Tuning Quality of Encoded Video

Table of Contents

The quality of encoded video depends on various factors. It is primarily a function of the target bit rate and the type of video content. However, there are some encoder parameters which can be used to adjust the video quality. This document describes the major parameters impacting video quality. These paramaters and the underlying concepts are applicable whether using FFmpeg, GStreamer or the C XMA APIs.

Various examples illustrating the effect of these settings can be found here:

- FFmpeg quality analysis examples.
- GStreamer quality analysis examples.

Number of B Frames

The default number of B frames is 2, but for most streams, the optimal number of B frames is 1. This provides the best tradeoffs for both video quality and objective quality use cases. The number of B frames can be adjusted according to the amount of motion in the video content. Generally, more B-frames helps compression, but hurts very high motion scenes. Xilinx recommends the following B frames settings:

- 0 for gaming clips with fast motion, camera pan/rotation scenes
- 2 for static or slow moving scenes, talking heads, or video conferencing type of content
- 1 for all medium motion and all other content

To set the number of B frames:

- When using FFmpeg, use the `-bf` option
- When using GStreamer, use the `b-frames` option

The number of B frames can also be adjusted dynamically. Consult the Dynamic Encoder Parameters section for more details on how to change this option during runtime.

Lookahead

Lookahead is used to improve the accuracy of rate control by enabling the encoder to buffer a specified number of frames (using the parameter). Spatial and temporal complexity measures are computed for these frames. The rate control uses these measures to distribute more bits to frames which are hard to encode, and less bits to frames which are easy to encode. This redistribution results in better video quality. For spatial and temporal adaptive quantization to take effect, the lookahead depth must be set to at least 1. When latency is tolerable in applications, Xilinx recommends a lookahead depth of 20 frames to get optimum video quality.

To enable lookahead:

- When using FFmpeg, use the `-lookahead_depth` option
- When using GStreamer, use the `lookahead-depth` option

Adaptive Quantization

This feature improves the video qualitity by changing the quantization parameter (QP) within a frame. The QP for each frame is determined by the rate control, and adaptive quantization (AQ) adjusts QP on top of that for different regions within a frame. It exploits the fact that the human eye is more sensitive to certain regions of a frame and redistributes more bits to those regions. The Xilinx video encoders support two types of AQ: Spatial Adaptive Quantization and Temporal Adaptive Quantization.

Adaptive Quantization is enabled by setting `-qp-mode` to `relative-load` or `auto` (default) and when `-lookahead_depth` >= 1. When AQ is enabled, Spatial AQ and Temporal AQ can be individually controlled as described in the following sections.

Adaptive Quantization is disabled by setting `-qp-mode` to `uniform`. In this case, all CU's within the frame are quantized using the same QP.

Spatial Adaptive Quantization

Spatial AQ adjusts the QP within a frame based on the spatial characteristics. The human eye is more sensitive to regions which are flat and have low texture than regions which have lots of detail and texture. Spatial AQ exploits this and provides more bits to the low texture and flat regions at the expense of high texture regions. This redistribution of bits to visually perceptible regions of the frame brings about visual improvement. Although spatial AQ improves video qualitity, it hurts objective metrics and causes a drop in PSNR and VMAF. It is recommended to turn this feature off when performing PSNR/VMAF based evaluation.

The spatial AQ algorithm can be controlled using the spatial AQ gain option. The range of this option is from 0 to 100 and indicates the strength of the redistribution of data within the frame as a percentage. The default gain is 50.

To control spatial AQ:

- When using FFmpeg, use the `-spatial-aq` option to enable or disable spatial AQ, and use the `-spatial-aq-gain` option to set the gain
- When using GStreamer, use the `spatial-aq` option to enable or disable spatial AQ, and use the `spatial-aq-gain` option to set the gain

Spatial quantization can also be adjusted dynamically. Consult the Dynamic Encoder Parameters section for more details on how to change this option during runtime.

Temporal Adaptive Quantization

Temporal AQ adjusts the QP based on the temporal characteristics of the sequence. It utilizes the lookahead frames to capture the temporal characteristics where static/low motion or background is differentiated with high motion regions. The high motion regions are not very sensitive to the human eye as compared with low motion regions. Temporal AQ exploits this fact and redistributes more bits to static or low motion regions.

To control temporal AQ:

- When using FFmpeg, use the `-temporal-aq` option to enable or disable temporal AQ
- When using GStreamer, use the `temporal-aq` option to enable or disable temporal AQ

Temporal quantization can also be adjusted dynamically. Consult the Dynamic Encoder Parameters section for more details on how to change this option during runtime.

Scaling List

Scaling list offers a mechanism to scale the transform coefficients by specifying scaling matrices. This influences the quality of encoded video. There are two options to specify the scaling lists mode: flat (0) and default (1).

For subjective video quality, the scaling list mode must be set to default. The default scaling mode gives more importance to low-frequency coefficients and less importance to high-frequency coefficients.

To improve the objective numbers (such as PSNR and VMAF), the scaling mode must be set to flat, where all the coefficients are scaled equally.

To control the scaling list mode:

- When using FFmpeg, use the `-scaling-list` option (0 = flat, 1 = default)
- When using GStreamer, use the `scaling-list` option (0 = flat, 1 = default)

Dynamic Encoder Parameters

Dynamic parameters are parameters which can be changed during runtime. This is useful to optimize video quality and compression rate for different segments of the video. This capability is supported through FFmpeg, GStreamer and the Video SDK C APIs.

The following encoder parameters can be dynamically modified:

- Number of B frames (0 to 4)
- Bitrate (1000 to INT_MAX)
- Temporal AQ mode (0 to 1)
- Spatial AQ mode (0 to 1)
- Spatial AQ gain (0 to 100)

When using FFmpeg or GStreamer, the encoder parameters which should be changed dynamically are specified in a configuration file as key-value pairs. This means that these key-value pairs must be known ahead of time. While this can used for file-based processing, this method is primarily intended as a testing mechanism.

When using the C APIs, the dynamic encoder parameters are updated on a frame-by-frame basis using side data. With this approach the top-level application can analyze the incoming frames, determine how to adjust the encoder parameters and update them before encoding the frame. This is primary intended use case for this capability.

Considerations regarding dynamic parameters settings:

- Recommended settings for dynamic B frames are:
  - 0 for gaming clips with fast motion, camera pan/rotation scenes
  - 2 for static or slow moving scenes, talking heads, or video conferencing type of content
  - 1 for all medium motion and all other content
- Dynamic B frames changes do not happen at the exact frame number specified. Instead, the change comes into effect one or two frames from the actual frame number specified in the config file.
- The maximum value for the number of B frames is the value configured at initilization
- In low latency mode, a dynamic bitrate changes is reflected after 1 or 2 GOPs. In normal latency mode, a bitrate change occurs within the next second.
- The configuration files for dynamic parameters must comply with the format specified above. Ill-formed files may result in unexpected behavior.

FFmpeg

Encoder parameters which should be changed dynamically are specified as key-value pairs in a configuration file. This configuration file is provided to FFmpeg using the `-expert-options` encoder switch with the `dynamic-params=<file>` option.

The `-expert-options` option is specific to an encoded output. For use cases with multiple encoded outputs (such as ABR ladders), each output can have its own `-expert-options` option and associated configuration file.

The configuration file should contain one line for each frame where one or more parameters are changed. Each line should start with the frame number followed by a list of key-value pairs for all the modified parameters:

```
<frameNumberN1>:<key1>=<value1>
<frameNumberN2>:<key2>=<value2>,<key3>=<value3>
```

Below is a table listing the parameters which can be changed at runtime, the corresponding key and valid values.

Dynamic Parameter	Key	Valid Values
Number of B frames	`NumB=<int>`	0 to 4
Bitrate (in bits per second)	`BR=<int>`	1000 to INT_MAX
Temporal AQ mode	`tAQ=<int>`	0 to 1
Spatial AQ mode	`sAQ=<int>`	0 to 1
Spatial AQ gain	`sAQGain=<int>`	0 to 100

Sample FFmpeg encode command:

```
ffmpeg -c:v mpsoc_vcu_hevc -i input.h265 -c:v mpsoc_vcu_hevc -b:v 2.5M -expert-options
dynamic-params=dynparams.txt -y output.h265
```

Sample configuration file for dynamic parameters:

```
300:NumB=1
600:BR=6000000
1200:sAQ=1,sAQGain=50
1800:tAQ=1
2400:NumB=0,BR=10000000,sAQ=0,sAQGain=50,tAQ=0
```

GStreamer

Encoder parameters which should be changed dynamically are specified as key-value pairs in a JSON configuration file, as follows:

```
"dynamic_params" :[
{
  "frame" : 600,
  "b-frames" : 2,
  "bitrate" : 6000,
  "temporal-aq" : false,
  "spatial-aq" : true,
  "spatial-aq-gain" : 50
},
{
  "frame" : 1500,
  "b-frames" : 0,
  "bitrate" : 3000
  "temporal-aq" : true,
  "spatial-aq" : true,
  "spatial-aq-gain" : 50
}
]
```

Refer to the GStreamer ABR Ladder Application for a working example leveraging dynamic parameters with GStreamer. This example includes a complete JSON file for configuring the dynamic parameters and a description of command line usage.

C APIs

Encoder parameters which should be changed dynamically must be specified in a data structure called `XlnxDynParams` and defined as follows:

struct **`XlnxDynParams`**

```
typedef struct XlnxLaDynParams {
    bool     is_spatial_gain_changed;
    uint32_t spatial_aq_gain;
    bool     is_temporal_mode_changed;
    bool     temporal_aq_mode;
    bool     is_spatial_mode_changed;
    bool     spatial_aq_mode;
} XlnxLaDynParams;

typedef struct XlnxEncDynParams {
    bool     is_bitrate_changed;
    uint32_t bit_rate;
    bool     is_bframes_changed;
    uint8_t  num_b_frames;
} XlnxEncDynParams;

typedef struct XlnxDynParams {
    XlnxEncDynParams enc_dyn_param;
    XlnxLaDynParams  la_dyn_param;
} XlnxDynParams;
```

The `XlnxDynParams` data structure is then passed along as side-data to the encoded frame using the `xma_frame_add_side_data()` API. The encoder plugin then takes care of updating the parameters automatically.

For of an example of how this can be implemented, refer to lines 279-291 of the examples/xma/common/enc_utils/src/xlnx_enc_dyn_params.c#L289 file of the XMA sample encoder application.

XMA Encoder and Transcoder Applications

The XMA Encoder and Transcoder sample applications also feature support for dynamically changing encoder parameters. These sample applications are developed using the Video SDK C APIs. Dynamic parameters are handled in the same way as for FFmpeg: they are specified in a configuration file following the same syntax as for FFmpeg, and the configuration file is passed to the application using the `-expert-options dynamic-params=<file>` option.

XMA H.264 Encode Example:

```
u30_xma_encode -w 1920 -h 1080 -pix_fmt yuv420p -i input_1080.yuv -fps 60 -g 120 -
periodicity-idr 120 -frames 600 -c:v mpsoc_vcu_h264 -expert-options dynamic-
params=cmdfile.txt -lookahead-depth 20 -spatial-aq 1 -temporal-aq 1 -spatial-aq-gain 80
-o out1.264
```

XMA HEVC Encode Example:

```
u30_xma_encode -w 1920 -h 1080 -pix_fmt yuv420p -i input_1080.yuv -fps 60 -g 120 -
periodicity-idr 120 -frames 600 -c:v mpsoc_vcu_hevc -expert-options dynamic-
params=cmdfile.txt -lookahead-depth 20 -spatial-aq 1 -temporal-aq 1 -spatial-aq-gain 80
-o out2.265
```

XMA ABR Transcode Example:

```
u30_xma_transcode -c:v mpsoc_vcu_h264 -i sample.h264 -multiscale_xma -num-output 2 -
out_1_width 1280 -out_1_height 720 -out_2_width 848 \
                  -c:v mpsoc_vcu_h264 -b:v 3000K -expert-options dynamic-
params=cmdfile.txt -qp-mode 2 -lookahead-depth 16 -temporal-aq 1 -spatial-aq 1 -
spatial-aq-gain 75 -o out_dp_test1.h264 \
                  -c:v mpsoc_vcu_h264 -b:v 4000K -expert-options dynamic-
params=cmdfile.txt -qp-mode 2 -lookahead-depth 16 -temporal-aq 1 -spatial-aq 1 -
spatial-aq-gain 75 -o out_dp_test2.h264
```

Dynamic IDR Frame Insertion

The dynamic IDR frame insertion feature allows encoding any incoming frame as IDR at runtime. This capability is supported through FFmpeg, GStreamer and the Video SDK C APIs.

Dynamic IDR frame insertion resets the GOP position. That is, if the stream is running with a GOP size of 120 (an IDR frame every 120 frames) and if an IDR frame is dynamically inserted at frame 50, the next IDR frame would be at frame 170 instead of 120.

To control dynamic IDR frame insertion:

- When using FFmpeg, use the `-force_key_frames` option
- When using GStreamer, use the `--forcekeyframe` option to `vvas_xabrladder` application
- When using the C APIs, set the `is_idr` flag in the `XmaFrame` data structure of frame to be encoded as IDR

FFmpeg

To control dynamic IDR frame insertion in FFmpeg, use the `-force_key_frames` option. This option can take either a list of timestamps (time,[time...]) or an expression (expr:*expr*).

- When using timestamps, FFmpeg will round the specified times to the nearest output timestamp as per the encoder time base and force a keyframe at the first frame having timestamp equal or greater than the computed timestamp.
- When using an expression, the string is interpreted like an expression and is evaluated for each frame. A key frame is forced in case the evaluation is non-zero.

Example using timestamps - Forcing IDR frames at specific time intervals:

```
ffmpeg -re -c:v mpsoc_vcu_hevc -i input.mp4 -c:v mpsoc_vcu_h264 –b:v 5M –
force_key_frames 1,2,3,5,8,13,21 -f mp4 out.mp4
```

Example using expressions - Forcing an IDR frame every 5 seconds:

```
ffmpeg -re -c:v mpsoc_vcu_hevc -i input.mp4 -c:v mpsoc_vcu_h264 –b:v 5M -
force_key_frames expr:gte(t,n_forced*5) -f mp4 out.mp4
```

For more details on using timestamps or expressions, see -force_key_frames in FFmpeg documentation (https://ffmpeg.org/ffmpeg-all.html).

GStreamer

To control IDR frame insertion in GStreamer, use the `--forcekeyframe` option in vvas_xabrladder application. This option takes IDR frame insertion frequency in number of frames.

Example to force an IDR frame every 30 frames:

```
vvas_xabrladder --devidx 0 --lookahead_enable 0 --codectype 1 --forcekeyframe 30 --file
<path to input file>
```

C APIs

To encode an incoming frame as IDR, the application needs to set the `is_idr` flag in the `XmaFrame` structure which is being sent to the lookahead module and then to the encoder:

```
XmaFrame la_in_frame;

la_in_frame.is_idr = 1;
```

For a complete source code reference, refer to `xlnx_enc_set_if_idr_frame()` function in the examples/xma/transcoder/lib/src/xlnx_transcoder.c file of the XMA transcoder application.

XMA Encoder and Transcoder Applications

In the XMA encoder and transcoder applications, use the -force_key_frame command line option to set IDR frames at specific frame numbers:

```
u30_xma_transcode -c:v mpsoc_vcu_h264 -i input.264 -c:v mpsoc_vcu_h264 -b:v 6000K -
force_key_frame "(200,300,250,200,180,400)" -o out.264
```

Objective Metrics versus Subjective Quality

Encoder settings can be used to optimize for objective quality (metrics such as PSNR, SSIM or VMAF) or subjective quality (visual appeal). This section summarizes recommended settings for these two goals.

Optimizing for Objective Metrics

To optimize for objective metrics, Adaptive Quantization should be disabled and a flat scaling list should be used. Doing so provides equal importance to all the blocks in the frame: the same quantization parameters and transform coefficients are used for all of them.

- When using FFmpeg, the `-tune-metrics` option can be set to 1 to automatically disable AQ and use a flat scaling list.

- When using GStreamer, the `qp-mode` and `scaling-list` options must be explicitly set.

Optimizing for Subjective Quality

To optimize for subjective quality, Adaptive Quantization should be enabled and the default scaling list should be used.

Adaptive Quantization (AQ) exploits the fact that the human eye is more sensitive to certain regions of a frame. This method drops information from high-frequency locations and keeps more information in low-frequency locations in a frame. The result appears more visually appealing. The Spatial AQ Gain parameter controls the strength of the redistribution of data within the frame. Setting too high a value may have a consequence of blurring edges. Experimentation across your clips is recommended if you wish to tune this parameter.

The default scaling list is used to scale up low-frequency data in the stream such that when it is quantized down during the encoding process, detail is retained.

For more details about Adaptive Quantization and the Scaling List, refer to the corresponding sections.