Daala Update

IETF 95 (Buenos Aires)

Progress Since Yokohama

- Focusing here on changes that impact compression performance
  - Lots of code clean-up, refactoring, optimization, tools work, etc. also
- Metrics from AWCY (only recently updated to follow draft-daede-netvc-testing)
- All metrics on ntt-short-1

Major Things

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

Keyframe Boosting

- We use a finer quantizer for keyframes than other frames
  - Better to code a good predictor than a crappy one that needs lots of updates
- Originally we were very conservative about this
  - Didn’t want “popping” when coding a keyframe without a scene cut
  - Boost was 1 quantizer step (smallest allowed)

Keyframe Boosting Adjustments

- Didn’t update DC quantizers when switching from 8-point to 4-point lapping
  - Caused an implicit keyframe boost
  - We corrected it, but was a ~1% regression
- Increased keyframe boost to 2
- Combination of both changes:

	RATE (%)	DSNR (dB)
PSNR	-3.02004	0.09553
PSNRHVS	-2.99342	0.14717
SSIM	-2.53231	0.06176
FASTSSIM	-2.44178	0.06971

More Keyframe Boosting

- 2 worked great, so why not 3...

	RATE (%)	DSNR (dB)
PSNR	-3.33421	0.10535
PSNRHVS	-2.23736	0.10973
SSIM	-2.95502	0.07231
FASTSSIM	-3.01988	0.08658

These gains are *in addition* to the prior slide...

More Keyframe Boosting?

- Boost of 3 caused one complaint on Github (thanks!)
- Developers unanimously thought 3 looked better
- Boosting to 4, 5, and even 6 gives more metrics improvements, but mixed developer opinions
- Really want this to be content-adaptive
  - Large boost not helpful with high motion
  - But that complicates testing, especially between codecs

B-Frames

- We added B-frames
  - MPEG-2 style references
    - Don’t use B-frame as references
    - Reference buffer management entirely implicit
      - One past and one future I/P frame
  - No blending mode
    - Each OBMC MV points to one reference
    - Normal OBMC blending still applies
- These are off by default, because they add latency

B-Frames

- Results using same quantizer as P frames was a small regression
- Using a coarser quantizer on B frames was a big improvement
- Results for 2 B-frames between each I/P frame:

	RATE (%)	DSNR (dB)
PSNR	-6.85495	0.21126
PSNRHVS	-2.39899	0.10919
SSIM	-6.68393	0.15477
FASTSSIM	-6.88149	0.19032

Better QP Modulation (Encoder Only)

- Adopted Thor's approach of using larger QP changes at lower bitrates
  - I frames (and "golden" frames): QP = BaseQP – 2
  - P frames: QP = BaseQP*1.05
  - B frames: QP = BaseQP*1.1+1
- Improvements over boost of 3:

	RATE (%)	DSNR (dB)
PSNR	-1.89545	0.05734
PSNRHVS	-1.70937	0.08181
SSIM	-1.93016	0.04450
FASTSSIM	-2.21168	0.06100

Improve Chroma Quantization

- Used to quantize chroma at a fixed multiple of the luma quantizer
- Now quantize chroma coarser than luma at high rates, and finer than luma at low rates
- Big penalty on luma-only metrics, huge gain on color-aware metric CIEDE2000

```
                    RATE (%) DSNR (dB)
          PSNR 3.65608  -0.11200
       PSNRHVS 3.61704  -0.17133
          SSIM 3.58143  -0.08467
     FASTSSIM 3.41183  -0.09387
CIEDE2000 (subset1):
             -10.7459   0.483104
```

64x64 Transforms

- Implemented a 64x64 DCT
  - Perfectly reversible, multiplier outputs 32 bits
  - As mentioned in Yokohama, don't code high bands
- Requires 64x64 Superblocks
  - Small (0.4%) regression
- Overall results of 64x64 SBs plus 64x64 DCT

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

32x32 and 64x64 Activity Masking Tuning

- With the addition of 64x64 transforms, we needed to tune the activity masking parameters
  - We'd never properly tuned 32x32, either
  - Metrics are useless for this, but somehow they all moved in the right direction (even PSNR!)

	RATE (%)	DSNR (dB)
PSNR	-0.07591	0.00233
PSNRHVS	-0.30077	0.01439
SSIM	-0.49197	0.01173
FASTSSIM	-1.05095	0.02923

Removed Bilinear Filter

- Filter intended to remove blocking artifacts in smooth regions after transition to fixed lapping
- Only ever ran on keyframes
- Combination of deringing filter and 64x64 transforms eliminated most of the benefit

	RATE (%)	DSNR (dB)
PSNR	-0.13489	0.00406
PSNRHVS	-0.07243	0.00340
SSIM	-0.19117	0.00439
FASTSSIM	-1.39138	0.03806

Deringing Filter Changes (1)

- Signal a filter strength (threshold)
  - Signaled once per 64x64 Superblock
  - One of 6 levels available (0 == off)
- Harms FastSSIM, but that's good
  - That means the deringing is working

	RATE (%)	DSNR (dB)
PSNR	-1.32890	0.04018
PSNRHVS	-0.25398	0.01196
SSIM	-0.68830	0.01581
FASTSSIM	1.93442	-0.05150

Deringing Filter Changes (2)

- Converted floating point calculations to fixed point
- Changed filter taps to [1,2,3,4,3,2,1]/16 from [2,2,3,2,3,2,2]/16
- Fixed several issues identified by NVIDIA during hardware review
  - Made block-level threshold calculation independent of other blocks
    - Used to have a term involving an average over the whole superblock
  - In the 45-degree case, changed second filter to run horizontally instead of vertically
    - Reduced the number of line buffers required in hardware by two
  - 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
- Quality impact of all of these changes was minimal

Fixed-Point PVQ Implementation (In Progress)

- Large set of incremental changes
- Can switch between fixed and float implementations at compile time
  - To test for regressions
- Currently < 0.1% change in metrics
- Expect to be complete before Berlin

New Coefficient Coder

- Based on splitting PVQ vector in half, coding sum of absolute values on one side
  - Plus special cases when the sum is 1
  - Computationally much simpler than prior approach
  - Requires more context memory
  - Not yet sure what the right trade-off for hardware is

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

Minor Things

Avoid Round-Tripping Skipped Bands Through PVQ

- Recall in Yokohama that we stopped coding very large (256+ coefficient) bands
- This was just a simple change to stop running them through our vector quantizer

	RATE (%)	DSNR (dB)
PSNR	0.09112	-0.00283
PSNRHVS	-0.27288	0.01308
SSIM	0.36275	-0.00866
FASTSSIM	-0.24695	0.00688

Reorder Skip Flags

- Previously coded a 2 to skip both AC and DC
- Now code a 0
  - Entropy coder overhead is minimized when 0 is the most probable symbol
  - If a packet is truncated or the decoder desyncs, reads past the end of the packet will be skips

	RATE (%)	DSNR (dB)
PSNR	-0.03658	0.00113
PSNRHVS	-0.13585	0.00650
SSIM	-0.10983	0.00261
FASTSSIM	-0.25036	0.00694

Flat Initialization of Probabilities for MV Valid Flags

- Previous MV valid flag probabilities were last trained when we only had 16x16 MV blocks
  - Already used flat probabilities when adding 32x32
  - Didn't change probabilities at all when moving from 4x4...32x32 to 8x8...64x64
- Just flat initialization was now better

	RATE (%)	DSNR (dB)
PSNR	-0.07483	0.00230
PSNRHVS	-0.07504	0.00358
SSIM	-0.06616	0.00157
FASTSSIM	-0.05619	0.00155

Don’t Code MV Reference Index When We Only Have One

- When both available references were the same (e.g., right after a keyframe), we still coded a reference index for every MV
- Instead, we now don’t do that
- Makes very little difference
  - Thanks, adaptive entropy coding

	RATE (%)	DSNR (dB)
PSNR	-0.01599	0.00049
PSNRHVS	-0.01745	0.00083
SSIM	0.00444	-0.00011
FASTSSIM	-0.03153	0.00087

Simplified Entropy Coder When Probabilities Sum to a Power of 2

- We support probabilities with arbitrary sum
  - No multiplies or divides, some approximation error
- If we can use a multiply, can do powers of 2 with lower overhead
- Currently implemented, but only used for some low-probability escape values (and headers)

	RATE (%)	DSNR (dB)
PSNR	-0.04686	0.00140
PSNRHVS	-0.04323	0.00204
SSIM	-0.06528	0.00148
FASTSSIM	-0.03908	0.00105

Move Where Quantization Matrices Are Applied

- Previously we scaled coefficients by the quantizer matrix before PVQ
- Moved to after normalization to a unit vector
  - We have higher precision in the normalized domain
  - Normalization still takes QM into account
  - Small (0.2%) rate reduction for subset1

	RATE (%)	DSNR (dB)
PSNR	0.00592	-0.00020
PSNRHVS	0.03283	-0.00160
SSIM	-0.06473	0.00156
FASTSSIM	0.06960	-0.00195

Don’t Apply Lapping Across Edge of Visible Region

- Video is padded to multiple of 64x64
- Visible region is smaller
- No longer apply lapping across the edge of the visible region
  - This breaks lossless cropping
  - Reduces visible edge artifacts

	RATE (%)	DSNR (dB)
PSNR	0.01072	-0.00033
PSNRHVS	0.00072	-0.00003
SSIM	-0.01913	0.00046
FASTSSIM	0.04135	-0.00116

Encoder-Only Improvements

Enable SATD in Motion Search (Encoder Only)

- Had tried this before, but it didn't seem to help
  - Tested both using 8x8 Walsh-Hadamard Transforms and a WHT that matches the MC partition size
  - All 8x8 was better
    - Recall from Yokohama that we dropped 4x4 MC support
- Now it helps:

	RATE (%)	DSNR (dB)
PSNR	-0.70911	0.02205
PSNRHVS	-0.75006	0.03614
SSIM	-0.61743	0.01479
FASTSSIM	-0.45986	0.01289

Don’t Code Updates Outside Viewable Area (Encoder Only)

- Our PVQ implementation doesn’t understand that some regions are padding
- MC ignores prediction errors in the padding
  - PVQ was then coding 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

Fixed Overflow in Skip Calculations (Encoder Only)

- Sometimes cheaper to code a block than skip
- We stored the bitrate difference in an unsigned variable
- Small metrics change, but fixes some visual glitches

	RATE (%)	DSNR (dB)
PSNR	-0.00898823	0.000430187
PSNRHVS	-0.0266512	0.00174076
SSIM	-0.0236835	0.000961467
FASTSSIM	-0.152803	0.00434009

Better AC/DC RDO (Encoder Only)

- Estimate the rate of coding a skip flag when skipping all AC coefficients in a block
  - Previously we ignored this cost because we were afraid of greedy decisions
  - But counting it seems to help

	RATE (%)	DSNR (dB)
PSNR	-0.55807	0.01734
PSNRHVS	-0.57831	0.02785
SSIM	-0.52872	0.01267
FASTSSIM	-1.10557	0.03110

Better Correction for QMs in Distortion Term (Encoder Only)

- Applying a quantization matrix reduces measured distortion
- We used to correct with a fixed scale factor
- Now apply one that varies by target quantizer

	RATE (%)	DSNR (dB)
PSNR	-0.74377	0.02299
PSNRHVS	-0.54053	0.02597
SSIM	-0.94424	0.02256
FASTSSIM	-0.90805	0.02531

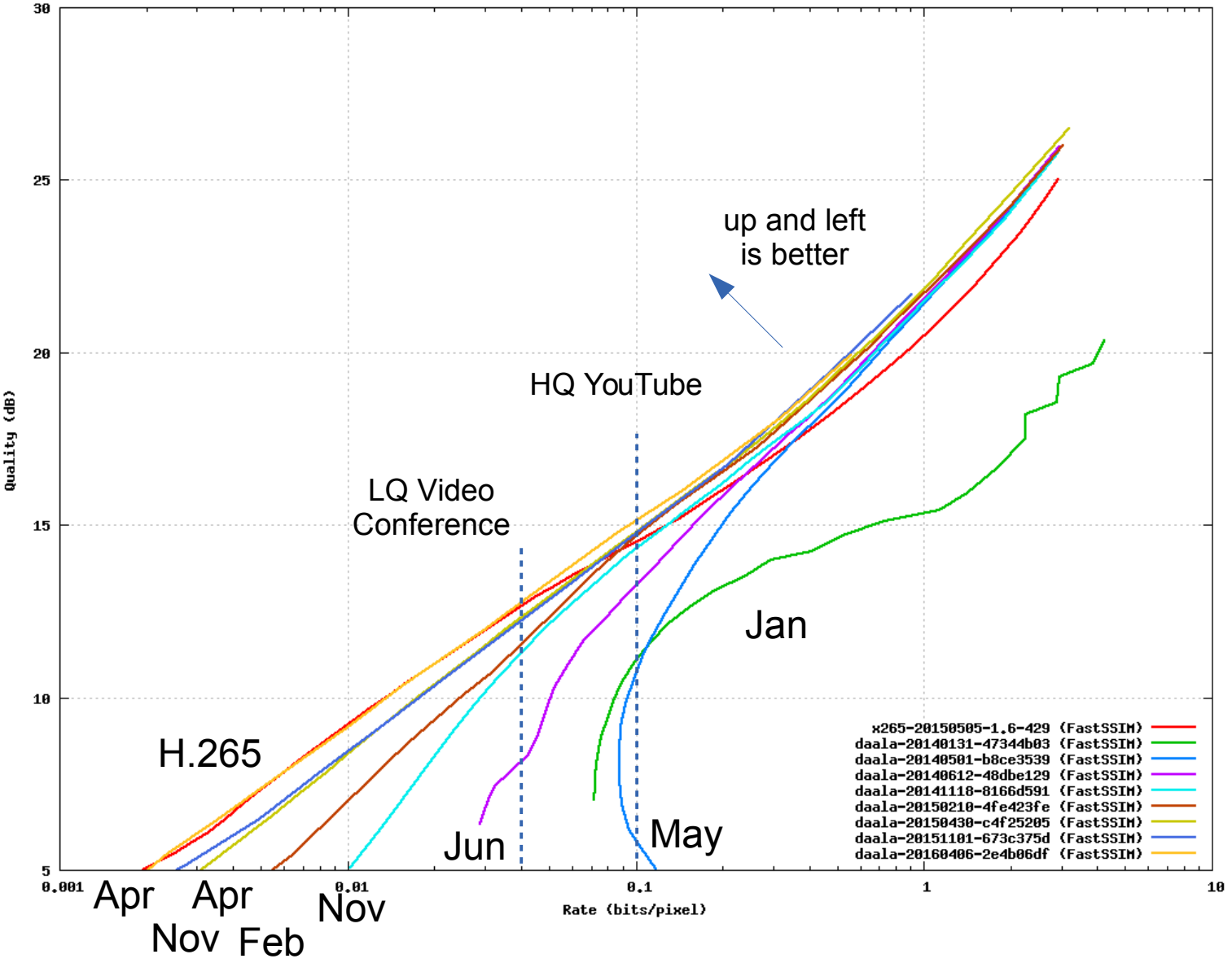
Summary

Summary

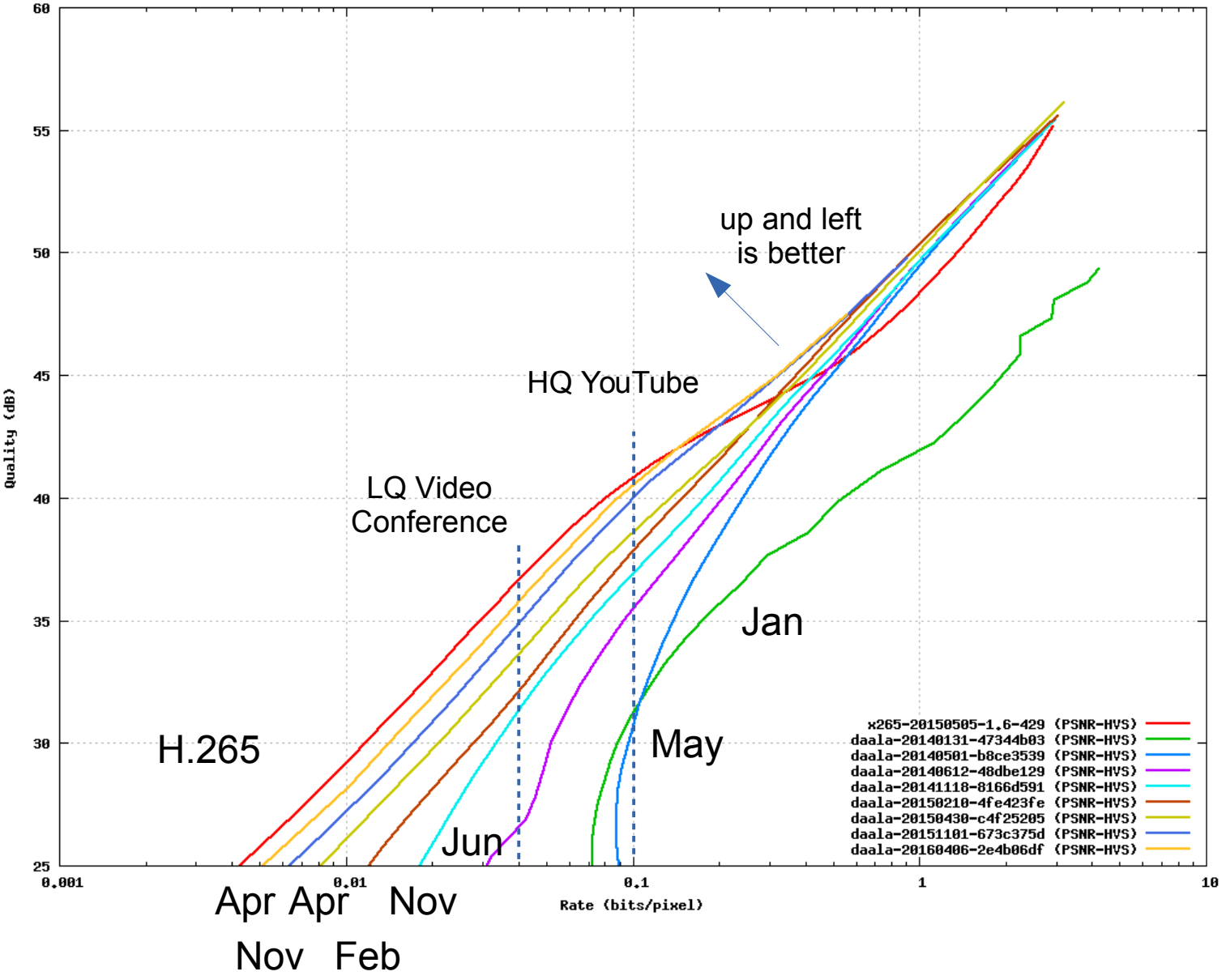
- 268 commits
- 4 new contributors
- David Michael Barr, Rostislav Pehlivanov, Luc Trudeau, Albert Villeneuve-Nguyen
- Aggregate results (with -b 2 --fpr)

	RATE (%)	DSNR (dB)
PSNR	-16.88311	0.56128
PSNRHVS	-13.14109	0.67286
SSIM	-17.37209	0.43484
FASTSSIM	-16.01262	0.47097

Daala Progress: FastSSIM January 2014 to April 2016



Daala Progress: PSNR-HVS January 2014 to April 2016



Questions?