Daala Update

IETF 96 (Berlin)

Progress Since Buenos Aires

- Main development switched to AV1
  - https://aomedia.googlesource.com/aom
- Daala project now primarily used as a testbed
  - Preparing integration of technologies into AV1
  - May continue as own codec some day when more mature
    - Submitted to 2016 ICIP still image challenge
- Test results reported on ntt-short-1
  - draft-ietf-netvc-testing changes now AV1-specific
  - Probably won't update Daala methodology to match

Summary

- 63 commits
- 4 new contributors
- Nguyen van Duc, Philippe Le, Rignon Noel, Arron Vuong
- Aggregate results (ntt-short-1, default options)

	RATE (%)	DSNR (dB)
PSNR	-1.15141	0.03467
PSNRHVS	-0.27349	0.01335
SSIM	-0.80811	0.01862
FASTSSIM	-1.15548	0.03196

Changes

Deringing Filter Changes

- 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

Deringing Filter Changes

- Reported at Buenos Aires that these were a small regression
- Retested shortly after and found this was no longer true

	RATE (%)	DSNR (dB)
PSNR	-0.31474	0.00944
PSNRHVS	-0.08200	0.00386
SSIM	-0.19829	0.00449
FASTSSIM	-0.68698	0.01850

Q15 Entropy Coder Adaptation (1)

- At Buenos Aires Hackathon, added a simplified entropy coder for power-of-2 probabilities
  - Eliminates most approximation overhead (~1%)
- Added new probability adaptation that keeps the sum of the probabilities constant
- Probability updates are more expensive
  - But benefit from lower overhead

Q15 Entropy Coder Adaptation (2)

- Fix total, T, at 32768
- Updates of the cumulative distribution, f_i, maintain this total
  - Coded value < symbol i
    - $f_i \rightarrow f_i - \lfloor (f_i + 2^{rate} - i - 2)/2^{rate} \rfloor$
  - Coded value ≥ symbol i
    - $f_i \rightarrow f_i - \lfloor (f_i + M - i - 32769)/2^{rate} \rfloor$
    - M = alphabet size
- M (alphabet size), i (symbol index), and *rate* are constants
  - Two 15-bit vector adds and one shift with pre-computed tables
- Additional rules for first few symbols in a given context to speed up adaptation

Q15 Entropy Coder Adaptation (3)

- Modified coefficient coder, split and skip coding, and generic coder to use new adaptation
- Currently only in a branch:
  - https://github.com/jmvalin/daala/tree/exp_dyadic_adapt9
- Not sure of the effect on hardware throughput

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

Fixed-Point PVQ

- Status since Buenos Aires
  - Completed replacements for reciprocal square roots, exp, log, pow, etc.
  - 11% of commits since IETF 95
  - Nearing completion: decoder float usage mostly gone
  - Impact on metrics remains small

	RATE (%)	DSNR (dB)
PSNR	0.01499	-0.00049
PSNRHVS	0.04322	-0.00212
SSIM	0.04482	-0.00118
FASTSSIM	0.18308	-0.00526

Rate Control

- Previously only supported constant quantizer
  - With fixed adjustments based on frame type
- Added 1-pass (no lookahead) rate control
  - Adapted from implementation in Theora
    - Extended to handle B frames, long-term reference frames
  - Targets “average bitrate” over some buffer interval
    - Typical intervals 12...250 frames
  - Not meant for hard-CBR (interactive)
    - Complement, not replacement, for rate control in Thor

Rate Control Model

- Basic model

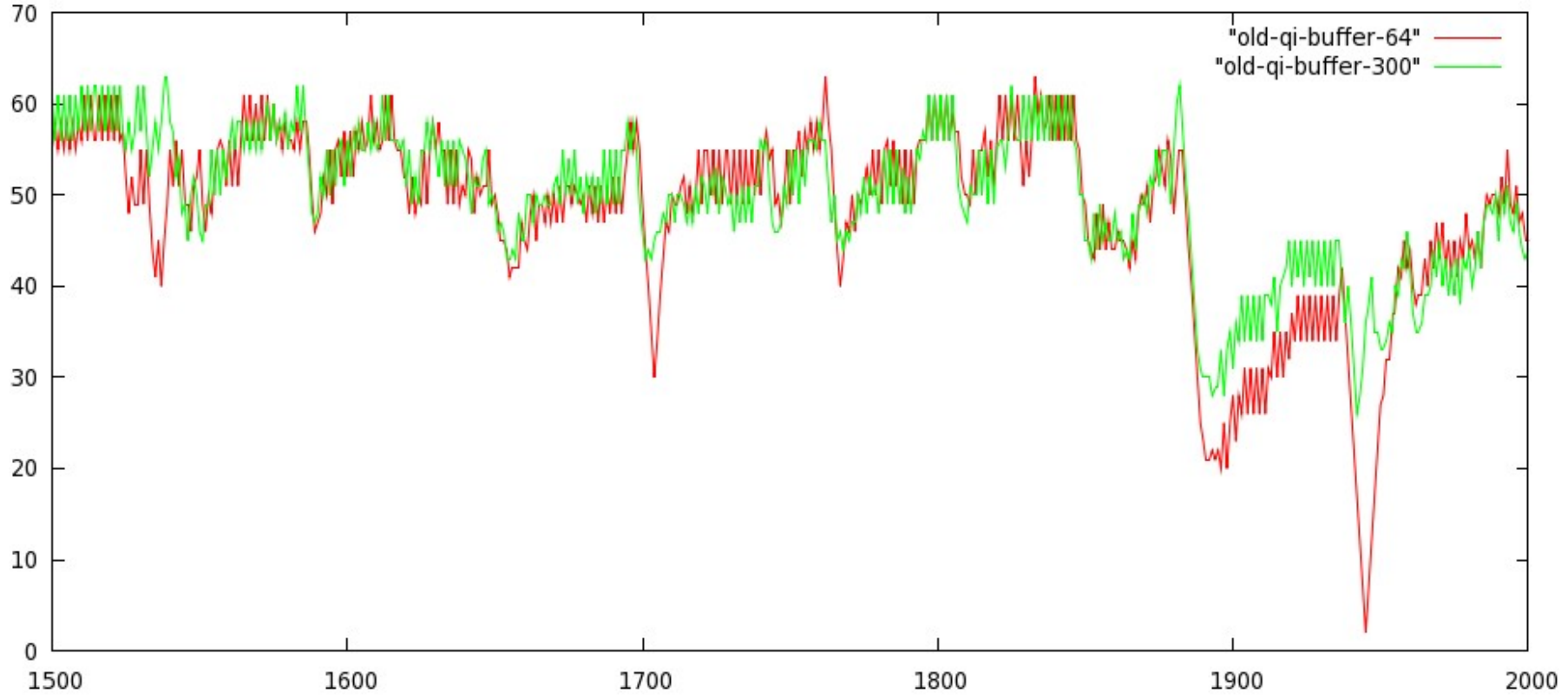
$$R = scale \cdot Q^{-\alpha}$$

- R = rate of a frame, in bits
- Q = actual quantizer (not QP)
- α = modeling exponent
  - Fixed for the whole sequence
  - 0.75...1.6, chosen based on frame type, bitrate range (bits per pixel)
- *scale* = estimate of scene/motion complexity
  - Measured during encoding for each frame type

Estimating *scale*

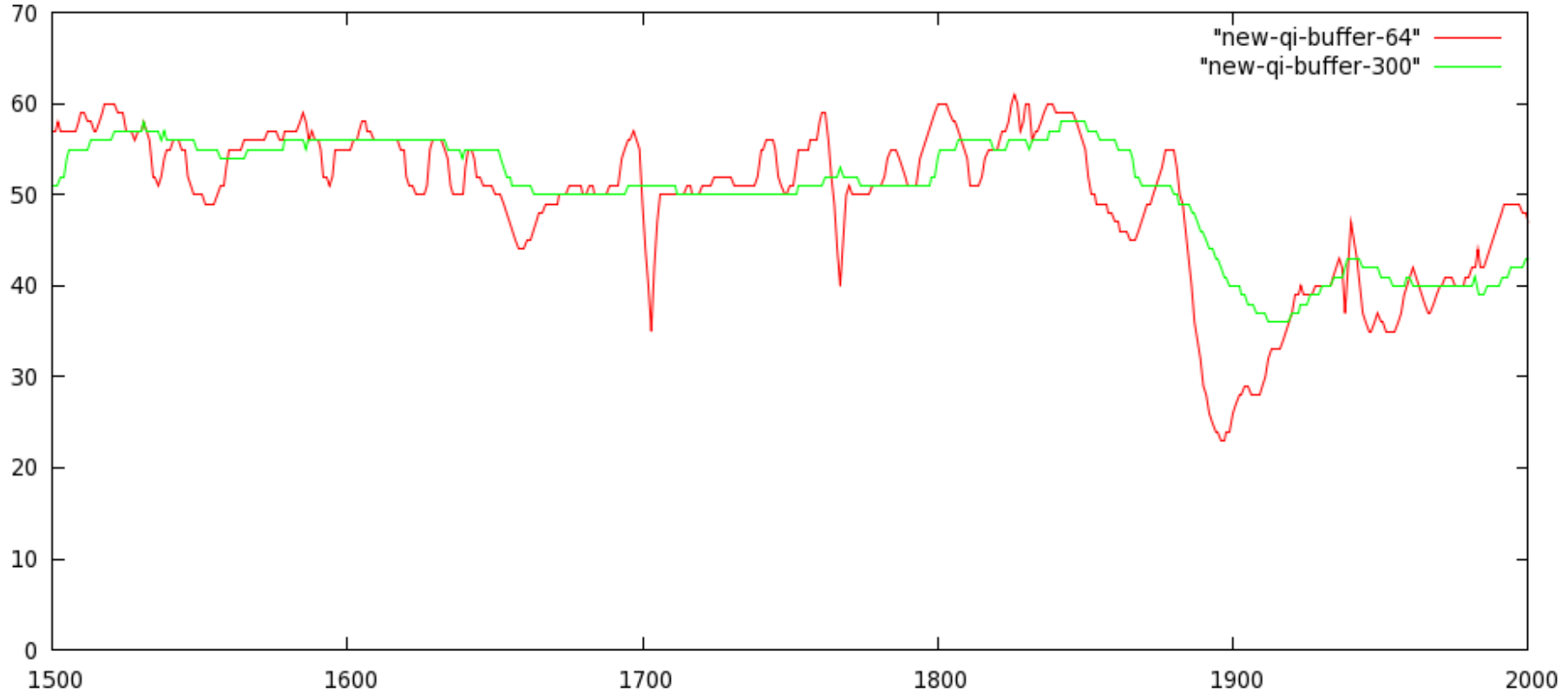
- Measure after encoding each frame
  - We know *R* and *Q,* and *α* is fixed, solve for *scale*
- Measured values fed into second order Bessel filter
  - Damps small oscillations
  - Reacts quickly to large changes
    - Time constant chosen to allow full-scale reaction in half the buffer interval
    - Faster adaptation at beginning of sequence (to handle, e.g., fade from black)
- One filter per frame type
  - Keyframes
  - Long-term reference (golden) frames
  - Regular P frames
  - B frames

Exponential Moving Average



- Highly oscillatory (inconsistent quality)
- Reaction time independent of buffer interval

2nd Order Bessel Filter



- Oscillations damped (more consistent quality)
- Longer buffer gives smoother reactions

Choosing *Q* (1)

- Every frame, plan out the whole buffer interval
  - After encoding this frame, throw the plan away and make a brand new one starting with the next frame
- Frame types chosen in fixed pattern
  - Regular keyframe, golden frame intervals
  - Regular B-frame spacing
- Aim for a fixed buffer fullness level at last keyframe in interval (or last frame)

Choosing Q (2)

- Pick constant quantizer that achieves target buffer fullness level
  - Taking into account fixed adjustments based on frame type, *i*

$$R = \sum_i N_i \cdot scale_i \cdot Q_i^{-\alpha_i}$$

- N_i is the number of frames of each type
- Each Q_i is a function of master Q (just like constant-quantizer)
- Adjust master Q until R hits target
  - Binary search (robust)

Two-Pass

- Measure *scale* for each frame in first pass
- Use measured *scale* instead of Bessel filter
  - $N_i \cdot scale_i$ is just the sum of the measured values in the buffer interval
  - Currently assuming frame types don't change between passes
- Add a fixed offset to correct for consistent over/under estimates
- Everything else is just like one-pass
  - Generalizes to one-pass-with-lookahead also
- Supports buffer intervals up to the whole sequence
  - "Unconstrained" VBR

Chunked Two-Pass (1)

- How video sharing sites work
  - Split video into many small (1...5 second) chunk
  - Encode each one in parallel to reduce latency
- Current libvpx rate control
  - Don’t want each chunk to be the same size
    - Complex scenes need enough rate to look good
    - Simple scenes only need so much quality
  - “Relax” buffer constraints
    - Intentionally over/under-shoot
  - Hope it works out on average over the whole sequence

Chunked Two-Pass (2)

- Better approach
  - Run first-pass for each chunk
  - Collect *scale* measurements from each chunk
    - Really only need average *scale* and count for each frame type over the whole sequence
  - Buffer plan can now take into account the rest of the sequence

Current Status and Future Plans

- 1-pass landed in Daala, 2-pass coming soon
- Next steps
  - Port to AV1
    - Handle adaptive frame type decisions
    - Handle additional frame types (alt-refs)
  - Compare with old libvpx rate control

Questions?