Fast Encoding Parameter Selection For Convex Hull Video Encoding

Ping-Hao Wu, Volodymyr Kondratenko, and Ioannis Katsavounidis

Facebook Inc., Menlo Park, CA 94025, USA

ABSTRACT

With the recent development of video codecs, compression efficiency is expected to improve by at least 30% over their predecessors. Such impressive improvements come with a significant, typically orders of magnitude, increase in computational complexity. At the same time, objective video quality metrics that correlate increasingly better with human perception, such as SSIM and VMAF, have been gaining popularity. In this work, we build on the per-shot encoding optimization framework that can produce the optimal encoding parameters for each shot in a video, albeit carrying itself another significant computational overhead. We demonstrate that, with this framework, a faster encoder can be used to predict encoding parameters that can be directly applied to a slower encoder. Experimental results show that we can approximate within 1% the optimal convex hull, with significant reduction in complexity. This can significantly reduce the energy spent on optimal video transcoding.

Keywords: Video encoding, video codec, per-shot encoding, convex hull

1. INTRODUCTION

Video encoding technologies have evolved rapidly in the past decade, with the development of modern video codecs such as MPEG's Versatile Video Coding (VVC)[1] and AOM's AV1.[2] These advanced video codecs have promised and are expected to improve on compression efficiency by at least 30% over their predecessors, HEVC[3] and VP9,[4] respectively. At the same time, objective video quality metrics that correlate increasingly better with human perception, such as SSIM[5] and VMAF,[6] have been gaining popularity, both as system monitoring tools but also as closed-loop mechanisms to improve quality of encoding. Such impressive improvements come with a significant increase in computational complexity, in particular on the encoder, which is typically orders of magnitude slower compared to previous generation codecs. As adaptive streaming – which requires multiple encoded versions of a source video – is gaining popularity and becomes the dominating form of entertainment video distribution, and as the amount of online videos increases rapidly, online video services face a tremendous volume of video encoding everyday; thus, the computational complexity of new video codecs has becoming increasingly important. With the ever growing popularity and exploding number of online videos, the amount of computations, and thus energy, spent on encoding – but also in measuring video quality – cannot be ignored.

In this work, we first look at several popular video codecs, their vastly different ranges of computational complexity and compression efficiency, and the disproportionate increase in complexity carried by the newer generations of codecs. Then we extend and build on the per-shot encoding optimization framework, which has been proven capable of producing the optimal resolution and quality for each shot in a video sequence, given any bitrate or quality target, albeit carrying itself another significant computational overhead. We demonstrate that, with this framework, a faster encoder can be used to predict encoding parameters that can then be directly applied to a slower encoder. More specifically, we use a faster encoder to find the convex hull, and the corresponding list of optimal encoding parameters for each shot. We then apply these encoding parameters to the same video but with a slower encoder to produce encodings of the source video at various quality or bitrate targets. Our experimental results show that, within the same codec family, we can approximate within 1% the optimal convex hull, with significant reduction in computational complexity. We also show that it is possible to leverage such projection across different codec families. This can significantly reduce the overall energy spent on compression and quality measurement.

Further author information: (Send correspondence to Ping-Hao Wu)
Ping-Hao Wu: E-mail: npw@fb.com; Volodymyr Kondratenko: E-mail: vkondratenko@fb.com
Ioannis Katsavounidis: E-mail: ikatsavounidis@fb.com

2. COMPUTATIONAL COMPLEXITY OF MODERN VIDEO CODECS

2.1 Encoders Used

The state-of-the-art in video coding has been advancing steadily over the past 20 years. For this work, we focused on 3 formats that are most relevant to the Facebook ecosystem: H.264/AVC, VP9 and AV1.

Although standardized in 2004, H.264/AVC[7] is the most popular codec used in some of the most popular video applications all over the world, including broadcast TV, Blu-ray disks and internet video streaming. H.264/AVC is the result of joint work by ISO/IEC and ITU-T, carried by the JCT-VC (Joint Collaborative Team in Video Coding), a team of experts also known as MPEG. One of the most popular software encoder open-source implementations of H.264/AVC is appropriately called x264[8] and it has become a de facto standard, especially for internet streaming applications. Among the key advantages of x264 are its significantly faster speed and improved BD-rate performance over the reference H.264 encoder, the ability to further tradeoff quality for additional speedup, multiple modes of operation, including constant bitrate (CBR), variable bitrate (VBR) and constant quality, referred to as ”constant rate factor” (CRF). We will use x264 in this work to encode test video sequences in H.264/AVC format.

VP9 is a video coding standard developed by Google and finalized in 2013.[4] It follows the same motion-compensated hybrid transform coding followed by quantization and entropy coding scheme that produced all MPEG codecs. Its coding efficiency is generally superior to AVC, but in various benchmarks it hasn't reached the level of performance of the more recent HEVC[3] codec. The key premise of VP9 – which is what made it very popular among multiple adaptive streaming content providers – is that it is royalty-free, unlike its MPEG counterparts that typically come with licensing fees that have hampered the deployment of HEVC, mostly because of uncertainties on the amount and structure of such licensing fees. VP9 offers an open-source implementation of a compliant encoder and decoder, known as libvpx.[9] Libvpx was created and is mostly maintained by Google, who have been constantly updating, by fixing various software bugs, improving its computational efficiency but also improving its coding efficiency. The current stable version of libvpx is 1.8.2 (released Dec. 2019), which is the encoder we used in our experiments

AV1[2] is yet another video coding standard developed by the Alliance for Open Media (AOM), a group of big companies in the field of video communications, under the premise of royalty-free licensing. Finalized in July of 2018, AV1 has demonstrated superior performance compared to both AVC and HEVC, but also VP9. Its superior performance, though, comes at the expense of very slow encoding speed, especially if one is trying to achieve its maximum coding performance. An open-source reference implementation of AV1 encoder and decoder is offered through the libaom[10] package. Currently in its 2.0.0 stable version (released May 2020), libaom has drastically improved in terms of computational efficiency over the past 2 years, while also delivering meaningful compression efficiency improvements. We used libaom to conduct AV1 encoding experiments in this work.

2.2 HW video encoding

On May 14, 2019, Facebook publicly announced a video transcoding custom-chip design, codenamed “Mount Shasta”.[11] The main functional blocks offered by this video transcoding ASIC (application-specific integrated circuit) are:

- Decoder – Accepts uploaded video; outputs uncompressed RAW video stream
- Scaler – Block to resize (shrink) the uncompressed video
- Encoders – Outputs compressed (encoded) video
- Quality measurement – Logic to measure the degradation in video quality after the encoding step

The density and power efficiency of such HW encoding ASICs is typically 2 orders of magnitude higher than a corresponding SW video encoder. Yet, given the lower flexibility that HW offers compared to SW, the compression efficiency of HW video encoders is typically lower to that of SW video encoders.

In this work, we used the H.264/AVC encoding functionality of our “Mount Shasta” HW video encoder for our experiments.

2.3 Computational complexity and video quality

Modern video encoders typically have many different features that can be turned on/off or adjusted differently, in a way that would result in a trade-off between compression efficiency and computational complexity. To enable and facilitate easier adoption, a set of curated configurations are typically provided to the users, which are usually exposed as various presets or speed settings.

This topic has been studied by a number of researchers and the key results are that these speed settings of modern encoders can be used to satisfy almost any available compute budget, while offering the best possible quality that the underlying codec can offer.

At the most recent "Video@Scale" event, organized by Facebook in Oct. of 2019,[12] Ioannis Katsavounidis presented some of these findings that show this compute complexity vs. compression efficiency tradeoff offered by x264, libvpx and libaom. These results are shown in the following Fig. 1. For each encoder shown, multiple points are plotted, corresponding to different speed setting.

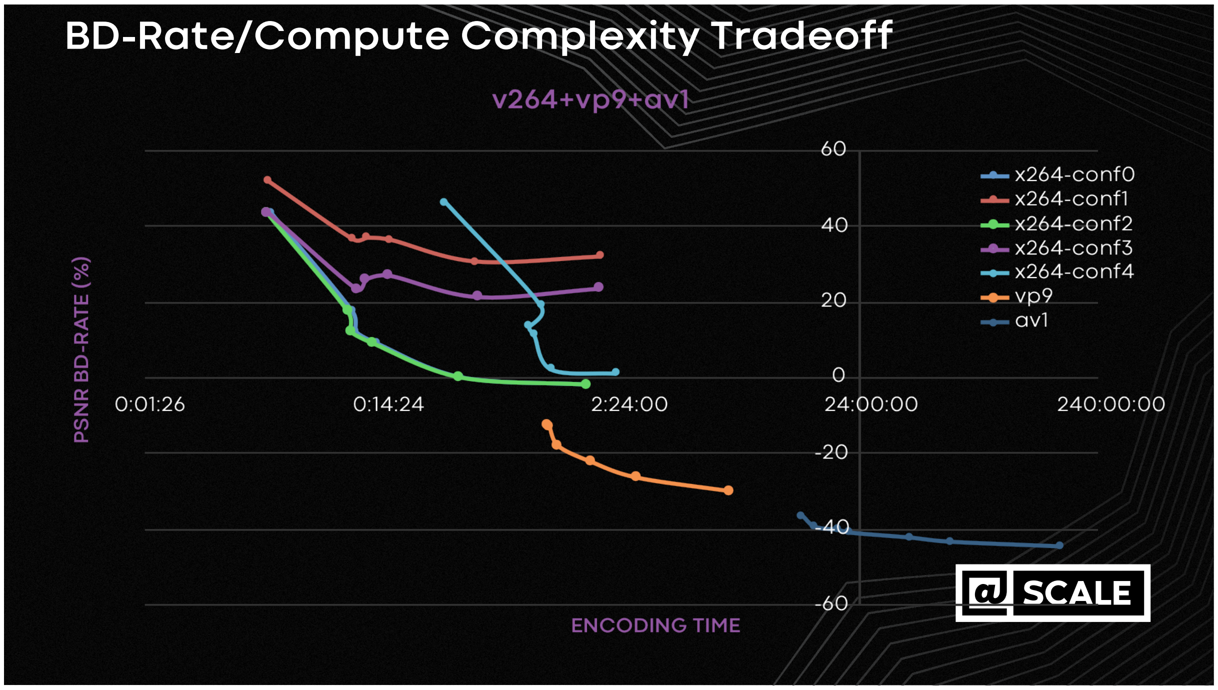


Figure 1. BD-rate vs. encoding compute complexity for various configurations of x264, libvpx and libaom; note log-x compute scale (execution time). Source: Video@Scale 2019

One can immediately notice the many orders of magnitude of compute complexity that the aforementioned encoders span. Such increasing compute complexity is prohibitive for social media video, given their huge volume.

3. DYNAMIC OPTIMIZER AND CONVEX HULL ENCODING

As mentioned earlier, in the modern era of internet video streaming, a video sequence is typically down-scaled to various resolutions, and encoded at different bitrates. This allows the client/player application to properly and quickly adapt to varying network conditions by switching to a different resolution and/or bitrate. Since stream switching is only feasible on I (or Key) frames, it is required that

- Encodings use closed GOP structure, i.e. key frames fully reset the state of a decoder
- All encoded versions of the same source video sequence have temporally aligned I-frames

In a 2018 Netflix tech blog,[13] Ioannis Katsavounidis laid out a framework that permits optimal encoding of multi-shot video sequences by splitting them into shots, applying constant-quality encoding within each shot and then combining encoded shots by maintaining constant quality-bitrate slope that meets a desired average quality of bitrate target. This framework, called "Dynamic Optimizer", permitted the use of constant-quality encoding to achieve a certain bitrate, without the use of traditional techniques such as in-loop rate control, that has been typically offered as either 1-pass or 2-pass encoding modes of various video encoders. The key result of that work was that Dynamic Optimizer achieved approximately 20% bitrate savings over baseline fixed QP encoding for a variety of content, encoders and quality metrics, but at the expense of a significant multiplier in encoding complexity, since each shot of a source video sequence was encoded multiple times, typically $6 - 10\times$.

As explained in the same tech blog, but further elaborated in the related work by Katsavounidis and Guo,[14] the exact same methodology can be applied to compare more accurately and in a way that is closer to actual adaptive streaming services, the relative performance of different encoders and coding standards. In fact, the use of Dynamic Optimizer is providing a truly fair comparison of the compression efficiency of different encoders, by removing any factor that rate control algorithms contribute to the efficiency or inefficiency of a given encoder.

In this section, we briefly describe the key components and ideas of the encoder comparison and evaluation methodology, as demonstrated in the work by Katsavounidis and Guo. This approach not only provides a comprehensive way to evaluate encoders, but also demonstrated a significant gain in compression efficiency, as measured by BD-rate,[15] compared to the traditional fixed-QP encoding method.

Key components of the methodology are:

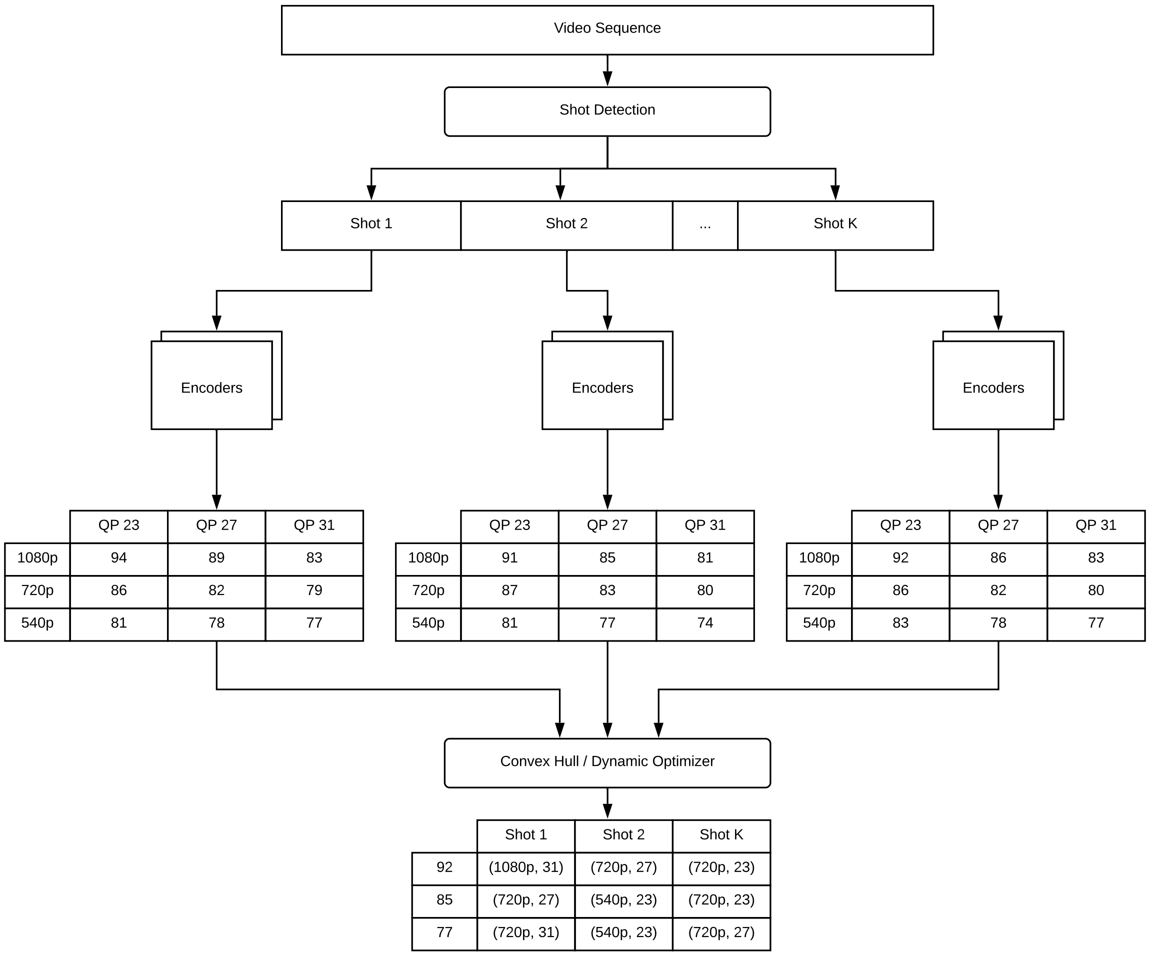
- Allow using multiple resolutions to better represent each video sequence in the adaptive streaming world.
- Use scaled quality metrics, where the lower-resolution versions are first decoded and up-scaled back to the original resolution for calculating quality metrics.
- Allow switching resolution and QP for different shots within the same video. The optimal selection of resolution and QP is based on the Dynamic Optimizer framework and the convex hull of each shot.

The process of selecting the optimal resolution and QP values via the Dynamic Optimizer framework is briefly described in the following. Interested readers are encouraged to refer to the prior works,[13].[14]

1. Perform shot detection on a long video sequence. Split it into multiple shots.
2. Each shot is downsampled to a number of pre-selected target resolutions.
3. Each shot, at each resolution, is encoded using various QP/CRF values that are allowed by the specific encoder and span a reasonable range of quality.
4. Each encode is decoded and upsampled back to the same resolution of the original video sequence.
5. VMAF, SSIM, PSNR and any other full-reference quality metrics are calculated at the same, original video sequence resolution
6. For each shot, all available encodes are filtered such that only those that are on the convex hull of the distortion-bitrate plane survive
7. Encodes from each shot's convex hull are matched according to their distortion/bitrate slope, thus offering optimal paths from beginning to the end of the source video sequence
8. The resulting aggregate distortion - or, equivalently, quality - and bitrate points for the entire sequence form what is called the convex hull of that sequence, in reference to the encoder used and the quality metric deployed.

A high-level diagram that illustrates the above flow can be seen in Fig. 2. One encode unit is illustrated by Fig. 3.

In this work, we conveniently refer to the encodings of a video sequence constructed and obtained using the above outlined methodology, as the convex hull encodings of the video sequence. One needs to always keep in mind that, since each one of the convex hull encodings of a sequence is obtained by the concatenation of a certain fixed (resolution, QP) encoding for each shot, we can view the convex hull of a sequence as a map that dictates what are these (resolution, QP) parameters that need to be used for each shot in order to achieve a given optimal (R, D) point. It is thus obvious that, any deviation from the optimal encoding parameters provided by a sequence's convex hull will result in encoding inefficiency, which we typically measure in terms of BD-rate, i.e. the increase in bitrate to achieve the same quality, typically expressed as a percentage.



	QP 23	QP 27	QP 31
1080p	94	89	83
720p	86	82	79
540p	81	78	77

	QP 23	QP 27	QP 31
1080p	91	85	81
720p	87	83	80
540p	81	77	74

	QP 23	QP 27	QP 31
1080p	92	86	83
720p	86	82	80
540p	83	78	77

	Shot 1	Shot 2	Shot K
92	(1080p, 31)	(720p, 27)	(720p, 23)
85	(720p, 27)	(540p, 23)	(720p, 23)
77	(720p, 31)	(540p, 23)	(720p, 27)

Figure 2. Illustration of per-shot convex hull encoding.

4. ENCODING PARAMETER SELECTION

One important outcome of convex hull encoding is, given a target bitrate or target quality, one can obtain the list of optimal resolutions and QPs for each shot in the video sequence, which we can simply call optimal encoding parameters. Because of the multiple encodings required to obtain the convex hull of each shot, one can immediately understand that, to optimally determine these encoding parameters, one would have to incur a much more significant computational complexity compared to traditional single-QP encoding across all shots.

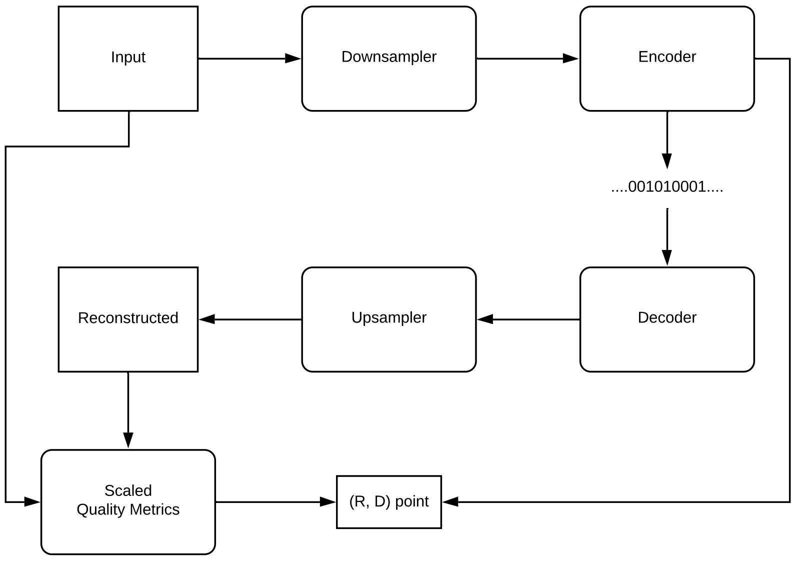


Figure 3. Illustration of one encode unit.

In this section, we examine the process of obtaining the optimal encoding parameters, and ways we can estimate them at a fraction of the compute cost.

4.1 Optimal Encoding Parameters

To obtain the optimal resolution and QP value for each shot, convex hull encoding employs multiple different resolutions and QP/CRF values and essentially conducts an exhaustive search on the underlying two dimensional parameter space. This would need a great number of encodes and introduce massive computational complexity.

Let us suppose that, for a given video sequence, we want to provide M different versions for adaptive video streaming. Most practical video streaming services choose a typical value of 5-10 for M, with each one of these versions corresponding to an encoding resolution, with some resolutions potentially receiving two encodings. For the rest of our analysis, we assume that each encoding resolution in conventional adaptive bitrate streaming use-case is assigned a single encoding, which means M is equal to the number of available encoding resolutions.

Traditionally for adaptive streaming, we would first choose a particular encoder and a set of encoding configurations such as GOP sizes, speed setting, etc. Then, as explained before, we would choose one resolution for each of these versions, and a target bitrate or target quality, specified by a QP/CRF value, for each of these resolutions. This would produce exactly M different encodings of the video sequence. These encodings are stored and the video player on the client side would choose a particular version depending on the network condition at a particular moment. Due to the adaptation that video players do in order to optimize video quality delivery under varying network conditions, the term “Adaptive BitRate streaming” (or ABR streaming) has been used widely in the industry and academia to describe this new paradigm of internet video delivery.

To employ convex hull encoding on the said video sequence, we would need to first split the video sequence into shots and encode all of them with N different QP/CRF values that can cover a reasonable range of visual quality. With all of these additional encodings, we would then be able to obtain the optimal selection of resolution and QP for every shot at a target quality or bitrate level.

Note that, this way, convex hull encoding introduces a total of N times the original computational complexity, or $(N - 1)$ times additional complexity, compared to the conventional single fixed-QP encoding per resolution. While this may be feasible for some codecs from previous generations that are much faster, such as H.264/AVC, it is certainly unrealistic with the latest generation of video codecs, such as VVC or AV1, which can be many orders of magnitude slower than previous generations.

Intuitively, we can say that, given a video codec and a configuration, there exists an optimal selection of encoding parameters that is content-dependent. Such optimal selection of encoding parameters would be different from shot to shot, and would depend on the spatiotemporal characteristics of each shot. Certain contents would be fine with a lower resolution - in particular those that are lacking fine details and/or have a high amount of camera-object motion. On the other hand, other contents may work better with a higher resolution and a lower QP. We shall demonstrate that, given a video codec, there is an inherent relationship between the content and its optimal encoding parameters that is nearly independent from other parameters that affect the operation of an encoder, such as speed settings. This intuition allows us to speed up the determination of encoding parameters, using different and much faster encoders to conduct the prerequisite multiple shot encodings.

4.2 Fast Parameter Selection

We now present a process for efficiently selecting optimal encoding parameters. Without loss of generality, it can be separated into three steps:

1. Preprocess step
2. Analysis step
3. Encoding step

In the following we describe each of these steps.

When a video sequence enters a video processing pipeline, it typically goes through a pre-process step first. This step can conceptually include multiple processes that can facilitate the subsequent analysis and encoding steps - and there are plenty of possibilities to do that. Here are some of the most relevant ones.

As it has been argued previously, primarily in the Dynamic Optimizer tech blog[13] and the subsequent work to compare video codecs using the Dynamic Optimizer framework,[14] the fundamental building blocks of video sequences are the continuous takes from a camera under fairly static lighting and environment conditions, called "shots". The key feature of shots is the homogeneity of frames within a shot, which makes it an ideal candidate to apply constant quality encoding into. It is thus beneficial to perform shot detection over an input video sequence, followed by segmentation into shots as part of the pre-processing step.

At the same time, it is necessary to create high-quality downsampled versions of these shots into different resolutions to facilitate analysis in the later step, and prepare for the eventual different versions of encodings generated for adaptive streaming. Many videos may also contain paddings such as letterbox or pillarbox, which would negatively impact coding efficiency, and may be desirable to be removed first, before analysis and encoding. Other image processing techniques, such as color enhancement, de-noising, artifacts removal, white balancing, etc. may also be selectively applied and performed during this step.

Once the video sequence is pre-processed, the analysis step is performed with the goal of determining - or, rather, estimating - the optimal encoding resolution and QP values for each shot, given the content and desired target average qualities or target average bitrates. As presented in Sec. 2, different presets or speed settings can be dramatically different in terms of computational complexity. What these different presets or speed settings do is turning on or off certain encoding features and/or applying heuristics to selectively enabling such features to a subset of the input video sequence. Yet, the fundamental quantization scheme that directly correlates with the amount of distortion introduced by an encoder, stays the same. On the other hand, the efficiency of an encoder, which at a given quality level translates to the amount of bits required, changes but in a way that is fairly predictable, constant and - most importantly - proportional to the complexity of each shot. In other words, the *percentage* of bits saved when switching from a faster encoder preset to a slower one remains fairly constant.

This implies that, in constructing the convex hull for a given shot, it is expected that a faster encoder preset will match to a large degree the results obtained by a slower preset, since the resulting distortion/bitrate slopes are expected to increase by a constant multiplicative factor. Moreover, when the time comes to match slopes from different shots in optimizing the aggregate quality or bitrate, the points chosen from each shot are expected to remain the same, regardless of the encoder preset used, even though the aggregate bitrate achieved by a faster encoder preset is always expected to be higher than the corresponding one from a slower encoder preset. In summary, optimality of encoding parameters, i.e. resolution and QP values for each shot, is expected to remain fairly constant when changing encoder speed presets.

With this intuition, we can apply the convex hull encoding concept, but with a faster encoder preset or speed setting. This way, we can encode the constituent shots much faster, obtain the rate-distortion performance of each shot at different resolution and QP values, and apply Dynamic Optimizer to determine the optimal encoding parameters.

At the final encoding step, these optimal encoding parameters that are obtained with a faster encoder setting, can be used directly with a slower encoder setting, in order to reap the additional benefit that these slower presets provide in terms of coding efficiency. In this way, we are utilizing the slower encoder setting at the minimum possible degree, which is to produce only the final M encodings, while we utilize a much faster encoder settings to perform the much higher - $(M \times N)$ - encodings.

Note that the pre-processing, analysis and final encoding steps must be done sequentially, but the multiple different operations within each, such as downsampling, encode, or quality metric calculations, can all be done in parallel.

In its simplest form, the procedure of the fast encoding parameter selection, as illustrated in Fig. 4, can be realized by the following:

1. Perform shot detection and split the video sequence into multiple shots.
2. Downsample and encode each shot using a faster encoder or a faster setting, at M different resolutions and N QP values.
3. Decode and upsample each encode back to the original resolution of the video sequence for metrics calculation.
4. Determine the convex hull for each shot
5. Determine the optimal selection of encoding parameters (resolution and QP) for each shot at a desired certain average quality level.
6. Encode using a slower encoder or encoding setting, by directly applying the optimal encoding parameters obtained in the previous step. The R-D curve obtained this way is referred in this work as the projected convex hull by a faster encoder into a slower encoder.

The proposed technique can also be considered as a prediction method, and as such, one expects some drop in BD-rate performance compared to the actual convex hull encoding performed by utilizing the same slower encoder for the analysis and final encoding steps. However, to our surprise, and as we will have a chance to elaborate further in Sec. 5, the drop is very small.

More specifically, and in reference to using the x264 encoder, the BD-rate difference originally observed between the convex hulls obtained when using the "veryfast" and "veryslow" settings is 16%. On the other hand, the projected convex hull by the "veryfast" preset into the "veryslow" preset is 1.25% on average behind the actual "veryslow" convex hull.

5. EXPERIMENTS

In the following, we present a few variants of the convex hull projection concept and thus provide a few different ways to achieve fast encoder parameter selection given the availability of different encoders.

1. Same encoder with different presets or speed settings.
2. Different encoders but within the same codec family.
3. Different codec families or implementations.

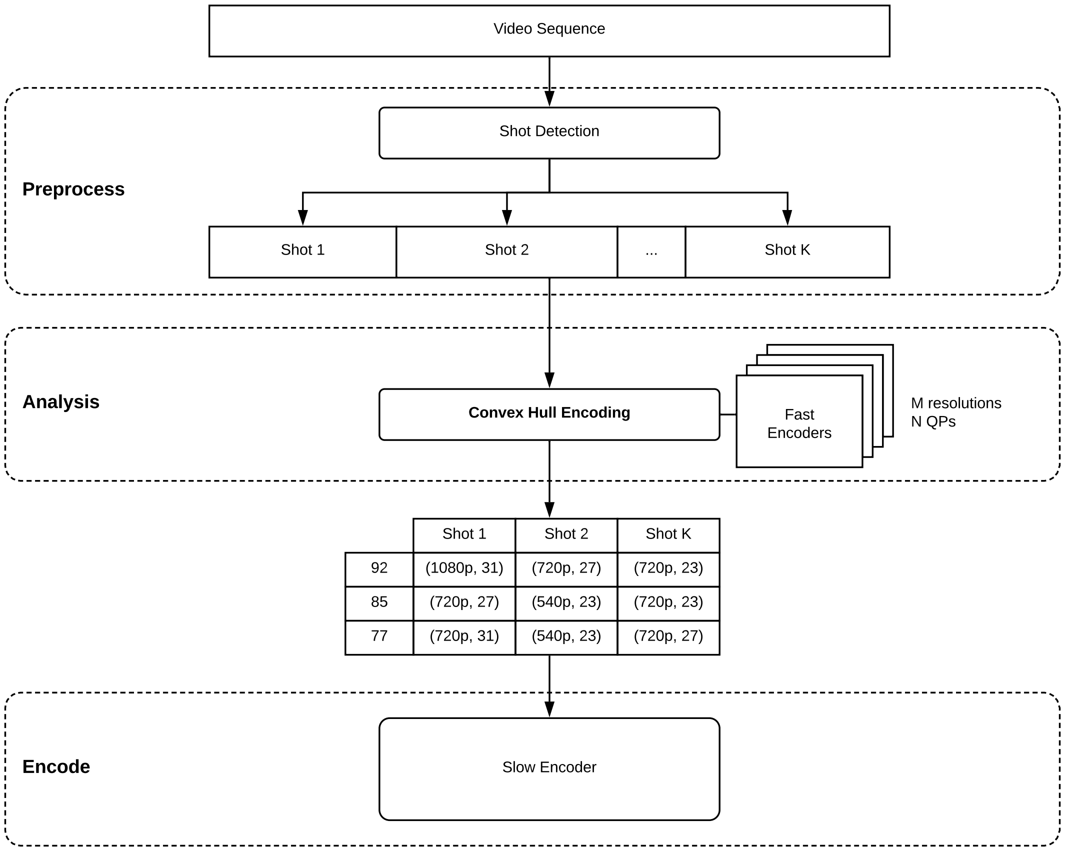


Figure 4. Illustration of fast encoding parameters selection for convex hull encoding.

5.1 Encoder Configurations

We decided to focus on 3 state-of-the-art video coding standards that have popular open-sourced software implementations: H.264/AVC, VP9, and AV1. As discussed in Sec. 2, all of these encoder implementations allow the trade-off between compression efficiency and computational complexity, along with a set of curated configurations, exposed as presets or speed settings. Below we describe the encoders and their speed settings that we used for our experiments.

For H.264/AVC, we used x264[8] and opted for three different presets: “veryslow”, “fast”, and “veryfast”. The common configurations we opted for our experiments include single-threaded mode, CRF mode, and tune PSNR. CRF, which stands for constant rate factor, is a constant quality mode offered by x264 to allow users to specify a value that is then mapped to the familiar QP values that H.264 standard uses, while also considering the characteristics of the source video. The “tune PSNR” mode was chosen initially to exclude visual quality tuning from the experiments.

For VP9, we used libvpx[9] and also opted for three different speed (cpu-used) settings: 0 (slowest), 3, and 4, along with 2-pass encoding, constant quality mode, and single-threaded configuration.

For AV1, we used libaom[10] and opted for speed settings: 1 (slower) and 4, along with 2-pass encoding, constant quality mode, and single-threaded configuration.

In addition, we used the “Mount Shasta” HW video encoder that was discussed in section 2.2, to provide HW-encoded H.264 streams.

5.2 Videos

We selected 10 public available videos. They are high quality videos, covering a wide variety of contents, ranging from professionally edited to amateur-produced videos. These are some more information on these videos:

- Resolutions: 1920x1080
- Total length: 38 min 24 sec
- Average number of shots per video sequence: 87.5

In our experiments, they were all downsampled to 8 different resolutions, including the original resolution, and encoded with 6 different QP/CRF values.

5.3 Quality metrics

Video quality metrics are essential in measuring and optimizing performance of video encoders. In video standardization, the default metric is peak signal-to-noise ratio (PSNR), which expresses the amount of pixel difference between the source and encoded video sequence. Although simple to calculate, PSNR has been shown through multiple subjective video quality assessment to correlate rather poorly with human eye's perception of quality.

For that reason, different perceptual full reference objective quality metrics have been developed. Among them, the two most popular and widely used in the video streaming industry are SSIM and VMAF.

SSIM[5] was developed in 2004 by Wang et al. as an image quality metric that captures the degree of luminance, contrast and structure difference between the source and distorted signals. It has been shown through multiple validation tests to correlate very well with human perception and for that reason it has gained popularity for both image and video automated quality assessment. It reports quality in a scale of 0 to 1, with 1 indicating no distortion and is typically applied on video signals by calculating SSIM values per frame and then averaging them along the duration of the video.

Given SSIM's highly non-linear scale, in particular on the top end, a number of logistic functions or other non-linear mappings have been proposed to translate SSIM values into a linear scale. Facebook has developed its own such non-linear mapping and the resulting linear values are called "FB-MOS". More details about FB-MOS are available in.[16]

VMAF stands for Video Multimethod Assessment Fusion and it was developed by Netflix through a research collaboration with USC.[17] It calculates multiple image metrics per frame, together a motion feature, which are then fused into a single score via support vector regression in a scale of 0 to 100, with 100 indicating no distortion. The support vector regression parameters used in VMAF have been obtained through subjective validation with a number of distorted stimuli that include typical compression and scaling artifacts. VMAF has been shown to correlate very highly with human perception (SROCC/PCC values higher than 0.9) for pristine sources and stimuli presented in 1080p displays at lab conditions ($3\times$ display height viewing distance). The entire VMAF implementation has been open-sourced by Netflix and is available.[18]

For our work, we used FB-MOS and VMAF since they are relevant perceptual video quality metrics.

5.4 Encoder with different speed settings

As described earlier, the majority of popular video encoder implementations provide a knob for trading off speed and encoding efficiency. For the first experiment, we utilize that and use the faster setting of the encoder to analyze and determine encoding parameters, which then would be used with the slower setting of the same encoder. In particular, with x264, we analyzed the video sequences with the "fast" and "veryfast" presets, and finally encode with the "veryslow" preset. With libvpx, we analyzed the video sequences with faster speed settings "cpu=3" or "cpu=4", and encode with slowest speed setting "cpu=0".

Table 1 shows the results with different x264 presets. The baseline used here is the conventional fixed CRF encode where all resolutions are encoded with a pre-selected CRF value, shown as the first row. Subsequent rows are the BD-rate w.r.t. FB-MOS and BD-rate w.r.t. VMAF obtained by analysis and encoding done with different speed settings. The row "x264 veryslow → x264 veryslow" represents the ground truth, which is obtained by applying convex hull encoding with the "veryslow" preset, for both the analysis and the final encode step. Followings rows are the results of analyzing with "fast" preset and "veryfast" presets, and encoding performed with "veryslow" preset. Table 2 shows similar results but with the libvpx encoder under different speed settings. For each of the fast selection, the BD-rate difference between the ground truth, or the slowest configuration of convex hull encoding, is also shown in the table.

Table 1. x264 BD-rate compared with "x264 veryslow" fixed CRF

Encoding configuration	FB-MOS	Δ	VMAF	Δ	Analysis Time	Total Time
x264 veryslow (fixed CRF 27)	0%	16.50%	0%	29.71%	-	100%
x264 veryslow → x264 veryslow	-16.50%	0%	-29.71%	0%	100%	600%
x264 fast → x264 veryslow	-16.23%	0.27%	-29.34%	0.38%	37.8%	326.7%
x264 veryfast → x264 veryslow	-15.25%	1.25%	-28.76%	0.96%	21.2%	227.3%

Table 2. libvpx BD-rate compared with libvpx "cpu=0" fixed QP

Encoding configuration	FB-MOS	Δ	VMAF	Δ	Analysis Time	Total Time
libvpx cpu=0 (fixed QP 38)	0%	21.17%	0%	38.27%	-	100%
libvpx cpu=0 → libvpx cpu=0	-21.17%	0%	-38.72%	0%	100%	600%
libvpx cpu=3 → libvpx cpu=0	-21.02%	0.16%	-38.57%	0.15%	21.8%	231.2%
libvpx cpu=4 → libvpx cpu=0	-20.94%	0.24%	-38.57%	0.15%	19.7%	218.4%

To demonstrate how much computation was saved with the proposed technique, we showed the speed differences in terms of the execution time, normalized by that of the slowest configuration, in the last two columns of the table. Both the analysis time and the total (analysis + encode) time are shown. As can been seen, we can achieve less than 1.25% BD-rate difference in terms of FB-MOS and less than 1% in terms of VMAF, with only 20% of computations in analysis. In terms of total time, the fast selection is about 2.3 times of traditional fixed QP/CRF encoding, while the original convex hull encoding would be 6 times as much.

Fig. 5 shows the bitrate vs. quality curves for one of the videos, using x264 and VMAF. It includes both "veryfast" and "veryslow" convex hulls, and the "veryfast → veryslow" projected convex hull. One can see that the projected convex hull matches almost exactly the actual "veryslow" convex hull.

5.5 Encoders within the same family

In the second experiment, we extended the fast selection technique to more than one encoders that are in the same family Specifically, since VP9 and AV1 have very similar quantization schemes, it would not be surprising to see the proposed technique to work across them.

All the final encodes were done with libaom cpu=1, as well as the baseline, which is libaom cpu=1 with fixed QP. Multiple speed settings were used on both libvpx and libaom for the analysis step. The results are shown in Table 3.

We expected larger BD-rate differences when first sequences were analyzed using libvpx and then encoded with libaom, compared to the case where we are using the same encoder in both steps. Surprisingly, the additional drop is very small, less than 1%, for both FB-MOS and VMAF. At the same time, because of the significant speed difference between libvpx and libaom, only approximately 2% of the original complexity is required. In terms of total time that includes both analysis and encode, only about 12% more computations are needed to get almost exactly the same impressive BD-rate gain.

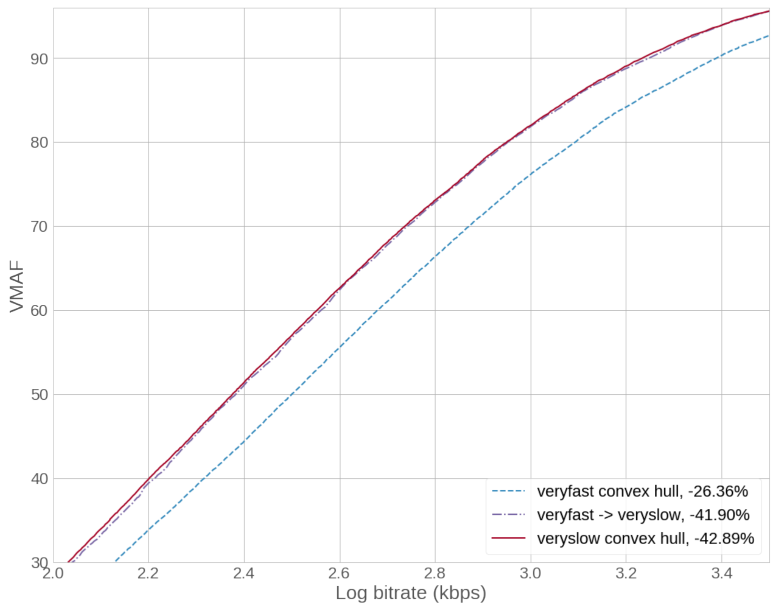


Figure 5. RD curves of x264 with different analysis configuration, with BD-rate against fixed CRF.

Table 3. libvpx and libaom BD-rate compared with libaom "cpu=1" fixed QP

Encoding configuration	FB-MOS	Δ	VMAF	Δ	Analysis Time	Total Time
libaom cpu=1 (fixed QP 38)	0%	25.30%	0%	41.65%	-	100%
libaom cpu=1 → libaom cpu=1	-25.30%	0%	-41.65%	0%	100%	600%
libaom cpu=4 → libaom cpu=1	-25.21%	0.09%	-41.61%	0.04%	20.3%	121.6%
libvpx cpu=0 → libaom cpu=1	-24.40%	0.90%	-41.24%	0.41%	10.8%	165.2%
libvpx cpu=3 → libaom cpu=1	-24.22%	1.08%	-41.06%	0.59%	2.4%	114.3%
libvpx cpu=4 → libaom cpu=1	-24.09%	1.21%	-41.07%	0.58%	2.1%	112.9%

5.6 Encoders with different implementations

Our third experiment used our "Mount Shasta" HW video encoder for the analysis step, which obviously has a different implementation from the software x264 encoder, and performed the final encode with x264 "veryslow" speed setting. The baseline, same as for the previous x264 experiments, is the x264 "veryslow" conventional fixed CRF, and the comparison target was convex hull encoding with both analysis and encoding performed with x264 "veryslow" speed setting. The result is shown in Table 4.

Table 4. HW H.264 BD-rate compared with "x264 veryslow" fixed CRF

	FB-MOS	Δ	VMAF	Δ
x264 veryslow (fixed CRF 27)	0%	16.50%	0%	29.71%
x264 veryslow → x264 veryslow	-16.50%	0%	-29.71%	0%
HW H.264 → x264 veryslow	-13.90%	2.59%	-28.08%	1.63%

5.7 The effect of encoder speed in quality and bitrate

The previous results show that, indeed, for constructing the convex hull for a given shot, encoding parameters from a faster encoder preset match to a large degree the results obtained by a slower preset. To examine the effect of encoder presets and its relationship with resolution and QP further, we perform a statistical analysis of the qualities and bitrates achieved by libvpx and libaom at different presets. We first aggregated qualities and bitrates from all the shots from the 10 videos for each combination of (resolution, QP), calculate the average bitrates and average quality metrics, and then we take the average again of these aggregated values across different (resolution, QP). Finally, we looked at the aggregate changes of bitrate and quality metrics with different encoder speed presets.

In particular, we chose the slowest encoder and speed setting, libaom with cpu=1 to serve as the comparison target, and examined the difference of bits, VMAF, FB-MOS, and PSNR, when switching to different encoder speed settings, including libaom cpu=4, and libvpx with cpu=0, 3, and 4. Table 5 shows the results. We can see that the quality metrics all stay quite the same when using an encoder with different speed settings. On the other hand, bits spent increase noticeably with faster and faster speed settings. This matches the intuition that, given an encoder, the same resolution/QP for a shot produces the same quality, but increases or decreases bitrate, depending on the encoder speed setting.

Table 5. Average bitrate and quality differences with same (resolution, QP), compared to libaom cpu=1

Encoding configuration	Δ bitrate	Δ VMAF	Δ FB-MOS	Δ PSNR
libaom cpu=1	0.00%	0.00	0.00	0.00
libaom cpu=4	4.54%	-0.62	-0.65	-0.09
libvpx cpu=0	11.11%	-1.70	-1.78	-0.26
libvpx cpu=3	20.93%	-2.44	-2.47	-0.39
libvpx cpu=4	23.99%	-2.86	-2.87	-0.45

6. DISCUSSION AND CONCLUSIONS

One can immediately notice that the improvements offered by the per-shot convex hull optimization framework are significant, since we measure average BD-rate improvements in the range 16.5-25.3% when using FB-MOS or 30-40% when using VMAF, over a fixed CRF/QP baseline for the 3 codecs in our study. Yet, those gains normally require a complexity/energy multiplier of about 6$\times$ over the baseline and for that reason, we would be able to offer it only for a very slim slice of our video content.

But, by performing the analysis step using a faster encoder, we notice how we can reduce the analysis time to a much smaller amount, with even the possibility to leverage our HW encoder for that purpose, and thus obtain the optimal encoding parameters at a negligible energy cost. This approximation turns out to be very accurate, since it carries as very small inefficiency, which is less than 2.6% in all cases.

In particular, we notice how we can predict the optimal encoding parameters for libaom cpu=1 by analyzing the content with libvpx cpu=4 encodings. In doing so, we use just 2.1% of the complexity that would otherwise be required, while achieving average BD-rate gains in terms of FB-MOS of 24.09%, a mere 1.2% less than what an implementation that uses libaom cpu=1 for the analysis would achieve. The corresponding BD-rate gains in terms of VMAF for this scenario are even more impressive: 41.07%, just 0.58% less than the maximum achievable gain.

Perhaps the most intriguing result, though, is the one achieved when we analyzed content using HW H.264 encoding to estimate encoding parameters that were then fed into x264 “veryslow”. In this case, the gains achieved (13.9% in terms of FB-MOS BD-rate and 28.08% in terms of VMAF) come at virtually no extra compute/energy cost.

Such compression efficiency gains traditionally required a new generation of video codec, for example rolling out AV1 in place of VP9, and that - as demonstrated before - came with a big increase in compute complexity. We intend to undertake additional experimentation in the near-term, as measured at scale, and hope to report our findings in our future work."

ACKNOWLEDGMENTS

We would like to thank our colleagues within Facebook's Video Infrastructure, and within RTP and CEA team for their help in running all these experiments.

REFERENCES

[1] Bross, B., Chen, J., Liu, S., and Wang, Y.-K., "Versatile Video Coding (Draft 10)," *ITU-T and ISO/IEC JVET-S2001* (2020).

[2] de Rivaz, P. and Haughton, J., "AV1 bitstream & decoding process specification," (2019).

[3] "High efficiency video coding," *ITU-T Rec. H.265 and ISO/IEC 23008-2 (HEVC)* (2013).

[4] Grange, A., de Rivaz, P., and Hunt, J., "VP9 bitstream and decoding process specification," (2016).

[5] Zhou Wang, Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P., "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing* **13**(4), 600–612 (2004).

[6] "VMAF, code repository." link: https://github.com/Netflix/vmaf.

[7] "Advanced video coding for generic audiovisual services," *ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC)* (2003).

[8] "x264, code repository - open-source AVC encoder software." link: https://github.com/mirror/x264.

[9] "libvpx, code repository - open-source VP9 encoder/decoder software." link: https://github.com/webmproject/libvpx.

[10] "libaom, code repository - open-source AV1 encoder/decoder software." link: https://aomedia.googlesource.com/aom.

[11] Lee, K. and Rao, V., "Accelerating facebook's infrastructure with application-specific hardware." https://engineering.fb.com/data-center-engineering/accelerating-infrastructure/ (2019).

[12] "Video @Scale 2019." link: https://atscaleconference.com/events/video-scale-2019/.

[13] Katsavounidis, I., "The NETFLIX tech blog: Dynamic optimizer — a perceptual video encoding optimization framework." https://netflixtechblog.com/dynamic-optimizer-a-perceptual-video-encoding-optimization-framework-e19f1e3a277f (Mar. 2018).

[14] Katsavounidis, I. and Guo, L., "Video codec comparison using the dynamic optimizer framework," in [*Applications of Digital Image Processing XLI*], Tescher, A. G., ed., **10752**, 266 – 281, International Society for Optics and Photonics, SPIE (2018).

[15] Bjontegaard, G., "Calculation of average psnr differences between rd curves," *ITU-T Q.6/SG16 VCEG 13th meeting* (2001).

[16] Regunathan, S., Wang, H., Zhang, Y., Liu, Y., Wolstencroft, D., Reddy, S., Stejerean, C., Gandhi, S., Chen, M., Sethi, P., Puntambekar, A., Coward, M., and Katsavounidis, I., "Efficient measurement of quality at scale in facebook video ecosystem," in [*Applications of Digital Image Processing XLIII*], International Society for Optics and Photonics, SPIE (2020).

[17] Li, Z., Aaron, A., Katsavounidis, I., Moorthy, A., and Manohara, M., "The NETFLIX tech blog: Toward a practical perceptual video quality metric." link: https://netflixtechblog.com/toward-a-practical-perceptual-video-quality-metric-653f208b9652 (2016).

[18] "Vmaf, code repository." link: https://github.com/Netflix/vmaf.