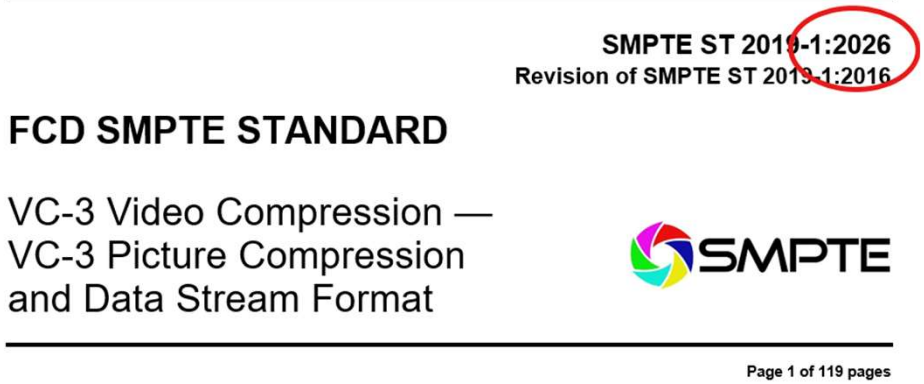

Avid DNx 4 SDK
Overview/ Specification

Avid DNx evolution

- **Older versions**
  - 8-bit bit depth limitation removed on levels LB, SQ and HQ
  - ST 2019-1: Amd1 (2023) support, based on ST 2019-1:**2016**
- **DNx SDK 2.7.5** (Aug ’23)
  - High core-count machine support
  - SDI clipping removed
- **DNx SDK 3.0** (old code base, long-term maintenance release; March ‘25)
  - ST 2019:**2016** compliant, no update to 2025 revision
  - Cleanup, redundancy purge (prep for 4.0 release)
  - Compatible implementation of customizable bitrates (requires use of VBR)
  - ACF control option (RGB encoding)

What's new?

- **ST 2019-1:2026 (VC-3 standard) revision**
  - Massive re-edit for clarification and ambiguity removal
  - Incorporates Amd 1:2023
  - Adds DNx GX (Filler) into the mainstream definition
  - New header field CUS to allow customizable CBR bitrates (previous: VBR only)
- **RP 2019-2:2026 (VC-3 compliance) revision**
  - New test set, covering the 2026 feature set
  - All new reference decoder code, extensively tested and reviewed
- **ST 2019-4: 2026 (MXF mapping) revision**
  - Cleanup of discrepancies between base standard and mapping document
  - Relaxed mapping constraints
  - **State**: *in drafting group*



SMPTE ST 2019-1:2026
Revision of SMPTE ST 2019-1:2016

FCD SMPTE STANDARD

VC-3 Video Compression —
VC-3 Picture Compression
and Data Stream Format

SMPTE

Page 1 of 119 pages

What's new?

- **DNxHD, DNxHR, (DNxGX) become "Avid DNx"**
- DNx GX (Filler) incorporated into the mainline Avid DNx codec
- All new code base (ST 2019-1:**2025** compliant)
- New interfaces, more control and customizations
- New rate controllers
- More platforms with native performance tuning
- Customizable bitrates (extended) and now including CBR
- **HDR editing at affordable bitrates** (see bit depth independence section)
- Compressed Merge
- **Version 4.5**: Asynchronous I/O interface

"DNxHD"/"DNxHR" become "Avid DNx"

- True resolution independence (all component formats)
  - **Spatial, bit depth, color coding**
- Simplified and converged naming scheme
  - **no more profile/bit depth differentiation**
- New license required
  - **No** change of terms (fees) *required*, but…
  - Updated compliance terms
  - More license options (you may find a cheaper option)
  - *License precondition*: SMPTE ST 2019-1:**2026** compliance
- RGB supported on all levels (progressive)
- Alpha supported on all levels
  - Alpha bit depth <u>separate</u> from filler

Avid DNx

Level	Bit depth	Color coding	Resolution	Sampling 4:4:4	Sampling 4:2:2	Sampling 4:2:0	Sampling RGB	Alpha	Bitrate Mbps
444	8-16	Any	up to 16384 x 16384	✓	-	-	✓	✓	116-440
HQX				-	✓	✓	-	✓	72-220
HQ				✓	✓	✓	✓	✓	36-220
SQ				✓	✓	✓	✓	✓	36-145
LB				-	✓	✓	-	✓	36-45[1]

Note: Bitrate shown is for HD (1920x1080) resolution
Lowest: 24 FPS, lowest target factor
Highest: 30 FPS, highest target factor

Actual Bitrate depends on: Resolution, Level, target factor, FPS

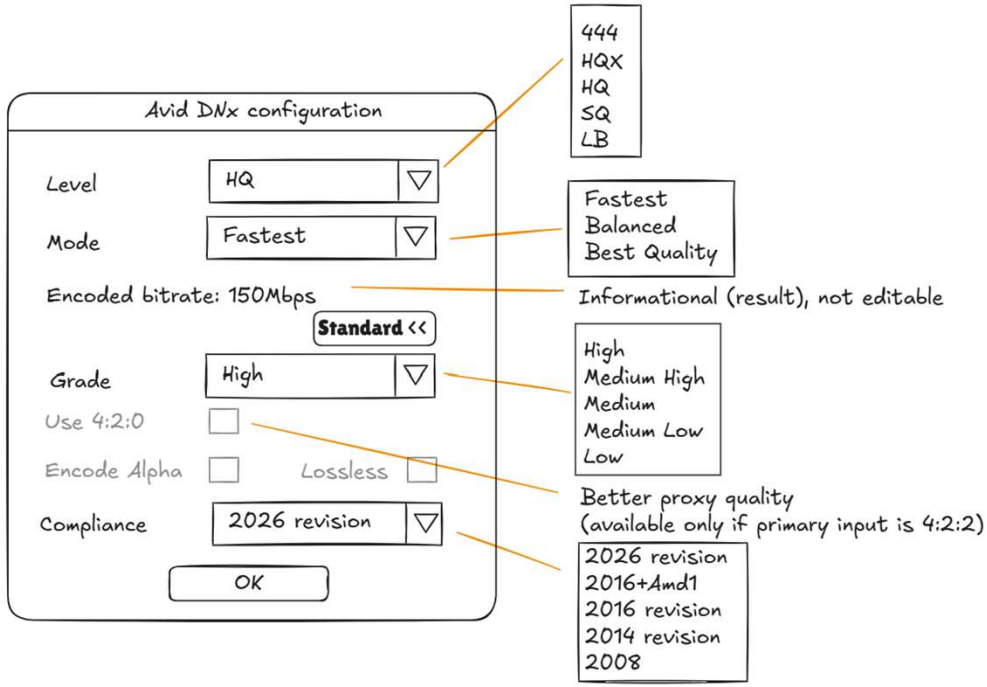
Table 6 – CDCI Picture Essence Descriptor Values

Item Name	Meaning	Source of derived data	Legal values
Component Depth	Number of active bits per sample	Component Depth of the uncompressed signal passed to the encoder	8 through 16
Horizontal	Specifies the horizontal color	SMPTE ST 2019-1	1,2

Reflecting the new naming schema

Goal: Penetrate the web-based, faulty information wall

- Refer to the codec only as "**Avid DNx**"
- **Do not** differentiate the Profiles, just Level
- **Do not** list a bit depth for the Level or in the codec configuration dialog (automatically pick the bit depth from the input, as defined in the standard)
- **Do not** differentiate color coding (pick from source; use OOB bitstream markup if not 709 or 2020)
- If possible, use DNx_FindBestCompressionID(). Otherwise: Use HD/709 only for interlaced.
- If possible, support customizable bitrates (Grade)
- If possible, offer a rate controller selection (Speed encoder is ~4 times faster)



* Concept

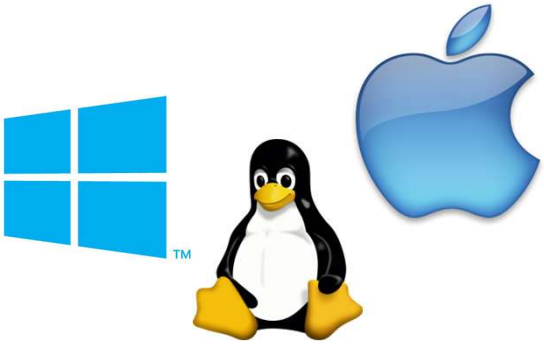
What's the difference between HQ and HQX?

The Levels HQ and HQX come with the same (max) bitrate. What's the difference?

- HQX is a compression which is less aggressive on quantization, specifically for bit depths > 8
  - Larger Quantization Range (+/-4096 for HQX vs +/- 1024 for HQ)
  - Less aggressive Q matrix
  - Different VLC coding table, specifically adapted to the higher (precision) quantization range
- Bottom line:
  - HQX may need slightly more compression pressure (qsf) to encode to the same bitrate target
  - HQ will, by default, produce more compression-based errors (PSNR) than HQX at full bitrate
  - Full bitrate is high enough that compression artefacts won't be visible (both), but when going for lower Grades visual quality of HQ will win.
  - If precision is your primary concern: Use HQX
  - If visual quality is your primary concern: Use HQ

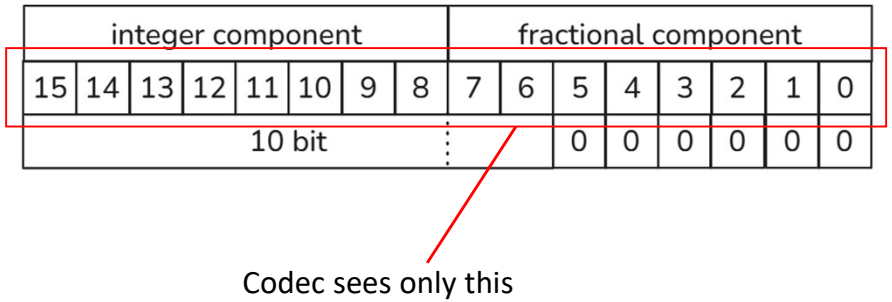
New codebase - Basics

- More platforms
  - Windows (Intel, **ARM64, ARM64EC**)
  - Linux (Intel)
  - MacOS (Neon)
- Fully SMPTE ST 2019-1:**2026** compliant
- Dedicated optimizations for each platform
- C++ API
  - Simplified instance usage
  - More detailed, verbose error information via error interface
  - Customizations



New codebase - Formats

- Only 2 canonical input/output component formats (8 bit/16 bit unsigned)
  - Bit depths 9 through 16 bits via 16-bit MSB alignment. No input clipping
  - Comprehensive viewport and line alignment support
  - Seldom used output formats dropped
  - Optional preview conversions included (zero/very low performance impact)
    - **Up-/downsample** 4:2:2 <-> 4:2:0 (enc/dec): Use 4:2:0 for proxies in 4:2:2 pipelines without buffer copy/conversion loop
    - **8-bit RGB output** with HDR/color space adoption for (confidence) preview/proxy editing on all monitor types
  - Alternate component ordering formats (BGR/RGB, UYV/YUV, UYUV/YUYV)
- Not new, but…
  - Avid DNx works with *any* color coding (primaries, transfer characteristic), specifically including *all* HDR definitions, if used in MXF
- **Custom input/output formatters** (interface)
  - Support *any* buffer format and layout
  - Requires client programming



New codebase – Quality, Performance, Compliance

- **New rate controllers**
  - High Quality (4.0): Presented in *SMPTE Motion Imaging Journal July/August 2025.* About 20% slower than the default (Balanced)
  - Balanced (4.3.2): Based on the same principle as High Quality but taking some shortcuts for faster operation. Starting with version 4.3.2 this is the new default.
  - Speed (4.3.2): Same as Balanced, but will not try to fill the bit budget if under limit after initial setup. Up to 10% faster than Balanced.
  - Highly improved adaptive, quality-optimized RGB encoding (beyond original DNxGX)
- **New Merge mode (4.3.2)**
  - Allows either binary copies without decode/re-encode, or fast transcodes, avoiding 60% of the overhead incurred in standard decode/encode multi-generation cycles.
- **Native** performance optimizations for all supported platforms
- Automatic multi-threading (number of used threads can still be defined manually, but no longer required), using oneTBB instead of proprietary solution
- Legacy format compliance mode (create bitstreams compliant to older revisions)

New codebase – Convenience/Utility functionality

- Reduced-size decodes for super-fast preview, browse and proxy workflows, specifically, when paired with the 8-bit RGB output options
- Compressed Merge support
- Viewport mapping
- Flexible alpha handling (keep, drop, insert)
- Automatic best Profile/Level identification function for encoder configuration
- Optional low-pass filtering for better proxy quality
- Energy efficiency mode for extended battery lifetime (on supporting CPUs)

Customizable bitrates (Grade)

- Already available in DNx 3.0, but tied to using VBR (requires index)
- ST 2019-1:**2026** also makes this available in CBR mode (Coding Unit Size)
- Bitrate (actually: compressed frame size) adjustable via Grade factor 0% (lowest logical) through 100% (full level bitrate)
- Use of Grade factor prevents mis-configurations
- DNX_GetTargetGrade(…) allows deriving Grade factor by specifying “desired” compressed frame size for actual resolution
- DNX_GetCompressedFrameSize(…) returns correct minimal buffer size

9.3.4 Coding Unit Size

The size of the Coding Unit (CUS) in bytes is specified as shown in Figure 17, Figure 18 and Figure 21 below:

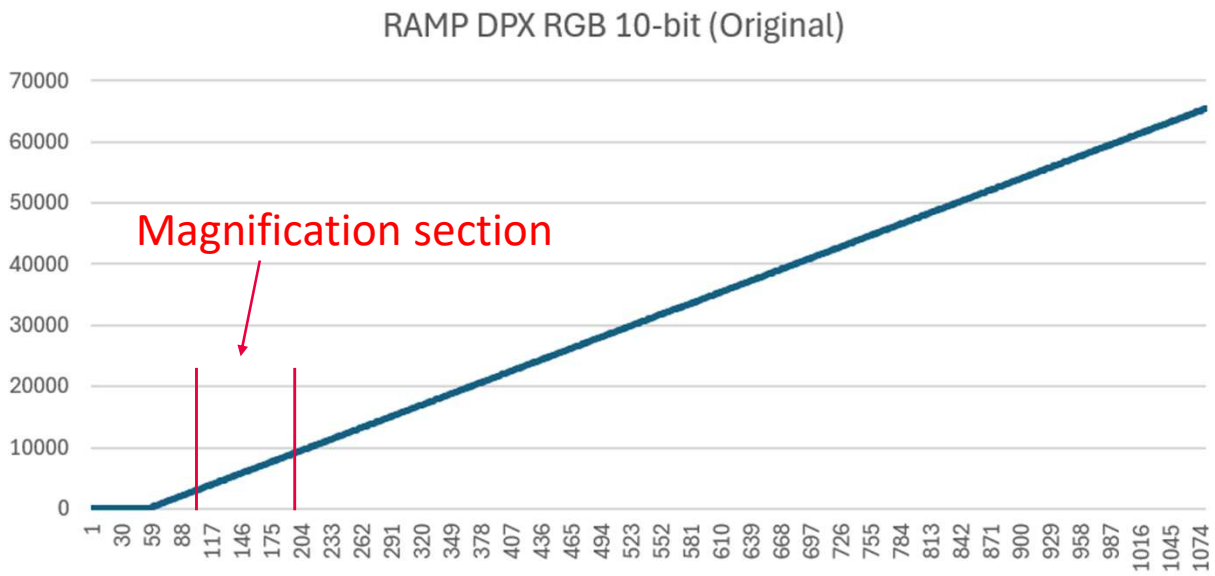
Byte Offset	MSB		LSB
0x008	CUS_3		
0x009	CUS_2		
0x00A	CUS_1		
0x00B	CUS_0		

Figure 21 — Coding Unit Size area

Note: Within the HR profile, an encoder can target any (suitable) compressed unit size within the range defined by formula (9.1) if it encodes using HVN ≥ 5, for both CBR and VBR mode. If backwards compatibility is a requirement, the encoder can target a lower bitrate only in VBR mode (HVN = 3 or 4, VBR=1, CUS=0), which will require the use of a full index in a file wrapper.

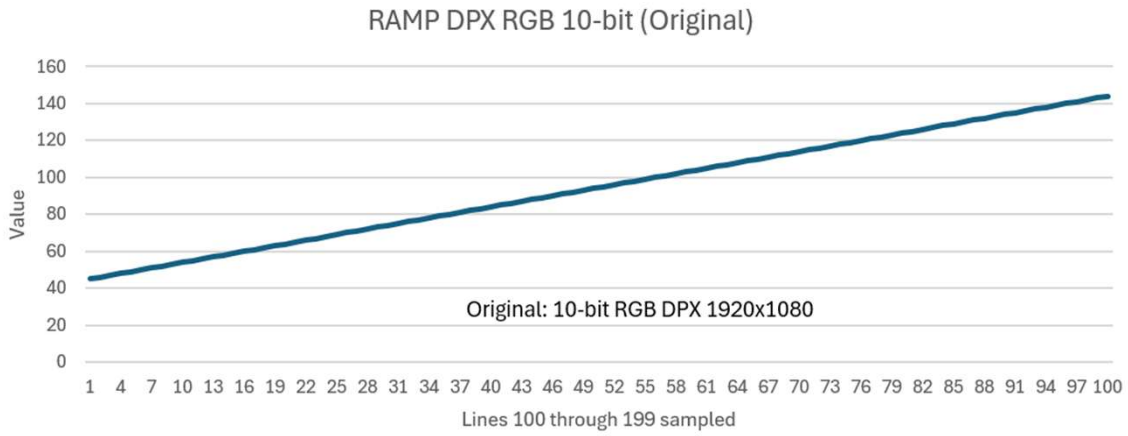
Under the hood: Bit depth independence

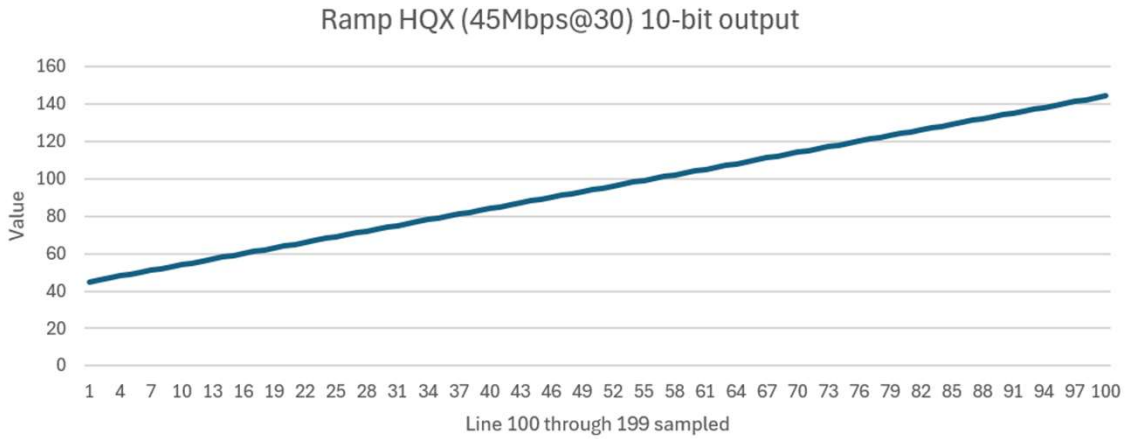
10-bit vertical ramp



In-depth description found in: *SMPTE Motion Imaging Journal July/August 2025*

Under the hood: Bit depth independence (magnified sections: Original vs HQX)

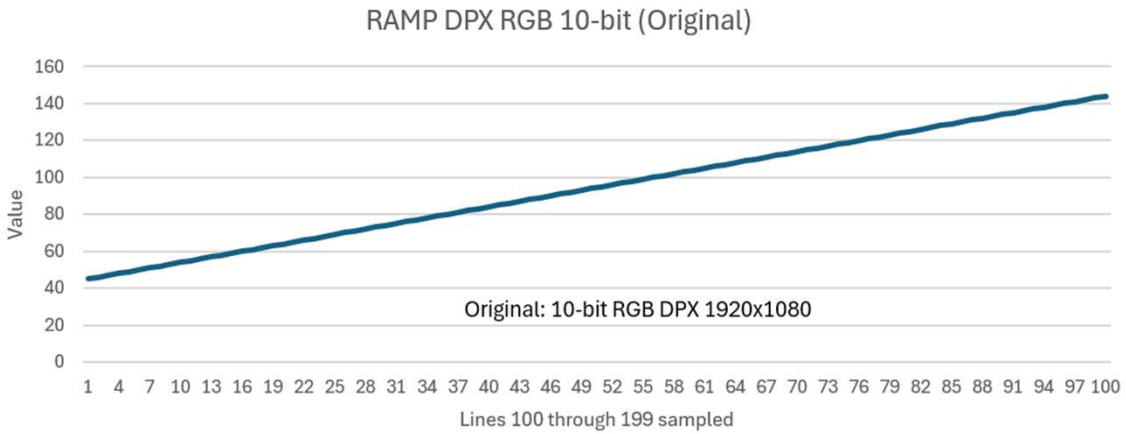


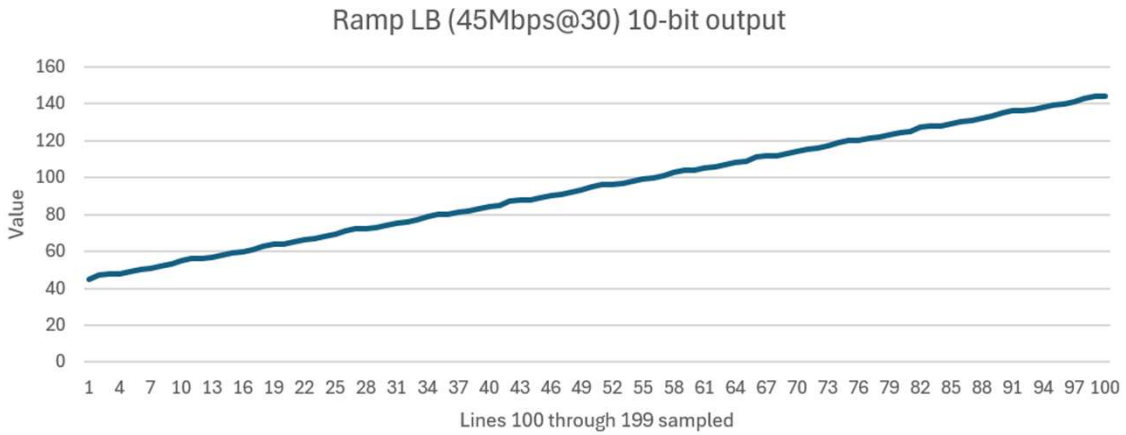


Results are **bit identical**

Note: The slight wiggles observable in both diagrams are artefacts of the line painting routine at this resolution. Values increase by 1 for each line

Under the hood: Bit depth independence (magnified sections: Original vs LB)

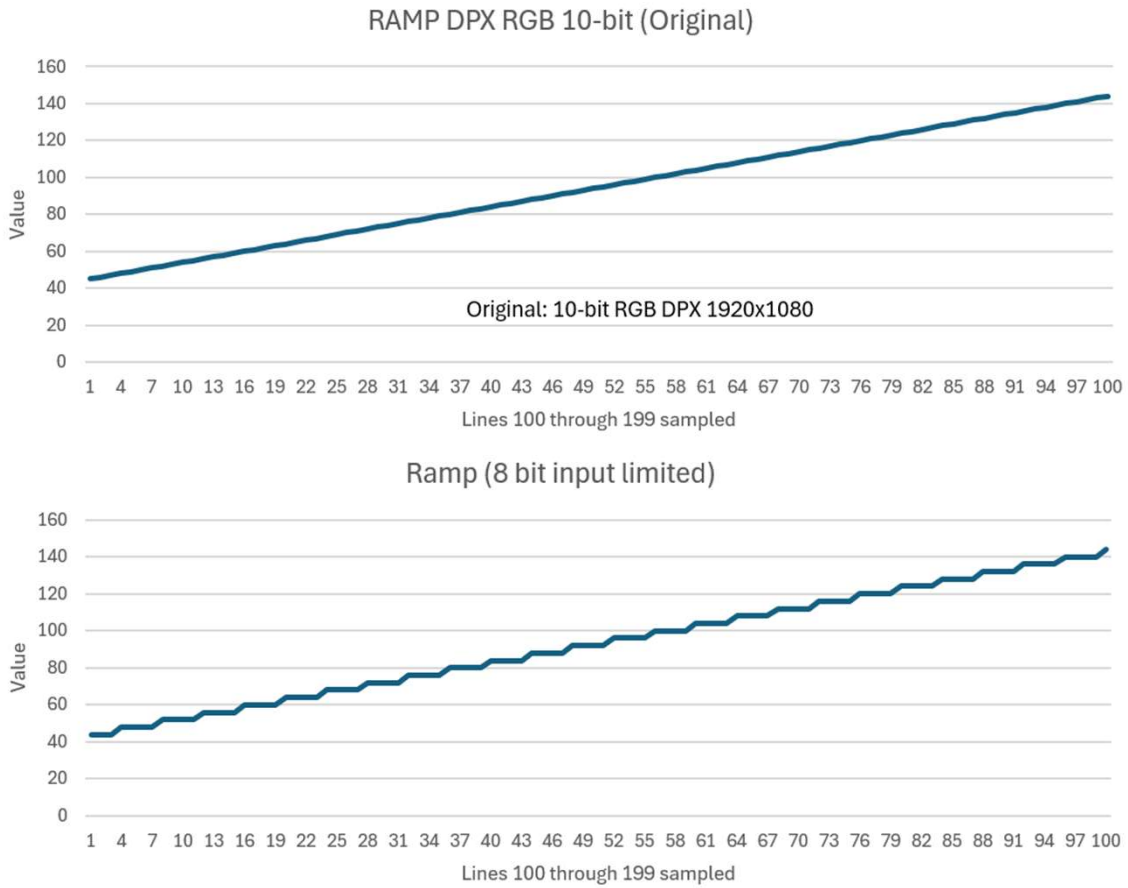




HDR editing at LB bitrates!

Compression artefacts present, but NO steps, no loss of monotonicity
Maximum error = 1

Under the hood: Bit depth independence (magnified sections: Original vs 8-bit limited LB)



If input is (erroneously) limited to 8-bits…

4.0 rate controller

Original DNx design motivation

Replace UNCOMPRESSED editorial content with COMPRESSED editorial content

- Prove using compressed signals does not mean a compromise on quality compared to uncompressed
- Focus on PSNR to keep differences small (OK for high bitrates, poor for low ones)

4.0 approach

Focus on achieving good visual quality for low bitrates (LB)

- Avoid “over-compression” in low complexity areas (highly visible)
- Hide compression noise in already existing image noise

4.0 rate controller

PSNR-based RDO approach (up to 3.0): Problem reckoning

- PSNR-based approach works well on the high-quality/high-bitrate, low compression media
- Bit budget preferentially allocated to complex macroblocks, as these contribute the most to PSNR
- Produces “visually unsatisfactory” results for highly compressed material (below SQ)

Reason:

- Rate controller tries to minimize qsf (quantization scale factor) on complex blocks, preferentially, to retain a low PSNR
- Accepts that low complexity areas will receive disproportionally large quantization as “compensation” to achieve overall bitrate

Effect: Quantization noise in complex areas is usually “masked” well by the complexity, but in low-complexity areas (“flattish”) it introduces massive blocking/banding artefacts, which are *highly visible*.

4.0 rate controller: Reference LB demonstrator

Uncompressed Original
(from 2016 SMPTE reference set)

- White noise background presents extremely complex content, but low visual relevance
- Complex content dominates the picture
- Insert presents mostly low-complexity content with some “slightly complex” areas
  - Feathers
  - Hairs
  - Fringes (transition to background)
- Human perception focuses on insert, ignores background

4.0 rate controller: DNx 3.0 LB and ProRes Proxy

DNx SDK (3.0) **LB**

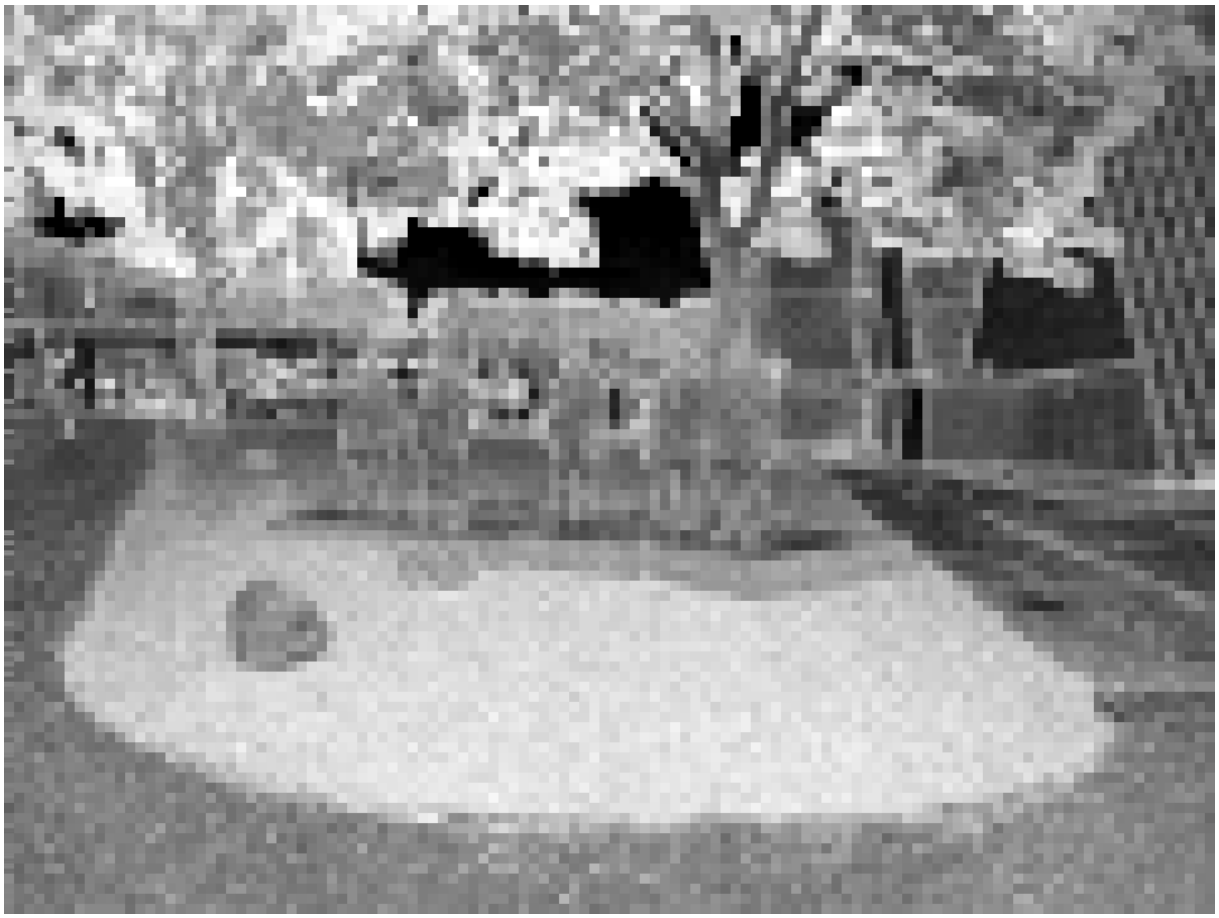
ProRes **Proxy**

4.0 rate controller (same bitrate as previous)

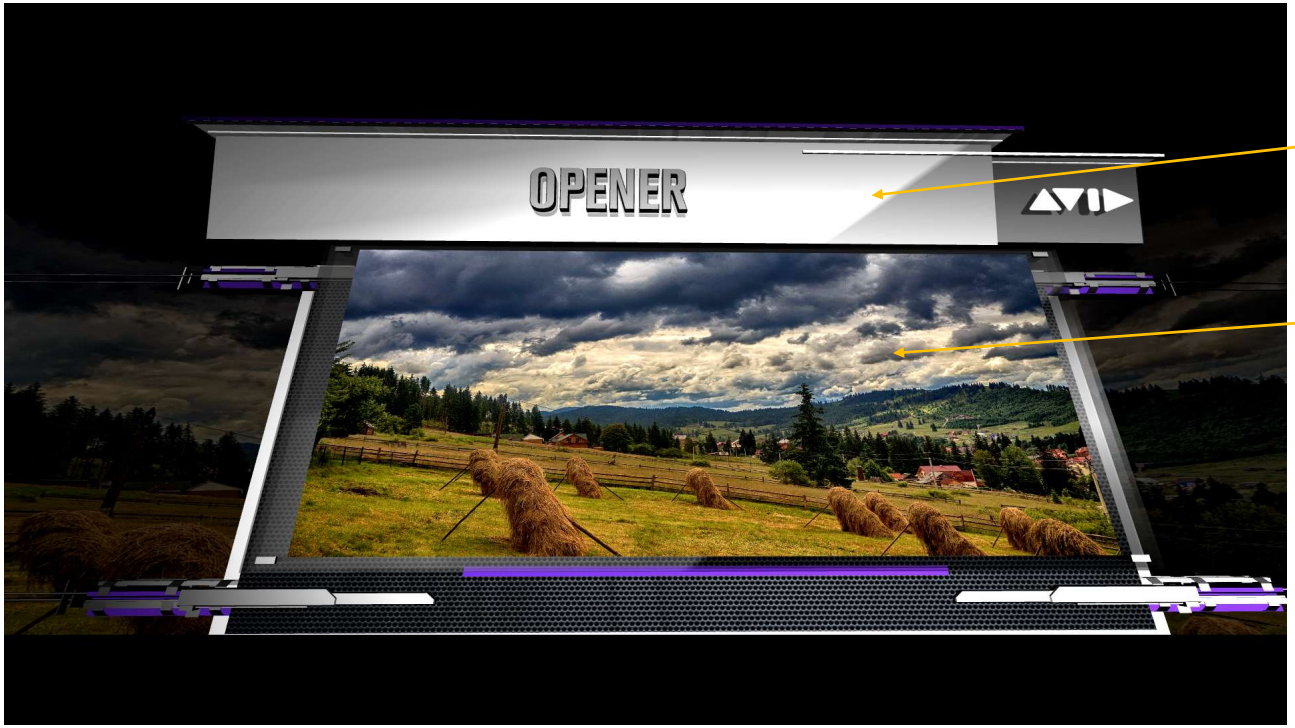
DNx **LB** (4.0)

4.0 rate controller: Noise distribution

4.0 rate controller: Noise distribution



4.0 rate controller: Quality booster for “GX”



- Graphic elements become super-clean, even at SQ level (RGB)
- Compression noise is hidden in “real-life” texture content
- Suppresses banding artefacts in “smooth” (graphic) elements very effectively

4.0 rate controller: Quality booster for “GX”

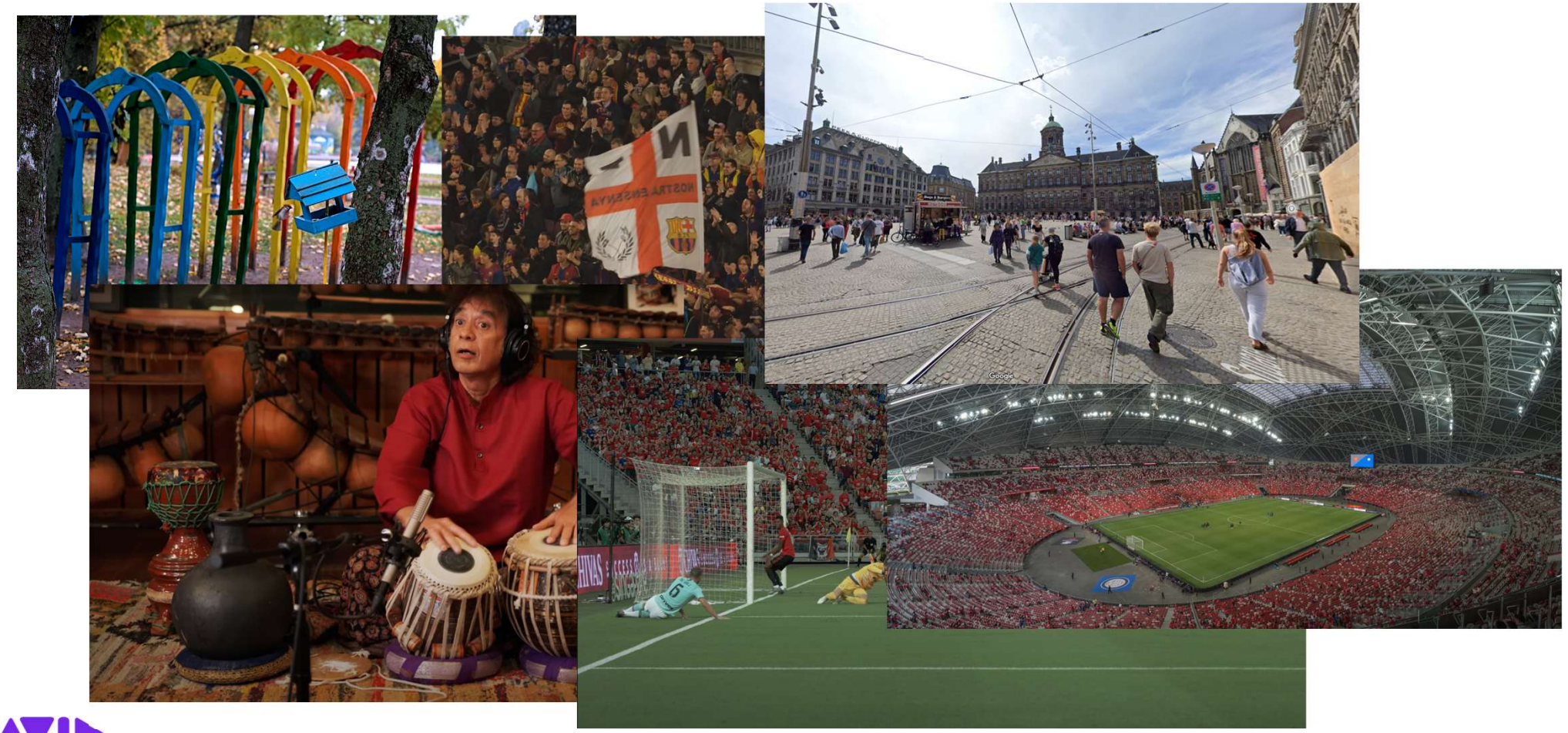
DNx GX SQ (RGB) at 145 Mbps 300% magnification

DNx 3.0 SDK

DNx 4.0 (4.0 encoder)

Loss of structural information

Non-realistic test cases?



Custom input/output formatters

- Generic option replacing large number of (seldom used) “exotic” pixel formats
- Automatically multi-threaded (no specific support required)
- Directly writes to/reads from internal codec memory
- Must conform I/O to codec’s internal memory configuration (fixed)
- Allows adapting to ANY input/output storage architecture, buffer layout, pixel layout (e.g. Tiled Memory)
- Constraint: Convenience functions included in default I/O handling (e.g. RGB output) do **NOT** ripple down to the custom I/O interfaces
- Clients seldom to never use ALL compression options (small sub-set, e.g. only 4:2:2) – low adoption threshold
- Can achieve better performance than generic conversions (code optimized for the specific format, avoids buffer copies compared to going through canonical formats)
- Samples (templates) provided in the SDK

Convenience functionality: Confidence preview

Rec 2020, HLG coded compressed picture

709 SDR monitor preview (8-bit RGB)

- 8-bit RGB output available for ALL component formats
- YC_BC_R->RGB conversion coefficients can be specified (=all primaries)
- Color space adaption optional via LUTs (sample project demonstrates the details)

Convenience functions: Up-/Downsampling

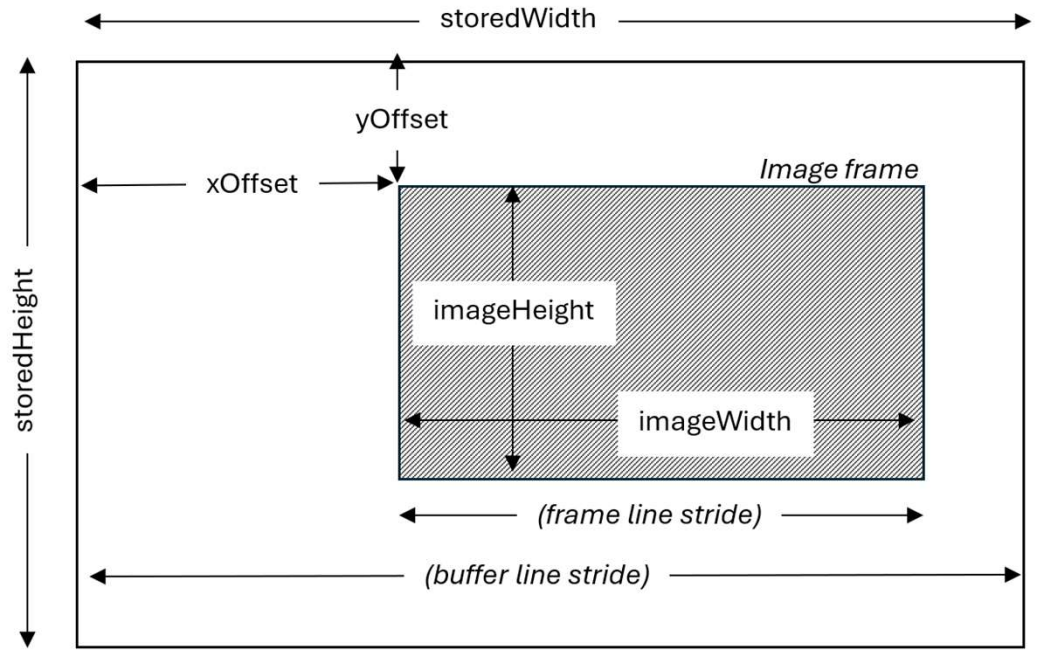
- 4:2:0 (planar) usually considered unfavorable for editing purposes
- 4:2:0 offers 12.5% bitrate advantage over 4:2:2 (= better quality at same bitrate)

- Difference matters a lot at proxy quality bitrates (LB (36 Mbps) to SQ (120 Mbps)
- (Simple) up-/downsampling 4:2:2 <-> 4:2:0 incorporated into I/O formatters
- Avoid extra buffer copies and conversion loops (practically no performance impact)
- Allows clients to use 4:2:0 encoding, but work in 4:2:2

Compressed Merge/Fast Transcode support

- Final output bitstreams can frequently be merged/adjusted from sources in compressed form, not requiring to go through an additional decode/encode generation
- SDK automatically handles the necessary adjustments, applying only the minimum of adjustment possible to ensure no generation loss
- Fast Transcode, skipping 60% of the decode/encode signal processing, is available as long as component formats match if a direct merge is not possible (transparently handled internally).
- Only changes requiring (external) component conversions (e.g. RGB -> YC_BC_R or scaling/cropping) need to go through the full decode/encode loop

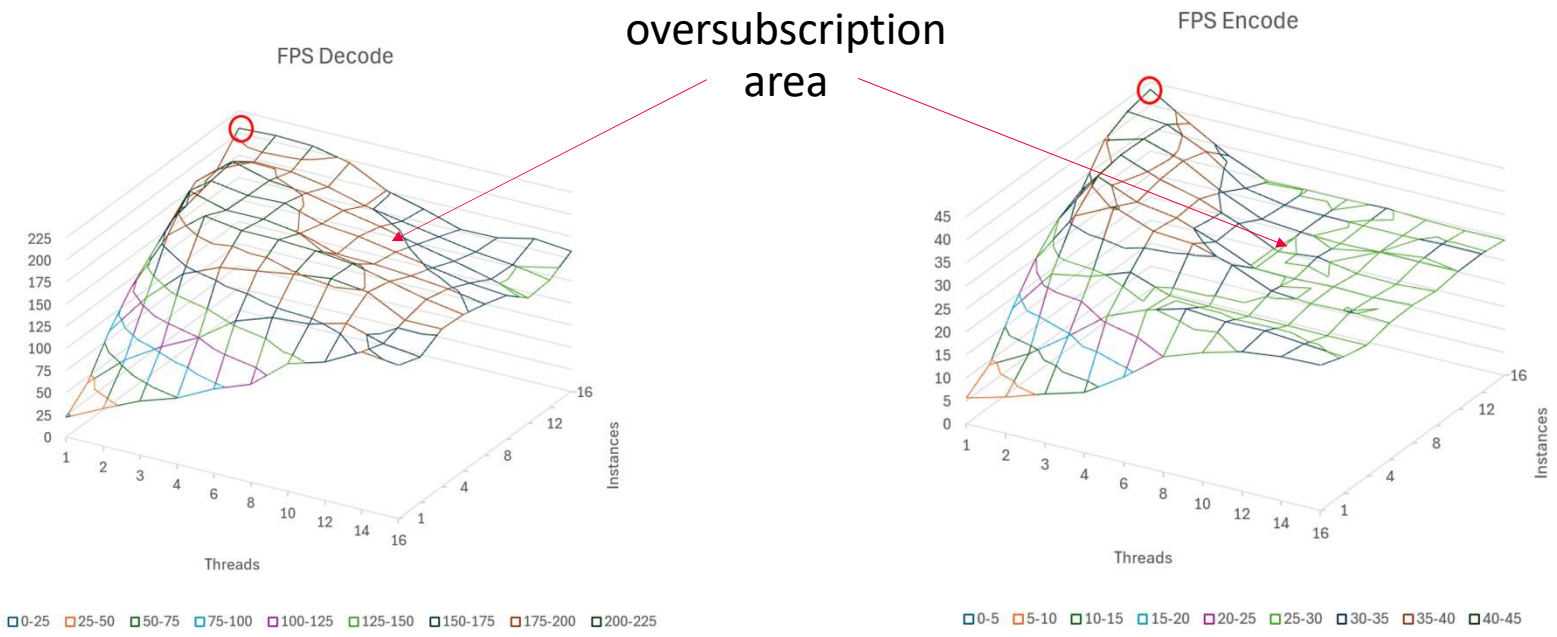
Convenience functions: Buffer viewport



Primary objectives

- Avoid buffer copies on cropping/insert workflows from/to larger image buffers
- Line stride alignment
- Any resolution for Any component format (odd/odd width/height for 4:2:2 and 4:2:0)

System Performance Tuning



* Diagrams shown measured on 16 logical processor system

- Best overall throughput performance always for single-threaded, instances=#processors configuration (both encode/decode), but also highest latency.
- Decoder ideally parallelizable internally (little impact from oversubscription)
- Encoder strongly impacted by oversubscription due to frame-wide bitrate distribution

Asynchronous processing mode

- Decouple expensive codec operations (specifically encoder) from rest of system
- Simplified “work-ahead processing” paradigm for clients without the need to build a complete asynchronous system themselves
- Simply submit frames for asynchronous processing
- No temporal ordering required (I-frames)
- Ideal for simple playback and recording operations (=streaming buffering)
- Dynamically switchable with synchronous mode

THE END