x265

Release

Jul 26, 2017

Contents

CHAPTER 1

Introduction

Increasing demand for high definition and ultra-high definition video, along with an increasing desire for video on demand has led to exponential growth in demand for bandwidth and storage requirements. These challenges can be met by the new High Efficiency Video Coding (HEVC) standard, also known as H.265. The x265 HEVC encoder project was launched by MulticoreWare in 2013, aiming to provide the most efficient, highest performance HEVC video encoder.

About HEVC

The High Efficiency Video Coding (HEVC) was developed by the ISO/IEC Moving Picture Experts Group (MPEG) and ITU-T Video Coding Experts Group (VCEG), through their Joint Collaborative Team on Video Coding (JCT-VC). HEVC is also known as ISO/IEC 23008-2 MPEG-H Part 2 and ITU-T H.265. HEVC provides superior video quality and up to twice the data compression as the previous standard (H.264/MPEG-4 AVC). HEVC can support 8K Ultra High Definition video, with a picture size up to 8192x4320 pixels.

About x265

The primary objective of x265 is to become the best H.265/HEVC encoder available anywhere, offering the highest compression efficiency and the highest performance on a wide variety of hardware platforms. The x265 encoder is available as an open source library, published under the GPLv2 license. It is also available under a commercial license, enabling commercial companies to utilize and distribute x265 in their solutions without being subject to the restrictions of the GPL license.

x265 is developed by MulticoreWare, leaders in high performance software solutions, with backing from leading video technology providers including Telestream and Doremi Labs (and other companies who want to remain anonymous at this time), and with contributions from open source developers. x265 leverages many of the outstanding video encoding features and optimizations from the x264 AVC encoder project.

The x265 software is available for free under the GNU GPL 2 license, from https://bitbucket.org/multicoreware/x265. For commercial companies that wish to distribute x265 without being subject to the open source requirements of the

GPL 2 license, commercial licenses are available with competitive terms. Contact license @ x265.com to inquire about commercial license terms.

While x265 is primarily designed as a video encoder software library, a command-line executable is provided to facilitate testing and development. We expect x265 to be utilized in many leading video hardware and software products and services in the coming months.

LEGAL NOTICES

The x265 software is owned and copyrighted by MulticoreWare, Inc. MulticoreWare is committed to offering the x265 software under the GNU GPL v2 license. Companies who do not wish to integrate the x265 Software in their products under the terms of the GPL license can contact MulticoreWare (license @ x265.com) to obtain a commercial license agreement. Companies who use x265 under the GPL may also wish to work with MulticoreWare to accelerate the development of specific features or optimized support for specific hardware or software platforms, or to contract for support.

The GNU GPL v2 license or the x265 commercial license agreement govern your rights to access the copyrighted x265 software source code, but do not cover any patents that may be applicable to the function of binary executable software created from the x265 source code. You are responsible for understanding the laws in your country, and for licensing all applicable patent rights needed for use or distribution of software applications created from the x265 source code. A good place to start is with the Motion Picture Experts Group - Licensing Authority - HEVC Licensing Program.

CHAPTER 2

Command Line Options

Note that unless an option is listed as **CLI ONLY** the option is also supported by x265_param_parse(). The CLI uses getopt to parse the command line options so the short or long versions may be used and the long options may be truncated to the shortest unambiguous abbreviation. Users of the API must pass x265_param_parse() the full option name.

Preset and tune have special implications. The API user must call x265_param_default_preset() with the preset and tune parameters they wish to use, prior to calling x265_param_parse() to set any additional fields. The CLI does this for the user implicitly, so all CLI options are applied after the user's preset and tune choices, regardless of the order of the arguments on the command line.

If there is an extra command line argument (not an option or an option value) the CLI will treat it as the input filename. This effectively makes the *--input* specifier optional for the input file. If there are two extra arguments, the second is treated as the output bitstream filename, making *--output* also optional if the input filename was implied. This makes **x265 in.y4m out.hevc** a valid command line. If there are more than two extra arguments, the CLI will consider this an error and abort.

Generally, when an option expects a string value from a list of strings the user may specify the integer ordinal of the value they desire. ie: *--log-level* 4 is equivalent to *--log-level* debug.

Executable Options

--help, -h
Display help text

CLI ONLY

--version, -V
Display version details

CLI ONLY

Command line executable return codes:

```
0. encode successful
1. unable to parse command line
2. unable to open encoder
3. unable to generate stream headers
4. encoder abort
```

Logging/Statistic Options

--log-level <integer|string>
Logging level. Debug level enables per-frame QP, metric, and bitrate logging. If a CSV file is being generated, frame level makes the log be per-frame rather than per-encode. Full level enables hash and weight logging. -1 disables all logging, except certain fatal errors, and can be specified by the string "none".

0.error

1.warning

2.info **(default)**

3.frame

4.debug

5.full

--no-progress
Disable periodic progress reports from the CLI

CLI ONLY

--csv <filename>
Writes encoding results to a comma separated value log file. Creates the file if it doesnt already exist, else adds one line per run. if *--log-level* is frame or above, it writes one line per frame. Default none

When frame level logging is enabled, several frame performance statistics are listed:

DecideWait ms number of milliseconds the frame encoder had to wait, since the previous frame was retrieved by the API thread, before a new frame has been given to it. This is the latency introduced by slicetype decisions (lookahead).

Row0Wait ms number of milliseconds since the frame encoder received a frame to encode before its first row of CTUs is allowed to begin compression. This is the latency introduced by reference frames making reconstructed and filtered rows available.

Wall time ms number of milliseconds between the first CTU being ready to be compressed and the entire frame being compressed and the output NALs being completed.

Ref Wait Wall ms number of milliseconds between the first reference row being available and the last reference row becoming available.

Total CTU time ms the total time (measured in milliseconds) spent by worker threads compressing and filtering CTUs for this frame.

Stall Time ms the number of milliseconds of the reported wall time that were spent with zero worker threads, aka all compression was completely stalled.

Avg WPP the average number of worker threads working on this frame, at any given time. This value is sampled at the completion of each CTU. This shows the effectiveness of Wavefront Parallel Processing.

Row Blocks the number of times a worker thread had to abandon the row of CTUs it was encoding because the row above it was not far enough ahead for the necessary reference data to be available. This is more of a problem for P frames where some blocks are much more expensive than others.

--cu-stats, --no-cu-stats
Records statistics on how each CU was coded (split depths and other mode decisions) and reports those statistics at the end of the encode. Default disabled

--ssim, --no-ssim
Calculate and report Structural Similarity values. It is recommended to use *--tune* ssim if you are measuring ssim, else the results should not be used for comparison purposes. Default disabled

--psnr, --no-psnr
Calculate and report Peak Signal to Noise Ratio. It is recommended to use *--tune* psnr if you are measuring PSNR, else the results should not be used for comparison purposes. Default disabled

Performance Options

--asm <integer:false:string>, **--no-asm**
x265 will use all detected CPU SIMD architectures by default. You can disable all assembly by using *--no-asm* or you can specify a comma separated list of SIMD architectures to use, matching these strings: MMX2, SSE, SSE2, SSE3, SSSE3, SSE4, SSE4.1, SSE4.2, AVX, XOP, FMA4, AVX2, FMA3

Some higher architectures imply lower ones being present, this is handled implicitly.

One may also directly supply the CPU capability bitmap as an integer.

Note that by specifying this option you are overriding x265's CPU detection and it is possible to do this wrong. You can cause encoder crashes by specifying SIMD architectures which are not supported on your CPU.

Default: auto-detected SIMD architectures

--frame-threads, -F <integer>
Number of concurrently encoded frames. Using a single frame thread gives a slight improvement in compression, since the entire reference frames are always available for motion compensation, but it has severe performance implications. Default is an autodetected count based on the number of CPU cores and whether WPP is enabled or not.

Over-allocation of frame threads will not improve performance, it will generally just increase memory use.

Values: any value between 0 and 16. Default is 0, auto-detect

--pools <string>, **--numa-pools** <string>
Comma seperated list of threads per NUMA node. If "none", then no worker pools are created and only frame parallelism is possible. If NULL or "" (default) x265 will use all available threads on each NUMA node:

```
'+'  is a special value indicating all cores detected on the node
'*'  is a special value indicating all cores detected on the node and all␣
↪remaining nodes
'-'  is a special value indicating no cores on the node, same as '0'
```

example strings for a 4-node system:

```
""        - default, unspecified, all numa nodes are used for thread pools
"*"       - same as default
"none"    - no thread pools are created, only frame parallelism possible
"-"       - same as "none"
"10"      - allocate one pool, using up to 10 cores on node 0
"-,+"     - allocate one pool, using all cores on node 1
```

```
"+,-,+"    - allocate two pools, using all cores on nodes 0 and 2
"+,-,+,-" - allocate two pools, using all cores on nodes 0 and 2
"-,*"      - allocate three pools, using all cores on nodes 1, 2 and 3
"8,8,8,8" - allocate four pools with up to 8 threads in each pool
```

The total number of threads will be determined by the number of threads assigned to all nodes. The worker threads will each be given affinity for their node, they will not be allowed to migrate between nodes, but they will be allowed to move between CPU cores within their node.

If the four pool features: *--wpp*, *--pmode*, *--pme* and *--lookahead-slices* are all disabled, then *--pools* is ignored and no thread pools are created.

If "none" is specified, then all four of the thread pool features are implicitly disabled.

Multiple thread pools will be allocated for any NUMA node with more than 64 logical CPU cores. But any given thread pool will always use at most one NUMA node.

Frame encoders are distributed between the available thread pools, and the encoder will never generate more thread pools than *--frame-threads*. The pools are used for WPP and for distributed analysis and motion search.

On Windows, the native APIs offer sufficient functionality to discover the NUMA topology and enforce the thread affinity that libx265 needs (so long as you have not chosen to target XP or Vista), but on POSIX systems it relies on libnuma for this functionality. If your target POSIX system is single socket, then building without libnuma is a perfectly reasonable option, as it will have no effect on the runtime behavior. On a multiple-socket system, a POSIX build of libx265 without libnuma will be less work efficient. See *thread pools* for more detail.

Default "", one thread is allocated per detected hardware thread (logical CPU cores) and one thread pool per NUMA node.

Note that the string value will need to be escaped or quoted to protect against shell expansion on many platforms

--wpp, --no-wpp

Enable Wavefront Parallel Processing. The encoder may begin encoding a row as soon as the row above it is at least two CTUs ahead in the encode process. This gives a 3-5x gain in parallelism for about 1% overhead in compression efficiency.

This feature is implicitly disabled when no thread pool is present.

Default: Enabled

--pmode, --no-pmode

Parallel mode decision, or distributed mode analysis. When enabled the encoder will distribute the analysis work of each CU (merge, inter, intra) across multiple worker threads. Only recommended if x265 is not already saturating the CPU cores. In RD levels 3 and 4 it will be most effective if –rect is enabled. At RD levels 5 and 6 there is generally always enough work to distribute to warrant the overhead, assuming your CPUs are not already saturated.

–pmode will increase utilization without reducing compression efficiency. In fact, since the modes are all measured in parallel it makes certain early-outs impractical and thus you usually get slightly better compression when it is enabled (at the expense of not skipping improbable modes). This bypassing of early-outs can cause pmode to slow down encodes, especially at faster presets.

This feature is implicitly disabled when no thread pool is present.

Default disabled

--pme, --no-pme

Parallel motion estimation. When enabled the encoder will distribute motion estimation across multiple worker threads when more than two references require motion searches for a given CU. Only recommended if x265 is not already saturating CPU cores. *--pmode* is much more effective than this option, since the amount of work

it distributes is substantially higher. With –pme it is not unusual for the overhead of distributing the work to outweigh the parallelism benefits.

This feature is implicitly disabled when no thread pool is present.

–pme will increase utilization on many core systems with no effect on the output bitstream.

Default disabled

--preset, -p <integer|string>
Sets parameters to preselected values, trading off compression efficiency against encoding speed. These parameters are applied before all other input parameters are applied, and so you can override any parameters that these values control. See *presets* for more detail.

0.ultrafast

1.superfast

2.veryfast

3.faster

4.fast

5.medium **(default)**

6.slow

7.slower

8.veryslow

9.placebo

--tune, -t <string>
Tune the settings for a particular type of source or situation. The changes will be applied after *--preset* but before all other parameters. Default none. See *tunings* for more detail.

Values: psnr, ssim, grain, zero-latency, fast-decode.

Input/Output File Options

These options all describe the input video sequence or, in the case of *--dither*, operations that are performed on the sequence prior to encode. All options dealing with files (names, formats, offsets or frame counts) are only applicable to the CLI application.

--input <filename>
Input filename, only raw YUV or Y4M supported. Use single dash for stdin. This option name will be implied for the first "extra" command line argument.

CLI ONLY

--y4m
Parse input stream as YUV4MPEG2 regardless of file extension, primarily intended for use with stdin (ie: *--input* - *--y4m*). This option is implied if the input filename has a ".y4m" extension

CLI ONLY

--input-depth <integer>
YUV only: Bit-depth of input file or stream

Values: any value between 8 and 16. Default is internal depth.

CLI ONLY

--dither
Enable high quality downscaling. Dithering is based on the diffusion of errors from one row of pixels to the next row of pixels in a picture. Only applicable when the input bit depth is larger than 8bits and internal bit depth is 8bits. Default disabled

CLI ONLY

--input-res <wxh>
YUV only: Source picture size [w x h]

CLI ONLY

--input-csp <integer|string>
YUV only: Source color space. Only i420, i422, and i444 are supported at this time. The internal color space is always the same as the source color space (libx265 does not support any color space conversions).

0.i400

1.i420 **(default)**

2.i422

3.i444

4.nv12

5.nv16

--fps <integer|float|numerator/denominator>
YUV only: Source frame rate

Range of values: positive int or float, or num/denom

--interlaceMode <false|tff|bff>**, --no-interlaceMode**

0.progressive pictures **(default)**

1.top field first

2.bottom field first

HEVC encodes interlaced content as fields. Fields must be provided to the encoder in the correct temporal order. The source dimensions must be field dimensions and the FPS must be in units of fields per second. The decoder must re-combine the fields in their correct orientation for display.

--seek <integer>
Number of frames to skip at start of input file. Default 0

CLI ONLY

--frames, -f <integer>
Number of frames of input sequence to be encoded. Default 0 (all)

CLI ONLY

--output, -o <filename>
Bitstream output file name. If there are two extra CLI options, the first is implicitly the input filename and the second is the output filename, making the *--output* option optional.

The output file will always contain a raw HEVC bitstream, the CLI does not support any container file formats.

CLI ONLY

--output-depth, -D 8|10
Bitdepth of output HEVC bitstream, which is also the internal bit depth of the encoder. If the requested bit

depth is not the bit depth of the linked libx265, it will attempt to bind libx265_main for an 8bit encoder, or libx265_main10 for a 10bit encoder, with the same API version as the linked libx265.

CLI ONLY

Profile, Level, Tier

--profile, -P <string>
Enforce the requirements of the specified profile, ensuring the output stream will be decodable by a decoder which supports that profile. May abort the encode if the specified profile is impossible to be supported by the compile options chosen for the encoder (a high bit depth encoder will be unable to output bitstreams compliant with Main or Mainstillpicture).

API users must use x265_param_apply_profile() after configuring their param structure. Any changes made to the param structure after this call might make the encode non-compliant.

Values: main, main10, mainstillpicture, main422-8, main422-10, main444-8, main444-10

CLI ONLY

--level-idc <integer|float>
Minimum decoder requirement level. Defaults to 0, which implies auto-detection by the encoder. If specified, the encoder will attempt to bring the encode specifications within that specified level. If the encoder is unable to reach the level it issues a warning and aborts the encode. If the requested requirement level is higher than the actual level, the actual requirement level is signaled.

Beware, specifying a decoder level will force the encoder to enable VBV for constant rate factor encodes, which may introduce non-determinism.

The value is specified as a float or as an integer with the level times 10, for example level **5.1** is specified as "5.1" or "51", and level **5.0** is specified as "5.0" or "50".

Annex A levels: 1, 2, 2.1, 3, 3.1, 4, 4.1, 5, 5.1, 5.2, 6, 6.1, 6.2, 8.5

--high-tier, --no-high-tier
If *--level-idc* has been specified, the option adds the intention to support the High tier of that level. If your specified level does not support a High tier, a warning is issued and this modifier flag is ignored. If *--level-idc* has been specified, but not –high-tier, then the encoder will attempt to encode at the specified level, main tier first, turning on high tier only if necessary and available at that level.

--ref <1..16>
Max number of L0 references to be allowed. This number has a linear multiplier effect on the amount of work performed in motion search, but will generally have a beneficial affect on compression and distortion.

Note that x265 allows up to 16 L0 references but the HEVC specification only allows a maximum of 8 total reference frames. So if you have B frames enabled only 7 L0 refs are valid and if you have *--b-pyramid* enabled (which is enabled by default in all presets), then only 6 L0 refs are the maximum allowed by the HEVC specification. If x265 detects that the total reference count is greater than 8, it will issue a warning that the resulting stream is non-compliant and it signals the stream as profile NONE and level NONE and will abort the encode unless *--allow-non-conformance* it specified. Compliant HEVC decoders may refuse to decode such streams.

Default 3

--allow-non-conformance, --no-allow-non-conformance
Allow libx265 to generate a bitstream with profile and level NONE. By default it will abort any encode which does not meet strict level compliance. The two most likely causes for non-conformance are *--ctu* being too small, *--ref* being too high, or the bitrate or resolution being out of specification.

Default: disabled

Note: *--profile*, *--level-idc*, and *--high-tier* are only intended for use when you are targeting a particular decoder (or decoders) with fixed resource limitations and must constrain the bitstream within those limits. Specifying a profile or level may lower the encode quality parameters to meet those requirements but it will never raise them. It may enable VBV constraints on a CRF encode.

Mode decision / Analysis

--rd <0..6>
Level of RDO in mode decision. The higher the value, the more exhaustive the analysis and the more rate distortion optimization is used. The lower the value the faster the encode, the higher the value the smaller the bitstream (in general). Default 3

Note that this table aims for accuracy, but is not necessarily our final target behavior for each mode.

Level	Description
0	sa8d mode and split decisions, intra w/ source pixels
1	recon generated (better intra), RDO merge/skip selection
2	RDO splits and merge/skip selection
3	RDO mode and split decisions, chroma residual used for sa8d
4	Currently same as 3
5	Adds RDO prediction decisions
6	Currently same as 5

Range of values: 0: least .. 6: full RDO analysis

Options which affect the coding unit quad-tree, sometimes referred to as the prediction quad-tree.

--ctu, -s <64|32|16>
Maximum CU size (width and height). The larger the maximum CU size, the more efficiently x265 can encode flat areas of the picture, giving large reductions in bitrate. However this comes at a loss of parallelism with fewer rows of CUs that can be encoded in parallel, and less frame parallelism as well. Because of this the faster presets use a CU size of 32. Default: 64

--min-cu-size <64|32|16|8>
Minimum CU size (width and height). By using 16 or 32 the encoder will not analyze the cost of CUs below that minimum threshold, saving considerable amounts of compute with a predictable increase in bitrate. This setting has a large effect on performance on the faster presets.

Default: 8 (minimum 8x8 CU for HEVC, best compression efficiency)

Note: All encoders within a single process must use the same settings for the CU size range. *--ctu* and *--min-cu-size* must be consistent for all of them since the encoder configures several key global data structures based on this range.

--rect, --no-rect
Enable analysis of rectangular motion partitions Nx2N and 2NxN (50/50 splits, two directions). Default disabled

--amp, --no-amp
Enable analysis of asymmetric motion partitions (75/25 splits, four directions). At RD levels 0 through 4, AMP partitions are only considered at CU sizes 32x32 and below. At RD levels 5 and 6, it will only consider AMP partitions as merge candidates (no motion search) at 64x64, and as merge or inter candidates below 64x64.

The AMP partitions which are searched are derived from the current best inter partition. If Nx2N (vertical rectangular) is the best current prediction, then left and right asymmetrical splits will be evaluated. If 2NxN (horizontal rectangular) is the best current prediction, then top and bottom asymmetrical splits will be evaluated, If 2Nx2N is the best prediction, and the block is not a merge/skip, then all four AMP partitions are evaluated.

This setting has no effect if rectangular partitions are disabled. Default disabled

--early-skip, --no-early-skip
Measure full CU size (2Nx2N) merge candidates first; if no residual is found the analysis is short circuited. Default disabled

--fast-intra, --no-fast-intra
Perform an initial scan of every fifth intra angular mode, then check modes +/- 2 distance from the best mode, then +/- 1 distance from the best mode, effectively performing a gradient descent. When enabled 10 modes in total are checked. When disabled all 33 angular modes are checked. Only applicable for *--rd* levels 4 and below (medium preset and faster).

--b-intra, --no-b-intra
Enables the evaluation of intra modes in B slices. Default disabled.

--cu-lossless, --no-cu-lossless
For each CU, evaluate lossless (transform and quant bypass) encode of the best non-lossless mode option as a potential rate distortion optimization. If the global option *--lossless* has been specified, all CUs will be encoded as lossless unconditionally regardless of whether this option was enabled. Default disabled.

Only effective at RD levels 3 and above, which perform RDO mode decisions.

--tskip-fast, --no-tskip-fast
Only evaluate transform skip for NxN intra predictions (4x4 blocks). Only applicable if transform skip is enabled. For chroma, only evaluate if luma used tskip. Inter block tskip analysis is unmodified. Default disabled

Analysis re-use options, to improve performance when encoding the same sequence multiple times (presumably at varying bitrates). The encoder will not reuse analysis if the resolution and slice type parameters do not match.

--analysis-mode <string|int>
Specify whether analysis information of each frame is output by encoder or input for reuse. By reading the analysis data writen by an earlier encode of the same sequence, substantial redundant work may be avoided.

The following data may be stored and reused: I frames - split decisions and luma intra directions of all CUs. P/B frames - motion vectors are dumped at each depth for all CUs.

Values: off(0), save(1): dump analysis data, load(2): read analysis data

--analysis-file <filename>
Specify a filename for analysis data (see *--analysis-mode*) If no filename is specified, x265_analysis.dat is used.

Options which affect the transform unit quad-tree, sometimes referred to as the residual quad-tree (RQT).

--rdoq-level <0|1|2>**, --no-rdoq-level**
Specify the amount of rate-distortion analysis to use within quantization:

At level 0 rate-distortion cost is not considered in quant

At level 1 rate-distortion cost is used to find optimal rounding values for each level (and allows psy-rdoq to be effective). It trades-off the signaling cost of the coefficient vs its post-inverse quant distortion from the pre-quant coefficient. When *--psy-rdoq* is enabled, this formula is biased in favor of more energy in the residual (larger coefficient absolute levels)

At level 2 rate-distortion cost is used to make decimate decisions on each 4x4 coding group, including the cost of signaling the group within the group bitmap. If the total distortion of not signaling the entire coding group is less than the rate cost, the block is decimated. Next, it applies rate-distortion cost analysis to the last non-zero

coefficient, which can result in many (or all) of the coding groups being decimated. Psy-rdoq is less effective at preserving energy when RDOQ is at level 2, since it only has influence over the level distortion costs.

--tu-intra-depth <1..4>
The transform unit (residual) quad-tree begins with the same depth as the coding unit quad-tree, but the encoder may decide to further split the transform unit tree if it improves compression efficiency. This setting limits the number of extra recursion depth which can be attempted for intra coded units. Default: 1, which means the residual quad-tree is always at the same depth as the coded unit quad-tree

Note that when the CU intra prediction is NxN (only possible with 8x8 CUs), a TU split is implied, and thus the residual quad-tree begins at 4x4 and cannot split any futhrer.

--tu-inter-depth <1..4>
The transform unit (residual) quad-tree begins with the same depth as the coding unit quad-tree, but the encoder may decide to further split the transform unit tree if it improves compression efficiency. This setting limits the number of extra recursion depth which can be attempted for inter coded units. Default: 1. which means the residual quad-tree is always at the same depth as the coded unit quad-tree unless the CU was coded with rectangular or AMP partitions, in which case a TU split is implied and thus the residual quad-tree begins one layer below the CU quad-tree.

--nr-intra <integer>, **--nr-inter** <integer>
Noise reduction - an adaptive deadzone applied after DCT (subtracting from DCT coefficients), before quantization. It does no pixel-level filtering, doesn't cross DCT block boundaries, has no overlap, The higher the strength value parameter, the more aggressively it will reduce noise.

Enabling noise reduction will make outputs diverge between different numbers of frame threads. Outputs will be deterministic but the outputs of -F2 will no longer match the outputs of -F3, etc.

Values: any value in range of 0 to 2000. Default 0 (disabled).

--tskip, --no-tskip
Enable evaluation of transform skip (bypass DCT but still use quantization) coding for 4x4 TU coded blocks.

Only effective at RD levels 3 and above, which perform RDO mode decisions. Default disabled

--rdpenalty <0..2>
When set to 1, transform units of size 32x32 are given a 4x bit cost penalty compared to smaller transform units, in intra coded CUs in P or B slices.

When set to 2, transform units of size 32x32 are not even attempted, unless otherwise required by the maximum recursion depth. For this option to be effective with 32x32 intra CUs, *--tu-intra-depth* must be at least 2. For it to be effective with 64x64 intra CUs, *--tu-intra-depth* must be at least 3.

Note that in HEVC an intra transform unit (a block of the residual quad-tree) is also a prediction unit, meaning that the intra prediction signal is generated for each TU block, the residual subtracted and then coded. The coding unit simply provides the prediction modes that will be used when predicting all of the transform units within the CU. This means that when you prevent 32x32 intra transform units, you are preventing 32x32 intra predictions.

Default 0, disabled.

Values: 0:disabled 1:4x cost penalty 2:force splits

--max-tu-size <32|16|8|4>
Maximum TU size (width and height). The residual can be more efficiently compressed by the DCT transform when the max TU size is larger, but at the expense of more computation. Transform unit quad-tree begins at the same depth of the coded tree unit, but if the maximum TU size is smaller than the CU size then transform QT begins at the depth of the max-tu-size. Default: 32.

Temporal / motion search options

--max-merge <1..5>
Maximum number of neighbor (spatial and temporal) candidate blocks that the encoder may consider for merging motion predictions. If a merge candidate results in no residual, it is immediately selected as a "skip". Otherwise the merge candidates are tested as part of motion estimation when searching for the least cost inter option. The max candidate number is encoded in the SPS and determines the bit cost of signaling merge CUs. Default 2

--me <integer|string>
Motion search method. Generally, the higher the number the harder the ME method will try to find an optimal match. Diamond search is the simplest. Hexagon search is a little better. Uneven Multi-Hexegon is an adaption of the search method used by x264 for slower presets. Star is a three step search adapted from the HM encoder: a star-pattern search followed by an optional radix scan followed by an optional star-search refinement. Full is an exhaustive search; an order of magnitude slower than all other searches but not much better than umh or star.

0.dia

1.hex **(default)**

2.umh

3.star

4.full

--subme, -m <0..7>
Amount of subpel refinement to perform. The higher the number the more subpel iterations and steps are performed. Default 2

-m	HPEL iters	HPEL dirs	QPEL iters	QPEL dirs	HPEL SATD
0	1	4	0	4	false
1	1	4	1	4	false
2	1	4	1	4	true
3	2	4	1	4	true
4	2	4	2	4	true
5	1	8	1	8	true
6	2	8	1	8	true
7	2	8	2	8	true

At –subme values larger than 2, chroma residual cost is included in all subpel refinement steps and chroma residual is included in all motion estimation decisions (selecting the best reference picture in each list, and chosing between merge, uni-directional motion and bi-directional motion). The 'slow' preset is the first preset to enable the use of chroma residual.

--merange <integer>
Motion search range. Default 57

The default is derived from the default CTU size (64) minus the luma interpolation half-length (4) minus maximum subpel distance (2) minus one extra pixel just in case the hex search method is used. If the search range were any larger than this, another CTU row of latency would be required for reference frames.

Range of values: an integer from 0 to 32768

--temporal-mvp, --no-temporal-mvp
Enable temporal motion vector predictors in P and B slices. This enables the use of the motion vector from the collocated block in the previous frame to be used as a predictor. Default is enabled

--weightp, -w, --no-weightp
Enable weighted prediction in P slices. This enables weighting analysis in the lookahead, which influences

slice decisions, and enables weighting analysis in the main encoder which allows P reference samples to have a weight function applied to them prior to using them for motion compensation. In video which has lighting changes, it can give a large improvement in compression efficiency. Default is enabled

--weightb, --no-weightb
Enable weighted prediction in B slices. Default disabled

Spatial/intra options

--strong-intra-smoothing, --no-strong-intra-smoothing
Enable strong intra smoothing for 32x32 intra blocks. Default enabled

--constrained-intra, --no-constrained-intra
Constrained intra prediction. When generating intra predictions for blocks in inter slices, only intra-coded reference pixels are used. Inter-coded reference pixels are replaced with intra-coded neighbor pixels or default values. The general idea is to block the propagation of reference errors that may have resulted from lossy signals. Default disabled

Psycho-visual options

Left to its own devices, the encoder will make mode decisions based on a simple rate distortion formula, trading distortion for bitrate. This is generally effective except for the manner in which this distortion is measured. It tends to favor blurred reconstructed blocks over blocks which have wrong motion. The human eye generally prefers the wrong motion over the blur and thus x265 offers psycho-visual adjustments to the rate distortion algorithm.

--psy-rd will add an extra cost to reconstructed blocks which do not match the visual energy of the source block. The higher the strength of *--psy-rd* the more strongly it will favor similar energy over blur and the more aggressively it will ignore rate distortion. If it is too high, it will introduce visal artifacts and increase bitrate enough for rate control to increase quantization globally, reducing overall quality. psy-rd will tend to reduce the use of blurred prediction modes, like DC and planar intra and bi-directional inter prediction.

--psy-rdoq will adjust the distortion cost used in rate-distortion optimized quantization (RDO quant), enabled by *--rdoq-level* 1 or 2, favoring the preservation of energy in the reconstructed image. *--psy-rdoq* prevents RDOQ from blurring all of the encoding options which psy-rd has to chose from. At low strength levels, psy-rdoq will influence the quantization level decisions, favoring higher AC energy in the reconstructed image. As psy-rdoq strength is increased, more non-zero coefficient levels are added and fewer coefficients are zeroed by RDOQ's rate distortion analysis. High levels of psy-rdoq can double the bitrate which can have a drastic effect on rate control, forcing higher overall QP, and can cause ringing artifacts. psy-rdoq is less accurate than psy-rd, it is biasing towards energy in general while psy-rd biases towards the energy of the source image. But very large psy-rdoq values can sometimes be beneficial, preserving film grain for instance.

As a general rule, when both psycho-visual features are disabled, the encoder will tend to blur blocks in areas of difficult motion. Turning on small amounts of psy-rd and psy-rdoq will improve the perceived visual quality. Increasing psycho-visual strength further will improve quality and begin introducing artifacts and increase bitrate, which may force rate control to increase global QP. Finding the optimal psycho-visual parameters for a given video requires experimentation. Our recommended defaults (1.0 for both) are generally on the low end of the spectrum.

The lower the bitrate, the lower the optimal psycho-visual settings. If the bitrate is too low for the psycho-visual settings, you will begin to see temporal artifacts (motion judder). This is caused when the encoder is forced to code skip blocks (no residual) in areas of difficult motion because it is the best option psycho-visually (they have great amounts of energy and no residual cost). One can lower psy-rd settings when judder is happening, and allow the encoder to use some blur in these areas of high motion.

--psy-rd <float>
Influence rate distortion optimizated mode decision to preserve the energy of the source image in the encoded image at the expense of compression efficiency. It only has effect on presets which use RDO-based mode decisions (*--rd* 3 and above). 1.0 is a typical value. Default 0.3

Range of values: 0 .. 2.0

--psy-rdoq <float>
Influence rate distortion optimized quantization by favoring higher energy in the reconstructed image. This generally improves perceived visual quality at the cost of lower quality metric scores. It only has effect when *--rdoq-level* is 1 or 2. High values can be beneficial in preserving high-frequency detail like film grain. Default: 1.0

Range of values: 0 .. 50.0

Slice decision options

--open-gop, --no-open-gop
Enable open GOP, allow I-slices to be non-IDR. Default enabled

--keyint, -I <integer>
Max intra period in frames. A special case of infinite-gop (single keyframe at the beginning of the stream) can be triggered with argument -1. Use 1 to force all-intra. Default 250

--min-keyint, -i <integer>
Minimum GOP size. Scenecuts closer together than this are coded as I or P, not IDR. Minimum keyint is clamped to be at least half of *--keyint*. If you wish to force regular keyframe intervals and disable adaptive I frame placement, you must use *--no-scenecut*.

Range of values: >=0 (0: auto)

--scenecut <integer>, **--no-scenecut**
How aggressively I-frames need to be inserted. The higher the threshold value, the more aggressive the I-frame placement. *--scenecut* 0 or *--no-scenecut* disables adaptive I frame placement. Default 40

--rc-lookahead <integer>
Number of frames for slice-type decision lookahead (a key determining factor for encoder latency). The longer the lookahead buffer the more accurate scenecut decisions will be, and the more effective cuTree will be at improving adaptive quant. Having a lookahead larger than the max keyframe interval is not helpful. Default 20

Range of values: Between the maximum consecutive bframe count (*--bframes*) and 250

--lookahead-slices <0..16>
Use multiple worker threads to measure the estimated cost of each frame within the lookahead. When *--b-adapt* is 2, most frame cost estimates will be performed in batch mode, many cost estimates at the same time, and lookahead-slices is ignored for batched estimates. The effect on performance can be quite small. The higher this parameter, the less accurate the frame costs will be (since context is lost across slice boundaries) which will result in less accurate B-frame and scene-cut decisions.

The encoder may internally lower the number of slices to ensure each slice codes at least 10 16x16 rows of lowres blocks. If slices are used in lookahead, they are logged in the list of tools as *lslices*.

Values: 0 - disabled (default). 1 is the same as 0. Max 16

--b-adapt <integer>
Set the level of effort in determining B frame placement.

With b-adapt 0, the GOP structure is fixed based on the values of *--keyint* and *--bframes*.

With b-adapt 1 a light lookahead is used to choose B frame placement.

With b-adapt 2 (trellis) a viterbi B path selection is performed

Values: 0:none; 1:fast; 2:full(trellis) **default**

--bframes, -b <0..16>
Maximum number of consecutive b-frames. Use *--bframes* 0 to force all P/I low-latency encodes. Default 4. This parameter has a quadratic effect on the amount of memory allocated and the amount of work performed by the full trellis version of *--b-adapt* lookahead.

--bframe-bias <integer>
Bias towards B frames in slicetype decision. The higher the bias the more likely x265 is to use B frames. Can be any value between -90 and 100 and is clipped to that range. Default 0

--b-pyramid, --no-b-pyramid
Use B-frames as references, when possible. Default enabled

Quality, rate control and rate distortion options

--bitrate <integer>
Enables single-pass ABR rate control. Specify the target bitrate in kbps. Default is 0 (CRF)

Range of values: An integer greater than 0

--crf <0..51.0>
Quality-controlled variable bitrate. CRF is the default rate control method; it does not try to reach any particular bitrate target, instead it tries to achieve a given uniform quality and the size of the bitstream is determined by the complexity of the source video. The higher the rate factor the higher the quantization and the lower the quality. Default rate factor is 28.0.

--crf-max <0..51.0>
Specify an upper limit to the rate factor which may be assigned to any given frame (ensuring a max QP). This is dangerous when CRF is used in combination with VBV as it may result in buffer underruns. Default disabled

--crf-min <0..51.0>
Specify an lower limit to the rate factor which may be assigned to any given frame (ensuring a min compression factor).

--vbv-bufsize <integer>
Specify the size of the VBV buffer (kbits). Enables VBV in ABR mode. In CRF mode, *--vbv-maxrate* must also be specified. Default 0 (vbv disabled)

--vbv-maxrate <integer>
Maximum local bitrate (kbits/sec). Will be used only if vbv-bufsize is also non-zero. Both vbv-bufsize and vbv-maxrate are required to enable VBV in CRF mode. Default 0 (disabled)

--vbv-init <float>
Initial buffer occupancy. The portion of the decode buffer which must be full before the decoder will begin decoding. Determines absolute maximum frame size. May be specified as a fractional value between 0 and 1, or in kbits. In other words these two option pairs are equivalent:

```
--vbv-bufsize 1000 --vbv-init 900
--vbv-bufsize 1000 --vbv-init 0.9
```

Default 0.9

Range of values: fractional: 0 - 1.0, or kbits: 2 .. bufsize

--qp, -q <integer>
Specify base quantization parameter for Constant QP rate control. Using this option enables Constant QP rate

control. The specified QP is assigned to P slices. I and B slices are given QPs relative to P slices using param->rc.ipFactor and param->rc.pbFactor unless QP 0 is specified, in which case QP 0 is used for all slice types. Note that QP 0 does not cause lossless encoding, it only disables quantization. Default disabled (CRF)

Range of values: an integer from 0 to 51

--lossless, --no-lossless
Enables true lossless coding by bypassing scaling, transform, quantization and in-loop filter processes. This is used for ultra-high bitrates with zero loss of quality. Reconstructed output pictures are bit-exact to the input pictures. Lossless encodes implicitly have no rate control, all rate control options are ignored. Slower presets will generally achieve better compression efficiency (and generate smaller bitstreams). Default disabled.

--aq-mode <0|1|2>
Adaptive Quantization operating mode. Raise or lower per-block quantization based on complexity analysis of the source image. The more complex the block, the more quantization is used. This offsets the tendency of the encoder to spend too many bits on complex areas and not enough in flat areas.

0.disabled

1.AQ enabled **(default)**

2.AQ enabled with auto-variance

--aq-strength <float>
Adjust the strength of the adaptive quantization offsets. Setting *--aq-strength* to 0 disables AQ. Default 1.0.

Range of values: 0.0 to 3.0

--qg-size <64|32|16>
Enable adaptive quantization for sub-CTUs. This parameter specifies the minimum CU size at which QP can be adjusted, ie. Quantization Group size. Allowed range of values are 64, 32, 16 provided this falls within the inclusive range [maxCUSize, minCUSize]. Experimental. Default: same as maxCUSize

--cutree, --no-cutree
Enable the use of lookahead's lowres motion vector fields to determine the amount of reuse of each block to tune adaptive quantization factors. CU blocks which are heavily reused as motion reference for later frames are given a lower QP (more bits) while CU blocks which are quickly changed and are not referenced are given less bits. This tends to improve detail in the backgrounds of video with less detail in areas of high motion. Default enabled

--pass <integer>
Enable multi-pass rate control mode. Input is encoded multiple times, storing the encoded information of each pass in a stats file from which the consecutive pass tunes the qp of each frame to improve the quality of the output. Default disabled

1.First pass, creates stats file

2.Last pass, does not overwrite stats file

3.Nth pass, overwrites stats file

Range of values: 1 to 3

--stats <filename>
Specify file name of of the multi-pass stats file. If unspecified the encoder will use x265_2pass.log

--slow-firstpass, --no-slow-firstpass
Enable a slow and more detailed first pass encode in multi-pass rate control mode. Speed of the first pass encode is slightly lesser and quality midly improved when compared to the default settings in a multi-pass encode. Default disabled (turbo mode enabled)

When **turbo** first pass is not disabled, these options are set on the first pass to improve performance:

- *--fast-intra*
- *--no-rect*
- *--no-amp*
- *--early-skip*
- *--ref* = 1
- *--max-merge* = 1
- *--me* = DIA
- *--subme* = MIN(2, *--subme*)
- *--rd* = MIN(2, *--rd*)

--strict-cbr, --no-strict-cbr
Enables stricter conditions to control bitrate deviance from the target bitrate in ABR mode. Bit rate adherence is prioritised over quality. Rate tolerance is reduced to 50%. Default disabled.

This option is for use-cases which require the final average bitrate to be within very strict limits of the target; preventing overshoots, while keeping the bit rate within 5% of the target setting, especially in short segment encodes. Typically, the encoder stays conservative, waiting until there is enough feedback in terms of encoded frames to control QP. strict-cbr allows the encoder to be more aggressive in hitting the target bitrate even for short segment videos. Experimental.

--cbqpoffs <integer>
Offset of Cb chroma QP from the luma QP selected by rate control. This is a general way to spend more or less bits on the chroma channel. Default 0

Range of values: -12 to 12

--crqpoffs <integer>
Offset of Cr chroma QP from the luma QP selected by rate control. This is a general way to spend more or less bits on the chroma channel. Default 0

Range of values: -12 to 12

--ipratio <float>
QP ratio factor between I and P slices. This ratio is used in all of the rate control modes. Some *--tune* options may change the default value. It is not typically manually specified. Default 1.4

--pbratio <float>
QP ratio factor between P and B slices. This ratio is used in all of the rate control modes. Some *--tune* options may change the default value. It is not typically manually specified. Default 1.3

--qcomp <float>
qComp sets the quantizer curve compression factor. It weights the frame quantizer based on the complexity of residual (measured by lookahead). Default value is 0.6. Increasing it to 1 will effectively generate CQP

--qpstep <integer>
The maximum single adjustment in QP allowed to rate control. Default 4

--qblur <float>
Temporally blur quants. Default 0.5

--cplxblur <float>
temporally blur complexity. default 20

--zones <zone0>/<zone1>/...
Tweak the bitrate of regions of the video. Each zone takes the form:

<start frame>,<end frame>,<option> where <option> is either q=<integer> (force QP) or b=<float> (bitrate multiplier).

If zones overlap, whichever comes later in the list takes precedence. Default none

Quantization Options

Note that rate-distortion optimized quantization (RDOQ) is enabled implicitly at *--rd* 4, 5, and 6 and disabled implicitly at all other levels.

--signhide, --no-signhide
Hide sign bit of one coeff per TU (rdo). The last sign is implied. This requires analyzing all the coefficients to determine if a sign must be toggled, and then to determine which one can be toggled with the least amount of distortion. Default enabled

--qpfile <filename>
Specify a text file which contains frametypes and QPs for some or all frames. The format of each line is:

framenumber frametype QP

Frametype can be one of [I,i,P,B,b]. **B** is a referenced B frame, **b** is an unreferenced B frame. **I** is a keyframe (random access point) while **i** is a I frame that is not a keyframe (references are not broken).

Specifying QP (integer) is optional, and if specified they are clamped within the encoder to qpmin/qpmax.

--scaling-list <filename>
Quantization scaling lists. HEVC supports 6 quantization scaling lists to be defined; one each for Y, Cb, Cr for intra prediction and one each for inter prediction.

x265 does not use scaling lists by default, but this can also be made explicit by *--scaling-list off*.

HEVC specifies a default set of scaling lists which may be enabled without requiring them to be signaled in the SPS. Those scaling lists can be enabled via *--scaling-list default*.

All other strings indicate a filename containing custom scaling lists in the HM format. The encode will abort if the file is not parsed correctly. Custom lists must be signaled in the SPS

--lambda-file <filename>
Specify a text file containing values for x265_lambda_tab and x265_lambda2_tab. Each table requires MAX_MAX_QP+1 (70) float values.

The text file syntax is simple. Comma is considered to be white-space. All white-space is ignored. Lines must be less than 2k bytes in length. Content following hash (#) characters are ignored. The values read from the file are logged at *--log-level* debug.

Note that the lambda tables are process-global and so the new values affect all encoders running in the same process.

Lambda values affect encoder mode decisions, the lower the lambda the more bits it will try to spend on signaling information (motion vectors and splits) and less on residual. This feature is intended for experimentation.

Loop filters

--deblock=<int>:<int>, **--no-deblock**
Toggle deblocking loop filter, optionally specify deblocking strength offsets.

<int>:<int> - parsed as tC offset and Beta offset <int>,<int> - parsed as tC offset and Beta offset <int> - both tC and Beta offsets assigned the same value

If unspecified, the offsets default to 0. The offsets must be in a range of -6 (lowest strength) to 6 (highest strength).

To disable the deblocking filter entirely, use –no-deblock or –deblock=false. Default enabled, with both offsets defaulting to 0

If deblocking is disabled, or the offsets are non-zero, these changes from the default configuration are signaled in the PPS.

--sao, --no-sao
Toggle Sample Adaptive Offset loop filter, default enabled

--sao-non-deblock, --no-sao-non-deblock
Specify how to handle depencency between SAO and deblocking filter. When enabled, non-deblocked pixels are used for SAO analysis. When disabled, SAO analysis skips the right/bottom boundary areas. Default disabled

VUI (Video Usability Information) options

x265 emits a VUI with only the timing info by default. If the SAR is specified (or read from a Y4M header) it is also included. All other VUI fields must be manually specified.

--sar `<integer|w:h>`
Sample Aspect Ratio, the ratio of width to height of an individual sample (pixel). The user may supply the width and height explicitly or specify an integer from the predefined list of aspect ratios defined in the HEVC specification. Default undefined (not signaled)

1.1:1 (square)

2.12:11

3.10:11

4.16:11

5.40:33

6.24:11

7.20:11

8.32:11

9.80:33

10.18:11

11.15:11

12.64:33

13.160:99

14.4:3

15.3:2

16.2:1

--crop-rect `<left,top,right,bottom>`
Define the (overscan) region of the image that does not contain information because it was added to achieve certain resolution or aspect ratio. The decoder may be directed to crop away this region before displaying the images via the *--overscan* option. Default undefined (not signaled)

--overscan <show|crop>
Specify whether it is appropriate for the decoder to display or crop the overscan area. Default unspecified (not signaled)

--videoformat <integer|string>
Specify the source format of the original analog video prior to digitizing and encoding. Default undefined (not signaled)

0.component

1.pal

2.ntsc

3.secam

4.mac

5.undefined

--range <full|limited>
Specify output range of black level and range of luma and chroma signals. Default undefined (not signaled)

--colorprim <integer|string>
Specify color primitive to use when converting to RGB. Default undefined (not signaled)

1.bt709

2.undef

3.**reserved**

4.bt470m

5.bt470bg

6.smpte170m

7.smpte240m

8.film

9.bt2020

--transfer <integer|string>
Specify transfer characteristics. Default undefined (not signaled)

1.bt709

2.undef

3.**reserved**

4.bt470m

5.bt470bg

6.smpte170m

7.smpte240m

8.linear

9.log100

10.log316

11.iec61966-2-4

12.bt1361e

13.iec61966-2-1

14.bt2020-10

15.bt2020-12

16.smpte-st-2084

17.smpte-st-428

--colormatrix <integer|string>
Specify color matrix setting i.e set the matrix coefficients used in deriving the luma and chroma. Default undefined (not signaled)

0.GBR

1.bt709

2.undef

3.**reserved**

4.fcc

5.bt470bg

6.smpte170m

7.smpte240m

8.YCgCo

9.bt2020nc

10.bt2020c

--chromaloc <0..5>
Specify chroma sample location for 4:2:0 inputs. Consult the HEVC specification for a description of these values. Default undefined (not signaled)

--master-display <string>
SMPTE ST 2086 mastering display color volume SEI info, specified as a string which is parsed when the stream header SEI are emitted. The string format is "G(%hu,%hu)B(%hu,%hu)R(%hu,%hu)WP(%hu,%hu)L(%u,%u)" where %hu are unsigned 16bit integers and %u are unsigned 32bit integers. The SEI includes X,Y display primaries for RGB channels, white point X,Y and max,min luminance values. (HDR)

Example for P65D3 1000-nits:

G(13200,34500)B(7500,3000)R(34000,16000)WP(15635,16450)L(10000000,1)

Note that this string value will need to be escaped or quoted to protect against shell expansion on many platforms. No default.

--max-cll <string>
Maximum content light level and maximum frame average light level as required by the Consumer Electronics Association 861.3 specification.

Specified as a string which is parsed when the stream header SEI are emitted. The string format is "%hu,%hu" where %hu are unsigned 16bit integers. The first value is the max content light level (or 0 if no maximum is indicated), the second value is the maximum picture average light level (or 0). (HDR)

Note that this string value will need to be escaped or quoted to protect against shell expansion on many platforms. No default.

Bitstream options

--annexb, --no-annexb
If enabled, x265 will produce Annex B bitstream format, which places start codes before NAL. If disabled, x265 will produce file format, which places length before NAL. x265 CLI will choose the right option based on output format. Default enabled

API ONLY

--repeat-headers, --no-repeat-headers
If enabled, x265 will emit VPS, SPS, and PPS headers with every keyframe. This is intended for use when you do not have a container to keep the stream headers for you and you want keyframes to be random access points. Default disabled

--aud, --no-aud
Emit an access unit delimiter NAL at the start of each slice access unit. If *--repeat-headers* is not enabled (indicating the user will be writing headers manually at the start of the stream) the very first AUD will be skipped since it cannot be placed at the start of the access unit, where it belongs. Default disabled

--hrd, --no-hrd
Enable the signalling of HRD parameters to the decoder. The HRD parameters are carried by the Buffering Period SEI messages and Picture Timing SEI messages providing timing information to the decoder. Default disabled

--info, --no-info
Emit an informational SEI with the stream headers which describes the encoder version, build info, and encode parameters. This is very helpful for debugging purposes but encoding version numbers and build info could make your bitstreams diverge and interfere with regression testing. Default enabled

--hash <integer>
Emit decoded picture hash SEI, so the decoder may validate the reconstructed pictures and detect data loss. Also useful as a debug feature to validate the encoder state. Default None

1.MD5

2.CRC

3.Checksum

--temporal-layers,--no-temporal-layers
Enable a temporal sub layer. All referenced I/P/B frames are in the base layer and all unreferenced B frames are placed in a temporal enhancement layer. A decoder may chose to drop the enhancement layer and only decode and display the base layer slices.

If used with a fixed GOP (`b-adapt` 0) and `bframes` 3 then the two layers evenly split the frame rate, with a cadence of PbBbP. You probably also want *--no-scenecut* and a keyframe interval that is a multiple of 4.

Debugging options

--recon, -r <filename>
Output file containing reconstructed images in display order. If the file extension is ".y4m" the file will contain a YUV4MPEG2 stream header and frame headers. Otherwise it will be a raw YUV file in the encoder's internal bit depth.

CLI ONLY

--recon-depth <integer>
Bit-depth of output file. This value defaults to the internal bit depth and currently cannot to be modified.

CLI ONLY

--recon-y4m-exec <string>

If you have an application which can play a Y4MPEG stream received on stdin, the x265 CLI can feed it reconstructed pictures in display order. The pictures will have no timing info, obviously, so the picture timing will be determined primarily by encoding elapsed time and latencies, but it can be useful to preview the pictures being output by the encoder to validate input settings and rate control parameters.

Example command for ffplay (assuming it is in your PATH):

–recon-y4m-exec “ffplay -i pipe:0 -autoexit”

CLI ONLY

CHAPTER 3

Application Programming Interface

Introduction

x265 is written primarily in C++ and x86 assembly language but the public facing programming interface is C for the widest possible portability. This C interface is wholly defined within `x265.h` in the source/ folder of our source tree. All of the functions and variables and enumerations meant to be used by the end-user are present in this header.

Where possible, x265 has tried to keep its public API as close as possible to x264's public API. So those familiar with using x264 through its C interface will find x265 quite familiar.

This file is meant to be read in-order; the narrative follows linearly through the various sections

Build Considerations

The choice of Main or Main10 profile encodes is made at compile time; the internal pixel depth influences a great deal of variable sizes and thus 8 and 10bit pixels are handled as different build options (primarily to maintain the performance of the 8bit builds). libx265 exports a variable **x265_max_bit_depth** which indicates how the library was compiled (it will contain a value of 8 or 10). Further, **x265_version_str** is a pointer to a string indicating the version of x265 which was compiled, and **x265_build_info_str** is a pointer to a string identifying the compiler and build options.

Note: **x265_version_str** is only updated when **cmake** runs. If you are making binaries for others to use, it is recommended to run **cmake** prior to **make** in your build scripts.

---

x265 will accept input pixels of any depth between 8 and 16 bits regardless of the depth of its internal pixels (8 or 10). It will shift and mask input pixels as required to reach the internal depth. If downshifting is being performed using our CLI application, the *--dither* option may be enabled to reduce banding. This feature is not available through the C interface.

Encoder

The primary object in x265 is the encoder object, and this is represented in the public API as an opaque typedef **x265_encoder**. Pointers of this type are passed to most encoder functions.

A single encoder generates a single output bitstream from a sequence of raw input pictures. Thus if you need multiple output bitstreams you must allocate multiple encoders. You may pass the same input pictures to multiple encoders, the encode function does not modify the input picture structures (the pictures are copied into the encoder as the first step of encode).

Encoder allocation is a reentrant function, so multiple encoders may be safely allocated in a single process. The encoder access functions are not reentrant for a single encoder, so the recommended use case is to allocate one client thread per encoder instance (one thread for all encoder instances is possible, but some encoder access functions are blocking and thus this would be less efficient).

Note: There is one caveat to having multiple encoders within a single process. All of the encoders must use the same maximum CTU size because many global variables are configured based on this size. Encoder allocation will fail if a mis-matched CTU size is attempted. If no encoders are open, **x265_cleanup()** can be called to reset the configured CTU size so a new size can be used.

An encoder is allocated by calling **x265_encoder_open()**:

```
/* x265_encoder_open:
 *      create a new encoder handler, all parameters from x265_param are copied */
x265_encoder* x265_encoder_open(x265_param *);
```

The returned pointer is then passed to all of the functions pertaining to this encode. A large amount of memory is allocated during this function call, but the encoder will continue to allocate memory as the first pictures are passed to the encoder; until its pool of picture structures is large enough to handle all of the pictures it must keep internally. The pool size is determined by the lookahead depth, the number of frame threads, and the maximum number of references.

As indicated in the comment, **x265_param** is copied internally so the user may release their copy after allocating the encoder. Changes made to their copy of the param structure have no affect on the encoder after it has been allocated.

Param

The **x265_param** structure describes everything the encoder needs to know about the input pictures and the output bitstream and most everything in between.

The recommended way to handle these param structures is to allocate them from libx265 via:

```
/* x265_param_alloc:
 *  Allocates an x265_param instance. The returned param structure is not
 *  special in any way, but using this method together with x265_param_free()
 *  and x265_param_parse() to set values by name allows the application to treat
 *  x265_param as an opaque data struct for version safety */
x265_param *x265_param_alloc();
```

In this way, your application does not need to know the exact size of the param structure (the build of x265 could potentially be a bit newer than the copy of `x265.h` that your application compiled against).

Next you perform the initial *rough cut* configuration of the encoder by chosing a performance preset and optional tune factor **x265_preset_names** and **x265_tune_names** respectively hold the string names of the presets and tune factors (see presets for more detail on presets and tune factors):

```
/*      returns 0 on success, negative on failure (e.g. invalid preset/tune name). */
int x265_param_default_preset(x265_param *, const char *preset, const char *tune);
```

Now you may optionally specify a profile. **x265_profile_names** contains the string names this function accepts:

```
/*      (can be NULL, in which case the function will do nothing)
 *      returns 0 on success, negative on failure (e.g. invalid profile name). */
int x265_param_apply_profile(x265_param *, const char *profile);
```

Finally you configure any remaining options by name using repeated calls to:

```
/* x265_param_parse:
 *  set one parameter by name.
 *  returns 0 on success, or returns one of the following errors.
 *  note: BAD_VALUE occurs only if it can't even parse the value,
 *  numerical range is not checked until x265_encoder_open().
 *  value=NULL means "true" for boolean options, but is a BAD_VALUE for non-booleans.
 ↪*/
#define X265_PARAM_BAD_NAME  (-1)
#define X265_PARAM_BAD_VALUE (-2)
int x265_param_parse(x265_param *p, const char *name, const char *value);
```

See *string options* for the list of options (and their descriptions) which can be set by **x265_param_parse**().

After the encoder has been created, you may release the param structure:

```
/* x265_param_free:
 *  Use x265_param_free() to release storage for an x265_param instance
 *  allocated by x265_param_alloc() */
void x265_param_free(x265_param *);
```

Note: Using these methods to allocate and release the param structures helps future-proof your code in many ways, but the x265 API is versioned in such a way that we prevent linkage against a build of x265 that does not match the version of the header you are compiling against. This is function of the X265_BUILD macro.

---

x265_encoder_parameters() may be used to get a copy of the param structure from the encoder after it has been opened, in order to see the changes made to the parameters for auto-detection and other reasons:

```
/* x265_encoder_parameters:
 *      copies the current internal set of parameters to the pointer provided
 *      by the caller.  useful when the calling application needs to know
 *      how x265_encoder_open has changed the parameters.
 *      note that the data accessible through pointers in the returned param struct
 *      (e.g. filenames) should not be modified by the calling application. */
void x265_encoder_parameters(x265_encoder *, x265_param *);
```

x265_encoder_reconfig() may be used to reconfigure encoder parameters mid-encode:

```
/* x265_encoder_reconfig:
 *       used to modify encoder parameters.
 *      various parameters from x265_param are copied.
 *      this takes effect immediately, on whichever frame is encoded next;
 *      returns 0 on success, negative on parameter validation error.
 *
 *      not all parameters can be changed; see the actual function for a
 *      detailed breakdown.  since not all parameters can be changed, moving
```

```
 *      from preset to preset may not always fully copy all relevant parameters,
 *      but should still work usably in practice. however, more so than for
 *      other presets, many of the speed shortcuts used in ultrafast cannot be
 *      switched out of; using reconfig to switch between ultrafast and other
 *      presets is not recommended without a more fine-grained breakdown of
 *      parameters to take this into account. */
int x265_encoder_reconfig(x265_encoder *, x265_param *);
```

Pictures

Raw pictures are passed to the encoder via the **x265_picture** structure. Just like the param structure we recommend you allocate this structure from the encoder to avoid potential size mismatches:

```
/* x265_picture_alloc:
 *  Allocates an x265_picture instance. The returned picture structure is not
 *  special in any way, but using this method together with x265_picture_free()
 *  and x265_picture_init() allows some version safety. New picture fields will
 *  always be added to the end of x265_picture */
x265_picture *x265_picture_alloc();
```

Regardless of whether you allocate your picture structure this way or whether you simply declare it on the stack, your next step is to initialize the structure via:

```
/***
 * Initialize an x265_picture structure to default values. It sets the pixel
 * depth and color space to the encoder's internal values and sets the slice
 * type to auto - so the lookahead will determine slice type.
 */
void x265_picture_init(x265_param *param, x265_picture *pic);
```

x265 does not perform any color space conversions, so the raw picture's color space (chroma sampling) must match the color space specified in the param structure used to allocate the encoder. **x265_picture_init** initializes this field to the internal color space and it is best to leave it unmodified.

The picture bit depth is initialized to be the encoder's internal bit depth but this value should be changed to the actual depth of the pixels being passed into the encoder. If the picture bit depth is more than 8, the encoder assumes two bytes are used to represent each sample (little-endian shorts).

The user is responsible for setting the plane pointers and plane strides (in units of bytes, not pixels). The presentation time stamp (**pts**) is optional, depending on whether you need accurate decode time stamps (**dts**) on output.

If you wish to override the lookahead or rate control for a given picture you may specify a slicetype other than X265_TYPE_AUTO, or a forceQP value other than 0.

x265 does not modify the picture structure provided as input, so you may reuse a single **x265_picture** for all pictures passed to a single encoder, or even all pictures passed to multiple encoders.

Structures allocated from the library should eventually be released:

```
/* x265_picture_free:
 *  Use x265_picture_free() to release storage for an x265_picture instance
 *  allocated by x265_picture_alloc() */
void x265_picture_free(x265_picture *);
```

Analysis Buffers

Analysis information can be saved and reused to between encodes of the same video sequence (generally for multiple bitrate encodes). The best results are attained by saving the analysis information of the highest bitrate encode and reuse it in lower bitrate encodes.

When saving or loading analysis data, buffers must be allocated for every picture passed into the encoder using:

```
/* x265_alloc_analysis_data:
 *  Allocate memory to hold analysis meta data, returns 1 on success else 0 */
int x265_alloc_analysis_data(x265_picture*);
```

Note that this is very different from the typical semantics of **x265_picture**, which can be reused many times. The analysis buffers must be re-allocated for every input picture.

Analysis buffers passed to the encoder are owned by the encoder until they pass the buffers back via an output **x265_picture**. The user is responsible for releasing the buffers when they are finished with them via:

```
/* x265_free_analysis_data:
 *  Use x265_free_analysis_data to release storage of members allocated by
 *  x265_alloc_analysis_data */
void x265_free_analysis_data(x265_picture*);
```

Encode Process

The output of the encoder is a series of NAL packets, which are always returned concatenated in consecutive memory. HEVC streams have SPS and PPS and VPS headers which describe how the following packets are to be decoded. If you specified *--repeat-headers* then those headers will be output with every keyframe. Otherwise you must explicitly query those headers using:

```
/* x265_encoder_headers:
 *      return the SPS and PPS that will be used for the whole stream.
 *      *pi_nal is the number of NAL units outputted in pp_nal.
 *      returns negative on error, total byte size of payload data on success
 *      the payloads of all output NALs are guaranteed to be sequential in memory. */
int x265_encoder_headers(x265_encoder *, x265_nal **pp_nal, uint32_t *pi_nal);
```

Now we get to the main encode loop. Raw input pictures are passed to the encoder in display order via:

```
/* x265_encoder_encode:
 *      encode one picture.
 *      *pi_nal is the number of NAL units outputted in pp_nal.
 *      returns negative on error, zero if no NAL units returned.
 *      the payloads of all output NALs are guaranteed to be sequential in memory. */
int x265_encoder_encode(x265_encoder *encoder, x265_nal **pp_nal, uint32_t *pi_nal,␣
↪x265_picture *pic_in, x265_picture *pic_out);
```

These pictures are queued up until the lookahead is full, and then the frame encoders in turn are filled, and then finally you begin receiving a output NALs (corresponding to a single output picture) with each input picture you pass into the encoder.

Once the pipeline is completely full, **x265_encoder_encode()** will block until the next output picture is complete.

Note: Optionally, if the pointer of a second **x265_picture** structure is provided, the encoder will fill it with data pertaining to the output picture corresponding to the output NALs, including the recontructed image, POC and decode

timestamp. These pictures will be in encode (or decode) order.

When the last of the raw input pictures has been sent to the encoder, **x265_encoder_encode()** must still be called repeatedly with a *pic_in* argument of 0, indicating a pipeline flush, until the function returns a value less than or equal to 0 (indicating the output bitstream is complete).

At any time during this process, the application may query running statistics from the encoder:

```
/* x265_encoder_get_stats:
 *       returns encoder statistics */
void x265_encoder_get_stats(x265_encoder *encoder, x265_stats *, uint32_t␣
↪statsSizeBytes);
```

Cleanup

At the end of the encode, the application will want to trigger logging of the final encode statistics, if *--csv* had been specified:

```
/* x265_encoder_log:
 *       write a line to the configured CSV file.  If a CSV filename was not
 *       configured, or file open failed, or the log level indicated frame level
 *       logging, this function will perform no write. */
void x265_encoder_log(x265_encoder *encoder, int argc, char **argv);
```

Finally, the encoder must be closed in order to free all of its resources. An encoder that has been flushed cannot be restarted and reused. Once **x265_encoder_close()** has been called, the encoder handle must be discarded:

```
/* x265_encoder_close:
 *      close an encoder handler */
void x265_encoder_close(x265_encoder *);
```

When the application has completed all encodes, it should call **x265_cleanup()** to free process global, particularly if a memory-leak detection tool is being used. **x265_cleanup()** also resets the saved CTU size so it will be possible to create a new encoder with a different CTU size:

```
/* x265_cleanup:
 *     release library static allocations, reset configured CTU size */
void x265_cleanup(void);
```

Multi-library Interface

If your application might want to make a runtime selection between a number of libx265 libraries (perhaps 8bpp and 16bpp), then you will want to use the multi-library interface.

Instead of directly using all of the **x265_** methods documented above, you query an x265_api structure from your libx265 and then use the function pointers within that structure of the same name, but without the **x265_** prefix. So **x265_param_default()** becomes **api->param_default()**. The key method is x265_api_get():

```
/* x265_api_get:
 *   Retrieve the programming interface for a linked x265 library.
 *   May return NULL if no library is available that supports the
 *   requested bit depth. If bitDepth is 0, the function is guarunteed
 *   to return a non-NULL x265_api pointer from the system default
```

```
 *   libx265 */
const x265_api* x265_api_get(int bitDepth);
```

Note that using this multi-library API in your application is only the first step.

Your application must link to one build of libx265 (statically or dynamically) and this linked version of libx265 will support one bit-depth (8 or 10 bits).

Your application must now request the API for the bitDepth you would prefer the encoder to use (8 or 10). If the requested bitdepth is zero, or if it matches the bitdepth of the system default libx265 (the currently linked library), then this library will be used for encode. If you request a different bit-depth, the linked libx265 will attempt to dynamically bind a shared library with a name appropriate for the requested bit-depth:

8-bit: libx265_main.dll 10-bit: libx265_main10.dll

(the shared library extension is obviously platform specific. On Linux it is .so while on Mac it is .dylib)

For example on Windows, one could package together an x265.exe statically linked against the 8bpp libx265 together with a libx265_main10.dll in the same folder, and this executable would be able to encode main and main10 bitstreams.

On Linux, x265 packagers could install 8bpp static and shared libraries under the name libx265 (so all applications link against 8bpp libx265) and then also install libx265_main10.so (symlinked to its numbered solib). Thus applications which use x265_api_get() will be able to generate main or main10 bitstreams.

CHAPTER 4

Threading

Thread Pools

x265 creates one or more thread pools per encoder, one pool per NUMA node (typically a CPU socket). `--pools` specifies the number of pools and the number of threads per pool the encoder will allocate. By default x265 allocates one thread per (hyperthreaded) CPU core on each NUMA node.

If you are running multiple encoders on a system with multiple NUMA nodes, it is recommended to isolate each of them to a single node in order to avoid the NUMA overhead of remote memory access.

Work distribution is job based. Idle worker threads scan the job providers assigned to their thread pool for jobs to perform. When no jobs are available, the idle worker threads block and consume no CPU cycles.

Objects which desire to distribute work to worker threads are known as job providers (and they derive from the JobProvider class). The thread pool has a method to **poke** awake a blocked idle thread, and job providers are recommended to call this method when they make new jobs available.

Worker jobs are not allowed to block except when abosultely necessary for data locking. If a job becomes blocked, the work function is expected to drop that job so the worker thread may go back to the pool and find more work.

On Windows, the native APIs offer sufficient functionality to discover the NUMA topology and enforce the thread affinity that libx265 needs (so long as you have not chosen to target XP or Vista), but on POSIX systems it relies on libnuma for this functionality. If your target POSIX system is single socket, then building without libnuma is a perfectly reasonable option, as it will have no effect on the runtime behavior. On a multiple-socket system, a POSIX build of libx265 without libnuma will be less work efficient, but will still function correctly. You lose the work isolation effect that keeps each frame encoder from only using the threads of a single socket and so you incur a heavier context switching cost.

Wavefront Parallel Processing

New with HEVC, Wavefront Parallel Processing allows each row of CTUs to be encoded in parallel, so long as each row stays at least two CTUs behind the row above it, to ensure the intra references and other data of the blocks above and above-right are available. WPP has almost no effect on the analysis and compression of each CTU and so it has

a very small impact on compression efficiency relative to slices or tiles. The compression loss from WPP has been found to be less than 1% in most of our tests.

WPP has three effects which can impact efficiency. The first is the row starts must be signaled in the slice header, the second is each row must be padded to an even byte in length, and the third is the state of the entropy coder is transferred from the second CTU of each row to the first CTU of the row below it. In some conditions this transfer of state actually improves compression since the above-right state may have better locality than the end of the previous row.

Parabola Research have published an excellent HEVC animation which visualizes WPP very well. It even correctly visualizes some of WPPs key drawbacks, such as:

1. the low thread utilization at the start and end of each frame
2. a difficult block may stall the wave-front and it takes a while for the wave-front to recover.
3. 64x64 CTUs are big! there are much fewer rows than with H.264 and similar codecs

Because of these stall issues you rarely get the full parallelisation benefit one would expect from row threading. 30% to 50% of the theoretical perfect threading is typical.

In x265 WPP is enabled by default since it not only improves performance at encode but it also makes it possible for the decoder to be threaded.

If WPP is disabled by *--no-wpp* the frame will be encoded in scan order and the entropy overheads will be avoided. If frame threading is not disabled, the encoder will change the default frame thread count to be higher than if WPP was enabled. The exact formulas are described in the next section.

Bonded Task Groups

If a worker thread job has work which can be performed in parallel by many threads, it may allocate a bonded task group and enlist the help of other idle worker threads in the same pool. Those threads will cooperate to complete the work of the bonded task group and then return to their idle states. The larger and more uniform those tasks are, the better the bonded task group will perform.

Parallel Mode Analysis

When *--pmode* is enabled, each CU (at all depths from 64x64 to 8x8) will distribute its analysis work to the thread pool via a bonded task group. Each analysis job will measure the cost of one prediction for the CU: merge, skip, intra, inter (2Nx2N, Nx2N, 2NxN, and AMP). At slower presets, the amount of increased parallelism is often enough to be able to reduce frame parallelism while achieving the same overall CPU utilization. Reducing frame threads is often beneficial to ABR and VBV rate control.

Parallel Motion Estimation

When *--pme* is enabled all of the analysis functions which perform motion searches to reference frames will distribute those motion searches as jobs for worker threads via a bonded task group (if more than two motion searches are required).

Frame Threading

Frame threading is the act of encoding multiple frames at the same time. It is a challenge because each frame will generally use one or more of the previously encoded frames as motion references and those frames may still be in the

process of being encoded themselves.

Previous encoders such as x264 worked around this problem by limiting the motion search region within these reference frames to just one macroblock row below the coincident row being encoded. Thus a frame could be encoded at the same time as its reference frames so long as it stayed one row behind the encode progress of its references (glossing over a few details).

x265 has the same frame threading mechanism, but we generally have much less frame parallelism to exploit than x264 because of the size of our CTU rows. For instance, with 1080p video x264 has 68 16x16 macroblock rows available each frame while x265 only has 17 64x64 CTU rows.

The second extenuating circumstance is the loop filters. The pixels used for motion reference must be processed by the loop filters and the loop filters cannot run until a full row has been encoded, and it must run a full row behind the encode process so that the pixels below the row being filtered are available. On top of this, HEVC has two loop filters: deblocking and SAO, which must be run in series with a row lag between them. When you add up all the row lags each frame ends up being 3 CTU rows behind its reference frames (the equivalent of 12 macroblock rows for x264). And keep in mind the wave-front progression pattern; by the time the reference frame finishes the third row of CTUs, nearly half of the CTUs in the frame may be compressed (depending on the display aspect ratio).

The third extenuating circumstance is that when a frame being encoded becomes blocked by a reference frame row being available, that frame's wave-front becomes completely stalled and when the row becomes available again it can take quite some time for the wave to be restarted, if it ever does. This makes WPP less effective when frame parallelism is in use.

--merange can have a negative impact on frame parallelism. If the range is too large, more rows of CTU lag must be added to ensure those pixels are available in the reference frames.

Note: Even though the merange is used to determine the amount of reference pixels that must be available in the reference frames, the actual motion search is not necessarily centered around the coincident block. The motion search is actually centered around the motion predictor, but the available pixel area (mvmin, mvmax) is determined by merange and the interpolation filter half-heights.

When frame threading is disabled, the entirety of all reference frames are always fully available (by definition) and thus the available pixel area is not restricted at all, and this can sometimes improve compression efficiency. Because of this, the output of encodes with frame parallelism disabled will not match the output of encodes with frame parallelism enabled; but when enabled the number of frame threads should have no effect on the output bitstream except when using ABR or VBV rate control or noise reduction.

When `--nr` is enabled, the outputs of each number of frame threads will be deterministic but none of them will match becaue each frame encoder maintains a cumulative noise reduction state.

VBV introduces non-determinism in the encoder, at this point in time, regardless of the amount of frame parallelism.

By default frame parallelism and WPP are enabled together. The number of frame threads used is auto-detected from the (hyperthreaded) CPU core count, but may be manually specified via *--frame-threads*

Cores	Frames
> 32	6..8
>= 16	5
>= 8	3
>= 4	2

If WPP is disabled, then the frame thread count defaults to **min(cpuCount, ctuRows / 2)**

Over-allocating frame threads can be very counter-productive. They each allocate a large amount of memory and because of the limited number of CTU rows and the reference lag, you generally get limited benefit from adding frame encoders beyond the auto-detected count, and often the extra frame encoders reduce performance.

Given these considerations, you can understand why the faster presets lower the max CTU size to 32x32 (making twice as many CTU rows available for WPP and for finer grained frame parallelism) and reduce *--merange*

Each frame encoder runs in its own thread (allocated separately from the worker pool). This frame thread has some pre-processing responsibilities and some post-processing responsibilities for each frame, but it spends the bulk of its time managing the wave-front processing by making CTU rows available to the worker threads when their dependencies are resolved. The frame encoder threads spend nearly all of their time blocked in one of 4 possible locations:

1. blocked, waiting for a frame to process
2. blocked on a reference frame, waiting for a CTU row of reconstructed and loop-filtered reference pixels to become available
3. blocked waiting for wave-front completion
4. blocked waiting for the main thread to consume an encoded frame

Lookahead

The lookahead module of x265 (the lowres pre-encode which determines scene cuts and slice types) uses the thread pool to distribute the lowres cost analysis to worker threads. It will use bonded task groups to perform batches of frame cost estimates, and it may optionally use bonded task groups to measure single frame cost estimates using slices. (see *--lookahead-slices*)

The function slicetypeDecide() itself is also be performed by a worker thread if your encoder has a thread pool, else it runs within the context of the thread which calls the x265_encoder_encode().

SAO

The Sample Adaptive Offset loopfilter has a large effect on encode performance because of the peculiar way it must be analyzed and coded.

SAO flags and data are encoded at the CTU level before the CTU itself is coded, but SAO analysis (deciding whether to enable SAO and with what parameters) cannot be performed until that CTU is completely analyzed (reconstructed pixels are available) as well as the CTUs to the right and below. So in effect the encoder must perform SAO analysis in a wavefront at least a full row behind the CTU compression wavefront.

This extra latency forces the encoder to save the encode data of every CTU until the entire frame has been analyzed, at which point a function can code the final slice bitstream with the decided SAO flags and data interleaved between each CTU. This second pass over the CTUs can be expensive, particularly at large resolutions and high bitrates.

CHAPTER 5

Preset Options

Presets

x265 has a number of predefined `--preset` options that make trade-offs between encode speed (encoded frames per second) and compression efficiency (quality per bit in the bitstream). The default preset is medium, it does a reasonably good job of finding the best possible quality without spending enormous CPU cycles looking for the absolute most efficient way to achieve that quality. As you go higher than medium, the encoder takes shortcuts to improve performance at the expense of quality and compression efficiency. As you go lower than medium, the encoder tries harder and harder to achieve the best quailty per bit compression ratio.

The presets adjust encoder parameters to affect these trade-offs.

	ultra-fast	super-fast	very-fast	faster	fast	medium	slow	slower	verys-low	placebo
ctu	32	32	32	64	64	64	64	64	64	64
min-cu-size	16	8	8	8	8	8	8	8	8	8
bframes	3	3	4	4	4	4	4	8	8	8
b-adapt	0	0	0	0	0	2	2	2	2	2
rc-lookahead	5	10	15	15	15	20	25	30	40	60
scenecut	0	40	40	40	40	40	40	40	40	40
refs	1	1	1	1	2	3	3	3	5	5
me	dia	hex	hex	hex	hex	hex	star	star	star	star
merange	57	57	57	57	57	57	57	57	57	92
subme	0	1	1	2	2	2	3	3	4	5
rect	0	0	0	0	0	0	1	1	1	1
amp	0	0	0	0	0	0	0	1	1	1
max-merge	2	2	2	2	2	2	3	3	4	5
early-skip	1	1	1	1	0	0	0	0	0	0
fast-intra	1	1	1	1	1	0	0	0	0	0
b-intra	0	0	0	0	0	0	0	1	1	1
sao	0	0	1	1	1	1	1	1	1	1
signhide	0	1	1	1	1	1	1	1	1	1
weightp	0	0	1	1	1	1	1	1	1	1
weightb	0	0	0	0	0	0	0	1	1	1
aq-mode	0	0	1	1	1	1	1	1	1	1
cuTree	0	0	0	0	1	1	1	1	1	1
rdLevel	2	2	2	2	2	3	4	6	6	6
rdoq-level	0	0	0	0	0	0	2	2	2	2
tu-intra	1	1	1	1	1	1	1	2	3	4
tu-inter	1	1	1	1	1	1	1	2	3	4

Placebo mode enables transform-skip prediction evaluation.

Tuning

There are a few `--tune` options available, which are applied after the preset.

Note: The *psnr* and *ssim* tune options disable all optimizations that sacrafice metric scores for perceived visual quality (also known as psycho-visual optimizations). By default x265 always tunes for highest perceived visual quality but if one intends to measure an encode using PSNR or SSIM for the purpose of benchmarking, we highly recommend you configure x265 to tune for that particular metric.

–tune	effect
psnr	disables adaptive quant, psy-rd, and cutree
ssim	enables adaptive quant auto-mode, disables psy-rd
grain	improves retention of film grain. more below
fastdecode	no loop filters, no weighted pred, no intra in B
zerolatency	no lookahead, no B frames, no cutree

Film Grain Retention

--tune grain tries to improve the retention of film grain in the reconstructed output. It helps rate distortion optimizations select modes which preserve high frequency noise:

- *--psy-rd* 0.5
- *--rdoq-level* 1
- *--psy-rdoq* 30

It lowers the strength of adaptive quantization, so residual energy can be more evenly distributed across the (noisy) picture:

- *--aq-strength* 0.3

And it similarly tunes rate control to prevent the slice QP from swinging too wildly from frame to frame:

- *--ipratio* 1.1
- *--pbratio* 1.1
- *--qcomp* 0.8

And lastly it reduces the strength of deblocking to prevent grain being blurred on block boundaries:

- *--deblock* -2

Fast Decode

--tune fastdecode disables encoder features which tend to be bottlenecks for the decoder. It is intended for use with 4K content at high bitrates which can cause decoders to struggle. It disables both HEVC loop filters, which tend to be process bottlenecks:

- *--no-deblock*
- *--no-sao*

It disables weighted prediction, which tend to be bandwidth bottlenecks:

- *--no-weightp*
- *--no-weightb*

And it disables intra blocks in B frames with *--no-b-intra* since intra predicted blocks cause serial dependencies in the decoder.

Zero Latency

There are two halves to the latency problem. There is latency at the decoder and latency at the encoder. *--tune zerolatency* removes latency from both sides. The decoder latency is removed by:

- *--bframes* 0

Encoder latency is removed by:

- *--b-adapt* 0
- *--rc-lookahead* 0
- *--no-scenecut*
- *--no-cutree*

- *--frame-threads* 1

With all of these settings x265_encoder_encode() will run synchronously, the picture passed as pic_in will be encoded and returned as NALs. These settings disable frame parallelism, which is an important component for x265 performance. If you can tolerate any latency on the encoder, you can increase performance by increasing the number of frame threads. Each additional frame thread adds one frame of latency.

CHAPTER 6

Lossless

Lossless Encoding

x265 can encode HEVC bitstreams that are entirely lossless (the reconstructed images are bit-exact to the source images) by using the *--lossless* option. Lossless operation is theoretically simple. Rate control, by definition, is disabled and the encoder disables all quality metrics since they would only waste CPU cycles. Instead, x265 reports only a compression factor at the end of the encode.

In HEVC, lossless coding means bypassing both the DCT transforms and bypassing quantization (often referred to as transquant bypass). Normal predictions are still allowed, so the encoder will find optimal inter or intra predictions and then losslessly code the residual (with transquant bypass).

All *--preset* options are capable of generating lossless video streams, but in general the slower the preset the better the compression ratio (and the slower the encode). Here are some examples:

```
./x265 ../test-720p.y4m o.bin --preset ultrafast --lossless
... <snip> ...
encoded 721 frames in 238.38s (3.02 fps), 57457.94 kb/s

./x265 ../test-720p.y4m o.bin --preset faster --lossless
... <snip> ...
x265 [info]: lossless compression ratio 3.11::1
encoded 721 frames in 258.46s (2.79 fps), 56787.65 kb/s

./x265 ../test-720p.y4m o.bin --preset slow --lossless
... <snip> ...
x265 [info]: lossless compression ratio 3.36::1
encoded 721 frames in 576.73s (1.25 fps), 52668.25 kb/s

./x265 ../test-720p.y4m o.bin --preset veryslow --lossless
x265 [info]: lossless compression ratio 3.76::1
encoded 721 frames in 6298.22s (0.11 fps), 47008.65 kb/s
```

Note: In HEVC, only QP=4 is truly lossless quantization, and thus when encoding losslesly x265 uses QP=4 internally

in its RDO decisions.

Near-lossless Encoding

Near-lossless conditions are a quite a bit more interesting. Normal ABR rate control will allow one to scale the bitrate up to the point where quantization is entirely bypassed (QP <= 4), but even at this point there is a lot of SSIM left on the table because of the DCT transforms, which are not lossless:

```
./x265 ../test-720p.y4m o.bin --preset medium --bitrate 40000 --ssim
encoded 721 frames in 326.62s (2.21 fps), 39750.56 kb/s, SSIM Mean Y: 0.9990703 (30.
↪317 dB)

./x265 ../test-720p.y4m o.bin --preset medium --bitrate 50000 --ssim
encoded 721 frames in 349.27s (2.06 fps), 44326.84 kb/s, SSIM Mean Y: 0.9994134 (32.
↪316 dB)

./x265 ../test-720p.y4m o.bin --preset medium --bitrate 60000 --ssim
encoded 721 frames in 360.04s (2.00 fps), 45394.50 kb/s, SSIM Mean Y: 0.9994823 (32.
↪859 dB)
```

For the encoder to get over this quality plateau, one must enable lossless coding at the CU level with *--cu-lossless*. It tells the encoder to evaluate trans-quant bypass as a coding option for each CU, and to pick the option with the best rate-distortion characteristics.

The *--cu-lossless* option is very expensive, computationally, and it only has a positive effect when the QP is extremely low, allowing RDO to spend a large amount of bits to make small improvements to quality. So this option should only be enabled when you are encoding near-lossless bitstreams:

```
./x265 ../test-720p.y4m o.bin --preset medium --bitrate 40000 --ssim --cu-lossless
encoded 721 frames in 500.51s (1.44 fps), 40017.10 kb/s, SSIM Mean Y: 0.9997790 (36.
↪557 dB)

./x265 ../test-720p.y4m o.bin --preset medium --bitrate 50000 --ssim --cu-lossless
encoded 721 frames in 524.60s (1.37 fps), 46083.37 kb/s, SSIM Mean Y: 0.9999432 (42.
↪456 dB)

./x265 ../test-720p.y4m o.bin --preset medium --bitrate 60000 --ssim --cu-lossless
encoded 721 frames in 523.63s (1.38 fps), 46552.92 kb/s, SSIM Mean Y: 0.9999489 (42.
↪917 dB)
```

Note: It is not unusual for bitrate to drop as you increase lossless coding. Having "perfectly coded" reference blocks reduces residual in later frames. It is quite possible for a near-lossless encode to spend more bits than a lossless encode.

Enabling psycho-visual rate distortion will improve lossless coding. *--psy-rd* influences the RDO decisions in favor of energy (detail) preservation over bit cost and results in more blocks being losslessly coded. Our psy-rd feature is not yet assembly optimized, so this makes the encodes run even slower:

```
./x265 ../test-720p.y4m o.bin --preset medium --bitrate 40000 --ssim --cu-lossless --
↪psy-rd 1.0
encoded 721 frames in 581.83s (1.24 fps), 40112.15 kb/s, SSIM Mean Y: 0.9998632 (38.
↪638 dB)

./x265 ../test-720p.y4m o.bin --preset medium --bitrate 50000 --ssim --cu-lossless --
↪psy-rd 1.0
```

```
encoded 721 frames in 587.54s (1.23 fps), 46284.55 kb/s, SSIM Mean Y: 0.9999663 (44.
↪721 dB)

./x265 ../test-720p.y4m o.bin --preset medium --bitrate 60000 --ssim --cu-lossless --
↪psy-rd 1.0
encoded 721 frames in 592.93s (1.22 fps), 46839.51 kb/s, SSIM Mean Y: 0.9999707 (45.
↪334 dB)
```

--cu-lossless will also be more effective at slower presets which perform RDO at more levels and thus may find smaller blocks that would benefit from lossless coding:

```
./x265 ../test-720p.y4m o.bin --preset veryslow --bitrate 40000 --ssim --cu-lossless
encoded 721 frames in 12969.25s (0.06 fps), 37331.96 kb/s, SSIM Mean Y: 0.9998108 (37.
↪231 dB)

./x265 ../test-720p.y4m o.bin --preset veryslow --bitrate 50000 --ssim --cu-lossless
encoded 721 frames in 46217.84s (0.05 fps), 42976.28 kb/s, SSIM Mean Y: 0.9999482 (42.
↪856 dB)

./x265 ../test-720p.y4m o.bin --preset veryslow --bitrate 60000 --ssim --cu-lossless
encoded 721 frames in 13738.17s (0.05 fps), 43864.21 kb/s, SSIM Mean Y: 0.9999633 (44.
↪348 dB)
```

And with psy-rd and a slow preset together, very high SSIMs are possible:

```
./x265 ../test-720p.y4m o.bin --preset veryslow --bitrate 40000 --ssim --cu-lossless -
↪-psy-rd 1.0
encoded 721 frames in 11675.81s (0.06 fps), 37819.45 kb/s, SSIM Mean Y: 0.9999181 (40.
↪867 dB)

./x265 ../test-720p.y4m o.bin --preset veryslow --bitrate 50000 --ssim --cu-lossless -
↪-psy-rd 1.0
encoded 721 frames in 12414.56s (0.06 fps), 42815.75 kb/s, SSIM Mean Y: 0.9999758 (46.
↪168 dB)

./x265 ../test-720p.y4m o.bin --preset veryslow --bitrate 60000 --ssim --cu-lossless -
↪-psy-rd 1.0
encoded 721 frames in 11684.89s (0.06 fps), 43324.48 kb/s, SSIM Mean Y: 0.9999793 (46.
↪835 dB)
```

It's important to note in the end that it is easier (less work) for the encoder to encode the video losslessly than it is to encode it near-losslessly. If the encoder knows up front the encode must be lossless, it does not need to evaluate any lossy coding methods. The encoder only needs to find the most efficient prediction for each block and then entropy code the residual.

It is not feasible for *--cu-lossless* to turn itself on when the encoder determines it is encoding a near-lossless bitstream (ie: when rate control nearly disables all quantization) because the feature requires a flag to be enabled in the stream headers. At the time the stream headers are being coded we do not know whether *--cu-lossless* would be a help or a hinder. If very few or no blocks end up being coded as lossless, then having the feature enabled is a net loss in compression efficiency because it adds a flag that must be coded for every CU. So ignoring even the performance aspects of the feature, it can be a compression loss if enabled without being used. So it is up to the user to only enable this feature when they are coding at near-lossless quality.

Transform Skip

A somewhat related feature, `--tskip` tells the encoder to evaluate transform-skip (bypass DCT but with quantization still enabled) when coding small 4x4 transform blocks. This feature is intended to improve the coding efficiency of screen content (aka: text on a screen) and is not really intended for lossless coding. This feature should only be enabled if the content has a lot of very sharp edges in it, and is mostly unrelated to lossless coding.

Index

Symbols

C