal Processing Society Best Young Author Paper award in 2015.

Adrian Grange is a member of the Chrome Media team at Google where he has contributed to the development of each generation of video codec from VP3 to VP9. Adrian also manages the Google Chrome University Research Program. Most recently, Adrian contributed to the technical development of the AV1 video codec, and to the formation of the Alliance for Open Media where he is Chair of the Codec Working Group.

Dr. Yaowu Xu is currently the Principal Software Engineer at Google, leading Google's video compression team that he helped to build and grow. His team is responsible for developing the core video technology that enables a broad range of products and services at Google including YouTube, Hangouts, Google Play and Stadia. His team has been the driving force behind open source video compression technology VP9 and AV1. Prior to joining Google through its acquisition of On2, Dr. Xu was the Vice President of Codec Development at On2 Technologies, where he co-created the VPx series codecs from VP3 to VP8. Dr. Xu holds a Ph.D. degree in Nuclear Engineering from Tsinghua University in Beijing, China, and a Ph.D. degree in Electrical and Computer Engineering from the University of Rochester in Rochester, NY. He has been granted more than one hundred thirty patents related to video compression.

Sarah Parker received her bachelors in Neuroscience at Brown University and became interested in image processing working on biological computer vision. She is currently a software engineer on the video coding team at Google, and has been working on AV1 2015. Sarah continues to enjoy learning about the video coding space, and is currently focusing on developing new tools for AV1's successor.

Cheng Chen received the B.S. degree in the Department of Automation from Tsinghua University in 2011, in Beijing, China. He received the M.S. and Ph.D. degrees in Electrical & Computer Engineering from the University of Iowa, in 2014 and 2016, respectively. He is now a software engineer at Google Inc. working on next generation video compression research and development.

Dr. Hui Su is currently a staff software engineer of the video coding team at Google. He has been working on video compression for many years. Before joining Google, he received the B.S. degree in Electrical Engineering from the University of Science and Technology of China in 2009, and the M.S. and Ph.D. degrees in Electrical and Computer Engineering from the University of Maryland at College Park in 2012 and 2014, respectively. His research interests include image/video compression, signal processing, and machine learning.

Urvang Joshi received his ME degree in Computer Engineering from Indian Institute of Science, Bangalore. He is currently a senior software engineer at Google, and has been contributing new compression techniques for open-source video codec AV1. He also worked on developing the open-source image format WebP. Before joining Google, he worked on Bing relevance team at Microsoft and face/object detection problems at Yahoo labs. His research interests include video compression and machine learning.

Ching-Han Chiang is currently a software engineer at Google and has been working on VP9 and AV1 for more than 4 years. Before joining Google, she was a software engineer at MStar Semiconductor, Taiwan. She also worked as a research assistant at Academia Sinica, Taiwan, conducting research on Computer Vision. She received her B.S. degree in Electrical Engineering from National Sun Yat-sen University, Kaohsiung, Taiwan. She received her M.S. degree in Electrical Engineering from National Tsing-Hua University, Hsinchu, Taiwan. She received her M.S. degree in Computer Science from New York University, USA. Her research interests include Computer Vision and ideo Compression. Her hobbies are Magic the Gathering, ukulele and climbing.

Yunqing Wang is a senior staff software engineer at Google, and has been working on AV1 and VPx series video codec development since 2007. Dr. Wang received the B.S. and Ph.D. degrees in Precision Instrument from Tsinghua University in 1991 and 1996, respectively, and the M.S. degree in Computer Science from University of Massachusetts at Amherst in 2002. Her research interests include video compression and optimization.

Paul Wilkins is an Educated at Cambridge University in the UK, Paul Wilkins has been working in the field of video compression since the early 1990s. He joined On2 in 1999 and ultimately became joint CTO alongside James Bankoski. After the acquisition of On2 by Google, Paul was technical lead for the VP9 project and is currently working on optimizing and improving the AV1 encoder.

Jim Bankoski is a Distinguished Engineer at Google leading the team responsible for Google's video, audio, and image compression efforts. Theteam was founded when On2 was acquired back in 2011. Jim was the CTO of On2 Technologies. He has contributed to the design of all of On2/Google's video codecs from Tm2x through VP9, including video codecs widely used in Flash, Skype and now WebM. His team also works on hardware implementations and VP9 is now adopted as a hardware component by most of the major TV manufacturers. He is currently leading Google's Alliance for Open Media Codec efforts at Google, the major new open source codec under development. The AOM organisation now comprises more than 40 companies.

Luc Trudeau received a Ph.D., M. Eng. and B.S. Eng. degrees from École de Technologie Supérieure, Montreal, in 2017, 2011 and 2009 respectively. Since 2018, he currently works as a research scientist with Two Orioles, LLC, New York, where he is involved in applications of video compression research in the EVE-AV1 and EVE-VP9 encoders. Luc is also a volunteer developer for the nonprofit organization VideoLAN, where he contributes to the dav1d decoder project. From 2016 to 2018, he was a research assistant for the Mozilla corporation and worked on the design and implementation of the Chroma from Luma coding tool in the AV1 video format.

Nathan Egge is a Senior Research Engineer at Mozilla and a member of the non-profit Xiph.Org Foundation. Nathan works on video compression research with the goal of producing best-in-class, royalty-free open standards for media on the Internet. He is a co-author of the AV1 video format from the Alliance for Open Media and contributed to the Daala project before that.

Jean-Marc Valin has a B.S., M.S., and Ph.D. in Electrical Engineering from the University of Sherbrooke. He is the primary author of the Speex speech codec and one of the main authors of the Opus audio codec. He also contributed to the Daala and AV1 video codecs. He has volunteered with the Xiph.Org Foundation since 2002. His expertise includes speech and audio coding, machine learning, speech enhancement, and other audio-related topics. He is currently a principal applied scientist at Amazon Web Services.

Thomas Davies graduated from Oxford University, Oxford, U.K., with the M.A. degree in Mathematics and Philosophy in 1991. He received the M.S. and Ph.D. degrees in Mathematics from Warwick University, Coventry, U.K., in 1992 and 1996, respectively. Having joined Cisco in 2011, he is currently a Principal Engineer in the Collaboration Technology Group. Here he has contributed to AV1 and HEVC video codec standard development, and the open source Thor codec. He works on video processing and video codec research, with a focus on real-time implementations for cloud conferencing applications. Previously, he led video codec research at BBC Research and Development, where in addition to working on HEVC he developed the open source Dirac video codec algorithms, the intra coding part of which is standardized as SMPTE VC-2. Before joining the BBC, he was a Technology Consultant in systems engineering and satellite networking.

Steinar Midtskogen holds a master's degree in informatics from the University of Oslo. He's an engineer at Cisco Systems in Norway with more than 20 years of experience with video compression technology. For much of this time he has focused on real-time video encoding and computationally-efficient codec implementations. In recent years he has participated in the research and development of the Thor and AV1 video codecs. His main contribution to the AV1 video codec was the Constrained Directional Enhancement Filter (CDEF), which was jointly developed with Jean-Marc Valin of Mozilla.

Andrey Norkin received the M.Sc. degree in computer engineering from Ural State Technical University, Yekaterinburg, Russia, in 2001 and the Doctor of Science degree in signal processing from Tampere University of Technology, Tampere, Finland, in 2007. From 2008 to 2015, he was with Ericsson, Sweden, conducting research on video compression and 3D video. In 2014, he worked on video encoding techniques for broadcast products at Ericsson TV, Southampton, UK. Since 2015, Dr. Norkin has been with Netflix, USA as a Senior Research Scientist working on encoding techniques for OTT video streaming, High Dynamic Range (HDR) video, and new video compression algorithms. He has participated in ITU-T and MPEG efforts on developing video compression standards, including HEVC, its extensions, and VVC, having multiple technical contributions and coordinating work in certain areas, such as deblocking and loop filters. He has also been involved in the Alliance for Open Media work and actively contributed to the development of AV1 video codec. Dr. Norkin's current research interests include video compression, OTT streaming, HDR video, and machine learning techniques.

Peter de Rivaz received his PhD in Image and Signal Processing from the University of Cambridge in 2000. In 2001 he was a co-founder of Alphamosaic where he designed the instruction set for the Videocore processor and implemented the MPEG-4 and H.264 video codecs for the Apple Video iPod. In 2009 he was a co-founder of the consultancy Argon Design where he designed video hardware engines for Raspberry Pi and participated in the VP9 and AV1 video codec development for Google. His research interests are machine intelligence and compiler design. His hobbies are Japanese, climbing, and recreational mathematics (as part of the Project Euler problem development team).

Zoe Liu received her PhD from Purdue University, West Lafayette, IN, and her ME/BE from Tsinghua University in Beijing. She is the Co-Founder & President of Visionular Inc., Palo Alto, CA, a startup delivering cutting edge video solutions to enterprise customers worldwide. Zoe was previously a software engineer with the Google WebM team and has been a key contributor to the newly finalized royalty free video codec standard AOM/AV1. She has devoted to the design and development of innovative products in the field of video codec and real-time video communications for almost 20 years. She was a 2018 Google I/O speaker. She has published more than 40 peer-reviewed international conferences and journal papers. Her main research interests include video compression, image processing, and machine learning.