Overview

of the Flexible Encode Infrastructure for High Efficiency Video Codec

October 2018

TABLE OF CONTENTS

1 INTRODUCTION

Intel® Media Server Studio provides rich set of interfaces to build different kinds of video processing applications. In this paper, we discuss one of them, Flexible Encode Infrastructure for High Efficiency Video Codec (HEVC FEI). This interface exposes low-level controls that are intended to improve the performance, visual quality, and flexibility of conventional SDK encoder by utilizing customer IP.

No prior knowledge of SDK or FEI is required to read this paper. The first chapters give a general idea on how an application can tune a hardware-accelerated video encoder and how different controls affect encoding quality and performance. At the same time, this paper can be used as developer's guide to build efficient transcoding pipeline. In the second half of the paper, we discuss a sample application that was specifically written to demonstrate HEVC FEI usage.

This paper starts with an overview of the HEVC FEI architecture and how it is mapped to conventional encoder building blocks. Next, the major part of the paper describes HEVC FEI controls and their impact on performance and visual quality. Then we discuss performance bottlenecks typical for common transcoding scenarios. We finish this paper with a deep discussion of a sample application, providing examples of how to reuse information from the input bitstream to improve visual quality using FEI and how to build efficient bitrate control.

1.1 Acronyms And Abbreviations

BRC	Bitrate control
CTU	Coding tree unit
CU	Coding unit
ENC	First part of the encoding process, including motion estimation and mode decision
EU	Execution unit
HME	Hierarchical motion estimation
HW BRC	Built-in driver BRC, used by conventional encoder
LA BRC	Look ahead bitrate control
ME	Motion estimation
MSE	Mean Squared Error, a measure of distortion between source and encoded frames
MV	Motion vector
MVP	Motion vector predictor
NNZ	Number of non-zero luma transform coefficients, used as visual distortion approximation
PAK	Last part of the encoding process, including motion compensation, transform, quantization, and entropy coding
QP	Quantization parameter
RDO	Rate distortion optimization
SSC	Sum of squared luma transform coefficients, used as a visual distortion approximation
TU1, TU4, TU7	SDK encoder presets:: TU1 quality, TU4 balanced, TU7 speed mode
VQ	visual quality

2 HEVC FEI OVERVIEW

FEI is built on top of the conventional hardware-accelerated encoder and exposes more controls over internal building blocks. These controls facilitate integration of customer algorithms into the encoding process. In addition, FEI provides standalone preprocessing functionality that can be used to gather different kind of statistics about input frame (e.g., frame complexity).

Figure 1 is a diagram of the conventional encoder. The encoding process starts with a hierarchical motion estimation (HME) that is done on the set or downscaled frames. Downscale ratios and the number of layers may differ depending on the original frame resolution and encoder settings, but in most cases at least two stages are present. ME starts on the smallest 16x downscaled frame, and then refines coarse MVs found in the first stage on a 4x downscaled frame. It concludes on the original resolution. After that, the encoder performs an intra-prediction stage to find the best intra-mode, and then makes a mode decision. Other stages are similar to FEI and conventional encoders and are not discussed here.

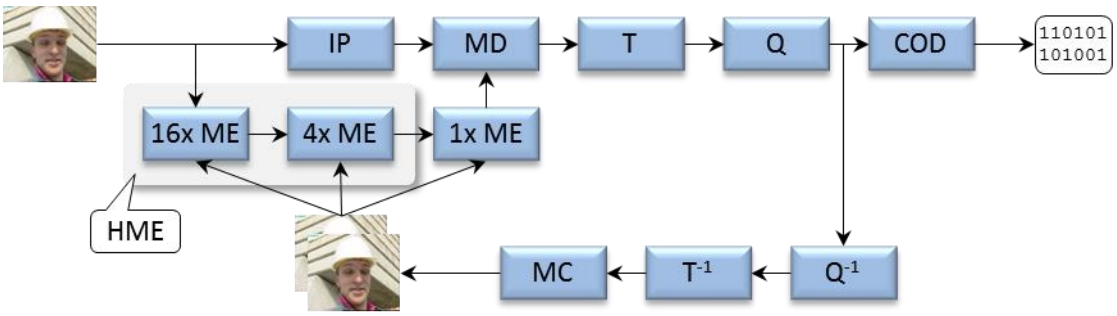


Figure 1. Conventional encoder, where

IP	*Intra prediction*
MD	*Mode decision*
HME	*Hierarchical motion estimation*
ME	*Motion estimation*
T, T^{-1}	*Transform and inverse transform*
Q, Q^{-1}	*Quantization and inverse quantization*
COD	*Entropy coding*
MC	*Motion compensation*

Figure 2 shows how preprocessing works. It uses the same hardware that a conventional encoder uses, but works significantly faster due to the reduced amount of processing. First, HME depth is limited to a single 4x layer followed by ME on the original resolution. Secondly, no dependencies between blocks are taken into account, making ME significantly faster. Finally, no actual mode decision is made here. The best inter-prediction partition is selected and the rest of the encoding stages, like transform and quantization, are completely skipped.

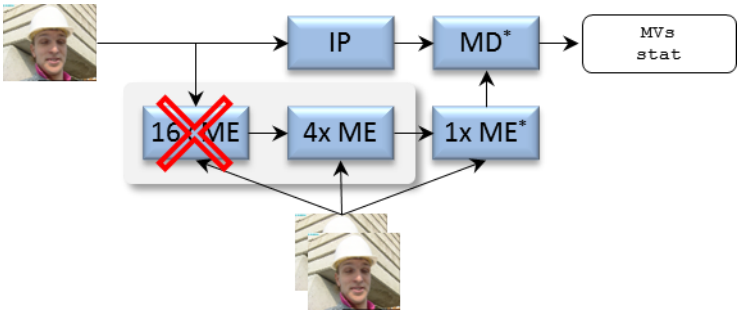


Figure 2. Preprocessing stage, also known as PreENC. MD and ME stages here are faster than in a conventional encoder.

Figure 3 is a diagram of the FEI encoder. In contrast to the conventional encoder, it does not have the HME stage and needs coarse MVs from application to do ME. (See "External MV Predictors" for details.) It also exposes additional controls to fine-tune motion estimation and mode decision. (See "HEVC FEI Controls" for details.)

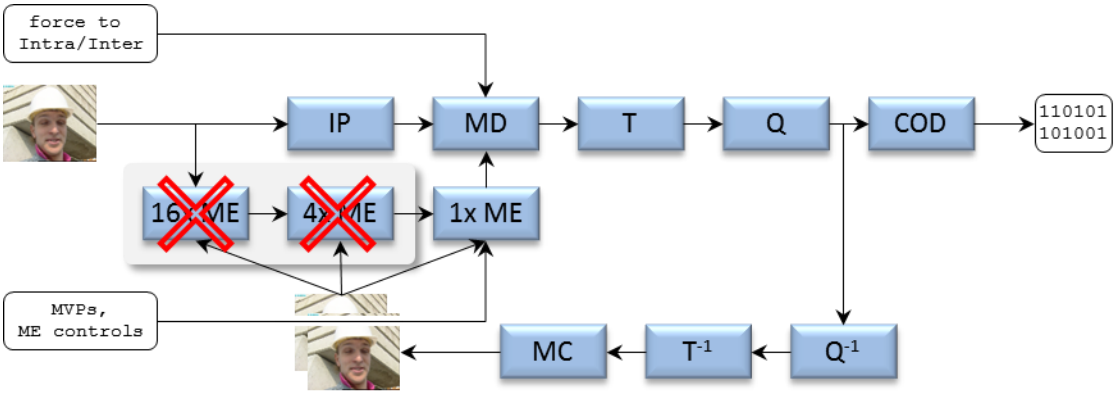


Figure 3. FEI ENCODEr. There is no HME stage. More controls over encoding process are exposed.

Figure 4 shows a typical encoding scenario that uses the PreENC and FEI encoders (also known as the PreENC + ENCODE pipeline). In the first stage of this pipeline, PreENC gathers preliminary statistics about the input frame. Then, it is fed to the customer algorithm that computes optimal control parameters for the FEI encoder. In this stage, actual VQ and performance are improved. Then, the computed parameters are sent to the encoder.

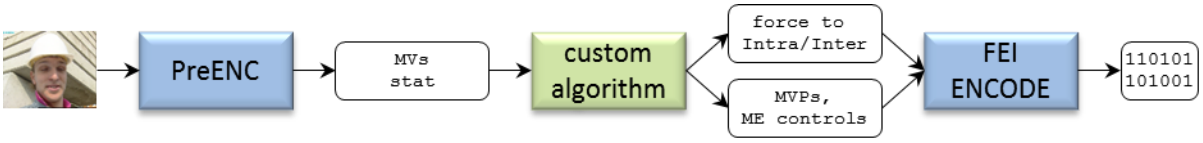


Figure 4. Encoding pipeline, PreENC+ENCODE. PreENC is used as a source of information for the customer algorithm that improves VQ.

For the transcoding use case shown on Figure 5, it is possible to use information from the coded input stream to guide the FEI encoder in the motion estimation and mode decision stages. In this case, statistics about the input frame come not from the PreENC stage, but directly from the decoder. In this

paper, we use term Decode Stream Out (DSO) for this kind of statistics and call this pipeline DSO + ENCODE. (See “Decode Stream Out” for details.)

Figure 5. Transcoding pipeline, DSO+ENCODE. Decode Stream is used as the source of information for the customer algorithm that improves VQ.

3 HEVC FEI CONTROLS

In this chapter, we discuss different controls over the encoding process exposed by the HEVC FEI interface. This description augments the HEVC FEI Reference Manual by illustrating the impact of each control on visual quality and performance. It also gives some recommendations on how to use these controls to achieve different goals like runtime performance/quality tradeoff.

BD rate is used as a visual quality metric. If not specified otherwise, “balanced” (TU4), a preset of the legacy encoder, is used as a base point for BD rate calculation. To avoid bitrate control influence, measurements were done in constant QP mode. Four-point RD curves were built based on the average PSNR, which included luma and chroma values. A positive BD rate means that HEVC FEI quality is better than conventional transcoding/encoding.

Density was selected as a performance metric. In this context, it represents the number of simultaneous transcoding that can run on the same system in real time. A fractional part in density means that last channel could not achieve real time. For example, if the density is 3.5, then three channels run at 30fps and one runs at 15 fps.

We evaluated 25 full HD streams with different content, including fast and slow motion, static content, and highly irregular scenes like waves on the water. (See a detailed description in Appendix E. Stream information.”) In addition, nine artificial streams were used that were generated from original content by applying a fade in/out affect every 25, 50, and 100 frames. Note that during encoding evaluation, no special tools (like weighted prediction) were used. Encoder settings were exactly the same for all streams in the test pool.

Two pipelines were used:

1. Conventional encoding from the YUV source to HEVC bitstream, based on PreENC followed by the ENCODE pipeline (see the “HEVC FEI Overview” chapter for details).
2. Transcoding from the AVC or HEVC source to the HEVC bitstream, based on the ENCODE pipeline with additional control parameters computed from DSO (see the “Sample Application” chapter for details). The AVC source bitstream was encoded by x264 encoder, the HEVC source bitstream and the auxiliary bitstream for DSO were encoded by HM16.17 encoder. To match the reference lists between the HM and FEI encoders, we made minor modifications to HM. We used 40Mbps source streams and 5Mbps auxiliary stream.

We used a single GOP pattern, IbBbP, where the P and middle B frames were used as reference. (The full test configuration is described in Appendix C. System configuration” and in Appendix D. Command Lines.”)

3.1 EXTERNAL MV PREDICTORS

The motion estimation part of a GPU-accelerated encoder has quite a limited search range. To catch long MVs, the encoder has to start a search in a reference frame, not from a collocated block, offsetting it by a motion vector predictor (MVP). A conventional encoder uses HME to find such predictors, but the FEI encoder relies on the application to provide them. From zero to four MVPs can be provided for each

block of 16x16 pixels. Each MVP specifies the location of the search window in the reference frame and reference frame index.

(See the “HEVC FEI Overview” chapter for details about HME, the “Search Window Size” chapter for supported search window sizes, and the “Internal MV Predictors” chapter for a list of additional internal MVPs.)

MVPs can be used to achieve different goals.

- Improve objective VQ. A mode decision made in a conventional encoder may be suboptimal. For example, an inter-prediction with a big MV difference may be selected instead of the more efficient intra-prediction or encoder may choose an inter-prediction from less efficient in the RDO sense location. In these cases, by providing a “good” MVP, application can prevent the encoder from choosing inefficient MVs. (See the “Force CTU To Intra/Inter” chapter for details how to force encoder to use specific prediction type.)
- Improve performance. By using MVPs, the application can control how much time the encoder spends on motion estimation. A conventional encoder estimates motion for several predictors for each reference frame. The FEI encoder runs a motion estimation only for provided MVPs, and the application can limit the number of searches, leading to the single best MVP.
- Improve subjective VQ. A conventional encoder does not use chroma channels during motion estimation, which in some cases may lead to visible color artifacts. Application can guide the FEI encoder with proper MVP to prevent such artifacts. MVP can also be useful for low-bitrate cases to keep the MV field even in areas with uniform textures. That, in turn, helps to improve subjective VQ.

Figure 6 shows how MVP affects performance and quality for different pipelines. Three pipelines are shown:

1. HEVC HW is conventional encoding. Its “balanced” TU4 preset is chosen as a reference point (i.e., zero BD rate). As we can see from the chart, in comparison to the ”balanced” preset “speed” one increases density from 3 to 12 channels and degrades VQ by 25%.
2. PreENC+HEVC FEI. In this mode, PreENC is run on four times downsampled frames, then the FEI encoder is called with MVPs from the first stage. This is the slowest mode due to PreENC overhead, and it has the closest to conventional encoder VQ because no additional IP is present in this pipeline—just a simple MVP repack from PreENC to ENCODE layout.
3. DSO+HEVC FEI. This is the most interesting case. Here, we calculate MVPs from DSO data extracted from auxiliary stream and feed them to FEI encoder (see the “Decode Stream Out” for details). Because the mode decision here has been made by the RDO-based encoder, we have better VQ than conventional encoder for both “balanced” and “speed” presets. For “speed” mode in this pipeline, the FEI encoder is significantly slower than a conventional encoder. This is because DSO processing is not optimized for performance, runs on CPU, and becomes a bottleneck for high-density cases.

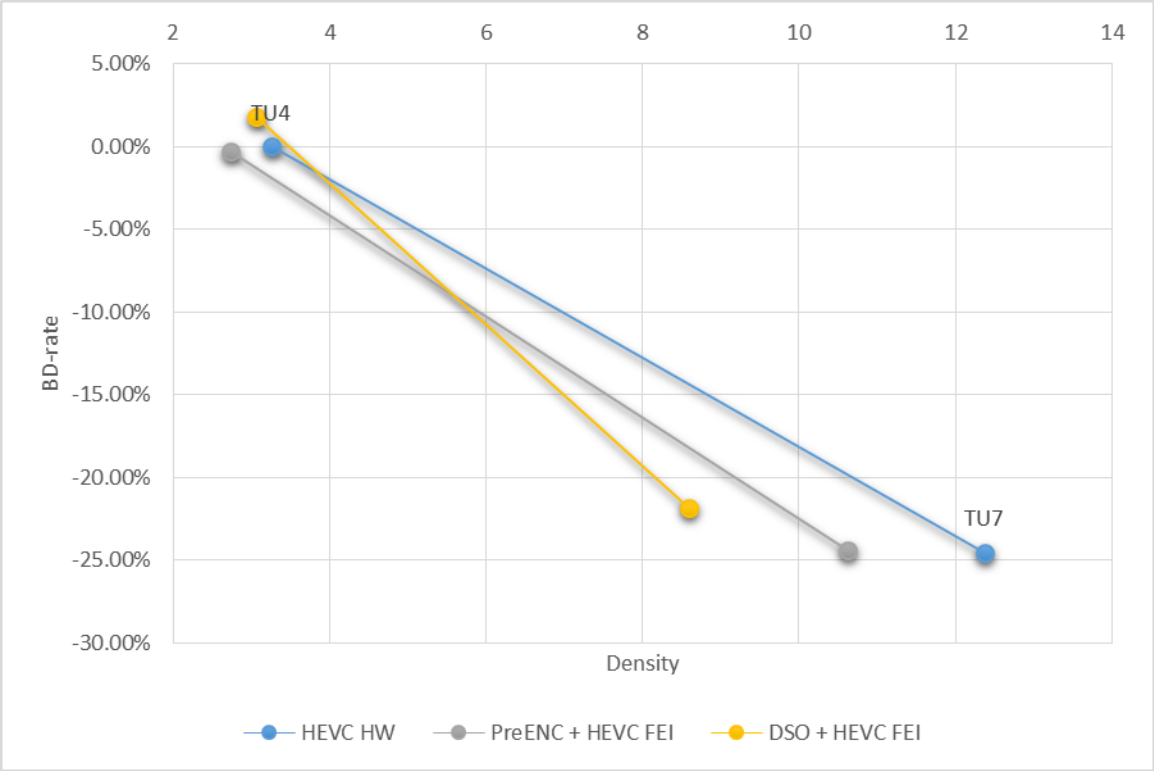


Figure 6. Impact of MVPs on VQ and density for different pipelines. Blue – conventional encoder, gray – FEI with MVPs from PreENC, yellow – FEI encoder with MVPs from DSO. Higher BD rate and density is better.

In addition to these pipelines, Figure 7 shows the HEVC FEI encoder without MVPs. This is the fastest mode, faster than conventional encoder by about 0.5 channel on TU4 and 1 channel on TU7, because the FEI encoder does less motion estimation here, just estimating internal MVPs, with no external MVPs and no HME stage. It also has the lowest visual quality due to significant degradation on high-motion streams where the encoder could not catch actual motion. Still, this mode can be used for slow-motion content.

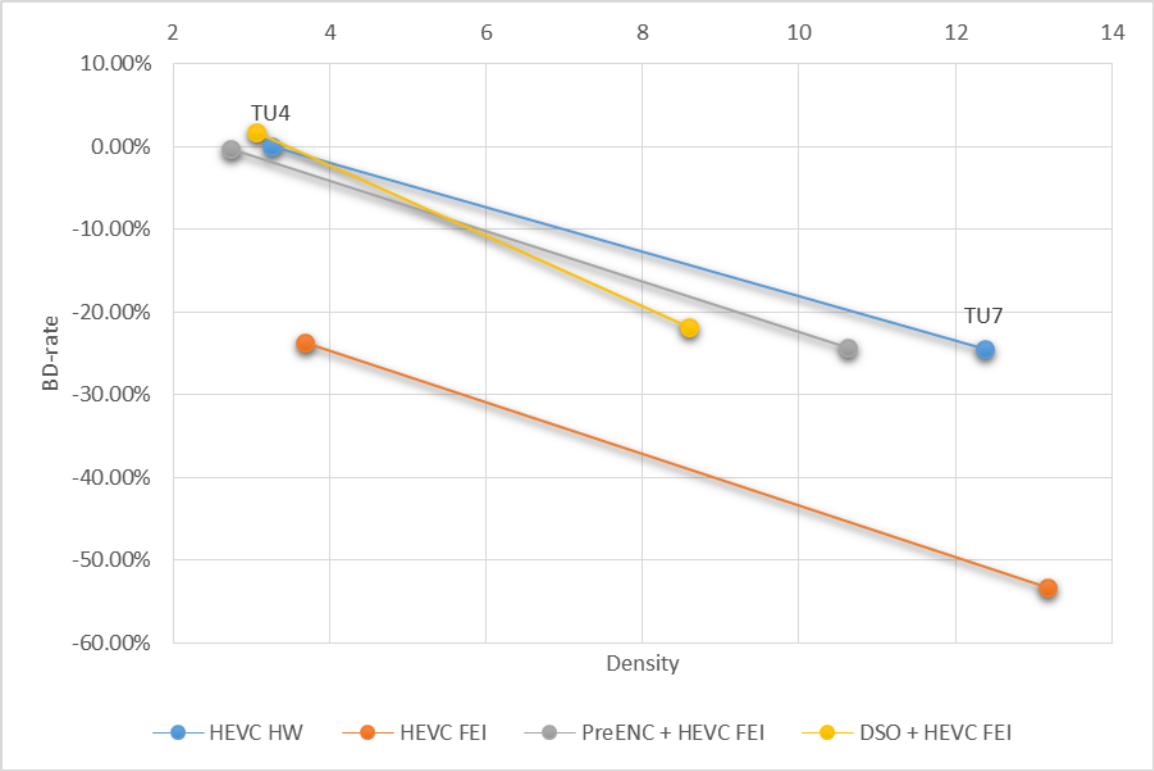


Figure 7. Impact of MVPs on VQ and density for different pipelines. Orange line is FEI encoder without MVPs.

Target Usage	Pipeline	Density	BD Rate
4	HEVC HW	3.26	0 %
	HEVC FEI	3.68	-23.64 %
	PreENC + HEVC FEI	2.74	-0.27 %
	DSO + HEVC FEI	3.06	1.76 %
7	HEVC HW	12.37	-24.53 %
	HEVC FEI	13.18	-53.26 %
	PreENC + HEVC FEI	10.62	-24.36 %
	DSO + HEVC FEI	8.59	-21.80 %

Table 1 Impact of MVPs on VQ and density for different pipelines.

Table 2 shows the BD rate gain/loss for each stream from the test pool for the three pipelines described above, PreENC+ENCODE, DSO+ENCODE, and ENCODE without MVPs. Each pipeline was measured for two presets, “balanced” TU4 and “speed” TU7. As we can see, the PreENC + ENCODE pipeline has no VQ gain in comparison to conventional transcoding, but DSO has. It is especially noticeable on streams with high motion like “Jockey,” streams with a fade in/out effect. It is less, but still noticeable, on streams with irregular content like “riverbed.” ENCODE without MVP shows acceptable visual quality on slow-motion streams like “Beauty” and “ducks_take_off,” but huge degradation on fast-motion streams like “Jockey.” “HoneyBee” with fades looks very interesting. FEI ENCODE without MVP has better VQ than VQ of the conventional encoder. This is because the encoder tends to use long and arbitrary motion vectors on such streams. Reducing the search range excludes such long MVs from consideration and the

encoder uses intra-prediction instead of inter-prediction with long MVs (see the chapter “Force CTU To Intra/Inter” for a discussion of how intra/inter prediction impacts VQ).

	TU4			TU7		
Sequence	**PreENC**	**DSO**	**No MVP**	**PreENC**	**DSO**	**No MVP**
Beauty	-0.16%	0.28%	-0.65%	-0.26%	-0.95%	-3.08%
big_buck_bunny	-0.24%	-0.30%	-6.32%	-0.41%	-0.67%	-6.65%
blue_sky	0.11%	0.32%	-60.22%	-0.14%	0.05%	-91.08%
Bosphorus	0.00%	0.15%	-19.28%	-0.48%	-0.26%	-20.55%
bq_terrace	0.48%	0.12%	-7.09%	0.11%	0.11%	-7.47%
crowd_run	-0.04%	-0.01%	-3.44%	0.20%	0.15%	-3.80%
ducks_take_off	-0.04%	-0.26%	-0.03%	-0.07%	-0.28%	-0.07%
elephants_dream	-0.21%	-0.29%	-2.18%	-0.39%	-0.49%	-2.95%
HoneyBee	-0.20%	-0.26%	-0.75%	-0.11%	-0.34%	-0.70%
in_to_tree	0.01%	-0.06%	-8.25%	0.26%	0.13%	-9.09%
Jockey	0.99%	10.72%	-95.50%	-0.92%	9.22%	-89.41%
kimono1	-0.09%	0.20%	-12.13%	-0.13%	-0.19%	-15.32%
old_town_cross	0.07%	0.02%	-0.49%	0.13%	0.06%	-0.79%
park_joy	-0.02%	0.16%	-6.48%	0.03%	0.25%	-7.78%
park_scene	-0.31%	-0.30%	-5.72%	-0.08%	-0.36%	-6.84%
pedestrian_area	-0.27%	0.08%	-12.46%	-0.64%	-1.43%	-15.83%
ReadySteadyGo	-0.87%	2.62%	-99.95%	-0.64%	1.61%	-97.30%
riverbed	0.16%	2.22%	1.79%	-0.39%	1.50%	1.51%
rush_hour	-0.27%	-0.94%	-8.62%	-0.38%	-1.01%	-9.91%
station2	-0.05%	-0.01%	-12.47%	-0.14%	-0.11%	-14.93%
sunflower	0.11%	0.33%	-30.12%	-0.02%	-0.15%	-32.72%
tears-of-steel	-0.62%	1.23%	-20.32%	-1.26%	0.74%	-22.10%
TouchDownPass	0.41%	0.44%	-29.69%	0.07%	-0.07%	-31.29%
tractor	-0.47%	0.98%	-95.22%	-0.32%	0.91%	-109.68%
YachtRide	-0.60%	-0.43%	-18.88%	-0.46%	-0.63%	-20.60%
HoneyBee_fade100	-0.82%	1.83%	1.86%	-2.81%	3.83%	4.49%
HoneyBee_fade50	-0.19%	5.88%	5.28%	-3.06%	9.58%	9.27%
HoneyBee_fade25	-0.13%	7.06%	6.15%	-3.04%	11.99%	12.14%
Jockey_fade100	-0.04%	8.09%	-46.99%	-1.89%	8.12%	-41.24%
Jockey_fade50	-0.74%	6.64%	-42.13%	-2.73%	8.48%	-34.19%
Jockey_fade25	-1.70%	6.14%	-27.29%	-3.75%	8.25%	-19.48%
tractor_fade100	-0.37%	1.49%	-69.13%	-0.87%	1.75%	-77.15%
tractor_fade50	-1.24%	2.77%	-45.34%	-2.02%	3.99%	-45.49%
tractor_fade25	-1.79%	3.09%	-31.59%	-3.05%	4.53%	-29.86%
Average	-0.27%	1.76%	-23.64%	-0.87%	2.01%	-24.70%

Table 2. BD-rates for PreENC+ENCODE, DSO+ENCODE and ENCODE only pipelines. For TU7 BD rate is calculated relative to TU7 preset of conventional encoder. Color bars that show VQ difference have the same scale for PreENC and DSO, but different for ENCODE only.

(For an API description look for `mfxExtFeiHevcEncMVPredictors` in “Reference Manual for HEVC FEI.”)

3.2 Internal MV Predictors

Apart from external MV predictors, FEI encoder also uses MV predictors derived from already encoded neighbor blocks. For streams with slow or smooth motion, that may be enough to provide good visual quality for the encoded stream, even without external predictors. On streams with high motion or with poor MVPs, it helps to improve visual quality.

Internal MVPs are calculated for the same block size as external ones (i.e., 16x16 pixels). Three neighbor blocks are used, left, top, and top right. They may belong to the current or a neighbor CTU. First, two predictors for reference frames with zero index are calculated. The first predictor is the median from three neighbors, according to the AVC standard. The second predictor is one of the neighbor MVs that is farthest from the median. Then two more predictors are calculated for the reference frames with indexes 1 and 2 from reference list L0. Note that the FEI encoder supports only three reference frames in L0 and one in L1 list, so these MVPs cover all reference frames.

If neighbor MVs are unavailable (for example, because block is located on the frame boundary or reference index of neighbor does not match to current reference), then zero MV is used. The encoder doesn't check the same MV predictors twice, so if no neighbor is available, only one zero MVP is estimated.

If the application can provide good MVPs, then the recommendation is to disable internal MVPs to speed up encoding. Disabling these internal MVPs has no impact on skip and merge modes. They will still be estimated.

Figure 8 and Figure 9 show the VQ and performance impact of this control. As we can see, disabling internal MVPs for the PreENC + ENCODE pipeline increases performance but decreases visual quality, because PreENC MVPs are not good enough. For the DSO case, the result is different. Disabling of internal predictors improves both VQ and performance. It is because the reference encoder makes better mode decisions. Also, limiting the number of MVPs to search decreases the probability for the FEI encoder to make a suboptimal decision.

Note that disabling both internal and external MVPs drastically decreases VQ and should never be used.

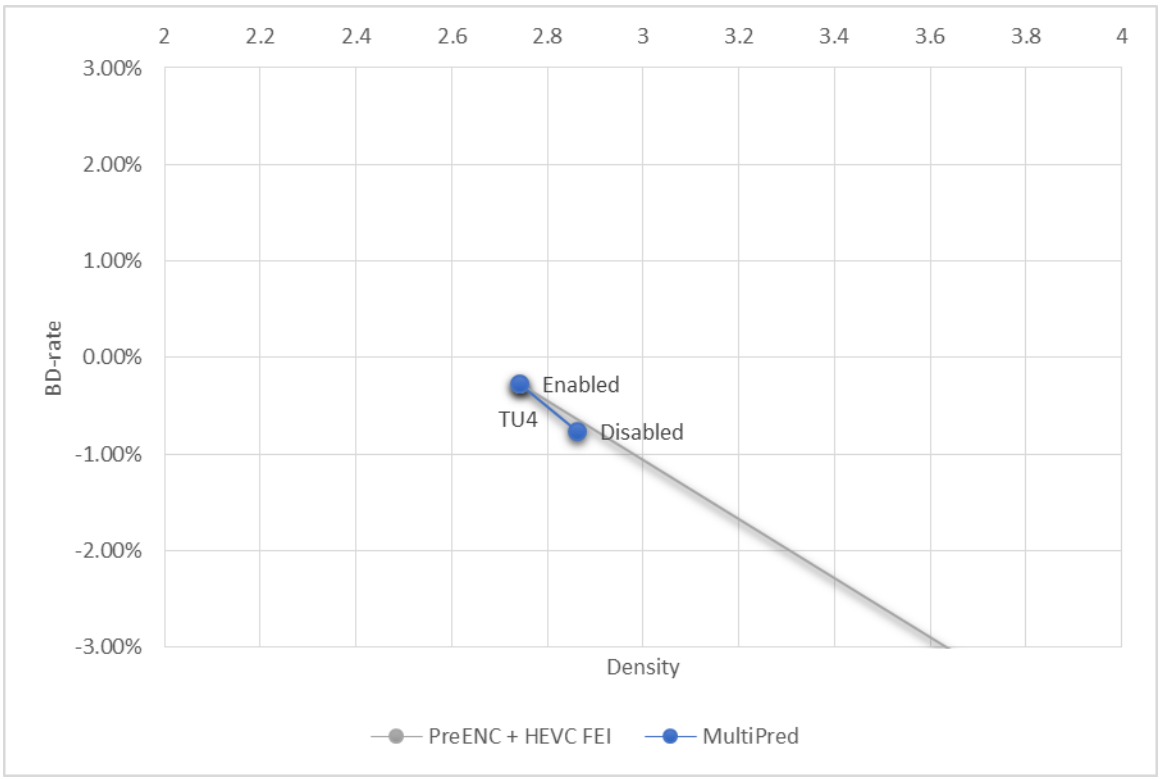


Figure 8. Impact of internal MVP control on VQ and density for PreENC + ENCODE pipeline

Internal MVPs	Density	BD Rate
Enabled	2.74	-0.27 %
Disabled	2.86	-0.76 %

Table 3. Impact of internal MVP control on VQ and density for PreENC + ENCODE pipeline

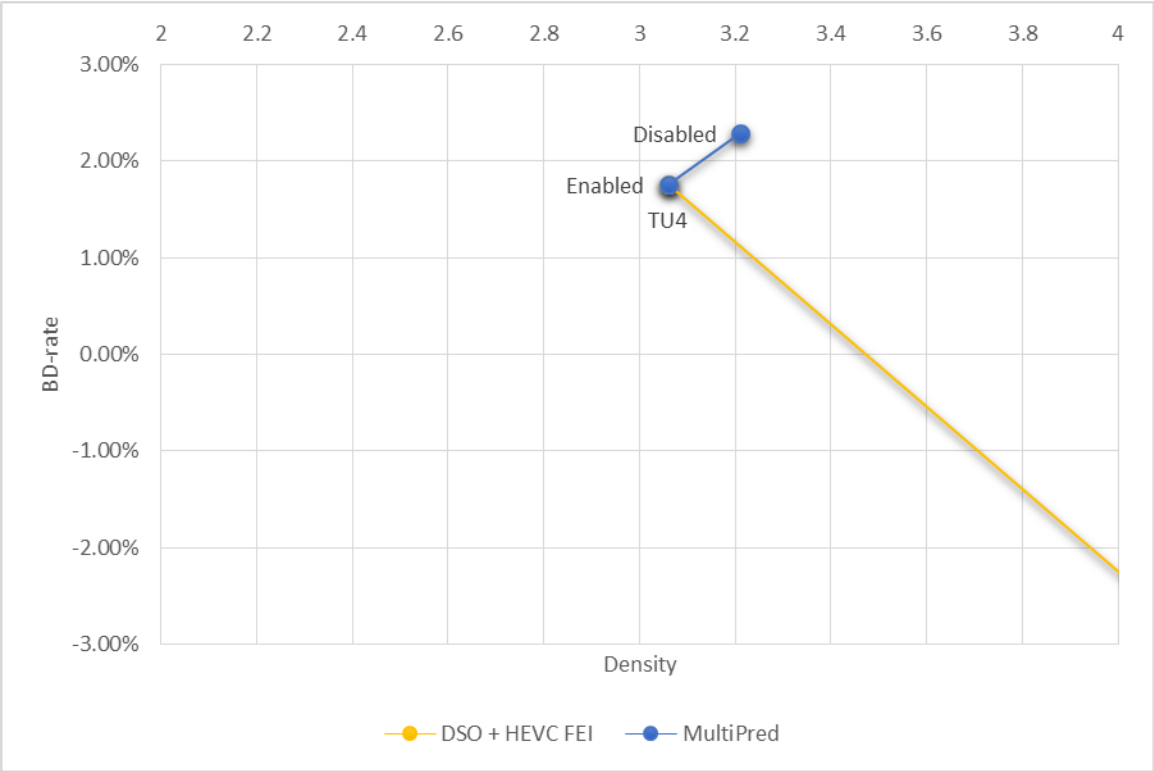


Figure 9. Impact of internal MVP control on VQ and density for DSO + ENCODE pipeline

Internal MVPs	Density	BD Rate
Enabled	3.06	1.76 %
Disabled	3.21	2.29 %

Table 4. Impact of internal MVP control on VQ and density for DSO + ENCODE pipeline

For API description look for `mfxExtFeiHevcEncFrameCtrl::MultiPredL0` and `MultiPredL1` in "Reference Manual for HEVC FEI".

3.3 FORCE CTU TO INTRA/INTER

This control is intended for use cases where the application has additional information about the encoded frame from which the prediction type may be deduced. For example, in the case of a screen capture, the CTU belonging to the just-appearing window will probably be efficiently coded as intra and unchanged background as inter.

This control can improve both visual quality and performance. VQ gain comes from making an optimal mode decision and performance gain comes from a reduced number of prediction types to be estimated.

The VQ and performance impact of this control is shown in Figure 10 and in Table 6. We used the DSO + ENCODE pipeline for this measurement. Force flag values were deduced from DSO data by the next algorithm:

- If all CUs in the current CTU of a 5Mb auxiliary stream are intra-coded, then the force to Intra flag was set.
- If all CUs in the current CTU were inter coded, then the force to Inter flag was set.
- Otherwise, both flags were reset and the mode decision was left to the encoder.

It is also possible to use a statistic from PreENC to deduce the prediction type, but we did not try this.

This algorithm works only if the target bitrate is close to the bitrate of the auxiliary stream. Because of this, we used three target bitrates: 4Mbps, 6MBps, and 8Mbps. To eliminate the bitrate control impact on the results, all encoding was done in constant QP mode. Different QP values were selected for different streams to make the bitrate as close to the target bitrate as possible.

As can we can from Figure 10 and Table 6, this very simple algorithm gives a noticeable VQ gain for the force to intra flag case. We especially achieved a big gain on streams with irregular content like “riverbed,” where the encoder tends to use inter prediction with long and arbitrary motion vectors. Good gain was also achieved on streams with fades. The faster the fade, the more significant the gain achieved. This is for the same reasons as the previous case. The encoder tends to use inter prediction in cases where intra is more efficient.

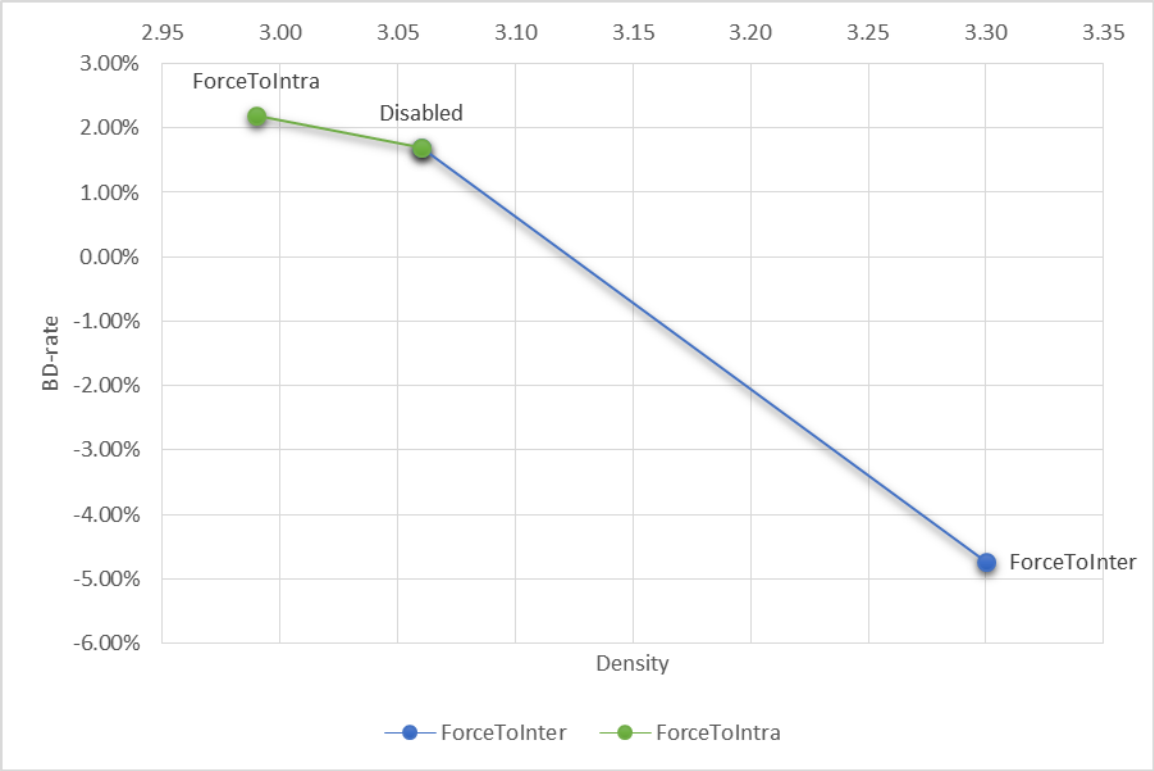


Figure 10. Impact of force to intra and inter flags on VQ and density for DSO + ENCODE pipeline

Force Flags	Density	BD Rate
No force flags	3.06	1.70 %
Force CTU to Intra	2.99	2.20 %
Force CTU to Inter	3.30	-4.73 %

Table 5. Impact of force to intra and inter flags on VQ and density for DSO + ENCODE pipeline

Sequence	Disabled ctrls	ForceToIntra	ForceToInter
Beauty	0.56%	0.90%	-3.98%
big_buck_bunny	-1.21%	-1.01%	-2.05%
blue_sky	0.11%	0.01%	-0.82%
Bosphorus	-0.09%	0.25%	-2.10%
bq_terrace	0.27%	0.27%	0.40%
crowd_run	0.32%	0.21%	0.22%
ducks_take_off	0.02%	-0.51%	-3.76%
elephants_dream	-0.36%	-0.59%	-2.56%
HoneyBee	-0.07%	-0.09%	0.35%
in_to_tree	-0.20%	-1.56%	0.25%
Jockey	10.20%	11.02%	5.37%
kimono1	-0.12%	-0.41%	-3.48%
old_town_cross	-0.19%	-0.48%	0.69%
park_joy	0.73%	0.80%	0.45%
park_scene	-0.24%	-0.28%	-0.34%
pedestrian_area	0.25%	-0.21%	-2.09%
ReadySteadyGo	2.52%	2.11%	2.51%
riverbed	3.30%	7.79%	0.75%
rush_hour	-0.30%	-0.53%	-2.60%
station2	-0.10%	-0.17%	-0.87%
sunflower	1.08%	0.93%	0.46%
tears-of-steel	0.37%	-0.18%	-1.26%
TouchDownPass	0.69%	1.01%	0.34%
tractor	0.76%	0.62%	-0.47%
YachtRide	-0.37%	-0.85%	-0.44%
HoneyBee_fade100	2.63%	2.79%	-11.36%
HoneyBee_fade50	4.91%	4.92%	-46.07%
HoneyBee_fade25	5.17%	5.16%	-86.43%
Jockey_fade100	8.86%	12.67%	3.62%
Jockey_fade50	6.89%	11.54%	-0.40%
Jockey_fade25	6.26%	11.21%	2.05%
tractor_fade100	1.02%	1.17%	-0.65%
tractor_fade50	1.97%	3.14%	-4.03%
tractor_fade25	2.01%	3.06%	-2.66%
Average	1.70%	2.20%	-4.73%

Table 6. BD-rates for DSO + ENCODE pipeline, with enabled force to intra and inter flags compared to conventional transcoding

On the other hand, this algorithm does not work for force to inter flag. In most cases, VQ became worse and on “HoneyBee,” with fades, the VQ degradation was disastrous. Analysis of the encoded streams showed that for some CTUs, the FEI encoder could not find an efficient inter prediction mode. For example, the reference encoder may use 32x32 partition, but FEI four 16x16 partitions, the reference encoder may use bidirectional prediction, but FEI may use unidirectional, and the MV itself can be suboptimal in length and direction. That is especially noticeable on streams with fades, where MVs often point not to the similar texture, but to the region with similar luminosity. All of these lead to costly inter predictions. And the FEI encoder chooses intra prediction for such CTUs and gets good overall VQ. However, if force to inter flag is on, then the encoder is forced to use a suboptimal mode and loses visual quality. Figure 11 shows the reference encoder decision, 32x32 partition with correct MVs pointing to the proper location in the reference frame, a very efficient mode. In Figure 12, the same CTU is shown. The FEI encoder could not find a good inter prediction and switched to intra instead. The result

is not as good as the inter prediction on Figure 11, but still acceptable. Figure 13 shows force to inter case, where the encoder is forced to use a bad decision, leading to a disastrous VQ loss.

Performance gain comes from reducing the amount of work the encoder does during the mode decision. For forced to inter CTUs, intra estimation is completely skipped (i.e., the total number of modes the encoder checks decreases). Because the relative number of inter CTUs is high, the performance gain is also significant.

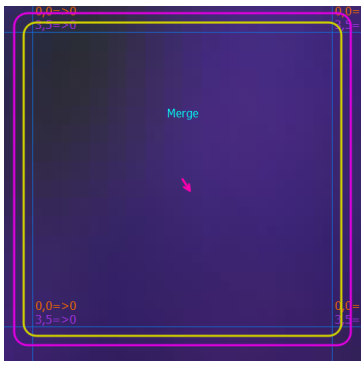


Figure 11. Reference encoder, CTU 367

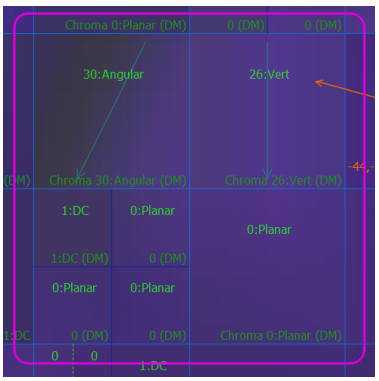


Figure 12. FEI encoder, force flags are OFF, same CTU

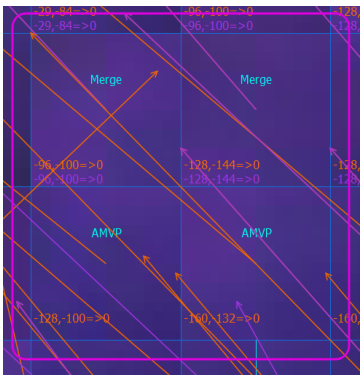


Figure 13. FEI encoder, force to inter flag is ON, same CTU

Note that the conclusion above does not mean that force to intra flag can be used only for VQ and force to inter only for performance improvement, or that the FEI encoder has suboptimal motion estimation. It just illustrates that these flags affect both VQ and performance and that the algorithm described above is not good enough. A more complicated algorithm is required to get both VQ and performance gains.

(For an API description, see `mfxFeiHevcEncCtuCtrl::ForceToIntra/ForceToInter` in "Reference Manual for HEVC FEI.")

3.4 Number of Frame Partitions

The HEVC FEI encoder shares compute resources with the 3D pipeline and, on a system with a high number of EUs, performance is usually limited by a lack of parallelism in processed frames rather than a lack of resources. (See also the “Performance Bottlenecks” chapter.) To mitigate this lack of parallelism, the encoder divides the frame into several regions and processes them in parallel. Note that these regions do not relate to tiles or slices defined by the HEVC standard. This is an encoder-specific notion and, apart from this control, it is not visible to the application in any way.

The FEI encoder supports from one to eight regions. The more regions are used, the more speed-up can be achieved. As we can see from Figure 14 and Figure 15, changing the number of regions from one to eight increases performance by 4x and decreases VQ by 4% for the PreENC pipeline and 3% for DSO. The default number of regions for the FEI encoder is four.

This is frame-level control. Application can turn it on and off depending on the frame type, current system workload, or any other criteria. (See also the “Performance/VQ Tradeoff” chapter.)

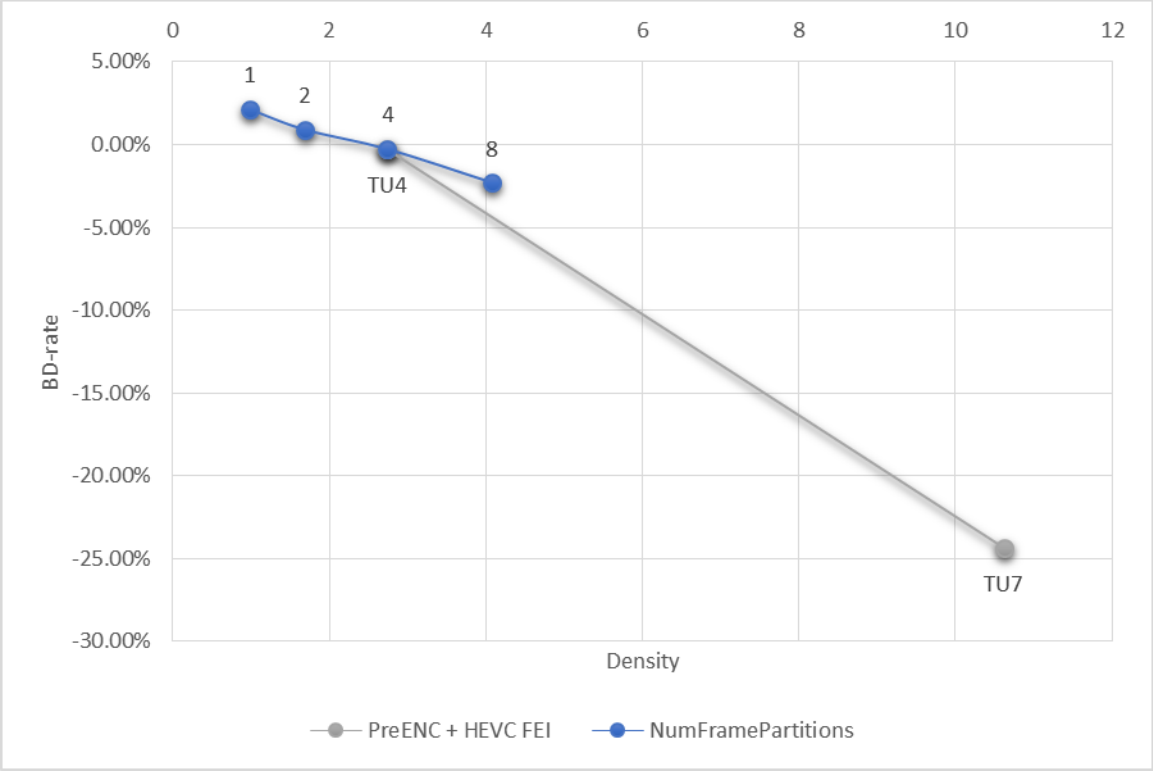


Figure 14. Impact of number of frame partitions on VQ and density for PreENC + ENCODE pipeline

Number of Frame Partitions	Density	BD Rate
1	0.98	2.13%
2	1.69	0.90%
4	2.74	-0.27%
8	4.07	-2.29%

Table 7. Impact of number of frame partitions on VQ and density for PreENC + ENCODE pipeline

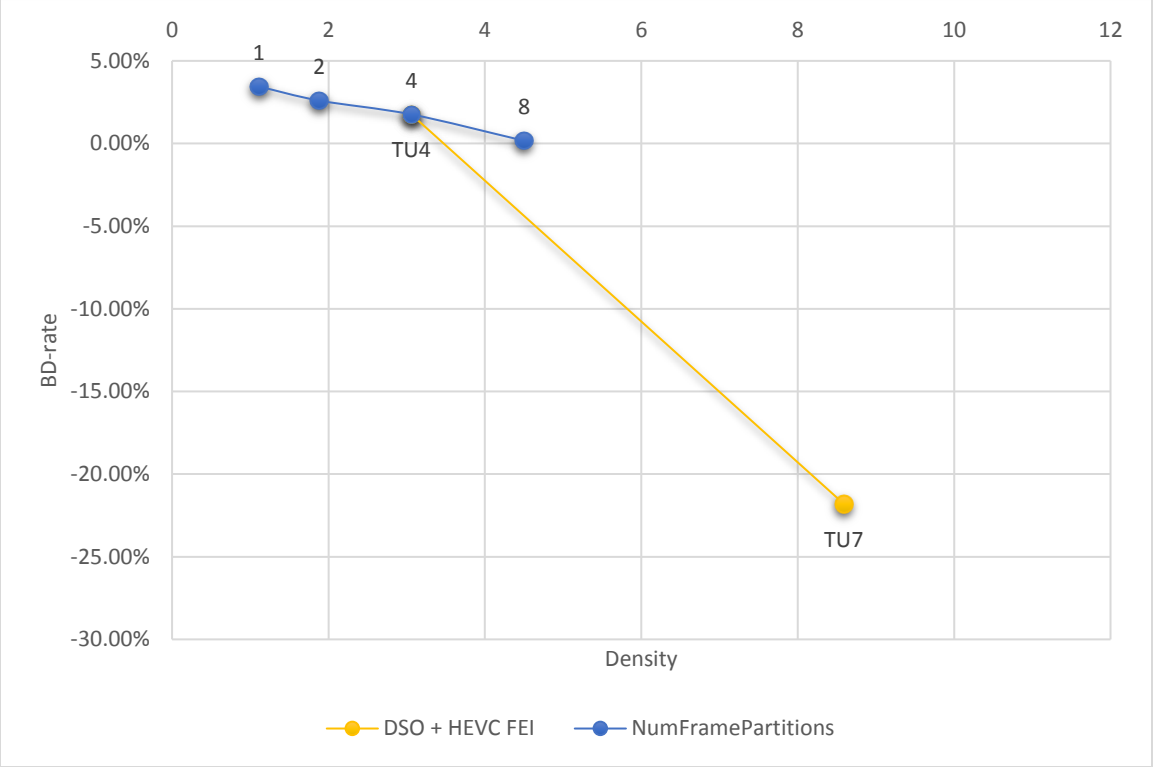


Figure 15. Impact of number of frame partitions on VQ and density for DSO + ENCODE pipeline

Number of Frame Partitions	Density	BD Rate
1	1.11	3.45%
2	1.88	2.60%
4	3.06	1.76%
8	4.50	0.17%

Table 8. Impact of number of frame partitions on VQ and density for DSO + ENCODE pipeline

(For an API description look for `mfxExtFeiHevcEncFrameCtrl::NumFramePartitions` in "Reference Manual for HEVC FEI.")

3.5 FORCE CTU SPLIT

This is another control that is intended to enable more parallelism during encoding. It is similar to Number of Frame Partitions, but instead of dividing the frame, it divides each 32x32 CTU into four CUs. That means that in an encoded bitstream, there will be no CUs larger than 16x16. Each CTU will be divided at least one time. This division creates more opportunities for parallel processing. Encoding of the 16x16 CU may start before the complete neighbor 32x32 CTU has been encoded. Another performance improvement comes from a reduced number of possible partitions to check. The 32x32 mode is not evaluated in this case.

Figure 16 and Figure 17 show the VQ and performance impact of this control.

This is frame-level control. The application can turn it on and off depending on the frame type or current system workload, or any other criteria. (See also the "Performance/VQ Tradeoff" chapter.) Note that this control has no effect on I frames.

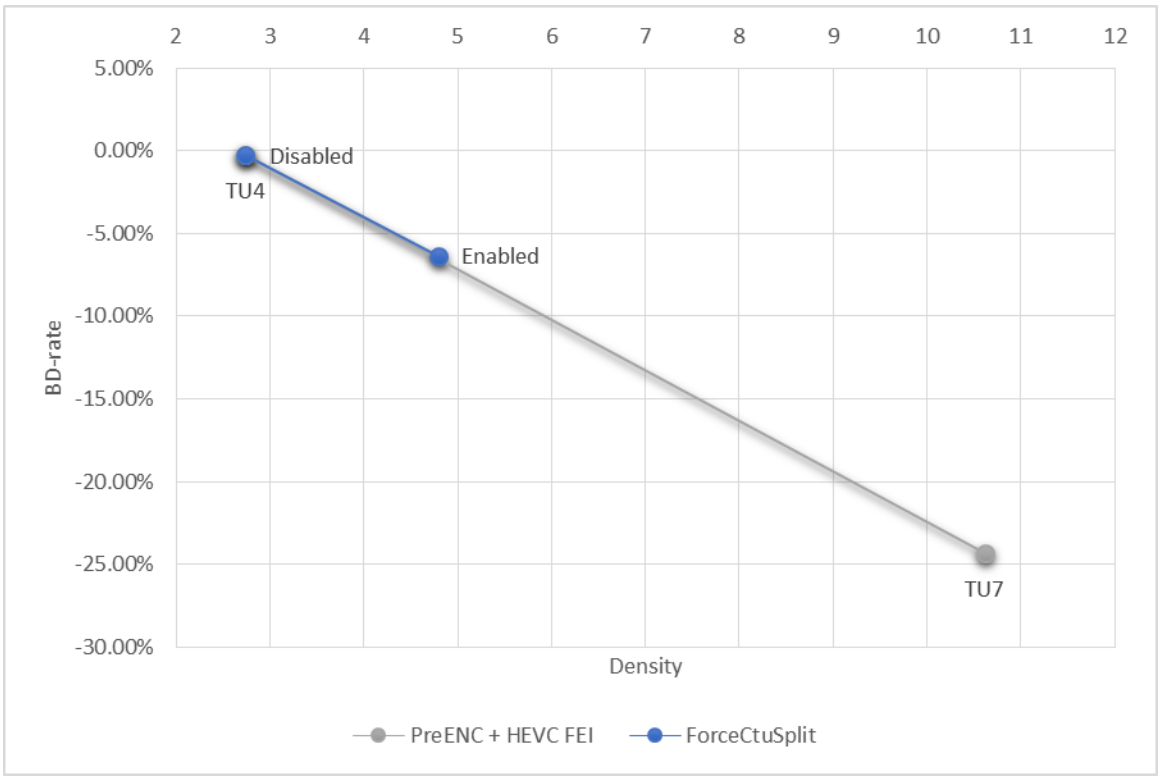


Figure 16. Impact of force CTU split on VQ and density for PreENC + ENCODE pipeline

Force CTU Split	Density	BD Rate
Disabled	2.74	-0.27 %
Enabled	4.79	-6.35 %

Table 9. Impact of force CTU split on VQ and density for PreENC + ENCODE pipeline

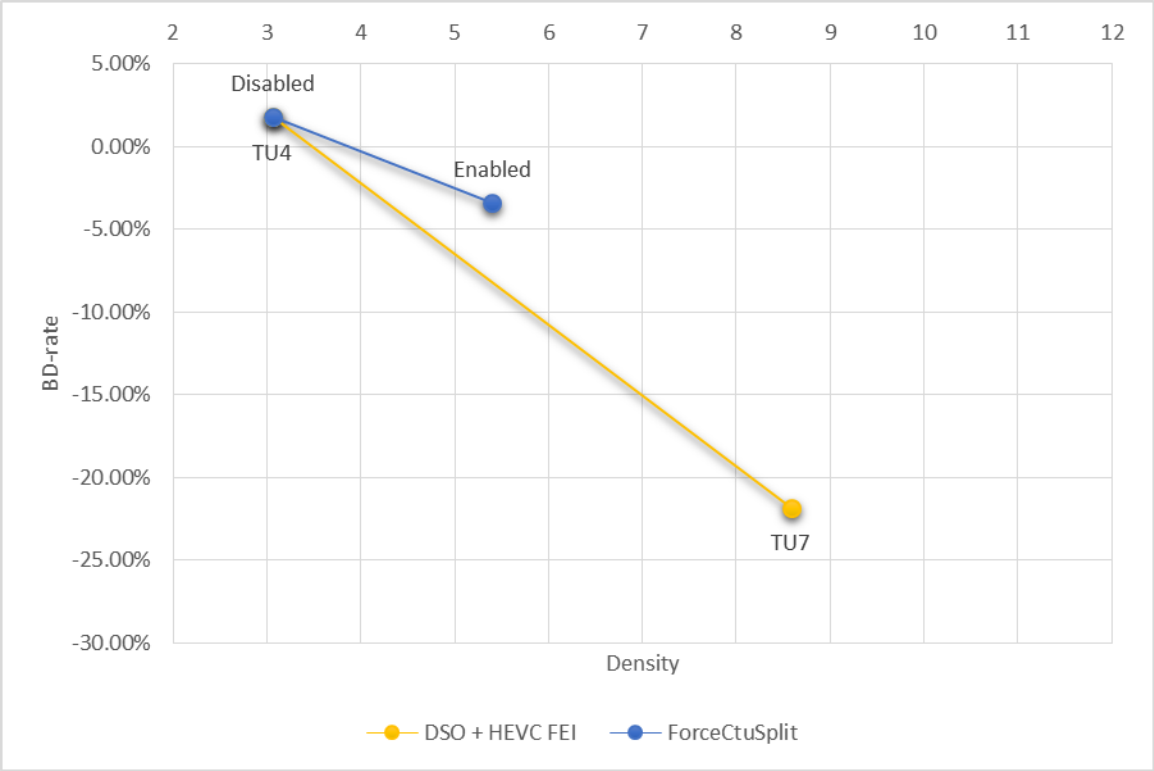


Figure 17. Impact of force CTU split on VQ and density for DSO + ENCODE pipeline

Force CTU Split	Density	BD Rate
Disabled	3.06	1.76 %
Enabled	5.40	-3.42 %

Table 10. Impact of force CTU split on VQ and density for DSO + ENCODE pipeline

(For an API description, look for `mfxExtFeiHevcEncFrameCtrl::ForceCtuSplit` in the Reference Manual for HEVC FEI.)

3.6 Fast Intra Mode

As follows from the name, this flag can be used to select different Intra prediction algorithms in the encoder. The FEI encoder does intra prediction in two steps. In the first step, nine AVC-specific intra prediction modes are estimated using fixed-function hardware. Then, additional refinement is done on EUs to select one of the 35 HEVC-specific intra prediction modes. If this flag is set, then the encoder skips the second step and just maps the AVC mode chosen in the first step to the corresponding HEVC mode without any additional refinement. That significantly improves performance, but obviously harms VQ.

Figure 18 and Figure 19 show the VQ and performance impact of Fast Intra Mode control on the encoding process. As we can see, this is very powerful control, with significant impacts on both VQ and performance.

This is frame level control. The application can turn it on and off depending on the frame type, current system workload, or any other criteria. (See also the "Performance/VQ Tradeoff" chapter.)

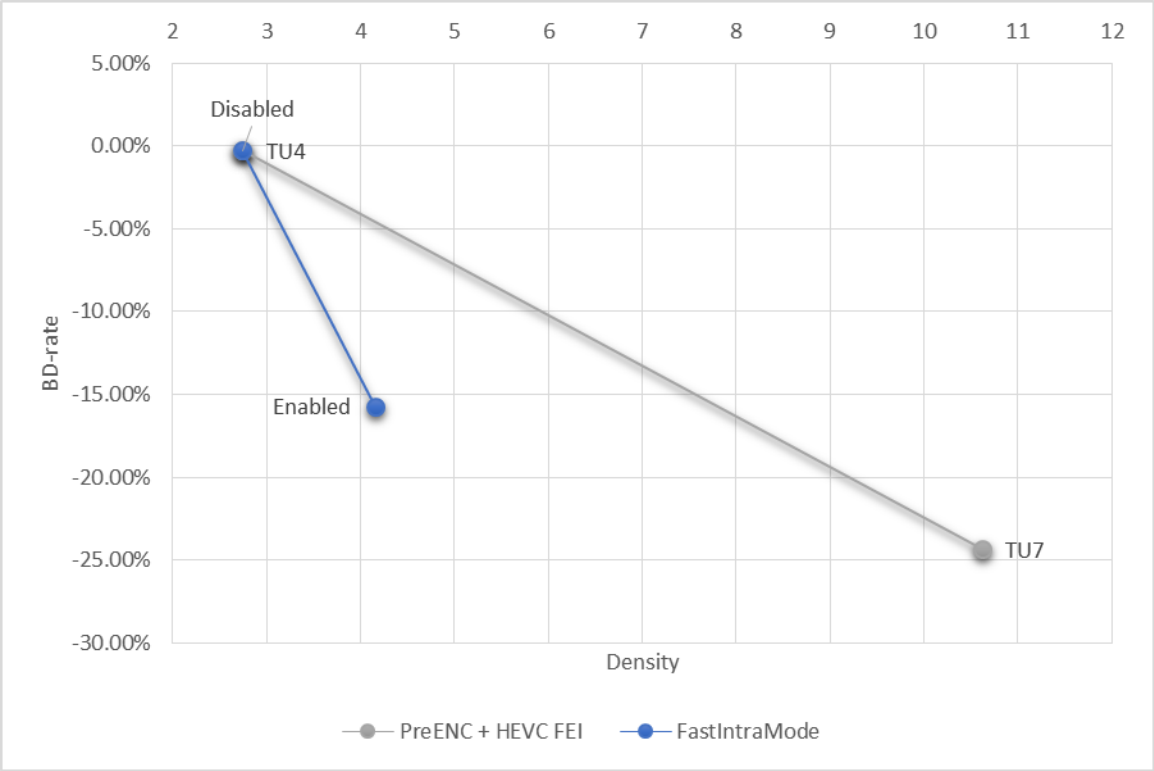


Figure 18. Impact of fast intra mode control on VQ and density for PreENC + ENCODE pipeline

Fast Intra Mode	Density	BD Rate
Disabled	2.74	-0.27 %
Enabled	4.16	-15.70 %

Table 11. Impact of fast intra mode control on VQ and density for PreENC + ENCODE pipeline

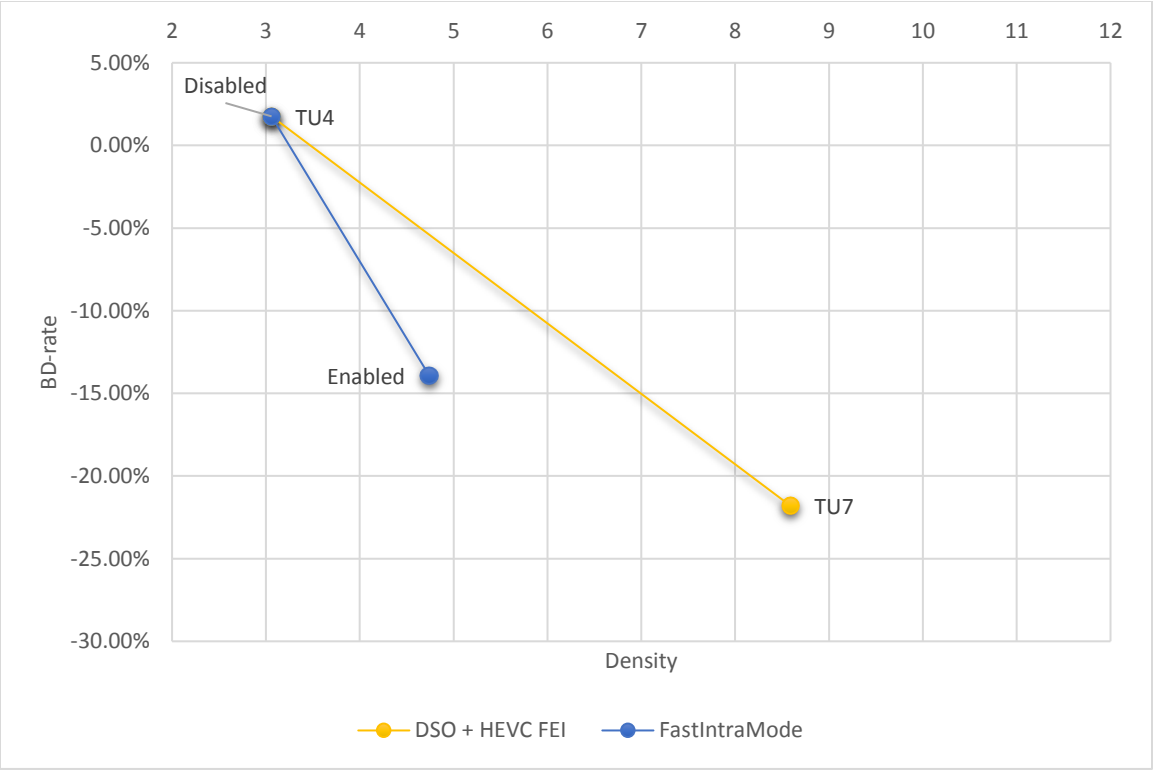


Figure 19. Impact of fast intra mode control on VQ and density for DSO + ENCODE pipeline

fast intra mode	density	BD rate
disabled	3.06	1.76 %
enabled	4.74	-13.92 %

Table 12. Impact of fast intra mode control on VQ and density for DSO + ENCODE pipeline

(For an API description, look for `mfxExtFeiHevcEncFrameCtrl:: FastIntraMode` in "Reference Manual for HEVC FEI.")

3.7 Motion Estimation Controls

In this chapter, we discuss several controls related to the motion estimation process.

3.7.1 Search Window Size

The FEI encoder does motion estimation in a rectangular area of a reference frame. The size of this area is specified via its horizontal and vertical dimensions by the `RefWidth` and `RefHeight` parameters.

During motion estimation, the FEI encoder compares a source block of pixels to a target block of exactly the same size. This target block is located completely inside the search window. Because of this, the total number of possible search locations is lower than the total number of pixels inside the search window (`RefWidth` * `RefHeight`) and depends on the source block size as shown in Figure 20. The area covered by these locations will be denoted as a "reference region" of the search window.

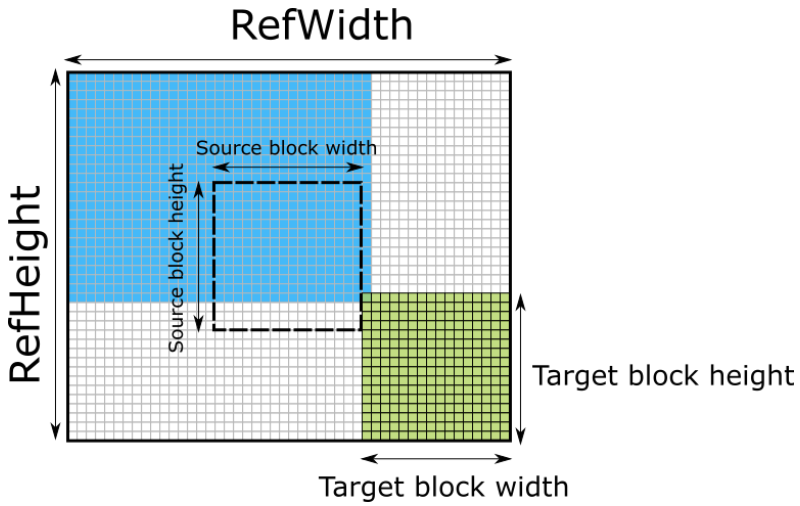


Figure 20. Search window for RefWidth=48, RefHeight=40 and 16x16 source block. The target block has the same size as the source block. One of the target block locations is highlighted in green and all possible locations for the top-left corner of the target blocks are highlighted in blue ("reference region").

Supported values of both the `RefWidth` and `RefHeight` parameters are 20, 24, …, 60, 64 for uni-directional motion estimation and 20, 24, …, 28, 32 for bi-directional motion estimation. Also, the `RefWidth * RefHeight` must be less than or equal to 2,048 for uni-directional motion estimation and less than or equal to 1,024 for bi-directional motion estimation.

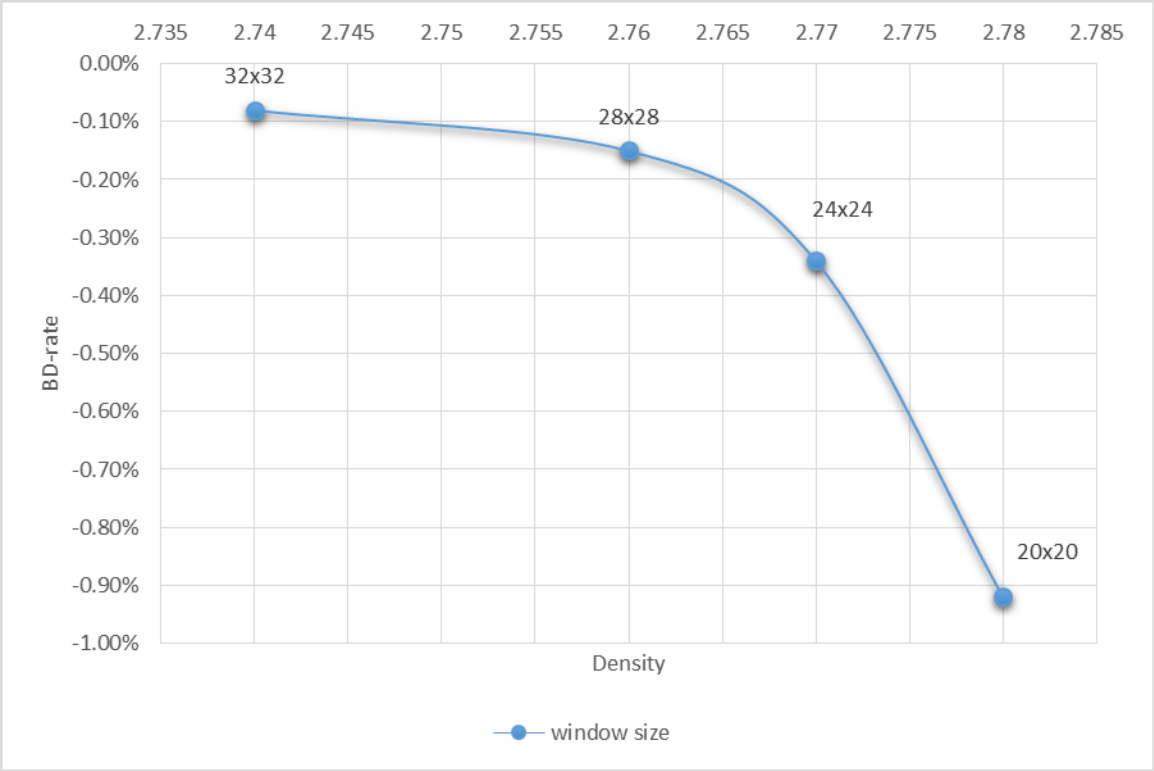


Figure 21. Impact of search window size on VQ and density for PreENC + ENCODE pipeline

Window Size	Density	BD Rate
20x20	2.78	-0.92 %
24x24	2.77	-0.34 %
28x28	2.76	-0.15 %
32x32	2.74	-0.08 %

Table 13. Impact of search window size on VQ and density for PreENC + ENCODE pipeline

(For an API description look for `mfxExtFeiHevcEncFrameCtrl::RefWidth` and `RefHeight` in Reference Manual for HEVC FEI.)

3.7.2 Search Path

For better utilization of hardware, motion estimation is performed on several search locations at once through so-called search units (SUs) (Figure 22). The size of each SU depends on the size of the source block of pixels that has to be estimated. 4x4 search locations are used for 16x16 source blocks, 8x8 for 8x8 source blocks, and 8x4 for 16x8 and 8x16 source blocks.

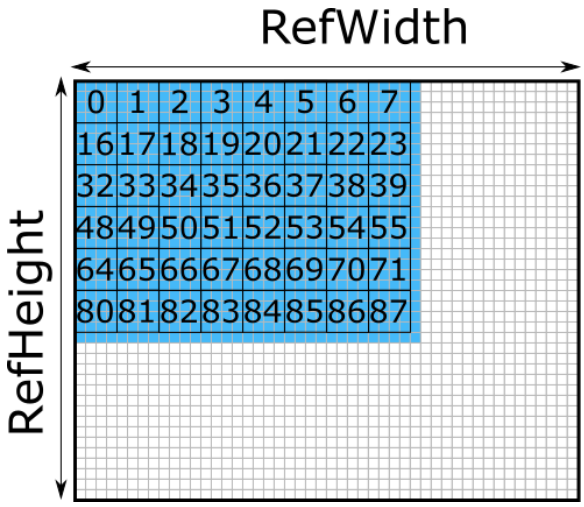


Figure 22. Search units for RefWidth=48, RefHeight=40 and 16x16 source block. Search units have a size of 4x4 search locations each. Each SU is assigned an index. Only locations inside the numbered SUs will be used during motion estimation as top left corners of the target block.

During motion estimation, the encoder goes through search units one by one, following predefined search path. Two are supported for now ("diamond" and "full"). See Figure 23. The application can also specify the number of SUs in the path and the length of the search path. For bi-directional prediction, the search path length specifies the sum of the searched SUs in both search windows. Supported values for search path length are 1, 2, …, 62, and 63 for uni-directional search and 2, 3, … , 63 for bi-directional search.

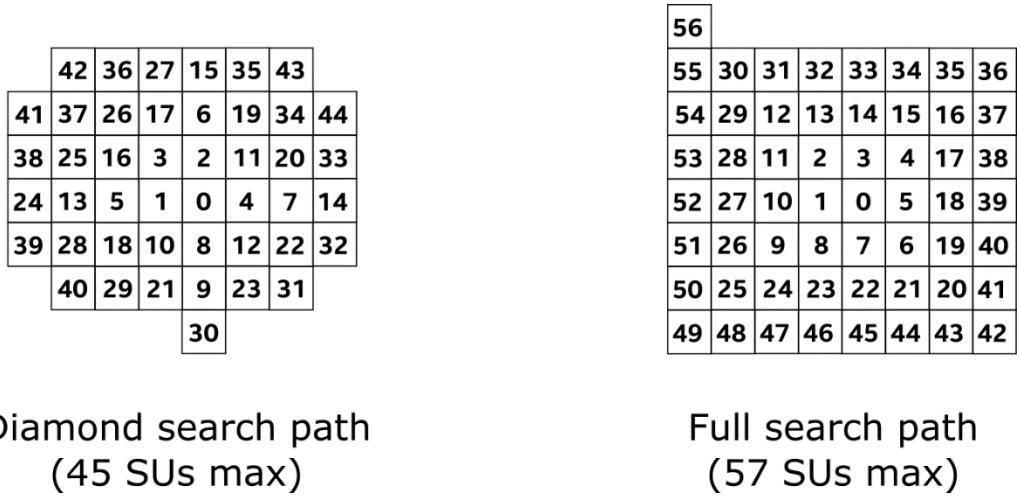


Figure 23. "Diamond" and "Full" search paths. Each square represents single SU. Indices represents order of evaluation during motion estimation. Note, that actual SUs shape depends on source block dimension and may be rectangular, not square.

Both “diamond” and “full” search paths start from the center of the reference region and cover most of the SUs. The major difference is the order of evaluation. It becomes important for fast search algorithms that have short search paths. In this case, the diamond path covers a wider area but skips some SUs in comparison to the full path. See Figure 24 and Figure 25.

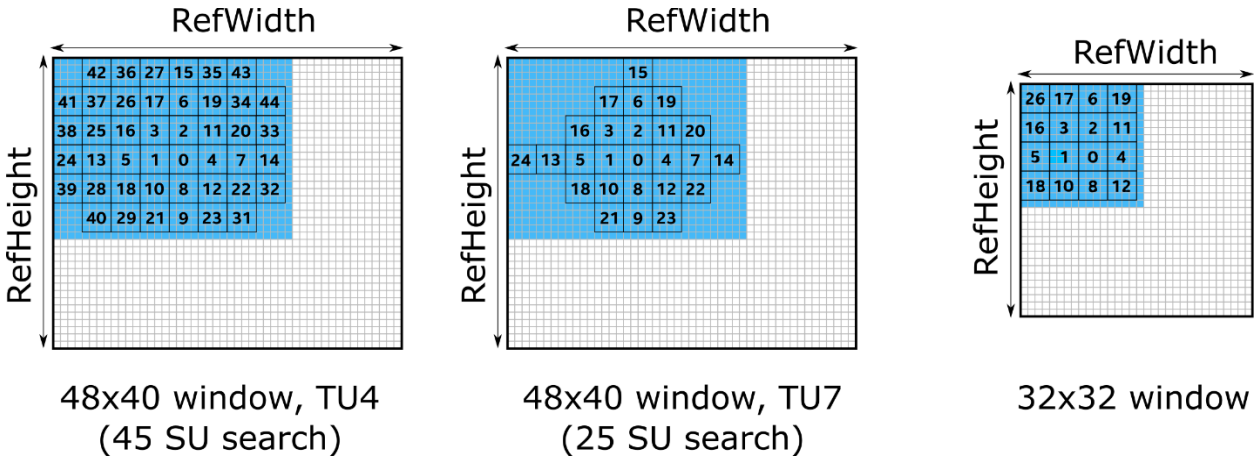


Figure 24. Diamond search path for different search window sizes and different search path lengths. SUs that fall outside the reference region (blue area) are not shown.

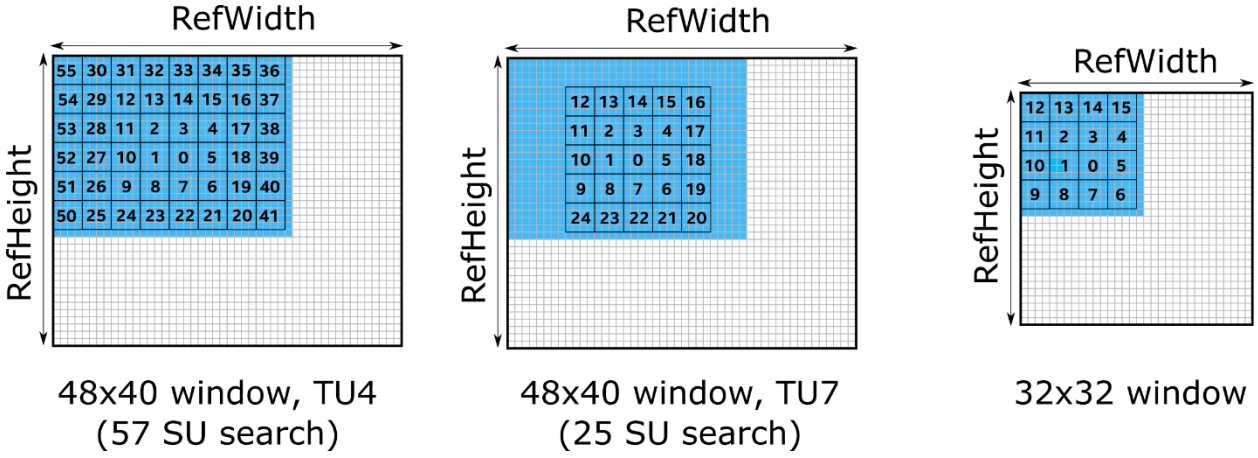


Figure 25. Full search path for different search window sizes and different search path lengths. SUs that fall outside the reference region (blue area) are not shown.

Usually, this control has a negligible impact on VQ and performance (Table 14 and Table 15). The only exception is very short search paths, where VQ slightly drops. Note that on some content, VQ drops on short search paths may be significantly higher than average, as shown in Table 14.

Search Path Length	Density	BD Rate, Average	BD Rate, Worst Case
2	2.77	-0.72%	-2.85%
8	2.77	-0.16%	-0.41%
16	2.76	-0.12%	-0.03%
24	2.74	-0.11%	-0.04%
32	2.74	-0.12%	-0.04%
40	2.74	-0.12%	-0.04%
48	2.74	-0.12%	-0.04%
57	2.73	-0.12%	-0.04%

Table 14. Impact of search path length on VQ and density for PreENC + ENCODE pipeline for 32x32 search window

Search Path Shape	Density	BD Rate
Full	2.74	-0.12%
Diamond	2.74	-0.12%

Table 15. Impact of search path shape on VQ and density for PreENC + ENCODE pipeline for 32x32 search window

(For an API description look for `mfxExtFeiHevcEncFrameCtrl::SearchPath` and `LenSP` in Reference Manual for HEVC FEI.)

3.7.3 Adaptive Search

Adaptive search is a variation of the standard gradient descent search. If it is disabled, then the encoder evaluates all SUs from the search path specified by the application and chooses the best location. If it is enabled, then after searching all SUs from the search path specified by the application, the encoder checks the best-found location. If it falls on the SU boundary, then the encoder continues to search by evaluating neighboring SUs of the best location found so far. The encoder continues this search until the best location falls inside the SU or all neighboring locations have been searched or the search window boundary is reached. (See Figure 26.)

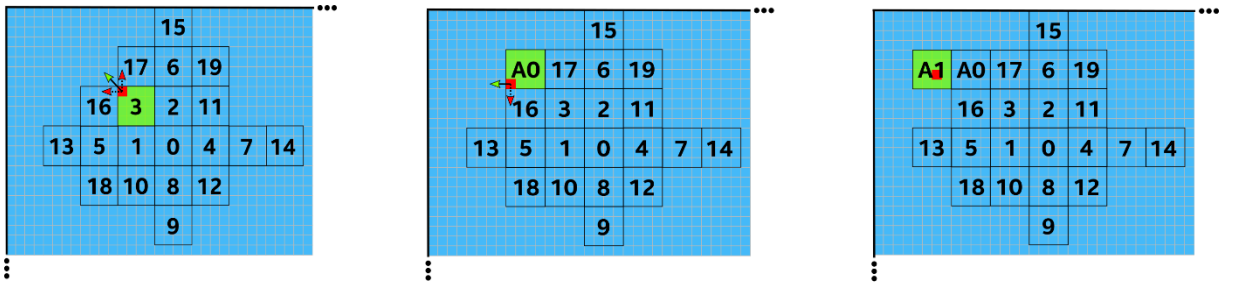


Figure 26. Adaptive search for search path length equal to 20. At step 1 20 SUs have been searched and the best pixel location (red square) has been found at the top-left corner of SU #3. Then SU #17 has been considered as the next to be searched, but rejected because it has been searched already. SU #16 has been eliminated by the same reason. So the search continues to the only remaining option SU #A0. After processing SU #A0, the best ME location once again falls onto its boundary and another adaptive search step has been taken. Finally, after searching SU #A1 the best ME location does not fall onto its boundary and the adaptive search ends.

Note that the total number of SUs searched during any type of search, including bi-directional, cannot exceed 63. For example, if the search path length has been set to 61 and adaptive search has been enabled, then at most two additional SUs will be searched as part of the adaptive search after searching the first 61 SUs.

Usually, this control has negligible impact on VQ and performance. (See Table 16.)

Target Usage	Adaptive Search	Density	BD Rate
4	Disabled	2.74	-0.30%
	Enabled	2.74	-0.27%
7	Disabled	10.63	-24.39%
	Enabled	10.62	-24.36%

Table 16. Impact of adaptive search control on VQ and density for PreENC + ENCODE pipeline

For the API description, look for `mfxExtFeiHevcEncFrameCtrl::AdaptiveSearch` in "Reference Manual for HEVC FEI."

3.7.4 Search Presets

The application can directly specify motion estimation parameters as described in previous chapters, or use one of the six presets showed in Table 17. The impact of these presets on VQ and performance is shown in Figure 27.

Preset Number	Name	Search Window Size	Search Path Shape	Search Path Length
1	Tiny	24x24	Diamond	4
2	Small	28x28	Diamond	9
3	Diamond	48x40	Diamond	16
4	Large diamond	48x40	Diamond	32
5	Exhaustive	48x40	Full	48

Table 17. Presets for motion estimation.

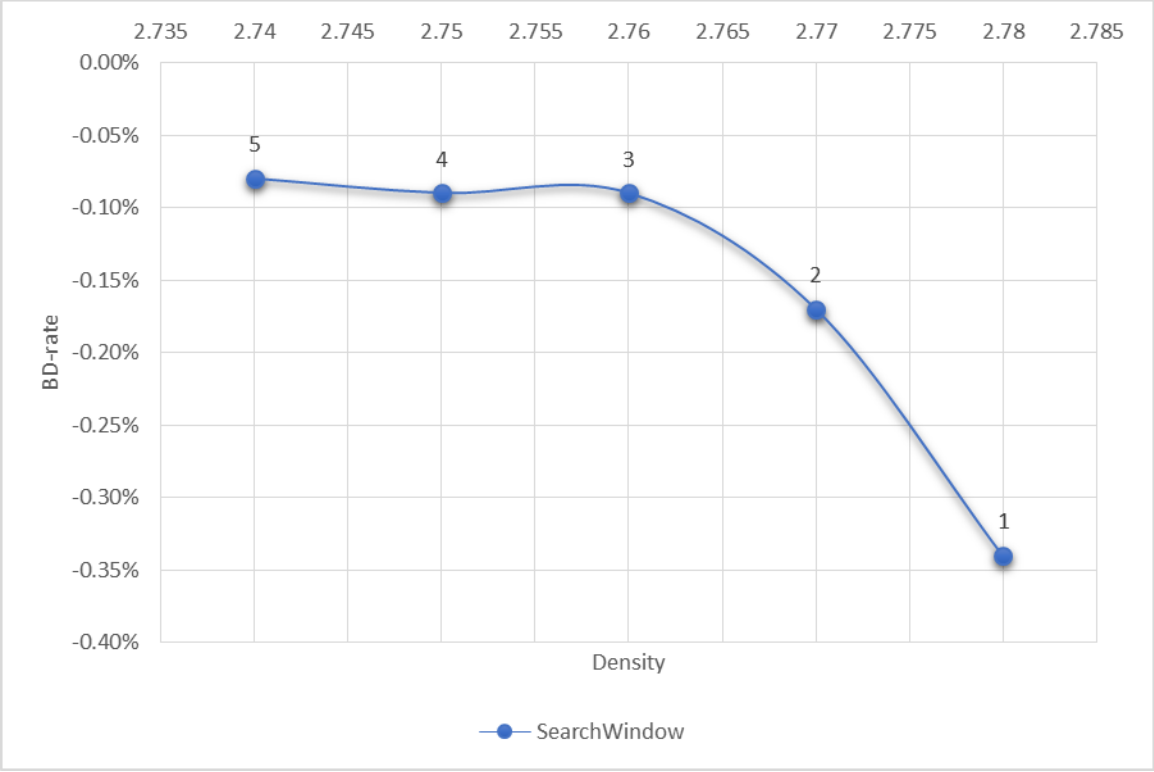


Figure 27. Impact of search presets on VQ and density for PreENC+ENCODE pipeline

For an API description, look for `mfxExtFeiHevcEncFrameCtrl::SearchWindow` in "Reference Manual for HEVC FEI."

3.8 Performance/VQ Tradeoff

A conventional SDK encoder has a Target Usage control for tradeoff between performance and quality. It has three settings:

- TU1: Quality
- TU4: Balance
- TU7: Speed

For some applications, this coarse control is not sufficient. A fine tradeoff between performance and visual quality may be desirable (e.g., to stay close to the balanced mode VQ but fit in one more channel). Yet another application may require runtime control over performance, at the same time keeping highest possible VQ (e.g., to adopt to changes in content or overall system workload). Both these goals can be achieved using FEI encoder controls. The application can gradually change performance and VQ and, in contrast to conventional encoder, do it in real time, on a frame-by-frame basis.

Table 18 shows a list of major VQ and performance controls supported by the FEI encoder, and their mapping to different target usages of a conventional encoder. This is not a complete list (some other settings also differ), but their impact is insignificant. All combinations of controls are supported. The four most interesting one are shown in Figure 28 for the PreENC+ENCODE pipeline and in Figure 29 for the DSO+ENCODE pipeline. For example, for the DSO pipeline, enabling force CTU split control on P frames, fast intra mode control on B frames, and increasing the number of frame partitions to 8 on the I and B frames doubles density in comparison to the conventional encoder at the cost of just 3% to the BD rate.

	Quality TU1	**Balanced TU4**	**Speed TU7**
Number of ref frames L0	3	3	1
Number of ref frames L1	1	1	1
Number of Frame Partitions	1	4	4
Force CTU split	off	off	on
Fast intra mode	off	off	on

Table 18. HEVC FEI presets to match different TU settings of conventional encoder

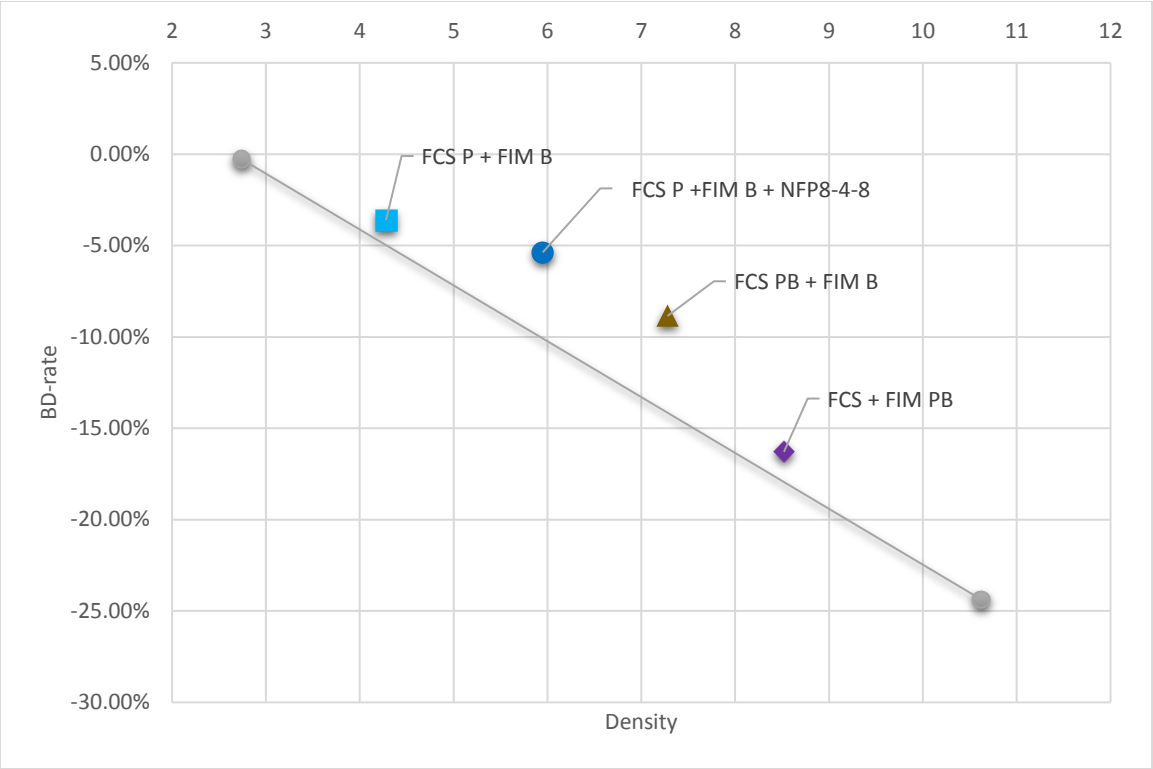


Figure 28. VQ/performance tradeoff for PreENC+ENCODE pipeline. FCS P means that Force CTU Split control is on on P frames, FIM B means that Fast Intra Mode is on on B frames, NFP8-4-8 means that Number of Frame Partitions is 8 for I frames, 4 for P frames, and 8 for B frames

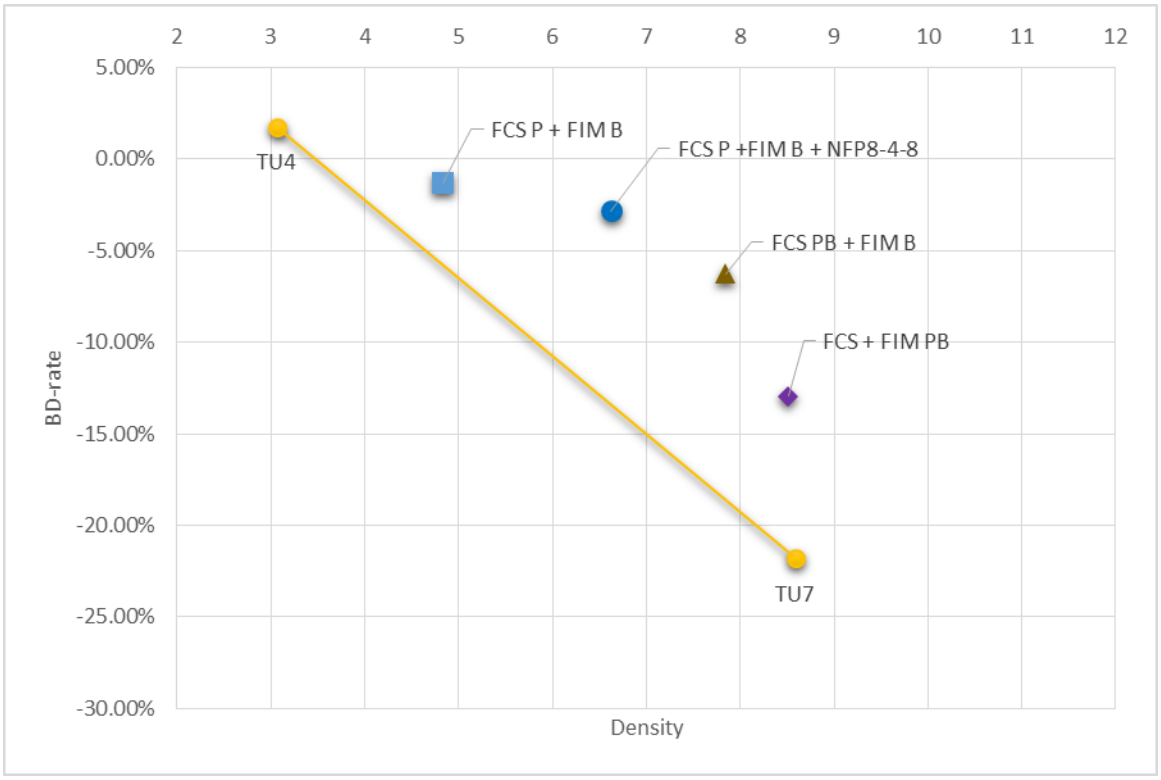


Figure 29. VQ/performance tradeoff for DSO+ENCODE pipeline. FCS P means that Force CTU Split control is on on P frames, FIM B means that Fast Intra Mode is on on B frames, NFP8-4-8 means that Number of Frame Partitions is 8 for I frames, 4 for P frames and 8 for B frames.

4 PERFORMANCE BOTTLENECKS

HEVC FEI provides a rich set of controls to improve VQ and performance. However, efficiently using them requires a basic understanding of possible performance bottlenecks. For example, it is useless to enable Fast Intra Mode if EUs are underutilized. The only thing it will achieve is to decrease VQ without any performance benefits.

To illustrate how GPU utilization depends on workload type, we selected three typical use cases for the PreENC+ENCODE pipeline (Table 19) and profiled them using Intel® VTune Amplifier 2018 and its GPU Hotspots analysis. For each use case, we selected a different number of channels to ensure real time performance (i.e., to ensure that transcoding frame rate for each stream is above 30fps). To eliminate start-time impact on the results, Intel VTune Amplifier traces were gathered from the fifth to tenth seconds of transcoding. As input, the 40Mbps “elephants_dream” stream was used for both AVC and HEVC cases. It was transcoded in constant QP mode to about 4Mbps.

Metric	AVC to HEVC TU4	AVC to HEVC TU7	HEVC to HEVC TU7
Channel count (density)	3	11	4
Total GPU utilization, %	100%	100%	100%
EU utilization, %	100%	100%	55%
VDBOX1 utilization, % (encode/decode)	15%/0%	76%/0%	38%/62%
VDBOX2 utilization, % (encode/decode)	0%/17%	0%/69%	not used

Table 19. GPU utilization metrics for PreENC+ENCODE pipeline

4.1 AVC TO TU4 HEVC TRANSCODING

As we can see from Table 19 and Figure 30 , when the encoder works in TU4 (balanced) mode, it spends most of the time on motion estimation and mode decision. EU utilization is 100%. But the bitstream packer is underutilized and VDBOX1 is almost idle at 15%. An additional load on EUs, like an increase in the number of External MV Predictors, degrades performance. Performance controls like Force CTU Split reduce EU usage and give performance gains.

HEVC-to-HEVC transcoding for the TU4 use case has a similar GPU utilization pattern. The only difference is that the VDBOX2 decoding workload moves to VDBOX1. EU remains a bottleneck.

Such utilization patterns may change only with very high decoding and/or encoding bitrates, where VDBOX may become a bottleneck. We won't estimate how high these bitrates are, because this heavily depends on stream content.

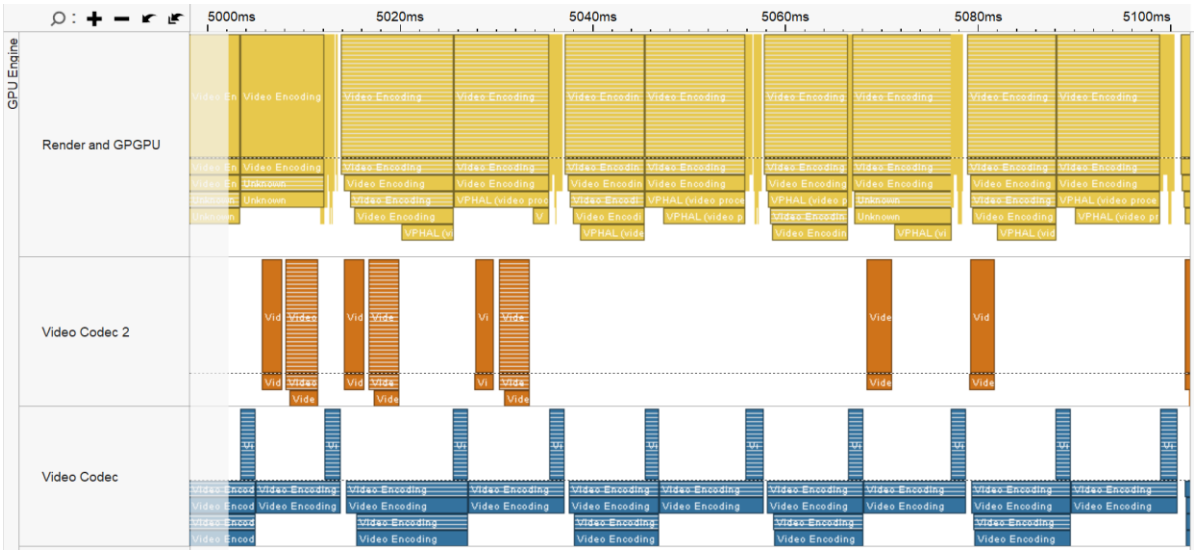


Figure 30. AVC to HEVC TU4. EU and VDBOXes task queues for a 100ms interval. Yellow denotes EU tasks, orange – VDBOX2, blue – VDBOX1.

4.2 AVC TO TU7 HEVC TRANSCODING

Table 19 and Figure 31 show how the encoder works in speed mode and uses fast versions of algorithms for motion estimation and mode decisions. Overall, processing here is significantly faster (11 channels in comparison to 3 in the previous case). But EUs are still a bottleneck and they are 100% busy. PAK utilization has significantly grown in comparison to the previous use case, but is still low. Also note that both VDBOXEs are utilized, one for decoding and one for encoding.

HEVC FEI controls behave similarly in this case, but the room for performance improvement is much smaller here. Most of the controls are already in fast mode (Table 18).

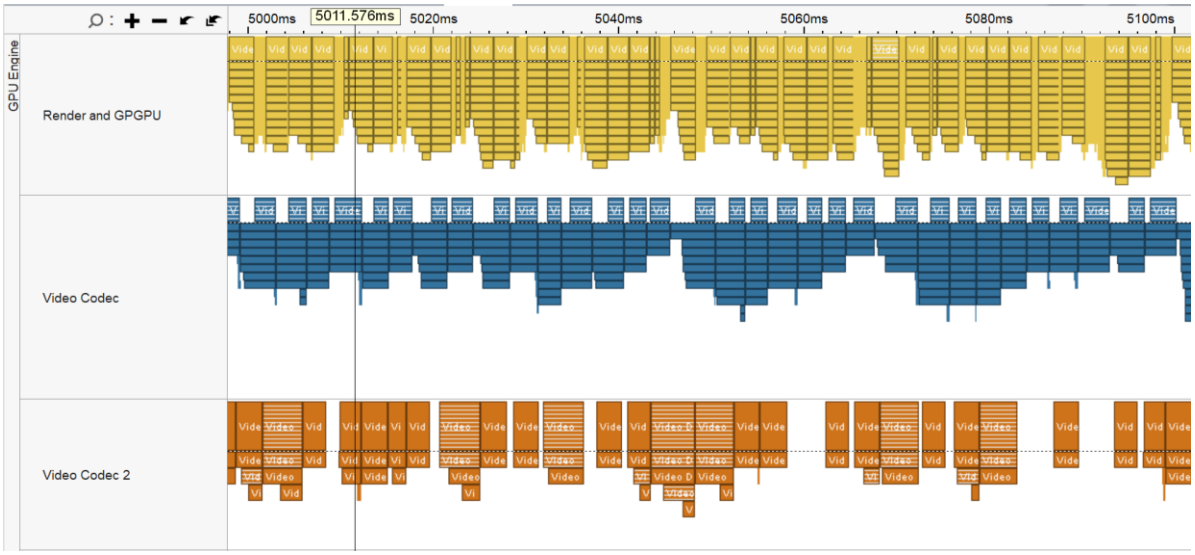


Figure 31. AVC to HEVC TU7. EU and VDBOXes task queues for a 100ms interval. Yellow denotes EU tasks, orange – VDBOX2, blue – VDBOX1.

4.3 HEVC TO TU7 HEVC TRANSCODING

In the next case (Table 19 and Figure 32), the picture changes. The encoder still works in the same speed mode, but the EUs are underutilized (just 55% busy). The reason for this change is redistribution of work between VDBOXes. In the previous case, one of them did decoding, another did encoding, and overall system performance was 11 channels. Here, all processing is done by one VDBOX and it becomes a bottleneck, decreasing performance to just 4 channels, close to TU4. VDBOX spends only 38% of the time on encoding and the rest on decoding. The second VDBOX is completely idle in this transcoding use case due to the hardware limitation of the 6th Generation Intel® Core™ Processors. Only one VDBOX supports HEVC processing, in contrast to AVC encoding/decoding, which is supported on both VDBOX1 and VDBOX2.

Because VDBOX is a bottleneck here, and its workload depends mostly on input/output bitrates and stream content, there is not much that can be done to improve performance. We assume that the bitrates and stream cannot be changed. On the other hand, EUs are underutilized, and it is possible to improve VQ by using more MVPs, reducing the number of frame partitions, and so on.

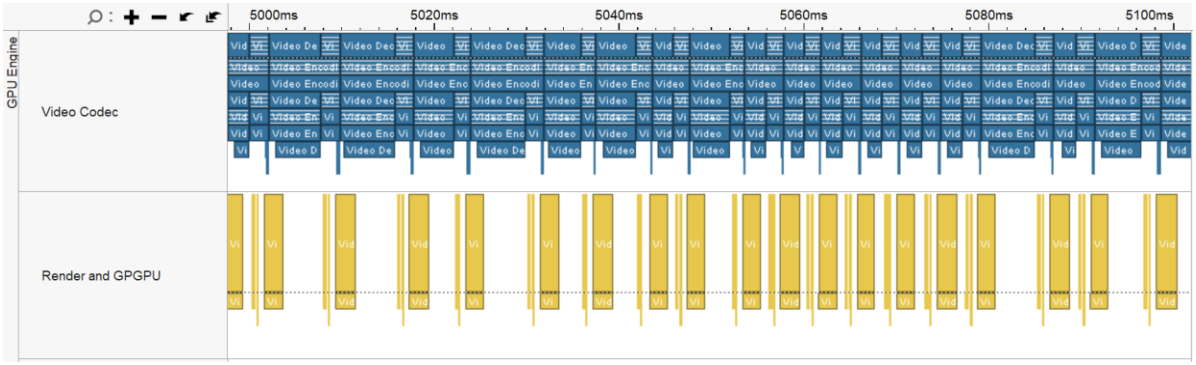


Figure 32. HEVC to HEVC TU7. EU and VDBOXes task queues for a 100ms interval. Yellow denotes EU tasks, blue represents VDBOX1.

5 SAMPLE APPLICATION

Usage of all FEI controls described in this paper is demonstrated by sample application that can be found on GitHub here. The application supports transcoding of one source stream to several output streams. This is called one-to-N transcoding. No resolution change is supported in this release, so it is a pure adaptive bitrate use case.

The sample supports input streams encoded by different codecs, but in this paper we discuss only one combination: two input streams encoded from the same YUV data with different settings. The streams are:

- The main stream, an AVC stream encoded with a high bitrate. It is used only as a source of raw data passed to the HEVC FEI encoder. In other words, this stream can be replaced by YUV file.
- The auxiliary stream, an HEVC stream from some high-quality encoder. The purpose of this stream is to emulate IP algorithms, which would tune HEVC FEI encoder settings to reach better quality. Besides, the information from the auxiliary stream is used in the BRC implementation provide by the sample.

Figure 33 shows the sample pipeline. The first stage produces decoded raw frames from the main input stream that goes through a regular AVC hardware decoder. We chose the AVC format for the main stream to reach higher density on the test system that has only one VDBOX that supports HEVC encoding/decoding (see the “Performance Bottlenecks” chapter for details). At the same stage, the auxiliary stream is processed by a bitstream parser and, based on extracted data, the repack stage produces MV predictors and other control parameters for the FEI encoder (see the “Decode Stream Out” and “MV Repacking” chapters for details). After the first stage, task objects (HevcTaskDSO) are produced that contain:

- Decoded frame
- DSO statistics
- Filled HEVC FEI control structures

Next, the task objects are passed to the Look Ahead (LA) part (a detailed description of LA is given in “Look Ahead BRC“ chapter). Note that LA may cache a big number of tasks. Since, in the current implementation, DSO statistics are bound to decoded surfaces, it leads to a high video memory consumption.

Once the LA cache is full, LA returns a task object, which is passed to the encoder(s). Each encoder is asynchronously operating in a separate thread. Encoders have queues of input tasks. In the absence of a new task, the encoder is waiting on a conditional variable representing a non-zero tasks queue. When the task comes, the encoder wakes up, encodes a surface, and produces a compressed bitstream.

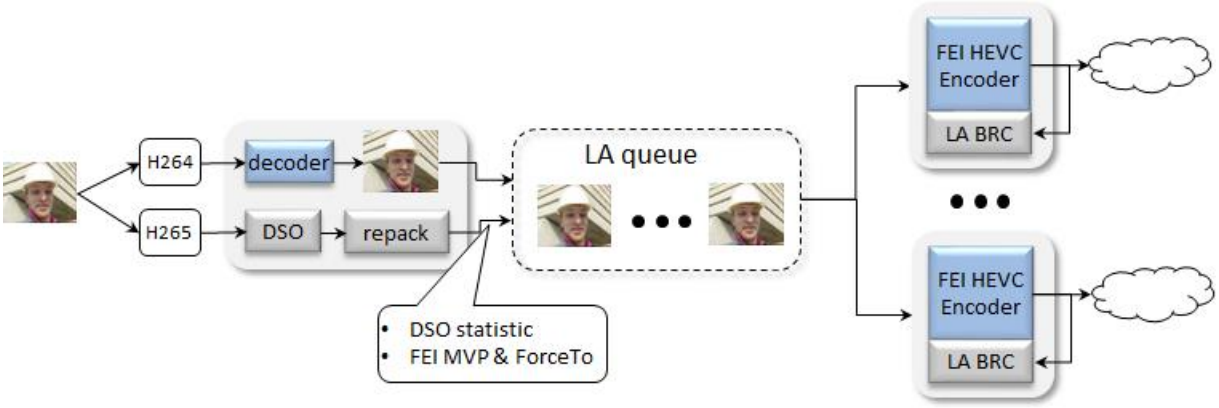


Figure 33. Sample configuration used for VQ evaluation

5.1 Decode Stream Out

Decode Stream Out (DSO) is a stage of information extraction from the input HEVC bitstream. It is done by input stream syntax parsing. Unfortunately, there is no support for HEVC decode stream-out in the GPU. In the sample, it is done on the CPU. The implementation goal was to show VQ improvements, so the code was not specifically optimized for performance. As a result, for speed target usage (TU7), this CPU-level parsing became a bottleneck and limited overall density.

The information extracted by the DSO stage includes Motion Vectors (MVs), Reference Lists (RefLists) and some additional statistics for LA BRC (see the "Look Ahead BRC" chapter for details).

The workflow diagram is represented in Figure 34. DSO accepts the encoded bitstream and extracts data related to next frame with the help of the bitstream parser. That includes all data associated with the frame NAL Units. Slice data contains several linked lists of HEVC basic syntax structures: CTU, CU, TU, and PU, where each CTU is a root for the CU list and each CU is a root for the TU and PU lists. Reference lists from the slice header are used to guarantee alignment of the reference structures between input and output streams. MVs from the PU become MV predictors after a repacking procedure. Some additional information from the CU level can be used as well. For example, the prediction type of the CU can give a hint to the encoder about which mode to use for the parent CTU. The DSO also plays an important role as a source of information for the BRC. QP and frame size are very important to know for proper bitrate control. The implementation of LA BRC in `sample_hevc_fei_abr` uses some more complex statistics as well, such as an approximation of Visual Distortion and the amount of predicted pixels from the reference frame for an accurate estimation of frame importance.

We extracted and used this data to improve the quality of the HEVC FEI encoder:

- Motion vectors. The FEI encoder doesn't use HME. It could miss long MVs or choose non-optimal ones. In the DSO stage, MVs extracted from the input stream are converted to MV predictors.
- Force intra/inter flags. These controls, if passed, influence the mode decision stage. During the DSO stage, force flags are set depending on the input PU type.

- Reference lists. These are used to align the input stream reference structure with the FEI encoder. It is not a mandatory step for general-purpose encoding, but very important for transcoding scenarios. If the reference lists in the input and output streams diverge, it will result in an incorrect MV predictor application because of mismatched reference indexes.
- POC. This is required for correct handling of the input surface by the MSDK lib.
- QP, encoded frame size, frame type, etc. This is data used by the BRC algorithm.
- Luma transform coefficients. The LA BRC algorithm uses them to approximate visual distortion.

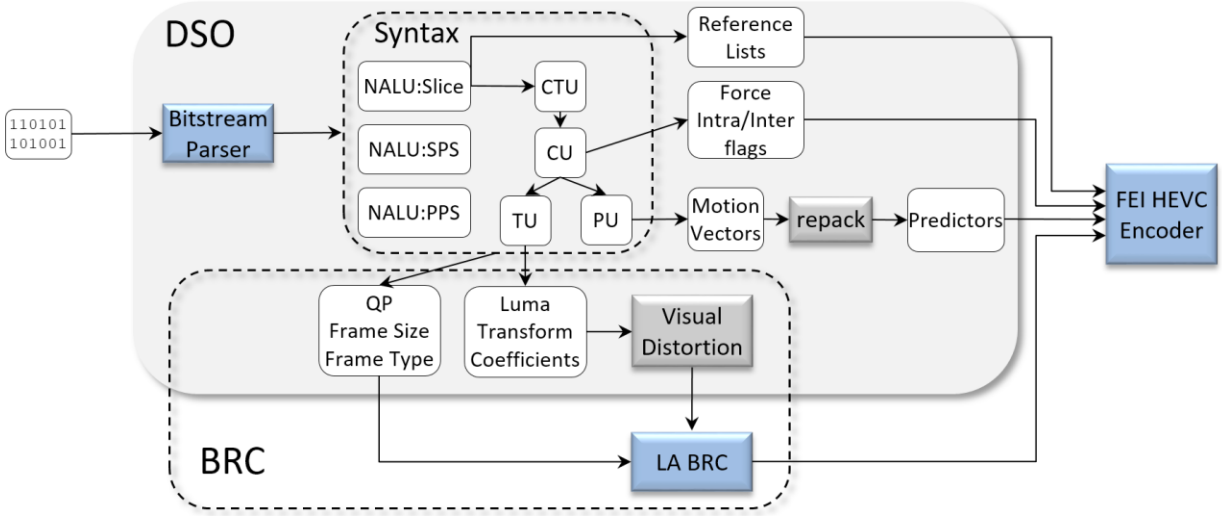


Figure 34. DSO part of pipeline workflow

5.2 Look Ahead BRC

The Look Ahead (LA) BitRate Control (BRC) algorithm assigns different QPs to each encoding frame using information from future input frames. The LA algorithm better distribute bits between frames and achieves better visual quality by adapting to stream content. It also better deals with scene changes.

The Look Ahead algorithm operates with frames in the LA window, including the current frame (which is not encoded yet) and some future frames. BRC tries to fit the target bitrate within the LA window with maximal visual quality. The Algorithm uses information from the input stream such as frame type and encoded size, as well as initial QPs, to estimate the importance of each frame and assign more bits (and lower QP) to more important frames.

The BRC in `sample_hevc_fei_abr` can be treated as a two-pass BRC because it operates with frames already passed through some other BRC during the encoding of the input stream.

The LA BRC in the sample has four important parameters, which can be set through the command line (see details on how each parameter affects encoding in the “BRC controls” chapter).

- `LookAheadDepth <n_frames>`. This shows how many frames from future to look ahead.
- `LookBackDepth <n_frames>`. This shows how many frames from the past to look back for calculation of statistics.

- `AdaptationLength <n_frames>`. This shows how many frames from the past to use to adjust the QP.
- `Algorithm::<MSE <YUV file> | NNZ | SSC>`. This shows which algorithm to use to approximate visual distortion:
  - `MSE` uses real mean squared error, calculated with YUV file (mostly for testing purposes in real use cases YUV is not available).
  - `NNZ` approximates distortion with the number of non-zero luma transform coefficients (default).
  - `SSC` approximates distortion with the sum of squared luma transform coefficients.

BRC has the following workflow:

- Input frames are buffered in a queue, with corresponding statistics gathered at the DSO stage.
- After the queue reaches `LookAheadDepth` size, frames start to go through the rest of the pipeline, pass through BRC block, and then go to the FEI encoder.

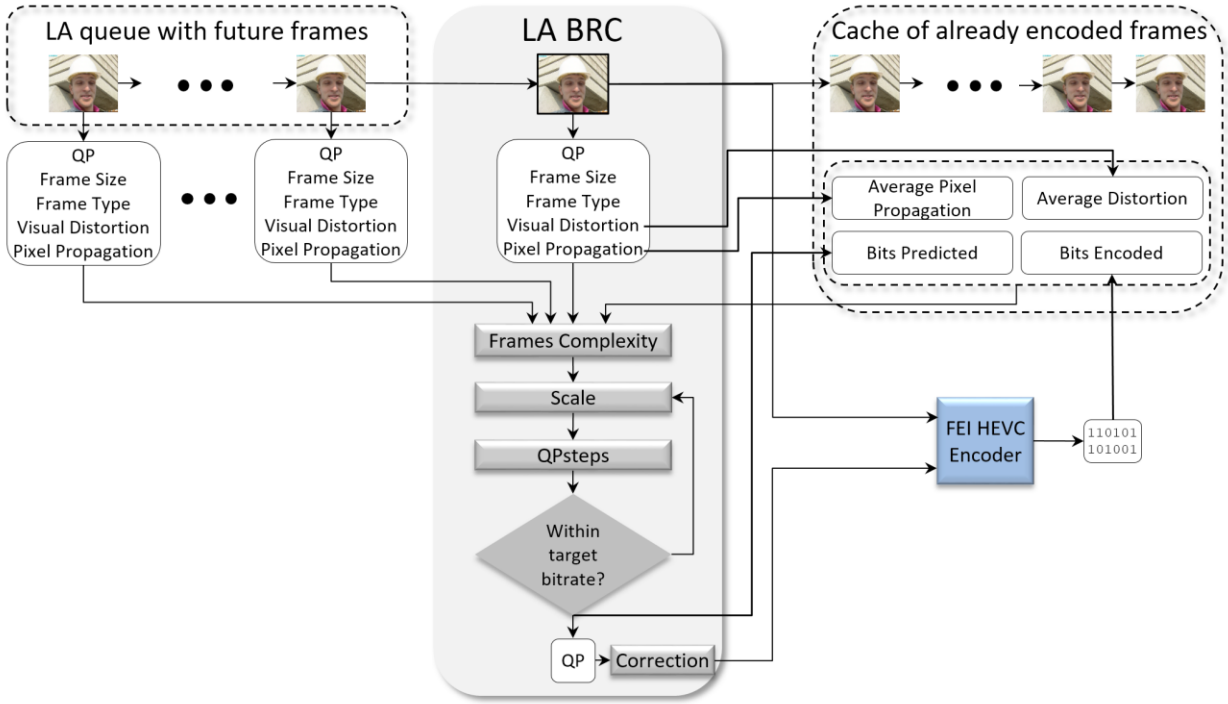


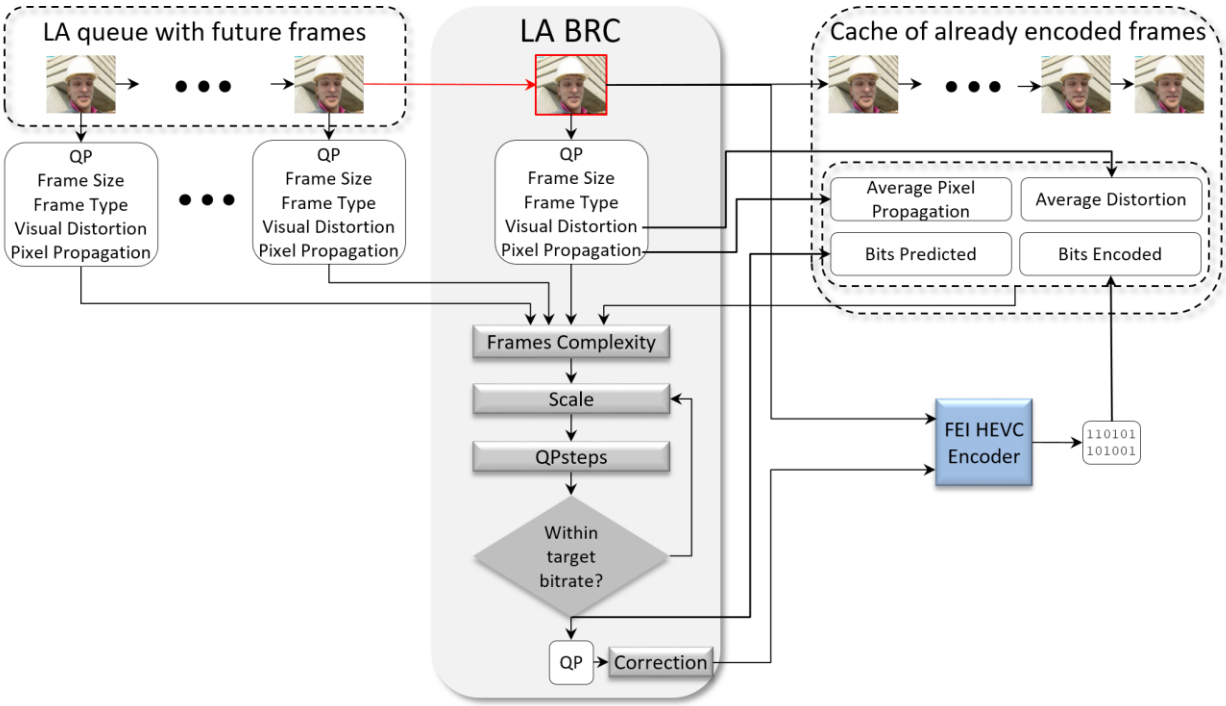
Figure 35. LA BRC workflow.

5.2.1 Algorithm

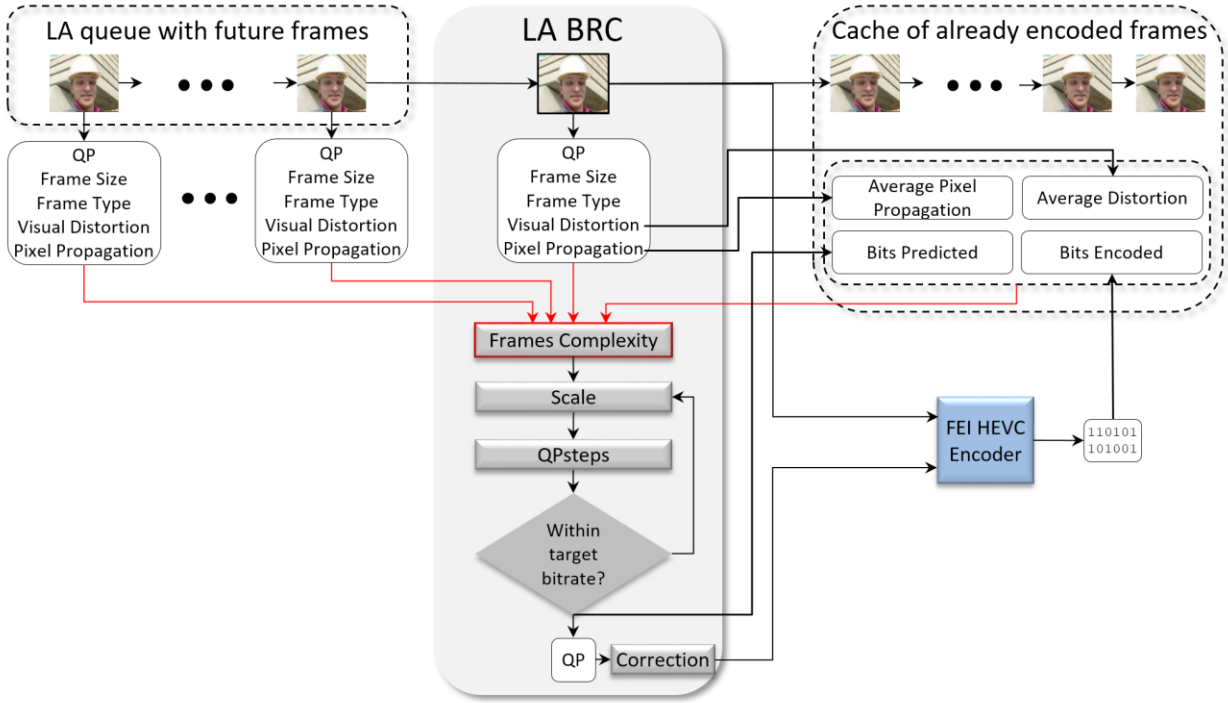
The main idea of the algorithm is to adjust the QP of the current frame using statistics from future frames and from some already encoded frames. Knowledge of some characteristics of future frames allows us to predict content changes and redistribute bits between frames more accurately to achieve better quality. The cache of already encoded frames is used to make our encoded size prediction for the current frame more robust.

The LA BRC algorithm can be subdivided into five main steps (a detailed explanation of the algorithm can be found in the ”Appendix B. Look Ahead BRC Algorithm”).

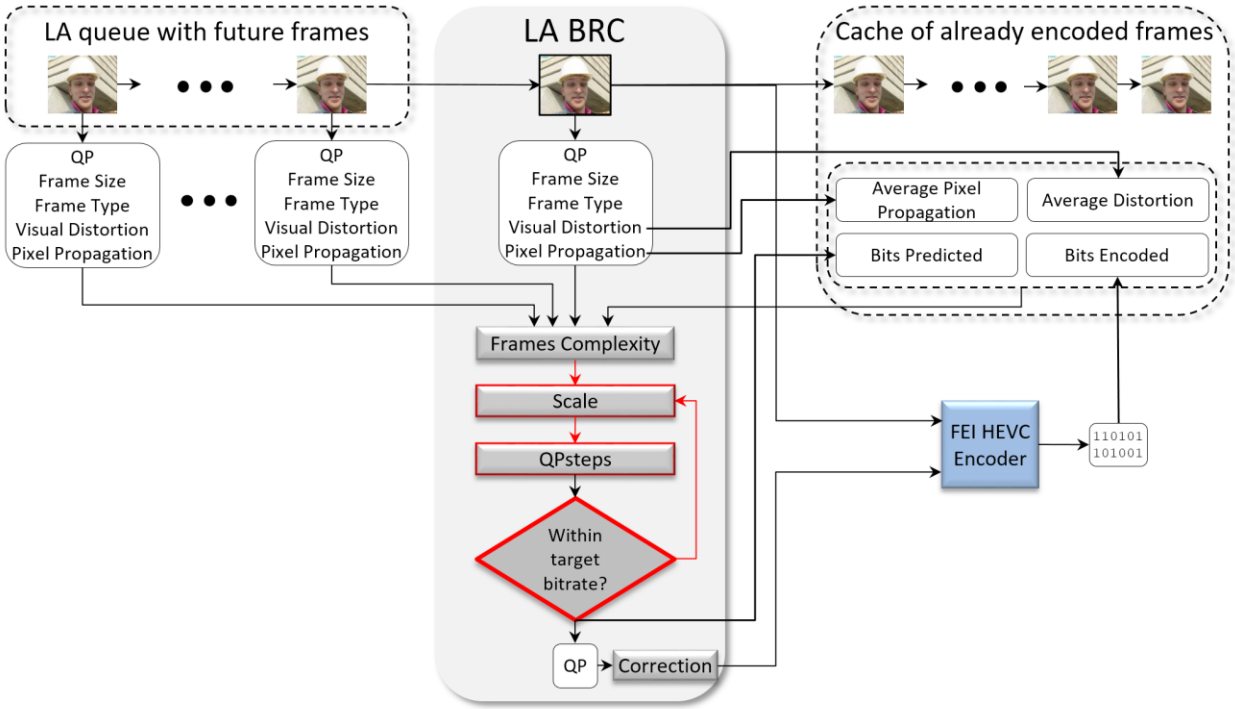
1. **Extract new frame from the LA queue.**



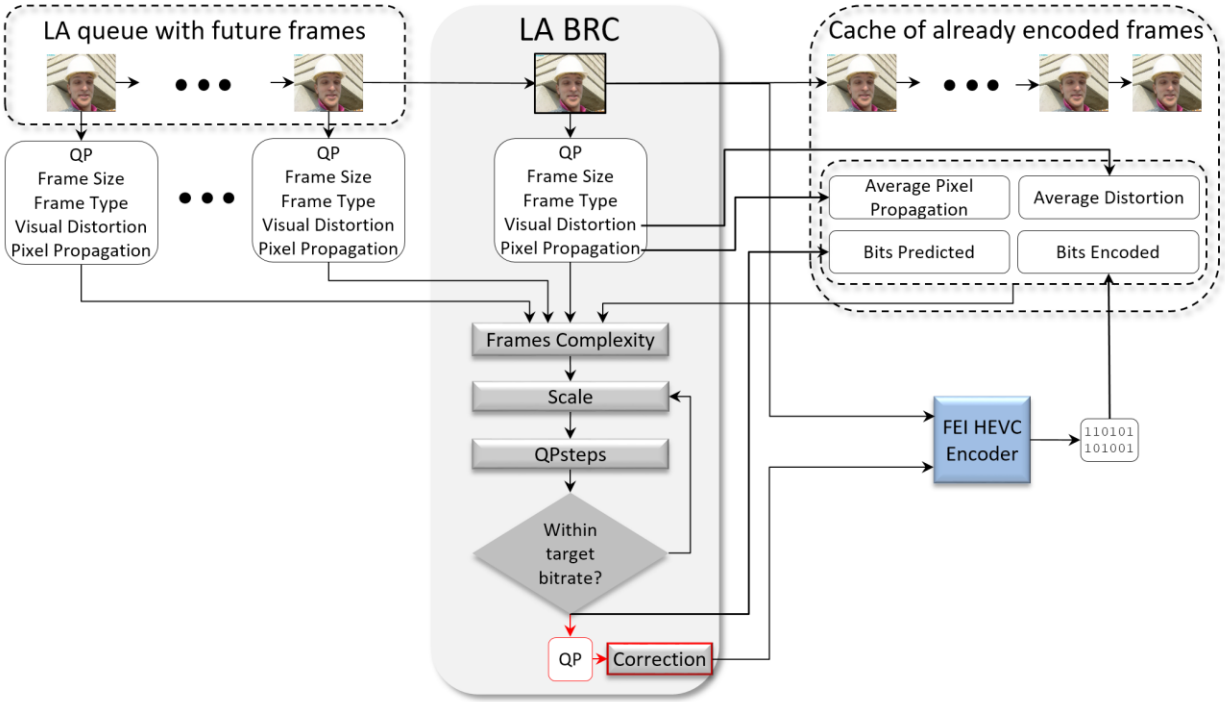
2. **Calculate the complexities of future frames in the LA queue and the current frame.**
   This step estimates the complexity of all non-encoded frames (current and all future). The algorithm uses some frame features such as QP, frame size, visual distortion, and pixel propagation. The aim is to obtain an integral value which shows how important this frame is for the sequence of frames within the LA window. It takes into account many indicators including how many pixels of the current frame would be used in future frames, visual quality, and the QP assigned by the previous BRC (during encoding of the input stream).



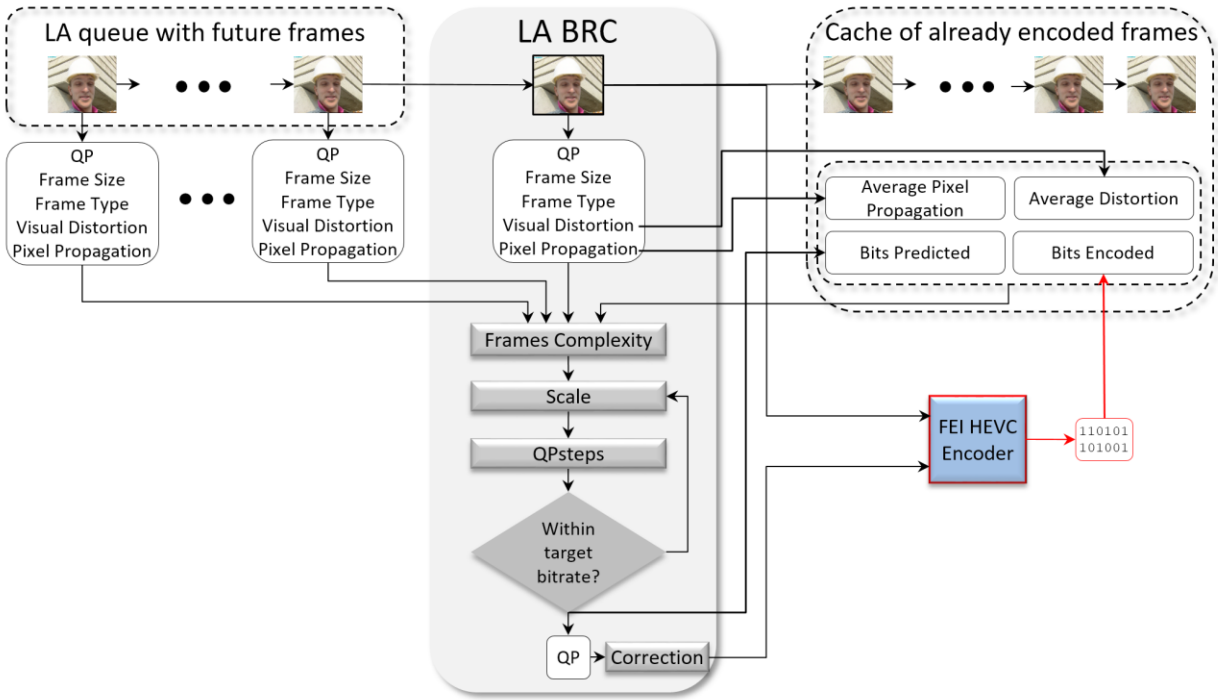
3. **Go to the iterative search procedure that assigns QSteps for the current frame and frames from the LA queue until reaching target bitrate.**
   Here, complexities are converted to QSteps (which are directly mapped to QPs). QStep assignment is performed by an iterative search procedure. At each search step, the algorithm assigns QSteps for all non-encoded frames and then estimates the bitrate. (For bitrate estimation, the encoded frames from frame cache are taken into account to stabilize the bitrate estimator and achieve robust estimation.) QStep correction is performed by scaling. If the calculated bitrate is far from the target, search chooses different scale. If the bitrate exceeds the target, the scale is decreased; otherwise, it is increased. Then the entire process is repeated until convergence is reached.



4. **After the QPstep is obtained, convert it to QP with additional corrections depending on the frame type, with the purpose of adding extra bits to non-B-frames and compensating for prediction errors.**
   This step performs QP correction if the current frame is a B-frame. The correction depends on reference frame types. This correction also compensates for prediction errors using information about previously predicted and encoded frame sizes.



5. **Send feedback after encoding to update encoded frame statistics with the size of the encoded frame.**



5.2.2 BRC controls

LA BRC exposes several parameters that influence the algorithm adaptation to content. Here, we give a brief explanation on how they affects actual quality adjustment.

- `LookBackDepth`. This parameter controls the left border of the look-ahead window. This affects statistics calculation and bitrate estimation. Increasing this parameter helps to improve bitrate estimation in future frames (mostly by stabilizing the estimation, making it less sensitive to prediction errors). However, it slows down the adaptation to content changes. In other words, too many frames from the past may result in a bad adaptation (Figure 36). On content where frame statistics change dramatically, a small window adapts much faster (Figure 37). On the other hand, in situations where stream statistics change but overall content complexity remains the same, a large window stabilizes the bitrate estimation and produces better quality (Figure 38).

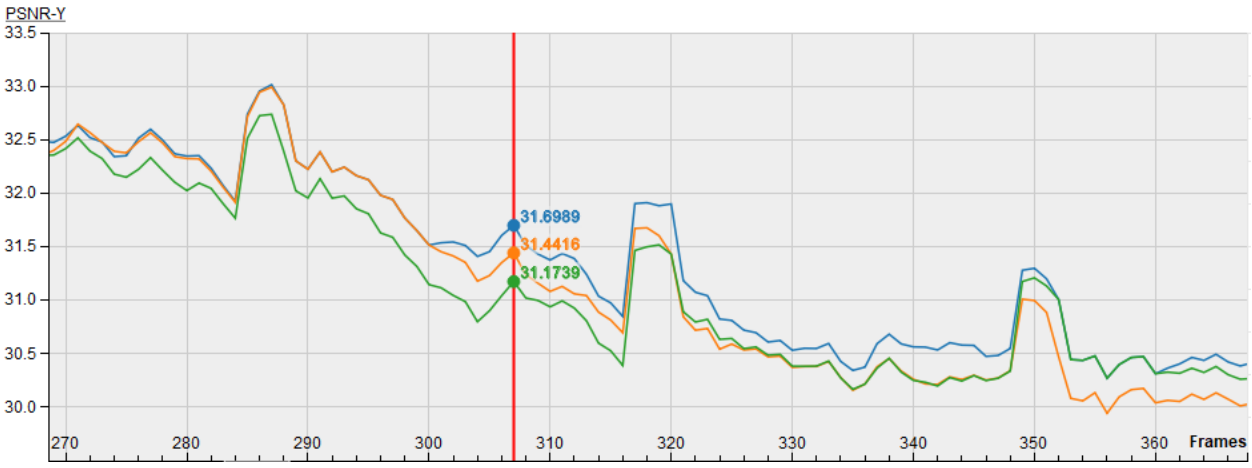


Figure 36. Adaptation on stream content change (panaramic shooting on BQ terrace stream). Blue – 300 frames of LookBack, orange – 100 frames, green – 0 frames.

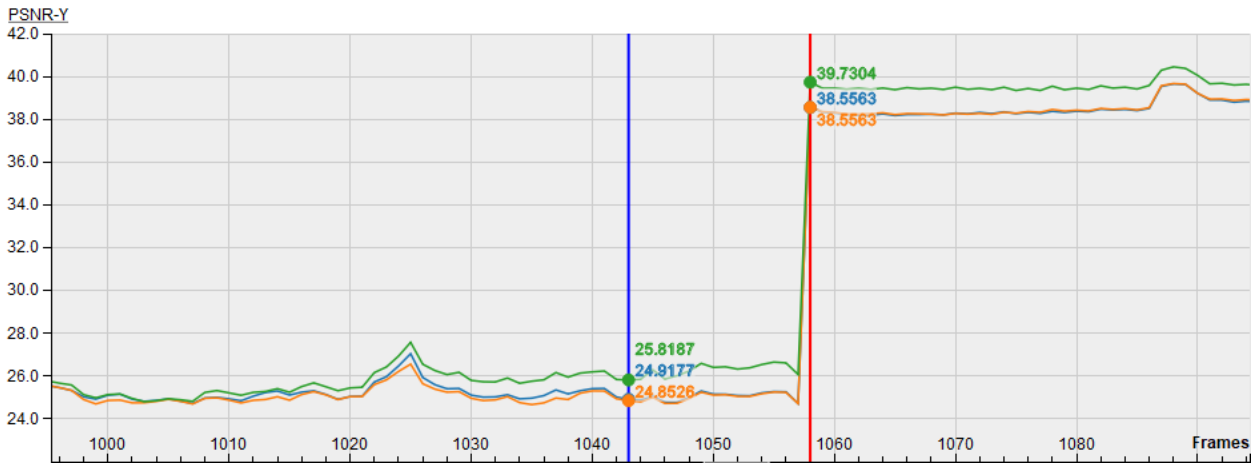


Figure 37. Adaptation to scene change: complex content to almost still image (crowd run to honey bee). Blue – 300 frames of LookBack, orange – 100 frames, green – 0 frames.

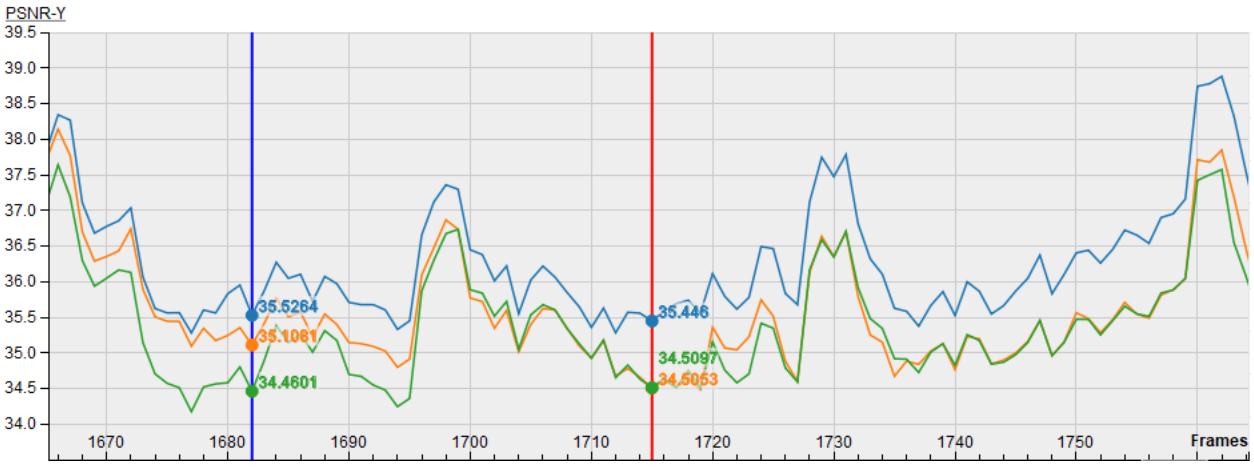


Figure 38. Adaptation to camera move on honey bee. Blue – 300 frames of LookBack, orange – 100 frames, green – 0 frames.

- `LookAheadDepth`. This is the number of frames from the future to look ahead, the same as LA queue length.
  A bigger parameter means more benefits gained after scene change (because the algorithm will start preparation to the upcoming scene change earlier). For example, Figure 39 shows where complex motion changes to static content ("crowd run" changes to "honey bee"). A large LA window allows the algorithm to notice such changes and to increase the bitrate. On the other side, if the complexity of a new scene is overestimated, the algorithm will decrease the bitrate earlier, with bigger LookAheadDepth, and will lose overall quality on this segment. For instance, Figure 40 shows panoramic motion changes to complex motion ("BQ terrace" changes to "crowd run"). The algorithm overestimated the complexity of the "crowd run" sequence and lost quality on the interval between frames 300 and 500.

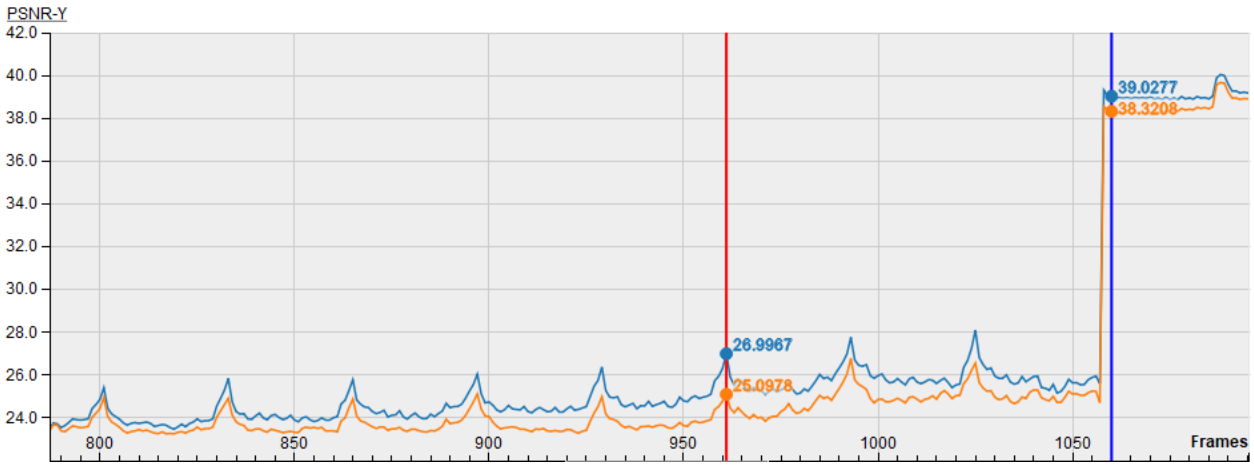


Figure 39. Benefit from noticing scene change earlier. Blue - LookAheadDepth = 300, orange - LookAheadDepth = 100

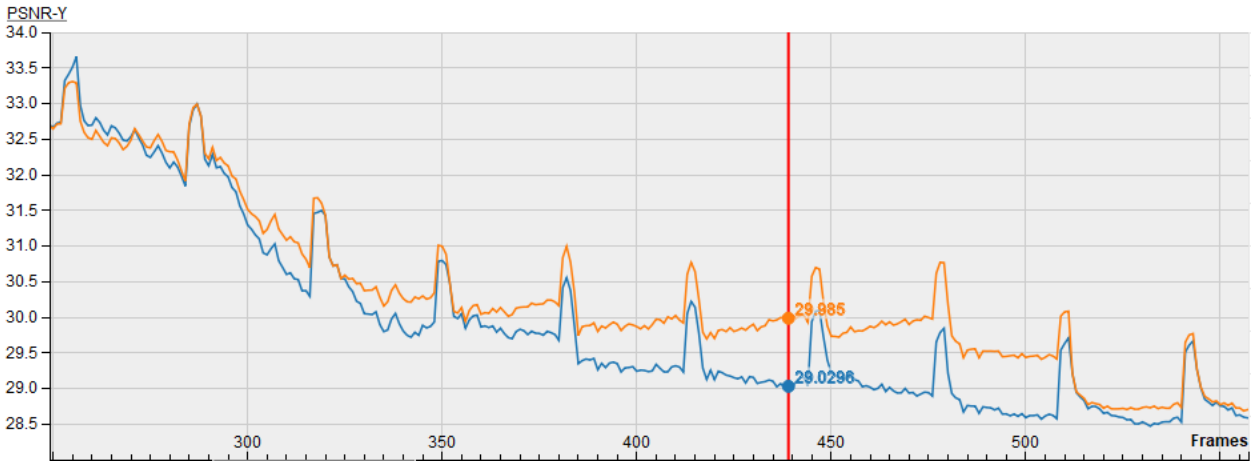


Figure 40. Panoramic motion will change to complex motion soon. Larger LA window lost quality. Blue - LookAheadDepth = 300, orange - LookAheadDepth = 100.

- `AdaptationLength`. This parameter controls number of frames from the past which are used to compute the bitrate adjustment ratio (see step 4 in the "Algorithm" chapter). A bigger amount stabilizes the adjustment estimation, but it makes it less sensitive to content changes. In general, this parameter targets bitrate accuracy. Figure 41 shows behavior in the case of changing a simple motion to a complex motion. The lowest adaptation window allows it to adapt faster. In Figure 42, the situation is the opposite. The larger window brings robustness to the estimation, when the small window fails to deal with a scene change where an almost-still image changes to simple motion.

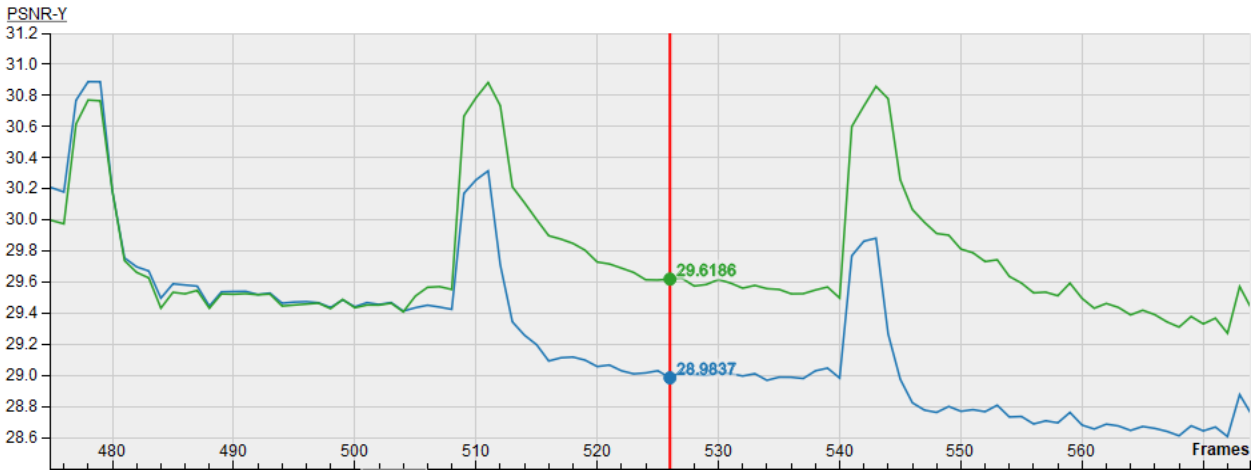


Figure 41. Behavior of algorithm with different AdaptationLength (simple panoramic shooting on "bqterrace" changes to complex motion of cars at the end of stream). Blue - 300 frames, green - 30 frames.

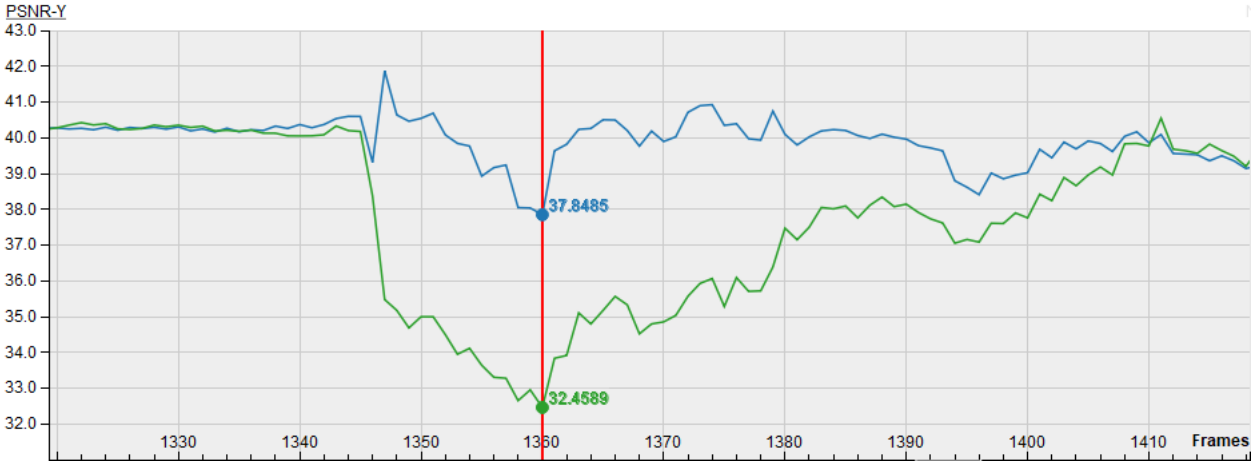


Figure 42. Behavior of algorithm with different AdaptationLength (almost still image of "honey bee" changes to simple motion on "sunflower" on frame 1347). Blue - 300 frames, green - 30 frames.

In conclusion, the optimal setting for the LA BRC parameters should be a tradeoff between prediction error and the speed of adaptation to content changes.

- `Algorithm`. This is the algorithm of visual distortion estimation. It is used to approximate the subjective quality of an encoded frame relative to the source frame. This value is one of the parameters of the frame complexity estimation formula (see Appendix B. Look Ahead BRC Algorithm," for details). Supported algorithms are:
    - `MSE` (Mean Squared Error), calculated as a per-pixel sum of the squared differences between the source and encoded frames divided by the number of pixels. This one requires an original YUV file, which is not available in most cases. It is implemented mostly for testing purposes, and not optimized for performance (it reads huge amount of data from disk, allocates a temporary buffer, and uses non-optimized straightforward way to calculate sum of squared differences). This has the best quality and worst performance among all methods (see the "Visual Quality" chapter for details).
    - `NNZ` (Number of Non-Zero transform coefficients), calculated as a total number of the non-zero luma transform coefficients in a whole frame. Based on the results from the "Visual Quality" chapter, it gives good quality in terms of BD-rate gain and almost the same performance as hardware BRC. Because of that, the `NNZ` algorithm is used by default.
    - `SSC` (Sum of Squared luma transform Coefficients). This is another way to map coefficients to distortion, which involve the magnitude of the coefficients. This method shows worse quality results than `NNZ`, with comparable performance.

5.2.3 Scene Change Handling

The Look Ahead BRC algorithm is much better for dealing with scene changes than regular BRC. Scene changes may be very harmful for BRC without LA if the motion type changes (i.e., fast motion content changes to slow motion and vice versa). In such a situation, past statistics won't help us guess optimal encoding parameters, so BRC will spend some time to adapt for the new type of content and will produce content of bad quality during this time, trying to stay within the allowed bitrate. Figure 43 demonstrates the different behavior of two BRCs during a scene change. We will use hardware BRC as a reference in quality and performance measurements. This standard built-in driver bitrate control algorithm is used by the conventional hardware encoder. This plot demonstrates the PSNR Y of HW BRC and LA BRC on a concatenated stream with several scene changes, where the type of motion radically changes. It is seen that the LA BRC experiences much less distortion on scene changes. For example, on frame 1058 PSNR Y, the difference between the two algorithms is above 9db. This results in a huge visual quality difference (Figure 44 and Figure 45).

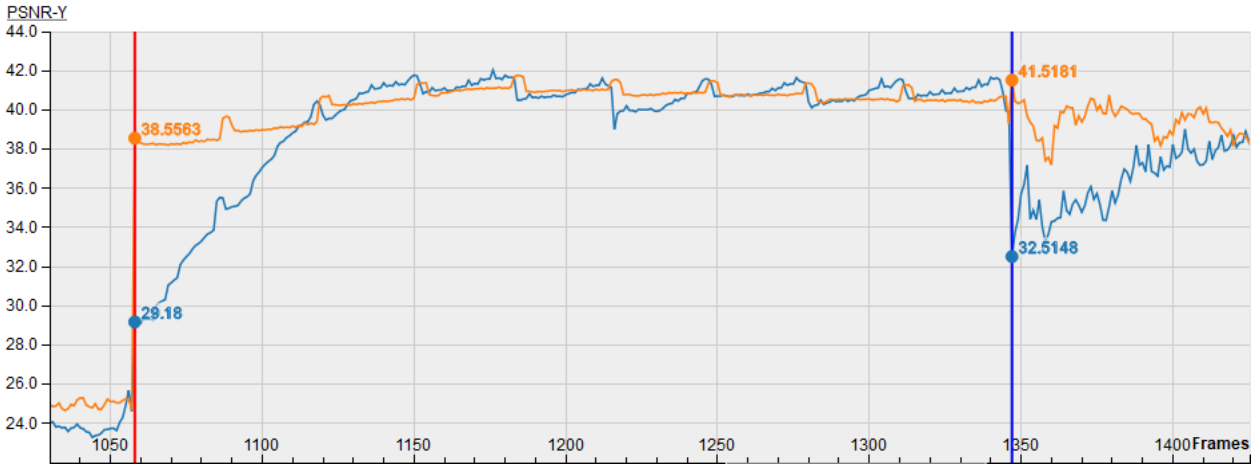


Figure 43. Scene change "crowd run" -> "honey bee" (frame 1056) -> "sunflower" (frame 1347). Blue line - HW BRC without LA, orange - LA BRC from sample_hevc_fei_abr

Figure 44. HW BRC PSNR-Y 29.18 (frame 1058 after "crowd run" to "honey bee" scene change)

Figure 45. LA BRC PSNR-Y 38.5563 (frame 1058 after "crowd run" to "honey bee" scene change)

5.2.4 Visual Quality

Four streams were used to evaluate the impact of the BRC algorithm on visual quality. Each one was concatenated from four streams from the test pool described in Appendix E. Stream information." We selected streams with different complexities to make scene changes harder to handle by BRC and to emphasize LA benefits. We ran the DSO+ENCODE pipeline and compared the default BRC used by the hardware-accelerated encoder with LA BRC, as described above.

VQ results are shown in Table 20. On these streams, LA BRC shows better objective visual quality than HW BRC. The MSE algorithm is the best, but both NNZ and SSC have similar VQ gain on two streams, moderate gain on one, and are similar to HW BRC VQ on the other stream.

Performance results are presented in Table 21. LA BRC with the NNZ and SSC algorithms demonstrates performance on par with HW BRC. MSE is significantly slower.

sequence	MSE	NNZ	SSC
bq_terrace_crowd_run_HoneyBee_sunflower	3.43%	1.06%	0.55%
ducks_sunflower_park_joy_HoneyBee	4.88%	4.66%	3.53%
HoneyBee_sunflower_bq_terrace_crowd_run	2.45%	0.09%	-0.86%
sunflower_ducks_HoneyBee_park_joy	5.65%	6.36%	5.36%
average	4.10%	3.04%	2.15%

Table 20. BD-rate comparison. LA settings: LookAhead=100, LookBack=100, AdaptationLength=100. Positive numbers mean that LA BRC is better.

BRC Algorithm	Density
HW BRC	3.19
MSE	0.26
NNZ	3.02
SSC	3.02

Table 21. Average density in numbers of transcoding channels. LA settings: LookAhead=100, LookBack=100, AdaptationLength=100. Bigger is better.

6 APPENDIX A. SAMPLE APPLICATION DETAILS

6.1 DECODE STREAM-OUT

DSO implementation in the `sample_hevc_fei_abr` is performed by `HevcSwDso` class, which uses the `IYUVSource` interface and fills in data for the Encoder and BRC. Also, it owns an instance of `BS_HEVC2_parser`, which perform actual input stream parsing.

HEVC bitstream parser implementation is located in the `BS_HEVC2_parser` class. To initialize the parser, input stream should be opened with `BS_HEVC2_parser::open`. To start actual parsing, perform a call of the `BS_HEVC2_parser::parse_next_unit` function. Note that this call invalidates all pointers to data of previous frame's structures if they weren't locked by `BS_HEVC2_parser::lock`. After that step, obtain the pointer to the first NALU of the frame with a `BS_HEVC2_parser:: get_header` call. Then you can access all syntax elements: SPS/PPS/Slice Headers, CTU, CU, PU, and TU trees. Each syntax element tree is arranged in a linked list structure.

All the work is done inside `mfxStatus HevcSwDso::GetFrame(HevcTaskDSO & task)` and the following functions are called one-by-one:

- `void HevcSwDso::FillFrameTask(const BS_HEVC2::NALU* header, HevcTaskDSO & task)`
  `header` – pointer to the current NAL Unit being parsed
  `task` – DSO task which stores information required for proper encoding with the FEI encoder

  Here, the current NAL Unit is parsed and extracted information required for `HevcTaskDSO` task, such as: DPB state (`HevcTaskDSO::m_dpb`), RefLists (`HevcTaskDSO::m_refListActive[2]`), frame type (`HevcTaskDSO::m_frameType`). This data is required for actual encoding. RefLists are used for proper construction of `mfxExtHEVCRefLists` – the extension buffer which is used to align internal encoders RefList with the one from input stream.

- `void HevcSwDso::FillMVP(const BS_HEVC2::NALU* header, mfxExtFeiHevcEncMVPredictors & mvps, mfxU32 nMvPredictors[2])`
  `header` – pointer to current NAL Unit being parsed
  `mvps` – motion vector predictors buffer to fill
  `nMvPredictors[2]` – to report back number of MVPs for L0/L1 lists

  In this function, the `mfxExtFeiHevcEncMVPredictors` buffer is filled according to its data layout with the motion vectors of input frame. Those vectors will be used as predictors during motion estimation stage of FEI encoder.

- `void HevcSwDso::FillCtuControls(const BS_HEVC2::NALU* header, mfxExtFeiHevcEncCtuCtrl & ctuCtrls)`
  `header` – pointer to current NAL Unit being parsed
  `ctuCtrl` – extension buffer for CTU level controls

In this function, some additional tuning is performed on the CTU level by filling the `mfxExtFeiHevcEncCtuCtrl` buffer. Currently, the CTU is forced to Inter/Intra according to the mode decision in the auxiliary input stream.

- `void HevcSwDso::FillBRCParams(const BS_HEVC2::NALU* header, HevcTaskDSO & task)`
  `header` – pointer to current NAL Unit being parsed
  `task` – DSO task which stores information required for proper encoding with FEI encoder and also data for BRC

  This function fills `HevcTaskDSO::m_statData` parameters for LA BRC (LA BRC presence is indicated by the `m_bCalcBRCStat` variable). LA BRC uses various statistics, including original frame size, share of intra pixels, and shares of pixels predicted from each reference. If the user selected an algorithm which uses distortion approximation, the additionally calculated number of non-zero luma transform coefficient or sum of squared coefficients.

6.2 MV REPACKING

Repacking is the process of converting per PU motion vectors obtained from DSO to the `mfxExtFeiHevcEncMVPredictors` structure required for FEI encoder. It has two modes of operation. Both modes attempt to produce the MVP field in `mfxExtFeiHevcEncMVPredictors` that is as close as possible to the motion vector field in the corresponding input frame from auxiliary bitstream.

6.2.1 First mode

This is the default mode. It is specified via a `-DSOMVPBlockSize 7` command line option of the `sample_hevc_fei_abr` application. It sets the `mfxExtFeiHevcEncMVPredictor::BlockSize` field of each `mfxExtFeiHevcEncMVPredictors::Data` element according to the size of the corresponding CU in the frame from the auxiliary bitstream (Figure 46). For the sake of simplicity, only the L0 motion vectors will be considered, since the process is the same for L1 motion vectors.

For 32x32 CUs, each input CU corresponds to four `mfxExtFeiHevcEncMVPredictor` elements. The `BlockSize` field of the `mfxExtFeiHevcEncMVPredictor` element corresponding to the top-left 16x16 block of the 32x32 CU is set to 2, while the other three `BlockSize` fields are set to 0. This enables the 32x32 predictor mode (i.e., only MVPs from the first `mfxExtFeiHevcEncMVPredictor` element will be used for the whole 32x32 area occupied by the CU). The motion vectors corresponding to the PUs inside the 32x32 CU are copied into the `mfxExtFeiHevcEncMVPredictor::MV[4]` field of the same `mfxExtFeiHevcEncMVPredictor` element. Since there may be no more than four PUs inside a CU, all available bitstream motion vectors for this CU are copied and no loss of motion vector data occurs during repacking.

For 16x16 CUs, each 16x16 CU corresponds to a single `mfxExtFeiHevcEncMVPredictor` element. For this element, the `BlockSize` field is set to 1 (16x16 predictor mode) and up to four MVs from the PUs inside the 16x16 CU are copied into the `mfxExtFeiHevcEncMVPredictor::MV[4]` field.

For 8x8 CUs, each 8x8 CU has three sibling 8x8 CUs that correspond to the same `mfxExtFeiHevcEncMVPredictor`. For each of these four CUs, only the motion vector from the first PU inside the CU is copied into the `mfxExtFeiHevcEncMVPredictor::MV[4]` field of the corresponding `mfxExtFeiHevcEncMVPredictor` element.

Note that the process above applies only to inter CUs. For intra CUs, there are no motion vectors to repack and the corresponding `mfxExtFeiHevcEncMVPredictor` structure is left empty.

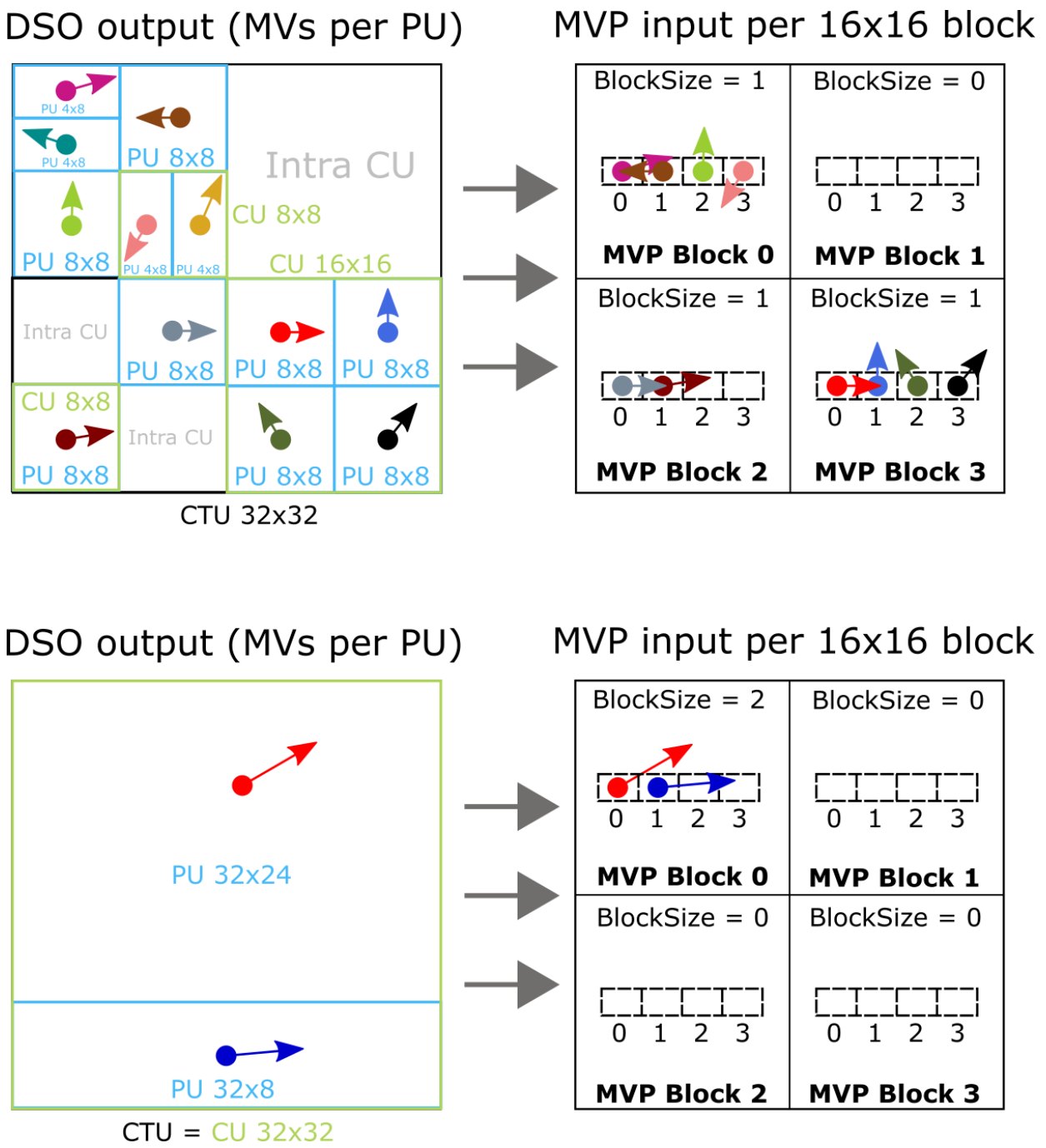


Figure 46. Illustration of the process of repacking the motion vectors from the DSO auxiliary stream into the motion vector predictor structure mfxExtFeiHevcEncMVPredictors. First mode, -DSOMVPBlockSize 7.

6.2.2 Second mode

The second mode of motion vector repacking is specified via the `-DSOMVPBlockSize 1` command line option of the `sample_hevc_fei_abr` application. It sets the `BlockSize` field of each `mfxExtFeiHevcEncMVPredictors::Data` element to 1 (16x16 MVP mode) and attempts to fill all the `mfxExtFeiHevcEncMVPredictor` elements corresponding to 16x16 blocks with valid motion vector predictor data by replication MVPs if necessary (Figure 47).

For 32x32 CUs, the motion vector inside each PU is put into each `mfxExtFeiHevcEncMVPredictor` element that has its corresponding 16x16 block intersected by this PU.

For 16x16 CUs, the repacking process is the same as for the first mode.

For 8x8 CUs, the motion vectors from all PUs of the four 8x8 CUs corresponding to a single `mfxExtFeiHevcEncMVPredictor` element are gathered and four motion vectors corresponding to the PUs with largest area are copied into the `mfxExtFeiHevcEncMVPredictor::MV[4]` field.

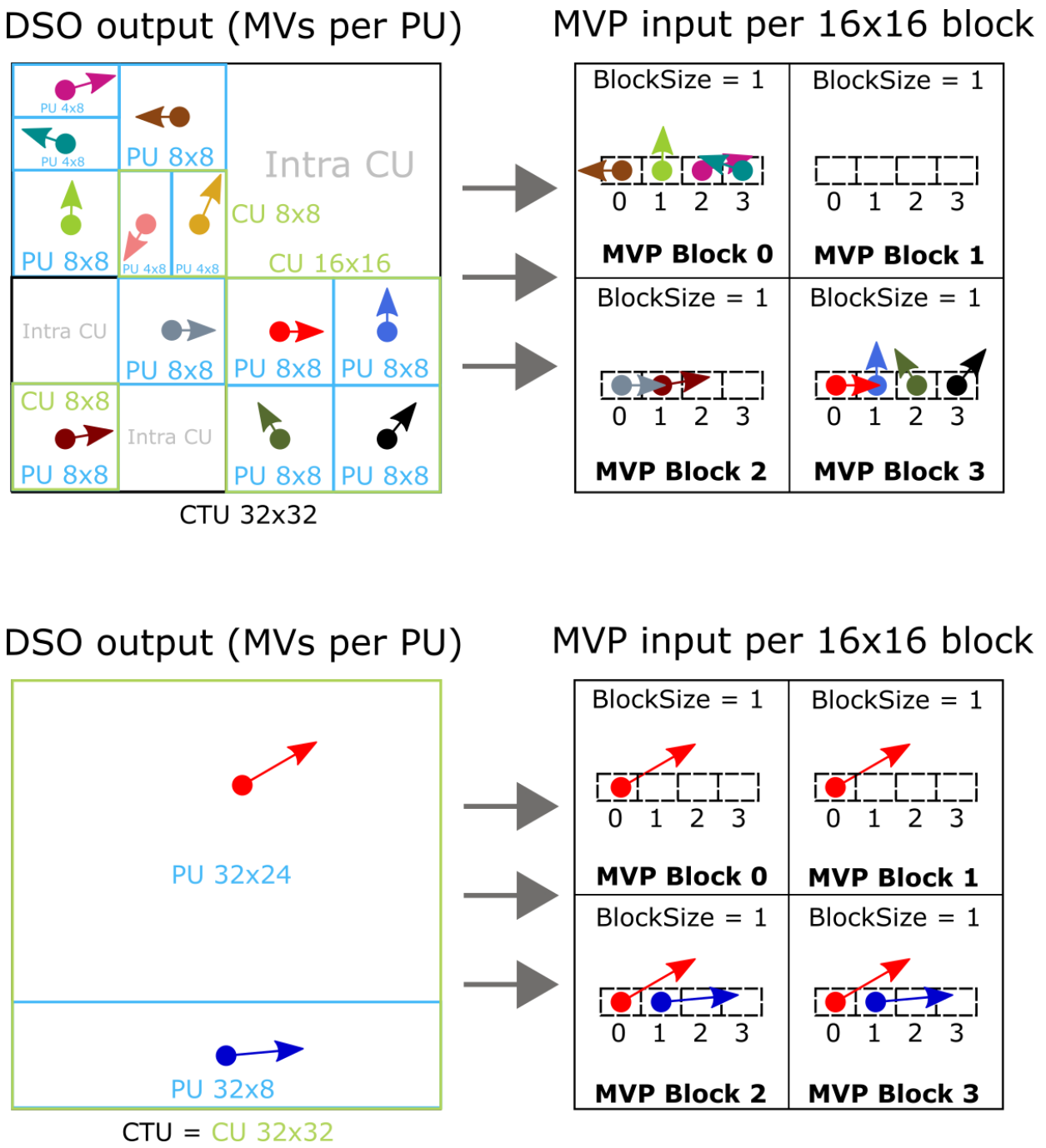


Figure 47. Illustration of the process of repacking the motion vectors from the DSO auxiliary stream into the motion vector predictor structure mfxExtFeiHevcEncMVPredictors. Second mode, -DSOMVPBlockSize 1.

6.3 LA BRC

The Look Ahead BRC is implemented in the `LA_BRC` class, which provides an interface for the encoder to submit a new frame and its statistics, and also to report the size of the encoded frame as feedback. The Look Ahead Queue is implemented in the `LA_Stat_Queue` class. All the work with the queue is hidden within LA_BRC. The LA queue stores some average statistics for frames in it, which is automatically updated when some frame leaves the queue or some other frame comes in. This data is used by the LA Algorithm for frame complexity estimation (step 3) and in the QP correction stage (step 4). Correspondence of the code location to the steps in section “Algorithm“ is:

1. `LA_Stat_Queue::StartNewFrameProcessing`
2. `LA_Stat_Queue::CalcComplexities`
3. `LA_BRC::UpdateStatData`
4. `LA_BRC::PreEnc`
5. `LA_BRC::Report`

7 APPENDIX B. LOOK AHEAD BRC ALGORITHM

7.1 ALGORITHM

In this section, we provide a precise explanation of each step from the "Algorithm" section.

The major steps of the algorithm are:

1. Extract new frame from LA queue.
2. Calculate the complexity of all frames in the LA queue, and for the current frame, using the statistics of each frame with the following formula:

$$Complexity[i] = QPstep\_orig[i] \cdot FrameSize\_orig[i] \sqrt{\frac{\overline{Distortion}}{1 + Distortion[i]}} \cdot \frac{Propagated[i]}{\overline{Propagated}}.$$

   Where $QPstep\_orig$ and $FrameSize\_orig$ are original QPstep and FrameSize from input stream. $Distortion$ is a current frame distortion and $\overline{Distortion}$ is an average distortion in the LookAhead + LookBack window. $Propagated$ is the total propagation of the current frame's pixels, $\overline{Propagated}$ is the average propagation on the LookAhead + LookBack window.

3. Go to the iterative search procedure, which assigns QPs to each frame and checks if the total bitrate of the LA queue is within the bitrate (sizes of already encoded frames from cache are also used here for more robust bitrate estimation). The QPstep estimation is performed with the following procedure.

$$QPstep[i] = \frac{Complexity[i]^{0.4}}{scale}$$

   This will set the QPsteps for **P** frames at correct level, but the **I** and **B** frames should be corrected by the following procedure, which is applied to all frames in the LA queue (non-encoded frames). At first, we will skip all the subsequent **B**-frames (frames are stored in encoded order, so reference frames precede to non-reference—i.e., the sequence **IPBBB** in the LA queue corresponds to the reference **I** and **P**-frames and 3 **B**-frames, which have **I**-frame as the L0 reference and **P**-frame as the L1 reference). And set their QPs as:

$$QstepB = QstepFirstNonB \cdot \left(\frac{ComplexityFirstNonB}{ComplexityB}\right)^{0.35}.$$

   Then, for all **P**-frames, we will compute two values, which initialized with 0 first.

$$LogQ = (LogQ + \log(QPstep[i])) \cdot Propagation[i]$$
$$norm = (norm + 1) \cdot Propagation[i]$$

   Both are used for I frames QPstep calculation.

$$QPstepPropagated = \frac{e^{\frac{LogQ}{norm}}}{1.4}$$

   If $norm \geq 1$then $QPstepI = QPstepPropagated$ else we blend the QPstep in following way:

$$QPstepI = QPstepPropagated \cdot norm + QPstepIoriginal \cdot (1 - norm).$$

Now we can estimate the bitrate for current $scale$.

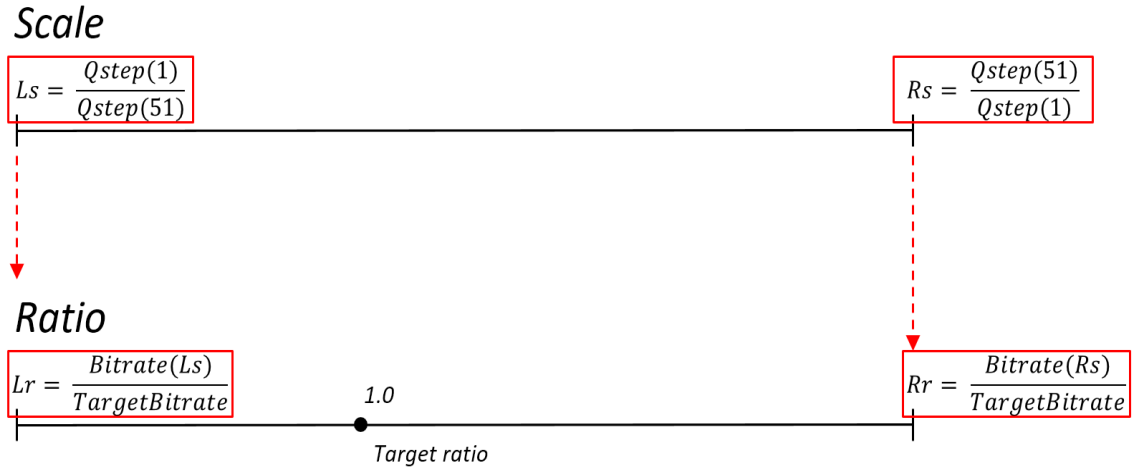
$$BitsPredicted[i] = \frac{QPstepOriginal[i]}{QPstepCalculated[i]} \cdot FrameSize[i]$$

$$Bitrate = \frac{1}{NFrames} \cdot \left(\sum_{FutureFrames} BitsPredicted[i] + \sum_{EncodedFrames} BitsEncoded[i] \right)$$

Where $NFrames$ – is a number of frames in current window, which is *NLookAheadFrames + NLookBackFrames*.

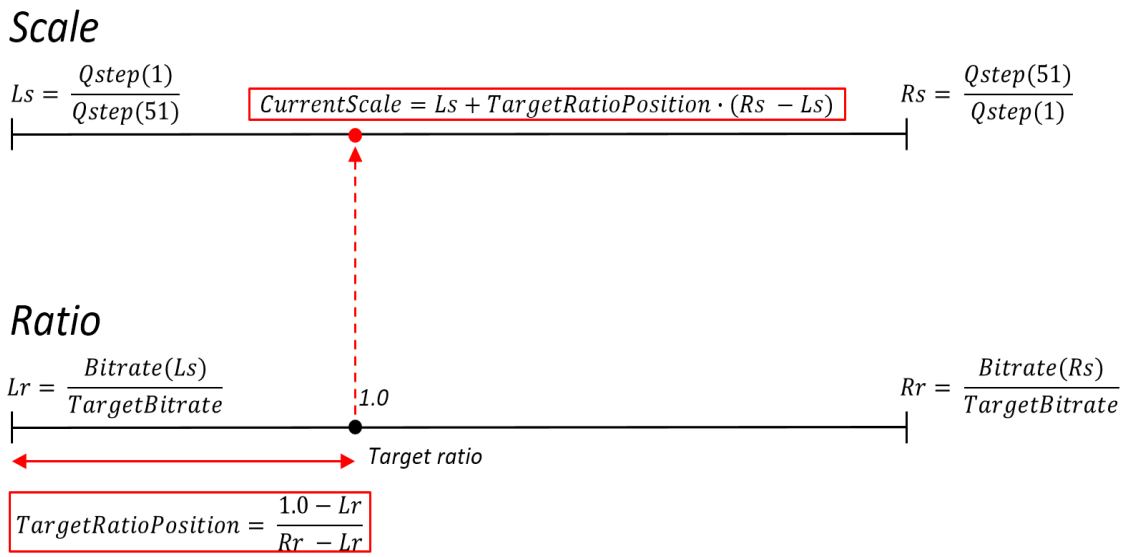
Search procedure checks the $Bitrate$ and if it doesn't match requested one it reruns with different $scale$.

At first, *scale* and bitrate *ratio* bounds initialized. The target is to reach ratio which equals (or very close) to 1.0.



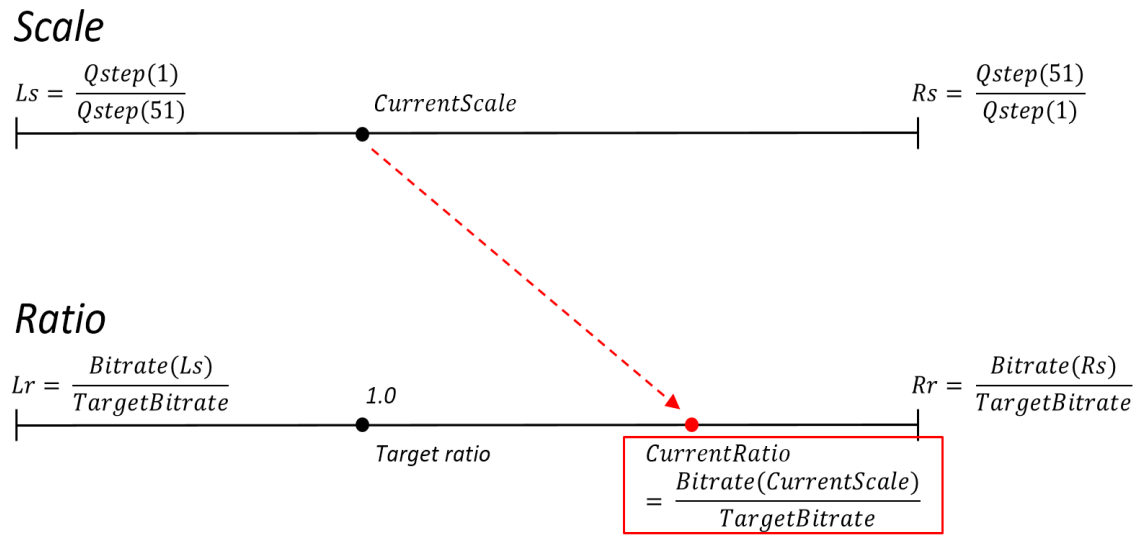
Here *Ls* and *Rs* are the left and right bounds of *scale*, represented by the lowest and biggest possible scale to apply. *Lr* and *Rr* are bounds of *ratio* corresponding to *scale* bounds. The target ratio 1.0 is somewhere between *Lr* and *Rr*.

Then projection of *target ratio* to *scale* is performed.



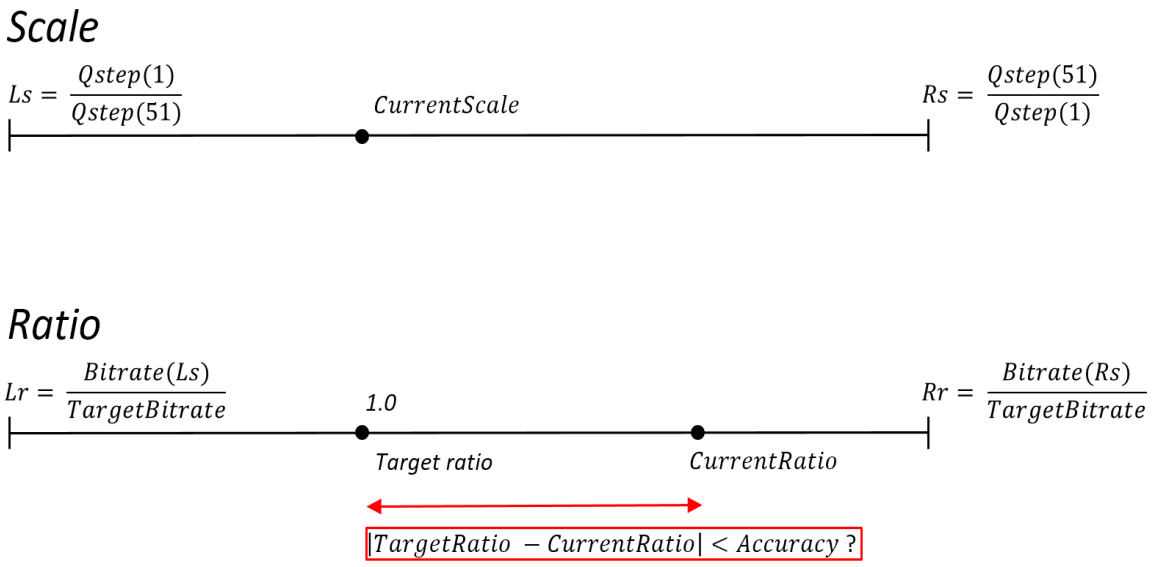
CurrentScale now is a candidate to be the *scale* which gives bitrate close to *target*.

The next step is projection of *CurrentRatio* back to the *ratio* axis.



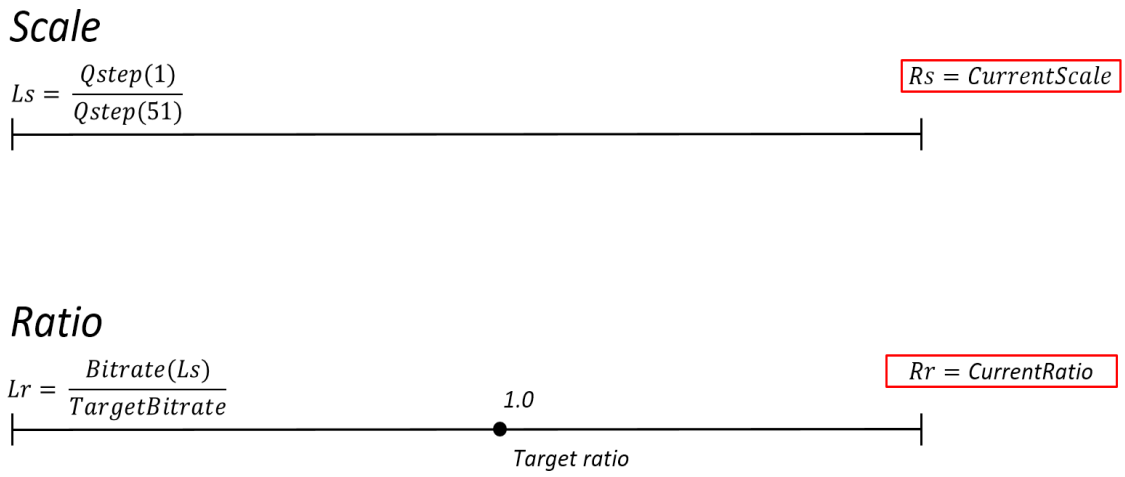
The scale to *ratio* projection (actually, bitrate calculation) procedure includes QPsteps calculation for all frames in the LA window.

Check if *CurrentRatio* is near *target* within some predefined accuracy.



If it is not (like on picture above, here bitrate exceeds *target*), move the next iteration of the search to one of the segments, which has *target ratio* inside.

Update one of the boundaries of *scale* and *ratio*. In the described situation, the next search will be performed in the left branch.



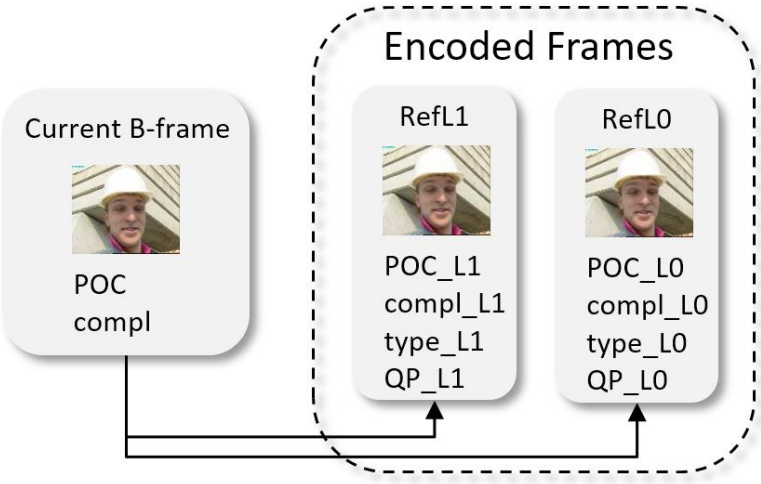
Repeat the procedure until convergence. At that moment, the final QPsteps are known.

4. After QPstep is obtained, convert it to QP with additional correction depending on the frame type, with the purpose of adding extra bits to non-**B**-frames. Multiply by the BitsEncoded / BitsPredicted ratio to compensate for prediction errors.

If the current frame is a non-**B**-frame, the resulting QP is just a simple formula:
$QP[i] = QPstep2QP\left(QPStep[i] \cdot \frac{BitsEncoded}{BitsPredicted}\right)$.

For **B**-frames, additional processing is performed (see explanation below).

Consider we have **B**-frame with 2 references (if there are more than 1 L0, only first one is taken into account) as in the picture below.



Where POC – is a current B-frame POC, compl – is a current B-frame complexity, POC_L0 (L1), compl_L0 (L1), QP_L0 (L1), type_L0 (L1) – are POC, complexity, QP and type of L0 (L1) references.

Correction is performed in two steps, where the first step depends on the type of L0 and L1 references and dedicated to QP smoothing.

Case 1: both references are **P**-frames.

Calculate the complexity distance in following way:

$$ComplDistL0 = \min\left(\frac{compl + 1.0}{compl\_L0 + 1.0}, \frac{compl\_L0 + 1.0}{compl + 1.0}\right).$$

This value indicates how close complexities are. For similar complexities, this value increases and tends to limit at 1.0 for identical complexities.

The formula for $ComplDistL1$ is similar.

POC distances are more straightforward:

$$POCdistL0 = POC - POC\_L0;$$
$$POCdistL1 = POC\_L1 - POC;$$

After that, QP is calculated as a weighted blending of reference frames QPs by following formula:

$$qp = \frac{qp\_L0 \cdot ComplDistL0 \cdot POCdistL0 + qp\_L1 \cdot ComplDistL1 \cdot POCdistL1}{ComplDistL0 \cdot POCdistL0 + ComplDistL1 \cdot POCdistL1}.$$

The same approach is used to calculate updated complexity:

$$ComplUpdated = \frac{compl\_L0 \cdot ComplDistL0 \cdot POCdistL0 + compl\_L1 \cdot ComplDistL1 \cdot POCdistL1}{ComplDistL0 \cdot POCdistL0 + ComplDistL1 \cdot POCdistL1}.$$

Case 2: only one of the references is a **P**-frame

In such a situation, values from that reference are simply inherited. For example, if the **P**-frame reference is the L0 reference, the following formula is used:

$$qp = qp\_L0;$$
$$ComplUpdated = compl\_L0.$$

Case 3:

Blend the QPs of references and keep the complexity of the current frame:

$$qp = \frac{qp\_L0 + qp\_L1}{2} + 6 \cdot log_2 1.4;$$
$$ComplUpdated = compl.$$

In the next step, additional adjustment is performed. QP is increased to save bits on **B**-frames encoding. It is done by adding some delta to QP:

$$\Delta qp_B = log_2\left(\frac{ComplUpdated}{compl} \cdot \frac{BitsEncoded}{BitsPredicted}\right) \cdot 6 \cdot 0.35;$$
$$qp += Clip3(0,6,\Delta qp_B).$$

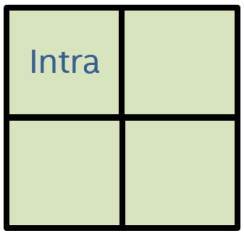
Where $Clip3$ truncates delta to 0 or 6 if it exceeds corresponding boundary.

5. After encoding, update the encoded frame statistics with the size of the encoded frame.

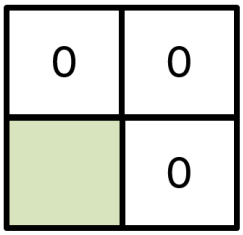
7.2 Pixel Propagation

This section provides more explanation of how pixel propagation is calculated.

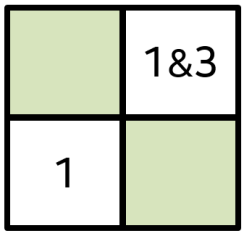
Let's consider four frames of 2x2 pixels resolution. Green blocks are intra pixels white are inter pixels. Numbers over inter pixels indicate reference frame numbers.



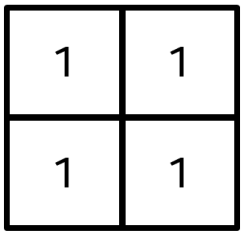
I-Frame
Display Order 0



P-Frame
Display Order 1



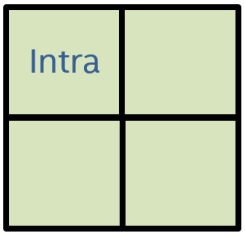
B-Frame
Display Order 2



P-Frame
Display Order 3

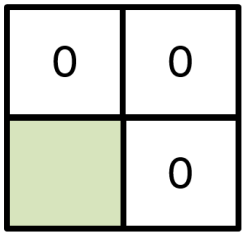
First, for each frame, we find how many pixels we predicted from each reference frame. For Bi-prediction, we take half of the pixels from each frame.

"0->3" means that we predict three pixels from frame number 0.



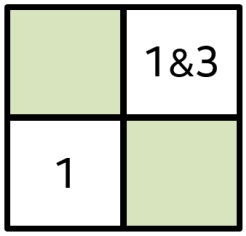
I-Frame
Display Order 0

Prediction map:
none



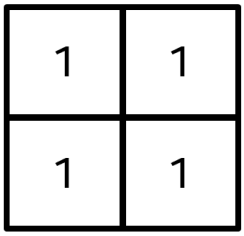
P-Frame
Display Order 1

Prediction map:
0 -> 3



B-Frame
Display Order 2

Prediction map:
1 -> 1.5
3 -> 0.5

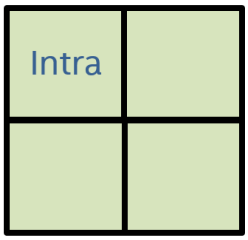


P-Frame
Display Order 3

Prediction map:
1 -> 4

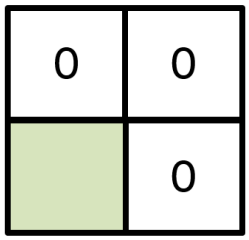
Then, for each frame, calculate the prediction share (share of total inter pixels predicted from each reference), which is the number of predicted pixels from a particular frame divided by the total number of inter pixels. It is a measure of prediction from each reference frame.

"3->0.5/2=0.25" means that 25% of the inter pixels in this frame are predicted from frame number 3.



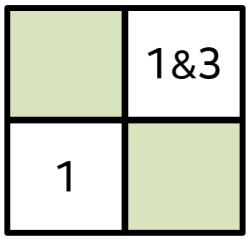
I-Frame
Display Order 0

Prediction map:
none
Prediction share:
none



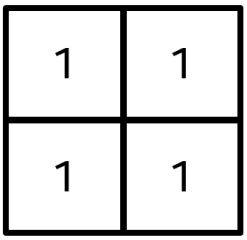
P-Frame
Display Order 1

Prediction map:
0 -> 3
Prediction share:
0 -> 3/3 = 1.0



B-Frame
Display Order 2

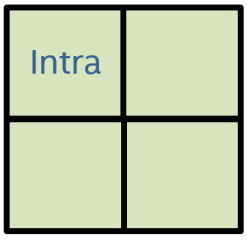
Prediction map:
1 -> 1.5
3 -> 0.5
Prediction share:
1 -> 1.5/2 = 0.75
3 -> 0.5/2 = 0.25



P-Frame
Display Order 3

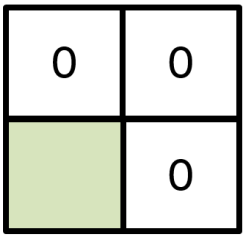
Prediction map:
1 -> 4
Prediction share:
1 -> 4/4 =1.0

On this step, we calculate how many pixels are directly propagated by each reference frame (i.e., for frame 0, how many 0 indicators are in following frames). For Bi-prediction, we take half of the pixels from each frame.



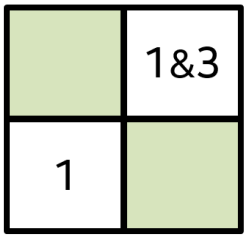
I-Frame
Display Order 0

Prediction map:
none
Prediction share:
none
How many propagated:
3.0



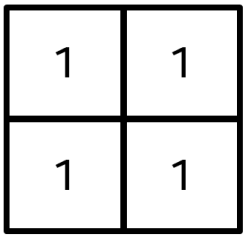
P-Frame
Display Order 1

Prediction map:
0 -> 3
Prediction share:
0 -> 3/3 = 1.0
Home many propagated:
1.5 + 4 = 5.5



B-Frame
Display Order 2

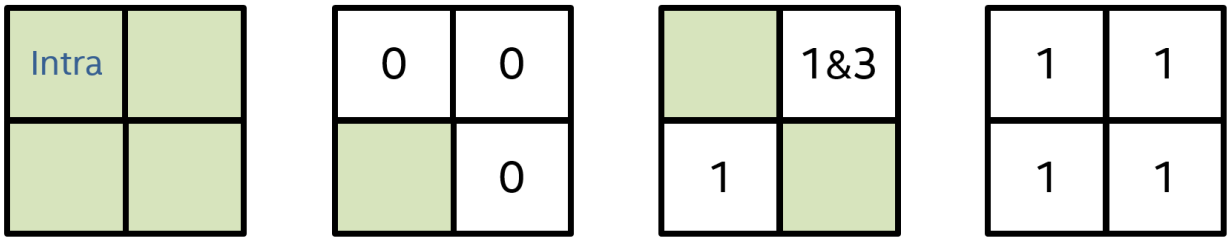
Prediction map:
1 -> 1.5
3 -> 0.5
Prediction share:
1 -> 1.5/2 = 0.75
3 -> 0.5/2 = 0.25
How many propagated:
0



P-Frame
Display Order 3

Prediction map:
1 -> 4
Prediction share:
1 -> 4/4 =1.0
How many propagated:
0.5

The tricky thing is taking into account indirect propagation, which is the number of pixels predicted from the current frame transitively by referencing other frames which have direct references to the current frame. In other words, if we consider three frames, each is a reference to only the previous one, **IPP**, and suppose, at each frame, 50% of pixels are predicted from the previous frame. First, **P** uses 50% of **I** pixels; second, **P** uses 50% of the first **P** pixels, but in the same moment it actually uses about 25% of the pixels of the **I** frame. So **I** frame is more important, since its pixels propagate more. From this point of view, any distortion in the **I** frame propagates more to the other frames.



I-Frame
Display Order 0

Prediction map:
none
Prediction share:
none
How many propagated:
3.0
How many propagated indirectly:
1 -> 1.0 * 5.5 = 5.5

P-Frame
Display Order 1

Prediction map:
0 -> 3
Prediction share:
0 -> 3/3 = 1.0
Home many propagated:
1.5 + 4 = 5.5
How many propagated indirectly:
none

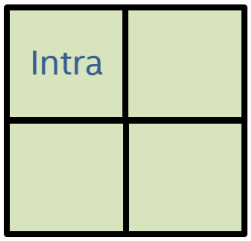
B-Frame
Display Order 2

Prediction map:
1 -> 1.5
3 -> 0.5
Prediction share:
1 -> 1.5/2 = 0.75
3 -> 0.5/2 = 0.25
How many propagated:
0
How many propagated indirectly:
none

P-Frame
Display Order 3

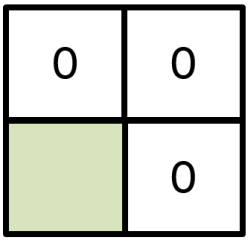
Prediction map:
1 -> 4
Prediction share:
1 -> 4/4 =1.0
How many propagated:
0.5
How many propagated indirectly:
none

The total propagation share is the sum of directly and indirectly propagated pixels divided by the total number of pixels, plus one for regularization (this is made to make non-ref B-frames have a total value equal to one, which makes it much simpler to use, because it is non-zero at any frame).



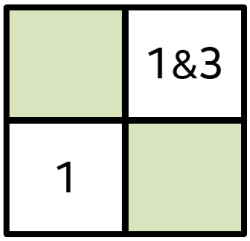
I-Frame
Display Order 0

Prediction map:
none
Prediction share:
none
How many propagated:
3.0
How many propagated indirectly:
1 -> 1.0 * 5.5 = 5.5
Share propagated total:
1.0 + (3.0 + 5.5)/4 = 3.125



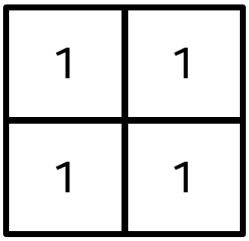
P-Frame
Display Order 1

Prediction map:
0 -> 3
Prediction share:
0 -> 3/3 = 1.0
Home many propagated:
1.5 + 4 = 5.5
How many propagated indirectly:
none
Share propagated total:
1.0 + 5.5/4 = 1.375



B-Frame
Display Order 2

Prediction map:
1 -> 1.5
3 -> 0.5
Prediction share:
1 -> 1.5/2 = 0.75
3 -> 0.5/2 = 0.25
How many propagated:
0
How many propagated indirectly:
none
Share propagated total:
1.0



P-Frame
Display Order 3

Prediction map:
1 -> 4
Prediction share:
1 -> 4/4 =1.0
How many propagated:
0.5
How many propagated indirectly:
none
Share propagated total:
1.0 + 0.5/4 = 1.125

8 APPENDIX C. SYSTEM CONFIGURATION

OS

- CentOS Linux* release 7.4.1708
- Kernel: 3.10.0-693.17.1.el7.x86_64
- UMD: 16.9.00092
- KMD: i915 1.6.0 20170818-00092-13ac7794

CPU

- Intel® Core™ i7-6770HQ CPU processor @ 1.80GHz
- CPU(s): 4
- Thread(s) per core: 2
- Stepping: 3
- L1d cache: 128 KB
- L1i cache: 128 KB
- L2 cache: 1 MB
- L3 cache: 6 MB
- L4 cache: 128 MB

GPU

- Intel® Iris™ Graphics 580
- eDRAM: 128 MB
- EU Count: 72
- Max Core Frequency: 950 MHz

Memory

- Number of channels: 1
- Size: 16384 MB
- Type: DDR4
- Speed: 2133 MHz

Three GPU slices (default) were enabled. CPU, GPU, and uncore frequencies were fixed. Performance mode and VDBox* load balancing were enabled. To do so, the following commands were executed:

```
echo 950 > /sys/class/drm/card0/gt_min_freq_mhz
echo 950 > /sys/class/drm/card0/gt_max_freq_mhz
echo 950 > /sys/class/drm/card0/gt_boost_freq_mhz

cpupower frequency-set -d 1800000 -u 1800000 -g performance

wrmsr 0x620 0x1212
wrmsr 0x621 0x12

echo -e "[KEY]\n\t0x00000001\n\tUFKEY_INTERNAL\\\n\t[VALUE]\n\t\t Enable VDBox
load balancing\n\t\t4\n\t\t1" > /etc/igfx_user_feature.txt
```

9 APPENDIX D. COMMAND LINES

This appendix provides command lines used during VQ and performance measurements. Sample applications can be found on GitHub here. A detailed description of command line parameters can be found in correspondent readme here.

9.1 CONVENTIONAL TRANSCODING

9.1.1 TU4

```
./sample_multi_transcode -i::h264 $AVC_INPUT -o::h265 $TU4_STREAM -cqp -qpi $QP
-qpp $QP -qpb $QP -DisableQPOffset  -u 4 -gop_size 32 -dist 4 -num_ref 4 -bref
-l 1 -hw -async 1
```

9.1.2 TU7

```
./sample_multi_transcode -i::h264 $AVC_INPUT -o::h265 $TU7_STREAM -cqp -qpi $QP
-qpp $QP -qpb $QP -DisableQPOffset  -u 7 -gop_size 32 -dist 4 -num_ref 4 -bref
-l 1 -hw -async 1
```

9.2 PREENC + ENCODE

9.2.1 TU4 encoding

```
./sample_hevc_fei -i $YUV_INPUT -w $WIDTH -h $HEIGHT -f $FPS -o $FEI_TU4_STREAM
-preenc 4 -encode -qp $QP -DisableQPOffset -g 32 -gpb:on -idr_interval 0 -
GopRefDist 4 -NumRefFrame 4 -NumRefActiveP 3 -NumRefActiveBL0 3 -NumRefActiveBL1
1 -bref -NumFramePartitions 4 -AdaptiveSearch -SearchWindow 5 -l 1 -EncodedOrder
-MVPBlockSize 1
```

9.2.2 TU4 transcoding

```
./sample_hevc_fei -i::h264 $AVC_INPUT -o $FEI_TU4_STREAM -preenc 4 -encode -qp
$QP -DisableQPOffset -g 32 -gpb:on -idr_interval 0 -GopRefDist 4 -NumRefFrame 4
-NumRefActiveP 3 -NumRefActiveBL0 3 -NumRefActiveBL1 1 -bref -NumFramePartitions
4 -AdaptiveSearch -SearchWindow 5 -l 1 -EncodedOrder -MVPBlockSize 1
```

9.2.3 TU7 encoding

```
./sample_hevc_fei -i $YUV_INPUT -w $WIDTH -h $HEIGHT -f $FPS  -o $FEI_TU7_STREAM
-preenc 4 -encode -qp $QP DisableQPOffset -g 32 -gpb:on -idr_interval 0 -
GopRefDist 4 -NumRefFrame 4 -NumRefActiveP 1 -NumRefActiveBL0 1 -NumRefActiveBL1
1 -bref -NumFramePartitions 4 -AdaptiveSearch -SearchWindow 5 -ForceCtuSplit -
FastIntraMode -l 1 -EncodedOrder -MVPBlockSize 1
```

9.2.4 TU7 transcoding

```
./sample_hevc_fei -i::h264 $AVC_INPUT -o $FEI_TU7_STREAM -preenc 4 -encode -qp
$QP DisableQPOffset -g 32 -gpb:on -idr_interval 0 -GopRefDist 4 -NumRefFrame 4 -
NumRefActiveP 1 -NumRefActiveBL0 1 -NumRefActiveBL1 1 -bref -NumFramePartitions
4 -AdaptiveSearch -SearchWindow 5 -ForceCtuSplit -FastIntraMode -l 1 -
EncodedOrder -MVPBlockSize 1
```

9.3 DSO + ENCODE

9.3.1 TU4 transcoding

```
./sample_hevc_fei_abr -i::h264 $AVC_INPUT -o $ABR_TU4_STREAM -dso
$5Mb_DSO_STREAM -qp $QP -DisableQPOffset -g 32 -gpb:on -idr_interval 0 -
GopRefDist 4 -NumRefFrame 4 -NumRefActiveP 3 -NumRefActiveBL0 3 -NumRefActiveBL1
1 -bref -l 1 -DSOMVPBlockSize 7 -NumFramePartitions 4
```

9.3.2 TU7 transcoding

```
./sample_hevc_fei_abr -i::h264 $AVC_INPUT -o $ABR_TU7_STREAM -dso
$5Mb_DSO_STREAM -qp $QP -DisableQPOffset -g 32 -gpb:on -idr_interval 0 -
GopRefDist 4 -NumRefFrame 4 -NumRefActiveP 1 -NumRefActiveBL0 1 -NumRefActiveBL1
1 -bref -l 1 -DSOMVPBlockSize 7 -NumFramePartitions 4 -ForceCtuSplit -
FastIntra:I -FastIntra:P -FastIntra:B
```

9.4 x264 ENCODING

```
./x264-r2762-90a61ec --input-res $WIDTHx$HEIGHT --bitrate 40000 --frames
$NFRAMES --fps $FPS --no-mbtree  -I 32 --no-fast-pskip --no-dct-decimate --no-
scenecut --ref 4 --bframes 3 --b-pyramid normal --b-adapt 0 --open-gop --weightp
0 --no-weightb --ipratio 1.4 --pbratio 1.3 --merange 24 --me umh --subme 11 --
partitions all --trellis 0 --no-psy --psnr --deblock 0:0 -o $AVC_INPUT
$YUV_INPUT
```

9.5 HM ENCODING

9.5.1 auxiliary stream for DSO

```
#======== File I/O =====================
#BitstreamFile                 : str.bin
#ReconFile                     : rec.yuv

#======== Profile ================
Profile                        : main

#======== Unit definition ================
MaxCUWidth                     : 32
MaxCUHeight                    : 32
MaxPartitionDepth              : 3
QuadtreeTULog2MaxSize          : 5
QuadtreeTULog2MinSize          : 2
QuadtreeTUMaxDepthInter        : 3
QuadtreeTUMaxDepthIntra        : 3

#======== Coding Structure =============
IntraPeriod                    : 32
DecodingRefreshType            : 1
GOPSize                        : 32
ReWriteParamSetsFlag           : 1

IntraQPOffset                  : -1
LambdaFromQpEnable             : 1

Frame1:  B  4  0 0.0 0.0 0 0 1.0 0 0 0 1 1 -4     0
Frame2:  B  2  1 0.0 0.0 0 0 1.0 0 0 0 1 2 -2  2    1 2 2 1 1
Frame3:  B  1  2 0.0 0.0 0 0 1.0 0 0 0 1 3 -1  1  3   1 1 3 1 1 1
Frame4:  B  3  2 0.0 0.0 0 0 1.0 0 0 0 2 3 -1 -3  1   1 -2 4 1 1 1 0
Frame5:  B  8  0 0.0 0.0 0 0 1.0 0 0 0 3 3 -4 -6 -8   1 -5 4 1 1 1 0
Frame6:  B  6  1 0.0 0.0 0 0 1.0 0 0 0 3 4 -2 -4 -6 2  1 2 4 1 1 1 1
Frame7:  B  5  2 0.0 0.0 0 0 1.0 0 0 0 2 4 -1 -3  1 3  1 1 5 1 1 0 1 1
Frame8:  B  7  2 0.0 0.0 0 0 1.0 0 0 0 3 4 -1 -3 -5 1  1 -2 5 1 1 1 1 0
Frame9:  B 12  0 0.0 0.0 0 0 1.0 0 0 0 3 3 -4 -6 -8   1 -5 5 1 1 0 1 0
Frame10: B 10  1 0.0 0.0 0 0 1.0 0 0 0 3 4 -2 -4 -6 2  1 2 4 1 1 1 1
Frame11: B  9  2 0.0 0.0 0 0 1.0 0 0 0 2 4 -1 -3  1 3  1 1 5 1 1 0 1 1
Frame12: B 11  2 0.0 0.0 0 0 1.0 0 0 0 3 4 -1 -3 -5 1  1 -2 5 1 1 1 1 0
Frame13: B 16  0 0.0 0.0 0 0 1.0 0 0 0 3 3 -4 -6 -8   1 -5 5 1 1 0 1 0
Frame14: B 14  1 0.0 0.0 0 0 1.0 0 0 0 3 4 -2 -4 -6 2  1 2 4 1 1 1 1
Frame15: B 13  2 0.0 0.0 0 0 1.0 0 0 0 2 4 -1 -3  1 3  1 1 5 1 1 0 1 1
Frame16: B 15  2 0.0 0.0 0 0 1.0 0 0 0 3 4 -1 -3 -5 1  1 -2 5 1 1 1 1 0
Frame17: B 20  0 0.0 0.0 0 0 1.0 0 0 0 3 3 -4 -6 -8   1 -5 5 1 1 0 1 0
Frame18: B 18  1 0.0 0.0 0 0 1.0 0 0 0 3 4 -2 -4 -6 2  1 2 4 1 1 1 1
Frame19: B 17  2 0.0 0.0 0 0 1.0 0 0 0 2 4 -1 -3  1 3  1 1 5 1 1 0 1 1
Frame20: B 19  2 0.0 0.0 0 0 1.0 0 0 0 3 4 -1 -3 -5 1  1 -2 5 1 1 1 1 0
```

```
Frame21: B 24  0 0.0 0.0 0 0 1.0 0 0 0 3 3 -4 -6 -8   1 -5 5 1 1 0 1 0
Frame22: B 22  1 0.0 0.0 0 0 1.0 0 0 0 3 4 -2 -4 -6 2  1 2 4 1 1 1 1
Frame23: B 21  2 0.0 0.0 0 0 1.0 0 0 0 2 4 -1 -3  1 3  1 1 5 1 1 0 1 1
Frame24: B 23  2 0.0 0.0 0 0 1.0 0 0 0 3 4 -1 -3 -5 1  1 -2 5 1 1 1 1 0
Frame25: B 28  0 0.0 0.0 0 0 1.0 0 0 0 3 3 -4 -6 -8   1 -5 5 1 1 0 1 0
Frame26: B 26  1 0.0 0.0 0 0 1.0 0 0 0 3 4 -2 -4 -6 2  1 2 4 1 1 1 1
Frame27: B 25  2 0.0 0.0 0 0 1.0 0 0 0 2 4 -1 -3  1 3  1 1 5 1 1 0 1 1
Frame28: B 27  2 0.0 0.0 0 0 1.0 0 0 0 3 4 -1 -3 -5 1  1 -2 5 1 1 1 1 0
Frame29: I 32 -1 0.0 0.0 0 0 1.0 0 0 0 0 3 -4 -6 -8   1 -5 5 1 1 0 1 0
Frame30: B 30  1 0.0 0.0 0 0 1.0 0 0 0 3 4 -2 -4 -6 2  1 2 4 1 1 1 1
Frame31: B 29  2 0.0 0.0 0 0 1.0 0 0 0 2 4 -1 -3  1 3  1 1 5 1 1 0 1 1
Frame32: B 31  2 0.0 0.0 0 0 1.0 0 0 0 3 4 -1 -3 -5 1  1 -2 5 1 1 1 1 0

#=========== Motion Search =============
FastSearch                     : 1
SearchRange                    : 512
ASR                            : 0
MinSearchWindow                : 96
BipredSearchRange              : 4
HadamardME                     : 1
FEN                            : 1
FDM                            : 1

#=========== Deblock Filter ============
LoopFilterOffsetInPPS          : 1
LoopFilterDisable              : 0
LoopFilterBetaOffset_div2      : 0
LoopFilterTcOffset_div2        : 0
DeblockingFilterMetric         : 0

#=========== Misc. ============
InternalBitDepth               : 8

#=========== Coding Tools =================
SAO                            : 0
AMP                            : 0
TransformSkip                  : 0
TransformSkipFast              : 0
SAOLcuBoundary                 : 0

#============ Slices ================
SliceMode                : 0
SliceArgument            : 1500
LFCrossSliceBoundaryFlag : 1

#============ PCM ================
PCMEnabledFlag                     : 0
PCMLog2MaxSize                     : 5
PCMLog2MinSize                     : 3
PCMInputBitDepthFlag               : 1
PCMFilterDisableFlag               : 0

#============ Tiles ================
TileUniformSpacing                 : 0
NumTileColumnsMinus1               : 0
NumTileRowsMinus1                  : 0
LFCrossTileBoundaryFlag            : 1

#============ WaveFront ================
WaveFrontSynchro                   : 0

#=========== Quantization Matrix =================
ScalingList                    : 0

#============ Lossless ================
TransquantBypassEnableFlag : 0
CUTransquantBypassFlagForce: 0

#============ Rate Control ======================
RateControl                        : 1
```

```
TargetBitrate                          : 5000000
KeepHierarchicalBit                    : 2
LCULevelRateControl                    : 0
RCLCUSeparateModel                     : 1
InitialQP                              : 0
RCForceIntraQP                         : 0
```

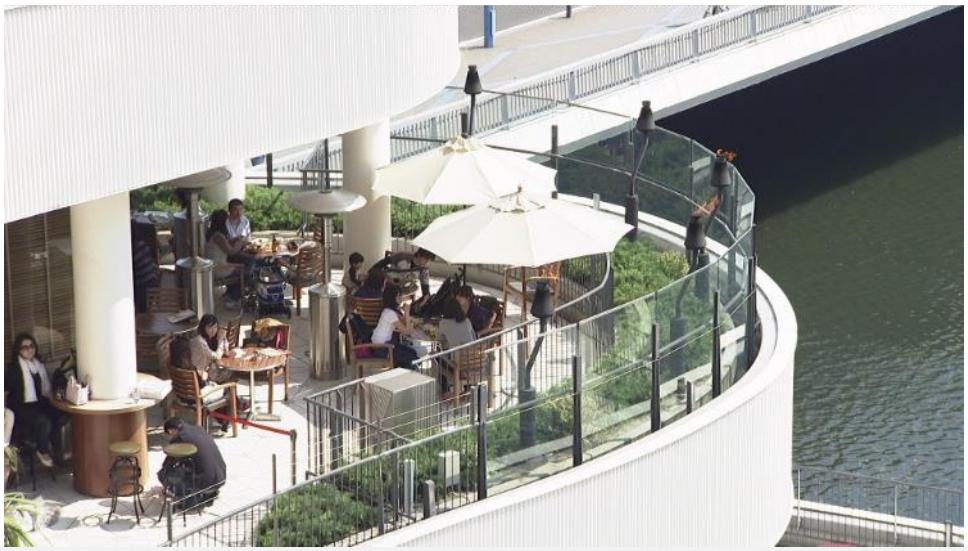
10 APPENDIX E. STREAM INFORMATION

Name: Beauty
Description: Closeup on female face, hair waving around. Black background
Source: http://ultravideo.cs.tut.fi/#testsequences, framerate conversion from 120 fps to 60 fps
Copyright: Digiturk

Name: big_buck_bunny
Description: Animation
Source: https://media.xiph.org/video/derf
Copyright: Blender Foundation / www.bigbuckbunny.org

Name: blue_sky
Description: Top of two trees against blue sky. Camera rotation.
Source: https://media.xiph.org/video/derf
Copyright:

Name: Bosphorus
Description: Zoomed in yacht, bridge on background. Panning right
Source: http://ultravideo.cs.tut.fi/#testsequences, framerate conversion from 120 fps to 60 fps
Copyright: Digiturk



Name: bq_terrace

Description: The camera pans in a diagonal direction from a terrace to the bridge. Plenty of vehicles moving on a bridge, and below the bridge are the water.

Source:

Copyright: NTT DOCOMO Inc.

Name: crowd_run

Description: A crowd of people running together, with big trees and the blue sky as the background.

Source: https://media.xiph.org/video/derf

Copyright: Sveriges Television AB (SVT), Sweden

Name: ducks_take_off
Description: Ducks are taking of water creating ripples effect.-
Source: https://media.xiph.org/video/derf
Copyright: Sveriges Television AB (SVT), Sweden

Name: elephants_dream
Description: Animation
Source: https://media.xiph.org/video/derf
Copyright: Blender Foundation / Netherlands Media Art Institute / www.elephantsdream.org

Name: HoneyBee, HoneyBee_fade25, HoneyBee_fade50, HoneyBee_fade100
Description: Bee harvesting flowers
Source: http://ultravideo.cs.tut.fi/#testsequences, framerate conversion from 120 fps to 60 fps
Copyright: Digiturk

Name: in_to_tree
Description: Camera approaching an old castle building and a tree next to it.-
Source: https://media.xiph.org/video/derf
Copyright: Sveriges Television AB (SVT), Sweden

Name: Jockey, Jockey_fade25, Jockey_fade50, Jockey_fade100
Description: Horse racing with camera panning to the left to follow
Source: http://ultravideo.cs.tut.fi/#testsequences, framerate conversion from 120 fps to 60 fps
Copyright: Digiturk

Name: kimono
Description: A woman walking slowly toward the camera in front of the woods. -
Source:
Copyright: Tokyo Institute of Technology, Nakajima Laboratory

Name: old_town_cross
Description: Panning view over the old town. Detailed houses, water and moving cars.-
Source: https://media.xiph.org/video/derf
Copyright: Sveriges Television AB (SVT), Sweden

Name: park_joy
Description: People are running in front of trees during a left to right camera movement.-
Source: https://media.xiph.org/video/derf
Copyright: Sveriges Television AB (SVT), Sweden

Name: park_scene
Description: People cycling in park. Panning right.-
Source:
Copyright: Tokyo Institute of Technology, Nakajima Laboratory

Name: pedestrian_area
Description: Shot of a pedestrian area. Low camera position, people pass by very close to the camera. Static camera.
Source: https:/media.xiph.org/video/derf
Copyright:

Name: ReadySteadyGo
Description: Horse racing track, riders getting ready for launch. The gates open and horses are running to the left
Source: http://ultravideo.cs.tut.fi/#testsequences, framerate conversion from 120 fps to 60 fps
Copyright: Digiturk

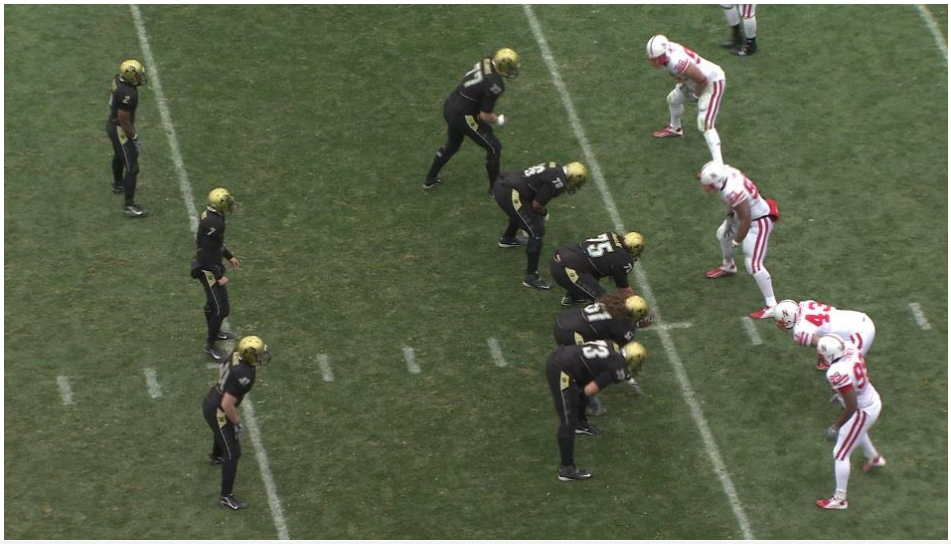
Name: riverbed
Description: Riverbed seen through the water.-
Source: https:/media.xiph.org/video/derf
Copyright:

Name: rush_hour
Description: Many cars moving slowly, high depth of focus. Fixed camera.-
Source: https:/media.xiph.org/video/derf
Copyright:

Name: station
Description: View from a bridge to railway station. Evening shot. Long zoom out. Many details, regular structures (tracks).
Source: https:/media.xiph.org/video/derf
Copyright:

Name: sunflower
Description: One bee at the sunflower, small color differences and very bright yellow. Fixed camera, small global motion.
Source: https:/media.xiph.org/video/derf
Copyright:

Name: tears-of-steel
Description: -Movie
Source: https:/media.xiph.org/video/derf
Copyright: Blender Foundation / mango.blender.org



Name: TouchDownPass
Description: American football.
Source: https:/media.xiph.org/video/derf, conversion from 422 to 420
Copyright: NTIA/ITS

Name: tractor, tractor_fade25, tractor_fade50, tractor_fade100
Description: A tractor in a field. Whole sequence contains parts that are very zoomed in and a total view. Camera is following the tractor, chaotic object movement, structure of a harvested field.
Source: https:/media.xiph.org/video/derf
Copyright:

Name: YachtRide
Description: Zoomed in yacht moves away from shot. Wavy water
Source: http://ultravideo.cs.tut.fi/#testsequences, framerate conversion from 120 fps to 60 fps
Copyright: Digiturk

LEGAL DISCLAIMER

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting **Intel's Web Site**.

MPEG is an international standard for video compression/decompression promoted by ISO. Implementations of MPEG CODECs, or MPEG enabled platforms may require licenses from various entities, including Intel Corporation.

Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

OPTIMIZATION NOTICE

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel.

Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

POST-PATCH DISCLAIMER

The benchmark results reported below may need to be revised as additional testing is conducted. The results depend on the specific platform configurations and workloads utilized in the testing, and may not be applicable to any particular user's components, computer system or workloads. The results are not necessarily representative of other benchmarks and other benchmark results may show greater or lesser impact from mitigations.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to www.intel.com/benchmarks.