Potential AV2 Research Directions

Caveats

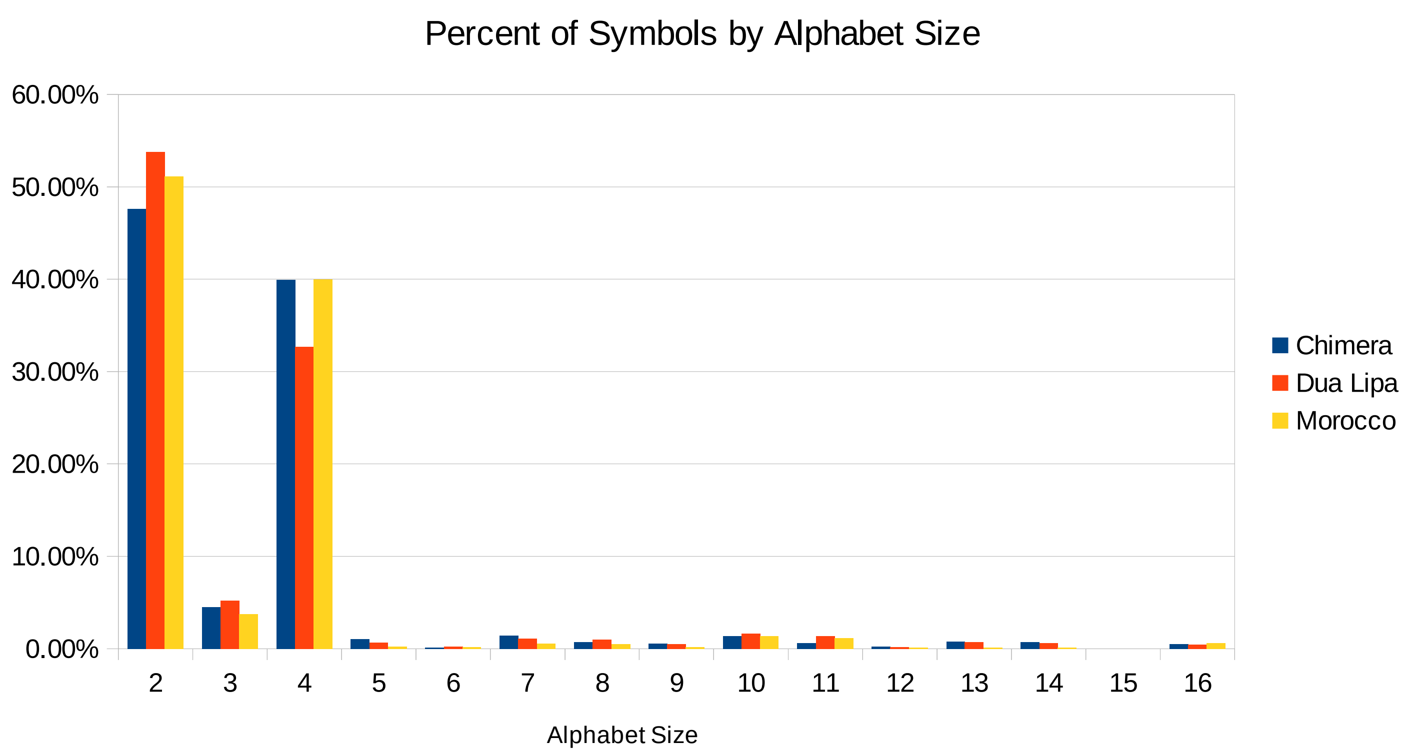
- These are just ideas
- We have not started working on any of them
- We don't know if any of them will work
  - If we knew what we were doing, it wouldn't be research
- They're not a complete list
  - Just some of the more thought-out ideas

Entropy Coding

- AV1 moved to "multi-symbol arithmetic coding"
  - Otherwise known as regular arithmetic coding, since 1979
  - Goal was to increase hardware throughput
    - More values per symbol → fewer symbols → lower clock rates (at the cost of a reasonable area increase)
  - Also helps software
    - Cost of coding a symbol dominated by overhead, not arithmetic
    - SIMD can also parallelize the arithmetic
- How can we move this forward in AV2?
  - In ways that do not add a lot of complexity!

Alphabet Size

- AV1 supports alphabet sizes up to 16
- But the majority of symbols use no more than 4



Larger Alphabets

- **Look for ways to increase alphabet sizes**
  - "Clean-up work" that we did not get to in AV1
  - Not always trivial to merge symbols, as adjacent symbols may have different contexts
    - Exterior product of combinations explodes table sizes
    - Need to evaluate if all contexts pay for themselves
- Daala was able to have zero (0) binary contexts
  - Same can be true for AV2!

Learning Rate

- AV1's CDF update learning rate is based on
  - The number of symbols coded in a given context in the current tile
    - Three bands: 0...15, 16...31, 32+
  - The number of symbols in the CDF (large alphabets adapt more slowly)
- Simple approximation of dynamic learning rates of frequency counts
- Assumes all contexts are equally stationary
  - This is definitely not true

Improvements to Learning Rate

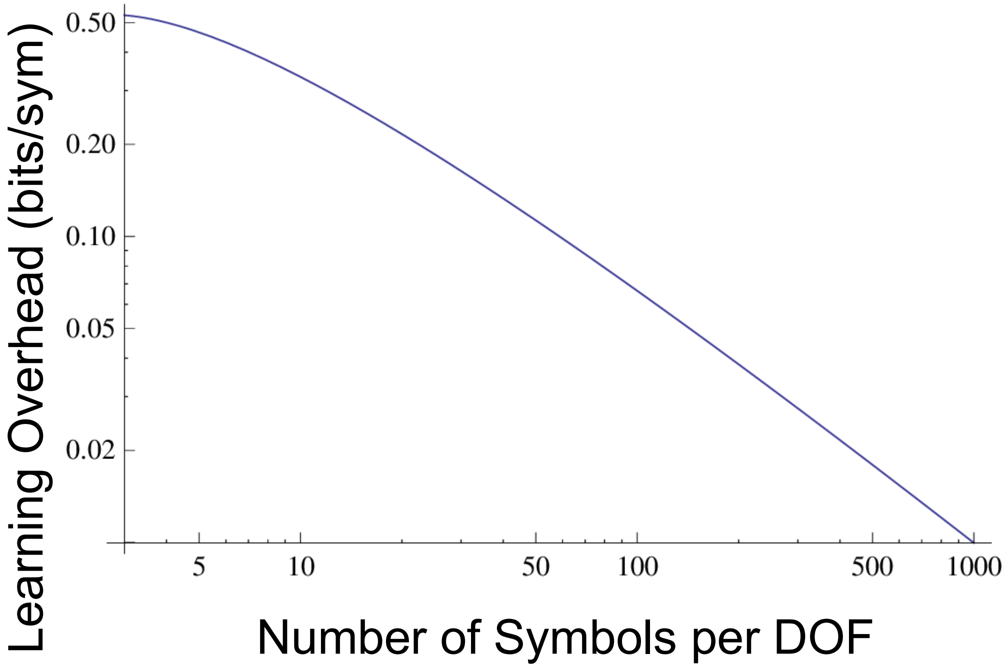
- Make learning rates symbol/context-specific (trained offline)
- Adapt learning rates dynamically
  - Code a low-probability value: start learning much faster
  - Code a high-probability value: start learning slightly slower
  - Picking thresholds for low/high in a multi-symbol world is an interesting problem!

Context Dilution

- What is the cost of adaptation in adaptive entropy coding?
  - Learning probabilities adds (roughly) **log(*N*) bits of overhead per Degree Of Freedom (DOF)** to code *N* symbols
    - Assumes a static distribution... doesn't count the cost of re-learning if the distribution changes
- How does the number of contexts impact this?
  - Lots of contexts means few symbols per context: ***context dilution***

Context Dilution

- Larger contexts amortize learning overhead



  - Need 1000 symbols per DOF to get under 1%
- Gain from better modeling must offset overhead

Static Context Merging

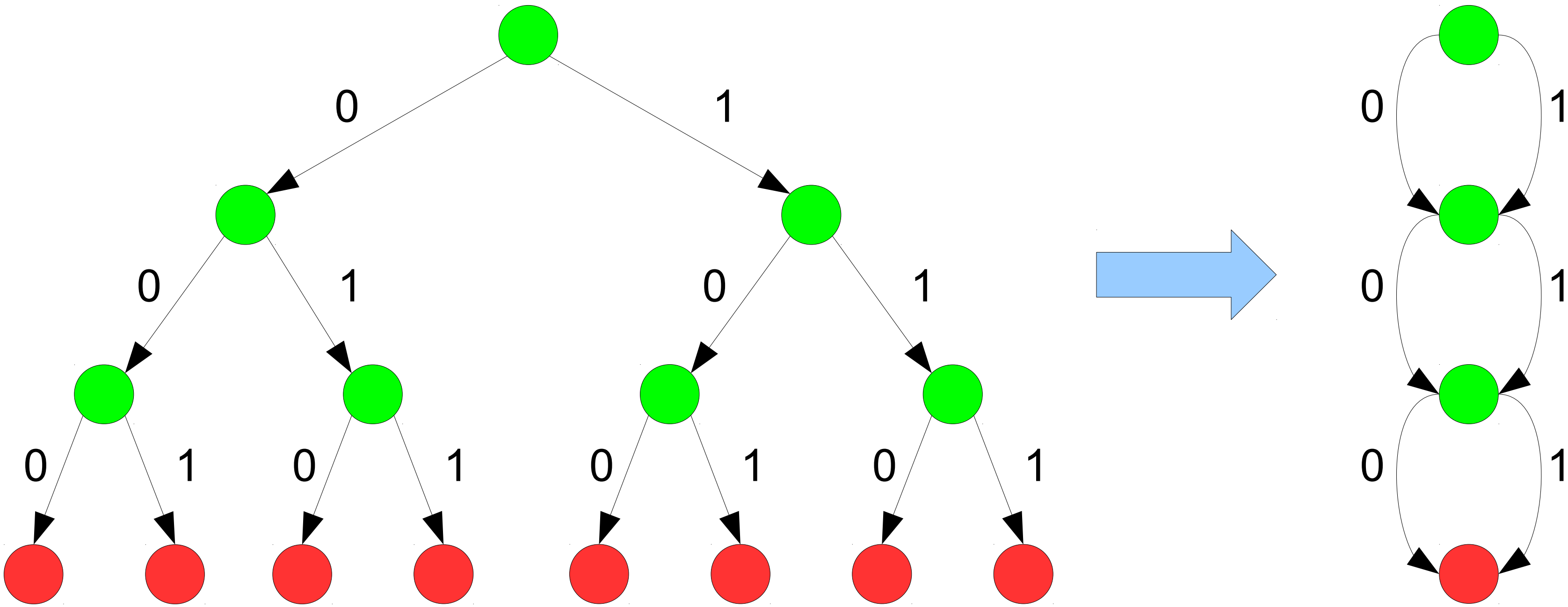
- Number of contexts in AV1 is completely fixed
- But number of symbols depends on resolution and bitrate… a *lot*
  - Current tradeoff cannot always be optimal
- **We can *merge* contexts by adding a level of indirection**
  - E.g., if the context is a ‘block size’, multiple block sizes can point to the same underlying CDF
  - Mappings can be parameterized by tile size and/or QP
  - If mapping is static per frame/tile, should be okay for hardware?
- Bitrate is also content-specific
  - Fast motion/poorly predicted content has lots more symbols
  - Also different for different frames in the prediction structure
    - Partially captured by QP, but temporal distance between frames matters, too
  - **Allow encoder to signal amount of merging?**
    - How does it know? Almost same problem as rate control
- **Challenge**: we lose differentiation in initial probabilities

Adaptive Context Splitting

- Can work in the other direction (prior art: FLIF)
  - Start by mapping all contexts for a given symbol to the same CDF
  - **Once some number of values is coded with a CDF, ‘split’ it**
    - Copy to multiple independent CDFs (distinguished by context)
    - Repeat as more symbols are coded until using whole context
- Advantage:
  - Automatic (adapts to resolution, QP, content, etc.)
- Disadvantages:
  - Increased complexity
    - Splitting should be rare in software, how bad is it for hardware?
  - Also loses differentiation in initial probabilities
    - Re-differentiate when we split?
  - How to propagate CDFs to future frames?
    - Can propagate splitting state, but is this optimal for a new frame?
    - May be an interesting idea even if restricted to intra frames?

Reduce DOFs per Context

- In addition to reducing context count, we can also reduce Degrees Of Freedom (DOFs)
- If you have worked with CABAC-based codecs, the following structures should be familiar:



- 8 possible values with 3 DOF instead of 7

Cumulative Distribution Functions (CDFs)

- In a multi-symbol world, we store probabilities as “Cumulative Distribution Functions” (CDFs)
  - I.e., the sum of the probabilities of all symbols less than a given symbol
  - Technically we store 1.0 - the CDF, or the “inverse CDF”, to save a subtraction while collecting rounding errors in the zero symbol
- CDFs are updated as the weighted average of the current CDF, cur_cdf[i], and an *elementary basis CDF*, cdf_e[k][i], for the coded symbol *k*
  - cdf_e[k][i] = 0.0 (i < k) or 1.0 (i ≥ k)
- Something interesting happened at the end of AV1 development
  - Initially had a complex mechanism to prevent probabilities going to zero during update
  - When we adopted lv_map, we reduced the precision of the arithmetic coder’s multiply to compensate for the increased symbol rate in hardware
  - That required dropping the low-order bits of the probabilities
    - But it still helped to adapt probabilities with 15 bits of precision
  - So we devised a way to enforce a minimum amount of code space for each value in the arithmetic coding engine itself
  - Which means we are now free to update CDFs in any arbitrary way we like

Reduce DOFs with CDFs

- How can we replicate the simplified CABAC structure with CDFs?
- Current CDF update is *equivalent* to a weighted average of PDFs
  - pdf_e[k][i] = 0.0 (i ≠ k) or 1.0 (i = k)
- **Can replace pdf_e[k] with sums and differences of only 3 basis vectors**
- E.g., pdf_e[0] might be

```
  {1,1,1,1,0,0,0,0}/12.0
+ {1,1,0,0,1,1,0,0}/12.0
+ {1,0,1,0,1,0,1,0}/12.0
  ---------------------
  {3,2,2,1,2,1,1,0}/12.0
```

- All PDFs are linear combinations of these three PDFs or their inverses
  - 3 DOF instead of 7

Reduce DOFs with CDFs

- No need to ape binary context models
  - How do we train a model from data?
    - Standard dimensionality reduction techniques (PCA, ICA, NMF)
- Software complexity: almost free
  - Pre-compute cdf_e[k] for each model, look up during SIMD update (just swap in different pointer)
  - Small cache penalty if we have a lot of models
- Hardware complexity:
  - Depends on the number of models
- Would want to used trained models for contexts
  - That are used a lot (e.g., coefficient coding), or
  - Where the same model can be reused for many contexts

Parametric CDFs

- Some contexts well-approximated by parametric distributions with 1 or 2 DOF
  - Laplace, etc.
- Must re-compute distribution after each update
  - Potentially okay for hardware if it can be pipelined with other symbols
  - Challenge for software depends on the distribution
- Could imagine this for the long tail of a coefficient coder
  - Easy to make strong distribution assumptions
  - Relatively infrequently used contexts
    - CDF update cost not that problematic (need to watch out for worst case!)
    - Benefit from reducing DOF relatively large

Raw Bits

- AV1 includes a “bit literal” similar in spirit to CABAC’s bypass mode
  - Nearly as expensive as full arithmetic coder in software
  - Much cheaper than full arithmetic coder in hardware (can process more bits/clock)
- Coding *actual* bit literals would be much better for software
  - Coding dozens of bits costs the same as coding one, and much less than arithmetic coding
- Solution from Daala: pack *raw bits* starting from the *end* of each packet
  - Essentially two “read pointers” from either end of the buffer that meet in the middle
- Costs one partial buffer copy during encode
  - Don’t know packet size in advance, so need to maintain two buffers and merge them
- Challenge: DRM encryption
  - No problems in CTR mode? (can encrypt/decrypt in any order)
  - Decryption is okay in CBC mode, but encryption must be done in-order
- Potential to use raw bits a lot more (e.g., in coefficient coding) if they can be made really cheap
- Is this an acceptable trade-off for hardware?

Entropy Coding Considerations

- When to do this work?
  - Refactoring tools to use larger alphabets, lower DOF models, etc., might get obsoleted by/need to be repeated for new tools
  - Optimizing context merging, per-context learning rates, etc., may change a lot as the codec changes
    - Can we automate this process enough to repeat it periodically as the codec evolves?
- Overfitting concerns
  - There are a *lot* of free parameters here
    - Context mappings for multiple bitrates, learning rates for each context, basis vectors for DOF reduction, etc.
  - Need to use large training sets, lots of encoder configurations, cross-validation, etc., to make sure we generalize sufficiently
- Implementation generalization concerns
  - Different encoders or encoder configurations may behave very differently
  - How does propagating probabilities through the prediction structure influence results?
  - Interactions with RDO

Quantization Rework (1)

- ~~Increase precision of tables~~
  - ~~Minimum step size is too large, especially for 8-bit~~
- ~~Make tables uniformly exponential~~
  - ~~Existing tables have a large linear section because of the minimum step size~~
- ~~Eliminate AC/DC offsets?~~
  - This is the role of quantization matrices
  - ~~Cannot have truly flat quantization over the whole range~~
    - ~~AC/DC values can't be made to match past a certain point~~
    - ~~DC quantizer range much smaller~~
  - ~~Current design is not documented, so unsure how to evaluate~~

Quantization Rework (2)

- Rebalance per-plane offsets
  - Per-plane weighting managed differently in various parts of libaom’s RDO
  - Should have defaults that are *consistent* and match human vision
    - And handle 4:2:2 and 4:4:4 properly!
- Allow per-plane segment offsets
  - Currently impossible to locally boost ~~all planes uniformly (because of the non-uniform tables), or~~ luma only, etc.
- ~~Extend table range?~~
- **Changing the bitstream is easy, updating the encoder is hard**
  - Many encoder tunings are based on QP
  - Changing what QP means requires understanding and reworking them all
  - libaom stuck in heavily-tuned local minimum

Frame Border Handling

- AV1 has many complicated special cases to deal with frames that are not a multiple of the superblock size
  - Which is basically every useful resolution
- Some really bizarre stuff
  - Partition size CDFs that are computed/updated unlike any others
  - Ragged partition sizes on the right/bottom edge
  - Intra prediction pixel availability for pixels outside coded region despite having no MI blocks out there (also CfL)
  - Loop filter rules that are different for every filter
- A burden on every implementer that slows down adoption
  - We've seen multiple encoders always split down partitions on the frame border to avoid many of these issues
  - Decoders are not so lucky!

simple_crop

- Simpler approach:
  - **Pad coded frame size to whole superblocks**
  - **Crop down to the visible region for display**
    - Already doing this at the 8x8 level in AV1
  - Make it the encoder's job to code outside the visible region in cheapest way
    - Set prediction residual to zero outside visible region for every prediction candidate during RDO
      - I.e., the "padding" values of the input frame are just whatever the prediction will be
- Moves all complexity to the encoder
  - Only need special cases for making *decisions*
  - Removes all special cases for syntax and semantics
- Still room for some *simple* special cases in decoder
  - Anywhere you can replace a whole symbol by a hardcoded value
  - E.g., assuming SKIP/NOSPLIT for blocks outside the visible region

simple_crop

- Tried this for AV1, but ran into implementation issues in libaom
  - Motion estimation uses tables of function pointers to SIMD routines with hard-coded block sizes to evaluate match error
  - Can’t even plumb custom sizes through most of the code
  - Fixing this requires refactoring basically the entire motion search
- Best result obtained while ignoring those problems was a 0.2% BDR loss
  - **No one was interested in taking a BDR hit to reduce implementation complexity**
    - Even though 0.2% is a lot smaller than the hit from always splitting down near frame borders! (1.5%...2.5% in rav1e)
  - Perhaps now opinions are different?
  - Results were for 64x64 superblocks, hit could be larger for 128x128

daala_tx

- Highly efficient, highly accurate factorized trigonometric transforms
- https://github.com/negge/daala_tx
- https://aomedia-review.googlesource.com/c/aom/+/37521/
- Not adopted for AV1 due to a strong preference from hardware to reuse as much of the VP9 transforms as possible
- Worth reconsidering for AV2?

Incontinent Horse Problem

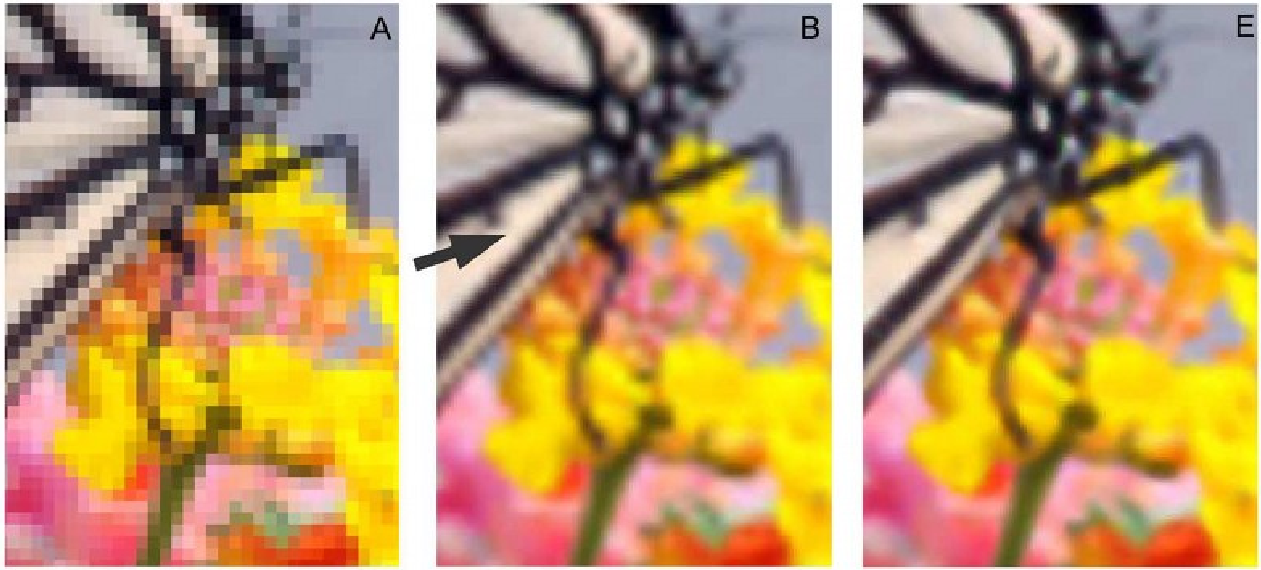
- With iterative intra prediction over small blocks
  - DC resolution is much coarser than with large blocks
  - Errors uncorrected as prediction extends to large areas

Haar DC

- Solution: Apply a Haar transform to DC coefficients
  - Each level of 2D Haar reduces the size of a DC quantization step by a factor of 2 (in pixel units)
  - Allows much smaller global shifts over large areas
- New problem
  - Need prediction residual to compute DC, but…
    - Need quantized DCs of neighbors to compute prediction, but…
      - Need DC to transform and quantize with neighbors, but...
  - DC term is fairly separable though
    - Were able to get something workable with Daala
    - Maybe possible to do the same with AV2?
- Fewer DC coefficients is also generally a good idea

Edge Directed Interpolation

- Lots of research into nonlinear interpolation
  - Can do better than linear filters, but expensive
  - Based on assumptions about natural images



From A. Giachetti and N. Asuni: “Real-Time Artifact-Free Image Upscaling.” IEEE Transactions on Image Processing, 20(10):2760–2768, Oct. 2011.

Edge Directed Subpel

- Tried this with Daala, but got poor results, even with expensive filters
- Working on CDEF gave us a clue why
  - Adding quantization noise makes per-pixel local orientation estimates *unreliable*
  - Needed to go up to 8x8 blocks to get reliable local orientation
- New proposed subpel mode
  - **Use CDEF direction search on 8x8 blocks in the reference frame** (offset by MV)
  - **Apply long(er) interpolation filter in the identified primary direction**
  - **Apply a filter with compact support in the cross direction**
- Expected benefits
  - Reduced ringing near edges in subpel motion, more SKIP usage
    - CDEF can clean up existing ringing, but only if you don’t SKIP
- Complexity
  - Re-uses highly optimized CDEF direction search (very cheap)
  - Actual filtering almost the same as existing subpel with slight pointer offsets

Inter Chroma from Luma (CfL)

- Goal: extend the very successful intra CfL to inter frames
- **Inter predict luma plane, code residual like normal**
- **Build a linear model of chroma from luma based on the reference frame pixels used for prediction**
- **Predict the chroma residual from the luma residual**
  - Added on top of the inter chroma prediction
- Not our idea: original idea from Cisco in Thor
  - draft-midtskogen-netvc-chromapred-02
- TODO: For Intra CfL, signaling the linear model was more reliable than building it from the prediction
  - What is the best way to harmonize the two approaches for inter?

Summary

- Large scale/systemic changes
  - Lots of potential entropy encoder improvements
    - Learning rates, alphabet sizes, context dilution/DOFs, raw bits
  - Many quantization improvements
    - ~~Table precision/uniformity, AC/DC offsets~~, plane offsets, per-plane segment offsets, ~~table range~~
- Old tools
  - simple_crop
  - daala_tx
- New tools
  - Haar DC
  - Edge Directed Subpel
  - Inter CfL

Questions?