Daala: One year later

Timothy B. Terriberry

The Xiph.Org Foundation & The Mozilla Corporation

Original Plan

- *Finish Daala by the end of 2015*
  - This obviously ain't gonna happen

Original Plan

- *Finish Daala by the end of 2015* ✓
  - ~~This obviously ain't gonna happen~~

Original Plan

- *Finish Daala by the end of 2015* ✓
  - ~~This obviously ain't gonna happen~~
- Not exactly:
  - Main development moved to AV1
  - Still using Daala as a research test bed
  - May have future life as a still-image codec
  - Techniques not ready for AV1 now may mature over time and become more compelling

VDD 2015

- Last year, we had
  - 32×32 transforms and MC
  - Multiple reference frames, but no B-frames
  - Bilinear loop filter and deringing filter
  - No intra mode in our motion search

VDD 2016

- Last year, we had
  - 32×32 transforms and MC
  - Multiple reference frames, but no B-frames
  - ~~Bilinear loop filter~~ and deringing filter
  - *No intra mode in our motion search*
- This year, we have
  - 64×64 transforms and MC
  - Basic MPEG-2 style B-frames (no bi-prediction, no direct mode)
  - No bilinear loop filter
  - Hardware-friendly deringing

Other major developments

- **Better handling of frame padding**
- **Full Precision References**
- **New transform coefficient coder**
- Fixed-point PVQ implementation (almost done, for real!)
- Rate control
- Better chroma quality
- Encoder turning and better SIMD

64×64 Motion Compensation

- Have restrictions on MC sizes for neighbors
  - Neighboring block size must be within factor of 2
- Complicated RDO scheme uses unwieldy tables
- Found disabling 4×4 MC was an improvement
- Just scaled up all the tables
  - Now we have 64×64 (down to 8×8)

	RATE (%)	DSNR (dB)
PSNR	-6.64171	0.19601
PSNRHVS	-6.12492	0.28447
SSIM	-6.62569	0.15629
FASTSSIM	-4.34795	0.11693

64×64 Transforms

- We have them

	RATE (%)	DSNR (dB)
PSNR	-1.10946	0.03470
PSNRHVS	-1.52479	0.07414
SSIM	-1.22348	0.02979
FASTSSIM	-1.16836	0.03324

- But now we have 64×64 padding

Better Handling of Frame Padding

- Our transform coding doesn't understand that some regions are padding
- MC ignores prediction errors in the padding
  - We then coded all of these errors
- After MC, replace the padding in the input frame by the MC predictor

	RATE (%)	DSNR (dB)
PSNR	-1.58367	0.04947
PSNRHVS	-1.69591	0.08251
SSIM	-1.57043	0.03814
FASTSSIM	-1.43134	0.04049

Full Precision References (Currently off by default)

- Daala always operates on transform coefficients in 12-bit precision
  - 8-bit inputs are shifted up by 4 before transforms
  - Used to shift inverse transform output back to 8 bits
    - Saves memory, but adds rounding noise
- FPR: Stop converting back to 8 bits

	RATE (%)	DSNR (dB)
PSNR	-1.95527	0.06122
PSNRHVS	-1.64452	0.07952
SSIM	-2.69109	0.06513
FASTSSIM	-1.97242	0.05554

Deringing Filter Updates

Deringing Filter Updates

- Fixed several issues from hardware review
  - Made block-level threshold calculation independent of other blocks
    - Removed term involving an average over the whole 64x64 superblock
  - In the 45-degree case, changed second filter to run horizontally instead of vertically
    - Reduced the number of line buffers required in hardware
  - Removed divisions in the direction search
    - Used to divide by small, fixed constants (1...8) when averaging pixels along each direction (implemented in practice by multiplies)
  - Multiply by the LCM instead: no rounding errors, still fits in 32 bits
- Also changed filter taps from [2,2,3,2,3,2,2]/16 to [1,2,3,4,3,2,1]/16

Bilinear Loop Filter

- Not a standard deblocking filter
  - Doesn't look outside of current block!
  - Compare decoded block to bilinear interpolation of corner pixels, blend with ~~optimal Wiener~~ filter gain

$$w = \min\left(1, \frac{\alpha Q^2}{12\sigma^2}\right)^2$$

Bilinear Loop Filter

- Not a standard deblocking filter
  - Doesn’t look outside of current block!
  - Compare decoded block to bilinear interpolation of corner pixels, blend with ~~optimal Wiener~~ filter gain

$$w=\min\left(1,\frac{\alpha Q^2}{12\sigma^2}\right)^2$$

New Coefficient Coder (1)

- Basic idea: sum of absolute values, *K*, known
  - True for PVQ, must be encoded for scalar
- Split coefficient vector in half

$$K = K_{\text{left}} + K_{\text{right}}$$

- Code K_{right}: K+1 possible values (0...K)
  - If K larger than 7, code top 3 bits of K_{right} with arithmetic coder, code rest with raw bits
  - Context chosen from vector dimension, top bits of K

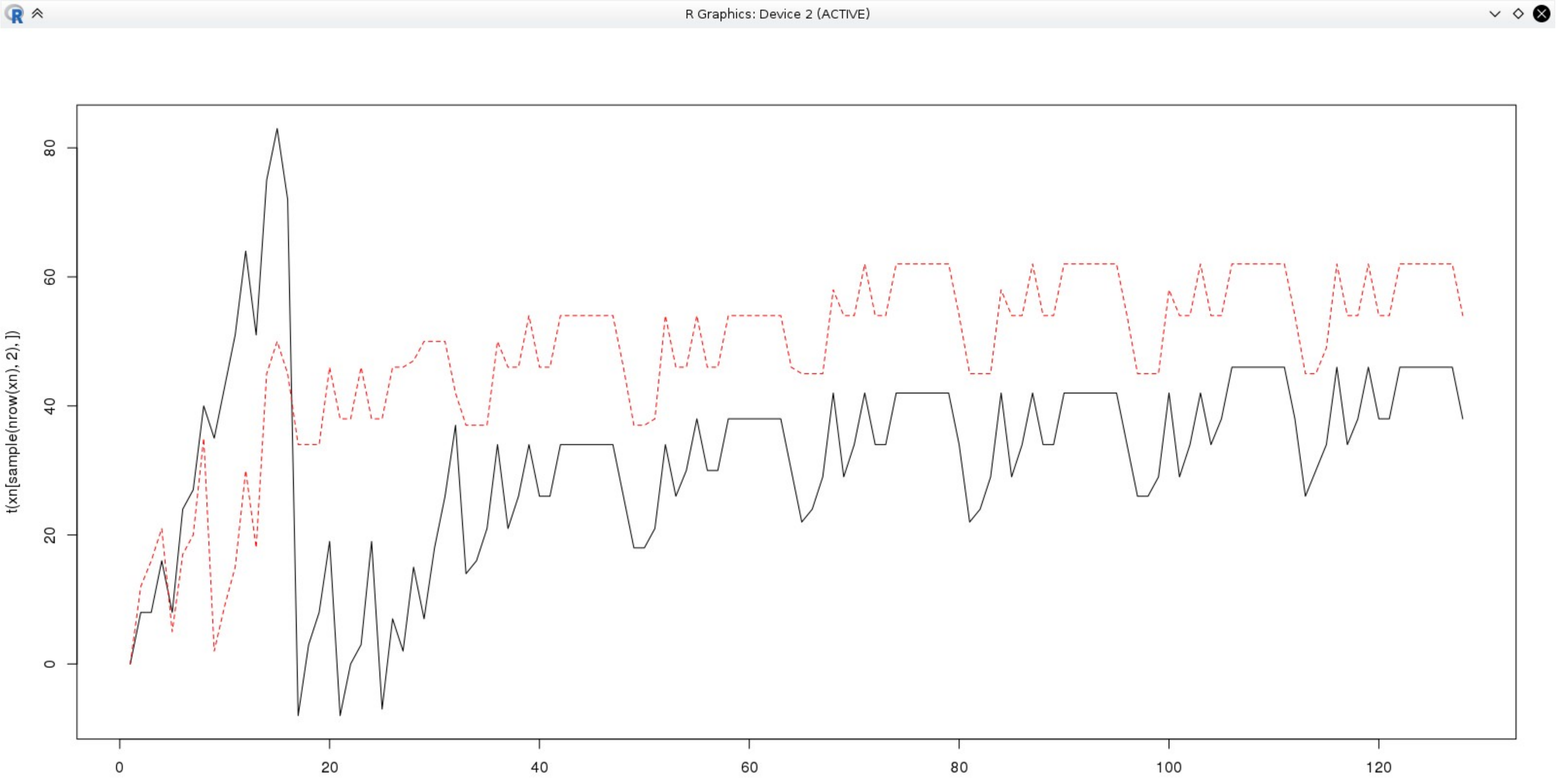
New Coefficient Coder (2)

- Special case: $K = 1$ and vector dimension ≤ 16
  - Code exact location of the 1 with one symbol
  - 12 contexts based on vector dimension
    - 4 for vectors that start out with dimension ≤ 16
    - 8 for vectors that get split down to dimension ≤ 16
- Sign bits coded with raw bits

	RATE (%)	DSNR (dB)
PSNR	-0.11934	0.00353
PSNRHVS	-0.06492	0.00298
SSIM	-0.36226	0.00815
FASTSSIM	-0.73242	0.01960

Room for Improvement

- Rate by pulse position for two random blocks



Dyadic Probability Adaptation (1)

- Probabilities that sum to a power of 2 can be coded with less overhead (~1%)
- Adaptation mostly done now with simple frequency counts
  - Total probability changes with each coded symbol
- Want: adaptation scheme with fixed total
  - Problem: need to ensure no probability goes to 0

Dyadic Probability Adaptation (2)

- Fix total, T, at 32768 (or any power of 2)
- Subtract probability floor $\{1,2,3,\ldots,M\}$ from CDF
  - M is the alphabet size (≤ 16)
- Blend with $\{0,0,\ldots,0,T\text{-}M,T\text{-}M,\ldots,T\text{-}M\}$ CDF
  - Weights 2^{-rate} and $(1 - 2^{-rate})$
- Add back in probability floor $\{1,2,3,\ldots,M\}$
- No matter how you round/truncate in the blending, total remains T, no probability is zero

Dyadic Probability Adaptation (3)

- Simplify
  - Symbol i < coded value
    - $f_i \rightarrow f_i - \lfloor (f_i + 2^{rate} - i - 2)/2^{rate} \rfloor$
  - Symbol i ≥ coded value
    - $f_i \rightarrow f_i - \lfloor (f_i + M - i - T - 1)/2^{rate} \rfloor$
- T (total probability), M (alphabet size), i (symbol index), and *rate* are constants
  - Two 15-bit vector adds and one shift with pre-computed tables
- Change rate for first few symbols in context to speed up adaptation

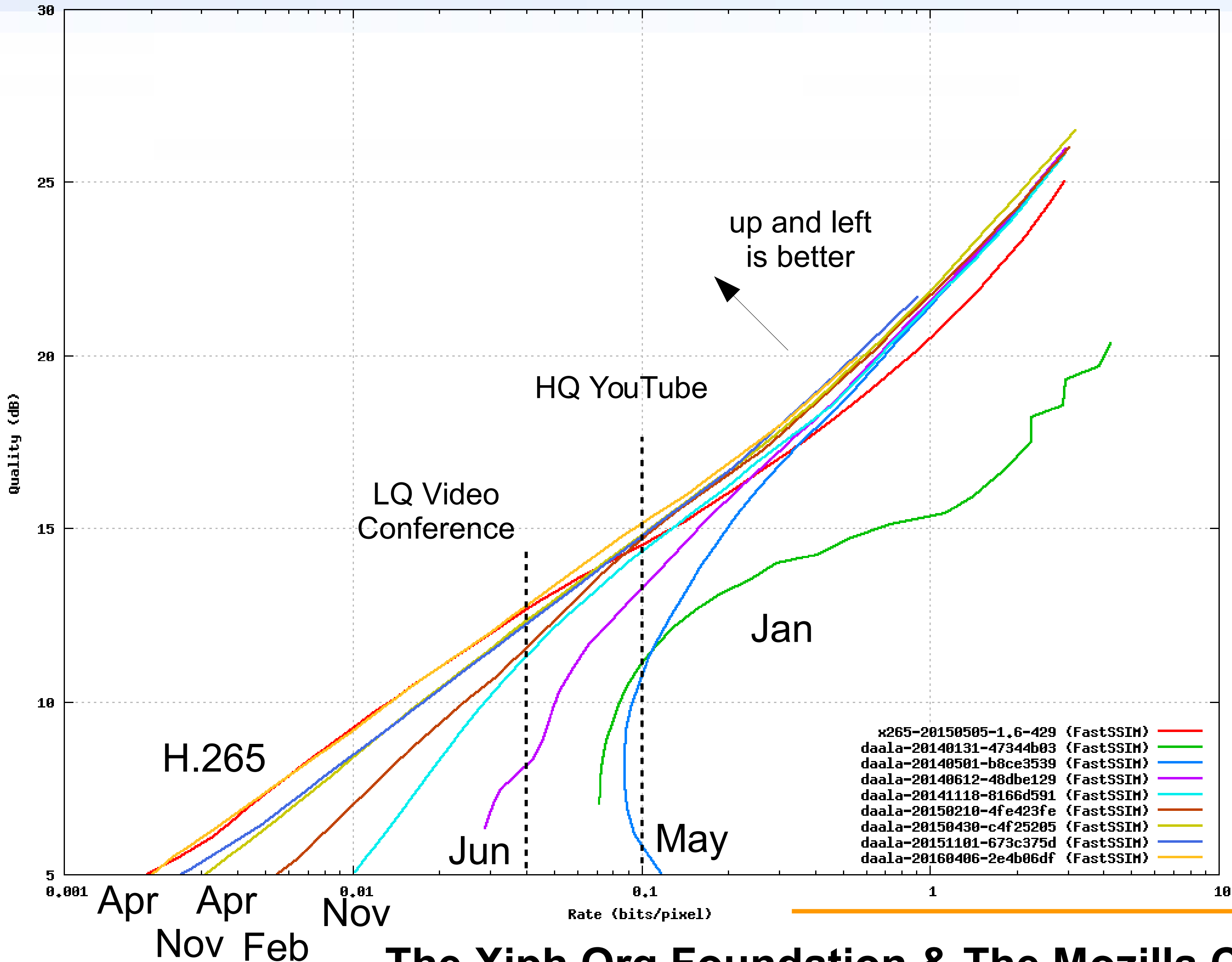
	RATE (%)	DSNR (dB)
PSNR	-0.45209	0.01360
PSNRHVS	-0.45243	0.02212
SSIM	-0.32941	0.00760
FASTSSIM	-0.47029	0.01296

Directions for AV1

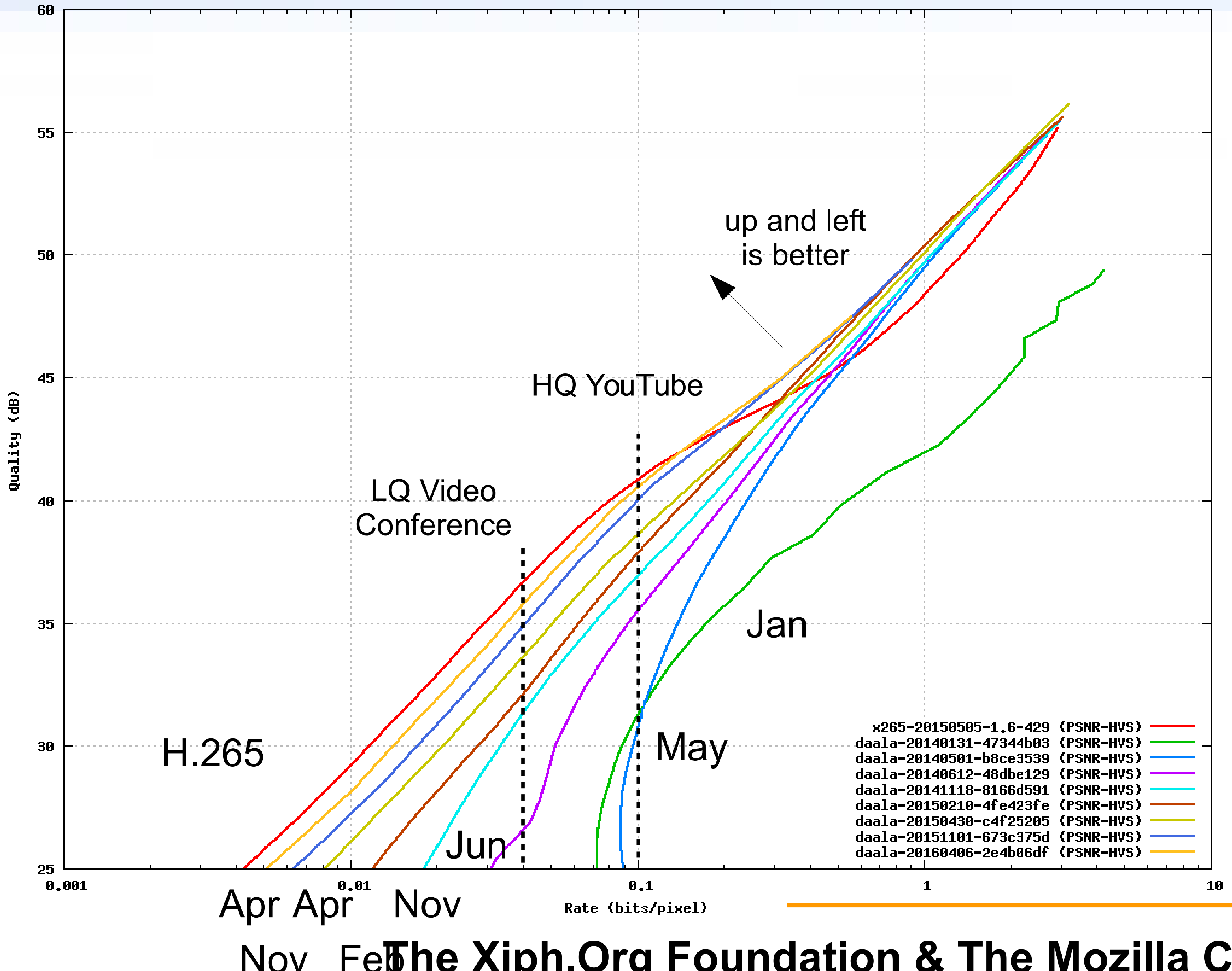
- Directional Deringing
  - Fully SIMDable, good perceptual improvements
- Non-binary Arithmetic Coding
  - Small effective parallelism in entropy coding
- Perceptual Vector Quantization
  - Already showing small gains vs. scalar on PSNR
  - Potential for large perceptual improvements
  - Enables freq. Domain Chroma-from-Luma, others
- Rate control improvements

Progress and Metrics

Daala Progress (Fast MS-SSIM): January 2014 to April 2016



Daala Progress (PSNR-HVS): January 2014 to April 2016



Questions?