Daala: A Perceptually-Driven Next Generation Video Codec

Thomas J. Daede*, Nathan E. Egge*, Jean-Marc Valin*, Guillaume Martres‡, Timothy B. Terriberry*

*Xiph.Org Foundation  †Mozilla  ‡EPFL
21 College Hill Road  331 E. Evelyn Ave.  1015 Lausanne
Somerville, MA 02144  Mountain View, CA 94041  Switzerland
United States

tterribe@xiph.org

Abstract

The Daala project is an attempt to design a royalty-free video codec capable of competing with the best patent-encumbered codecs. Part of our strategy is to replace core tools of traditional video codecs with alternative approaches, many of them designed to take perceptual aspects into account, rather than optimizing for simple metrics like PSNR. This paper documents some of our experiences with these tools, which ones worked and which did not, and what we’ve learned from them. The result is a codec which compares favorably with HEVC on still images, and is on a path to do so for video as well.

Introduction

Since its inception, Daala has used lapped transforms [1]. These promise to structurally eliminate blocking artifacts from the transform stage, one of the most annoying visual artifacts present at low bitrates [2,3]. Although extended to variable block sizes for still images and block sizes up to $16 \times 16$ [4], extensions to the larger block sizes found in modern video codecs become problematic due to the exponential search complexity. We now use a fixed-lapping scheme that admits an easy search and improves visual quality despite a lower theoretic coding gain.

Daala also employs OBMC [5] to eliminate blocking artifacts from the prediction stage. Early work demonstrated improvements simply by running OBMC as a post-process to simple block-matching algorithms [6]. This was later extended to multiple block sizes, including adaptively determining the overlap [7], but still running as a post-process. We use a novel structure borrowed from surface simplification literature that allows efficient searching and partition size selection using the actual prediction, instead of a block-copy approximation.

Daala also builds on the vector quantization work of the Opus audio codec [8], extending its gain-shape quantization scheme to support predictors and adaptive entropy coding [9]. This explicitly encodes the gain of bands of AC coefficients, that is, their contrast, and explicitly codes how well each band matches the predictor. By extracting a small number of perceptually meaningful parameters from an otherwise undifferentiated set of transform coefficients, this “Perceptual Vector Quantization” enables a host of new techniques. We have demonstrated its use for automatic activity masking (to preserve detail in low-contrast regions) [9] and frequency-domain Chroma-from-Luma (CfL) to enhance object boundaries in the color planes [10].
Both lapped transforms and PVQ are particularly susceptible to ringing artifacts, the former due to the longer basis functions and the latter due to the tendency either to skip entire diagonal bands (giving artifacts on diagonal edges similar to wavelets) or to inject energy into the wrong place when trying to preserve contrast. Furthermore, lapping prevents strong directional intra prediction, which could be used to create clean edges. Therefore we designed a sophisticated directional deringing filter, which can aggressively filter directional features with minimal side information.

**Methodology**

This section attempts to describe why we made many of the choices we did. All of the code, including the full commit history, is available in our public git repository [11]. Where appropriate, it includes the four metrics we commonly examine, PSNR, SSIM [12], PSNR-HVS-M [13], and multiscale FastSSIM [14]. Unless otherwise specified, Bjøntegaard-delta [15] (BD) rate changes and other results are from our automated testing framework [16]. By default this uses 18 sequences ranging in resolution from 416 × 240 to 1920 × 1080 and 48 to 60 frames in length.

**Lapped Transforms**

At least as far back as H.263, video codecs have used adaptive filters to remove blocking artifacts. However, there are also non-adaptive solutions to the blocking problem: lapped transforms. Daala uses the time-domain construction from [3], with the DCT and lapping implemented using reversible lifting transforms.

Originally, the application of lapping was done in an order similar to that of a loop filter. We applied the post-filter to rows of pixels first, for the entire image, and then columns. This allows maximal parallelism with minimal buffering. The pre-filter ran in the opposite order. However, this has two issues. First, it creates oddly-shaped basis functions in the presence of varying block sizes. Although possible to see in synthetic examples, we never observed an issue with this on real images, and adaptive deblocking filters have a similar issue. Second, it makes block size decisions NP-hard, because the order and amount of lapping to apply remains unknown until the size of both the current block and its neighbors is known, creating a two-dimensional dependency graph that does not admit a tree structure.

We developed a heuristic to make up-front block size decisions, without a rate-distortion optimization (RDO) search, based on the estimated visibility of ringing artifacts. However, it was clearly making sub-optimal decisions for video, often choosing large transforms when only a small portion of a block was updated. In order to make a real RDO search tractable, we made two changes. First, we made the order recursive: we first apply the lapping to the exterior edges of a 32 × 32 superblock, then if we are splitting to a 16 × 16, we filter the interior edges. This is essentially the same as the order proposed by Dai et al. [4]. Second, we fixed the lapping size to an 8-point filter (4 pixels on either side of a block edge). For 4 × 4 blocks, we apply 4-point lapping to the interior edges of an 8 × 8 block (overlapping with the 8-point lapping applied to the exterior edges). When subsampled, the chroma planes always use 4-point lapping.
This removed the dependency on the neighbors’ block sizes, allowing for a tree-structured dynamic programming search. This proceeds bottom up. At each level, we start with the optimal block size decision using blocks of at most $N/2 \times N/2$, undo the lapping on the interior edges of the quadtree split, and compare to using an $N \times N$ block. In both cases the exterior edges of the $N \times N$ block remain lapped, but we can at least make an apples-to-apples comparison of the relative distortions.

The result is an optimal solution using blocks of at most $N \times N$. This optimization procedure produced BD-rate reductions of 10.4% for PSNR, and 12.3% for SSIM. On our more perceptual metrics, the changes were smaller: 4.5% for PSNR-HVS-M and 5.2% for multiscale FastSSIM. This reflects the reduction in coding gain from the reduced lapping sizes. These gains are almost entirely due to the improved decisions. Using the same decisions produced by this fixed-lapping process with the previous variable-lapping scheme gave almost as much gain.

However, despite the smaller lapping size, this scheme actually increases ringing artifacts. We primarily code edges with $4 \times 4$ blocks, but it increases the support of the $4 \times 4$ blocks from 8 pixels to 12. To further reduce ringing, we are currently using 4-point lapping everywhere. This provided another 3.6% gain in PSNR, and 1.4% on PSNR-HVS-M, but lost 0.4% on multiscale FastSSIM. The visual impact is somewhat larger, and seems to be a regression for intra frames, but a win in some cases on video. This will require more systematic visual testing.

**Bilinear Smoothing Filter**

Because the smaller lapping size no longer completely eliminates blocking, especially in smooth gradients, we apply a bilinear smoothing filter to all $32 \times 32$ blocks in keyframes. The filter simply computes a bilinear interpolation between the four corners of a decoded block (after unlapping). Then, it blends the result of that interpolation with the decoded pixels. Unlike a conventional deblocking filter, it does not look outside of the current block at all.

For a given quantization step size, $Q$, we can compute the optimal Wiener filter gain

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

where $\alpha$ is a strength parameter (currently set to 5 for luma and 20 for chroma) and $D^2$ is the mean squared error between the decoded block and the bilinear interpolation. However, when we blend, we actually found using $w^2$ works better than $w$, as it applies less smoothing when we are uncertain if it is a good idea. The result is actually a small ($< 1\%$) regression on metrics on subset3, our large still image training subset, but provides a substantial visual reduction in blocking in gradients at low rates.

**Perceptual Vector Quantization**

Perceptual Vector Quantization is an extension of the gain-shape vector quantization used in Opus [8] to take advantage of prediction and adaptive entropy coding. The main idea is that it splits the AC coefficients into bands and explicitly codes the
magnitude ("gain"), $g$ of each band. Then, the band is normalized to a unit vector, and the "shape" of the spectrum is encoded separately. To handle prediction, we apply a Householder reflection to map the normalized prediction onto one of the axes, and then code the angle between that axis and the vector being quantized, $\theta$. What remains is a vector on a sphere of dimension $N - 1$ (where $N$ here is the size of the band) with a known radius $g \sin \theta$, which we code using Pyramid Vector Quantization [17] to reduce the number of degrees of freedom to $N - 2$, eliminating any redundancy with $g$ and $\theta$. For details, we refer readers to [9].

One thing this provides is explicit energy preservation, which is less important for video than audio, and can even be relaxed to save bits during RDO. More importantly, the gain directly tells us the contrast in each band. This allows us to implement activity masking without sending any extra side information. The gain directly encodes the amount of contrast in a band. Instead of using a linear quantizer, we can compand the gain to get more resolution for smaller gains. Then, once we know the gain, we can adjust the quantizer for $\theta$ and the shape vector to give more bits to smooth regions where errors are easily visible, and less to textured regions where they are not. We disable activity masking on $4 \times 4$ blocks, to avoid over-penalizing edges.

*Chroma-from-Luma Prediction*

Although color conversion to Y’CbCr decorrelates the color channels globally across the frame, there is still some local correlation. Lee and Cho created a spatial domain intra predictor using a linear model of the relationship between chroma and luma built from previously-decoded neighbors [18]. Because Daala uses lapped transforms, these neighbors are unavailable when they are needed. Although it would be possible, with additional complexity, to use more distant neighbors, this technique is even more efficient in the frequency domain. PVQ even makes it possible to remove the model fitting step entirely. For more details, see [10].

We can simply use the reconstructed luma coefficients directly as the chroma shape predictor in PVQ.

\[
C_{AC}(u, v) = \alpha_{AC} \cdot L_{AC}(u, v) \implies r = \alpha_{AC} \cdot \hat{x}_L
\] (2)
While the value of $\alpha_{AC}$ can be learned in the decoder, it is sometimes wrong. It is cheaper to simply code its sign. This technique allows us to predict features within a block that cannot be predicted by straight edge extension. Although this significantly reduces the number of bits spent on chroma, the perceptual impact is even larger, giving cleaner edges than the horizontal and vertical intra prediction we use for the luma plane.

**Deringing Filter**

The use of lapped transforms and the lack of directional intra prediction, along with energy preservation in the band structure of PVQ [9], make Daala particularly susceptible to ringing artifacts. To combat this, Daala uses an in-loop deringing filter that takes into account the direction of edges and patterns being filtered. On keyframes, this runs before the bilinear smoothing filter. The filter identifies the direction of each block and adaptively filters along that direction. A second filter runs across the lines filtered by the first filter, with more conservative thresholds to avoid blurring edges. We describe the process in some detail here, since it has not been published elsewhere.

First, the decoder splits the image into $8 \times 8$ blocks, and determines a dominant direction for each block from the decoded image. These directions do not need to be transmitted, reducing the overhead of side information for the filter. A perfectly directional block would have a constant value along all lines in a given direction. The decoder minimizes the “mean squared difference” (MSD) between the decoded block and a perfectly directional block formed by taking the mean of the pixels in each line.

For each direction, $d$, we partition the pixels into distinct lines, as illustrated in Fig. 3(a), indexing the lines by $k$. The MSD, $\sigma^2_d$, is then

$$\sigma^2_d = \frac{1}{N} \sum_{k=0}^{N_d-1} \left[ \sum_{p \in P_{d,k}} (x_p - \mu_{d,k})^2 \right],$$

(3)
Figure 3: Line numbers for pixels following one direction in an $8 \times 8$ block and the steps between pixels for each direction for each filter stage. Pixels are always sampled using nearest-neighbor filtering, with no subpel filter.

where $P_{d,k}$ is the set of pixels in line $k$ following direction $d$, $x_p$ is the value of the pixel at location $p$, $N_d$ is the number of lines in the block with direction $d$, and $N$ is the total number of pixels in the block. $\mu_{d,k}$ is the pixel average for line $k$ following direction $d$:

$$
\mu_{d,k} = \frac{1}{N_{d,k}} \sum_{p \in P_{d,k}} x_p ,
$$

where $N_{d,k}$ is the number of pixels in $P_{d,k}$.

Substituting (4) into (3) and simplifying, we get

$$
\sigma_d^2 = \frac{1}{N} \left[ \sum_{p \in \text{block}} x_p^2 - \sum_{k=0}^{N_d-1} \frac{1}{N_{d,k}} \left( \sum_{p \in P_{d,k}} x_p \right)^2 \right] .
$$

The first term is constant. The optimal direction, $d_{\text{opt}}$, is then just

$$
d_{\text{opt}} = \max_d s_d ,
$$

where

$$
s_d = \sum_{k=0}^{N_d-1} \frac{1}{N_{d,k}} \left( \sum_{p \in P_{d,k}} x_p \right)^2 .
$$

The decoder selects a direction for each block even when there are no strong directional features, as its main purpose is to allow stronger filtering without crossing directional edges.
We use what we term a “conditional replacement filter” to remove noise without blurring sharp edges, like a median filter or bilateral filter, but simpler and easier to vectorize with SIMD:

\[ y(n) = x(n) + \frac{1}{W} \sum_{k=-M}^{k=M} w_k \text{thresh} (x(n+k) - x(n), T) , \]

(8)

with the threshold function

\[ \text{thresh}(d, T) = \begin{cases} d, & |d| < T, \\ 0, & \text{otherwise}. \end{cases} \]

(9)

The effect is that the value of pixels whose difference from the center pixel, \( x(n) \), exceed the threshold \( T \) are simply replaced by the value of \( x(n) \). This keeps the normalization weight, \( W \), constant, and setting it to a power of two eliminates the division.

The first, or “directional” filter for pixel \((i,j)\) is the 7-tap conditional replacement filter

\[
y(i,j) = x(i,j) + \frac{1}{W} \sum_{k=1}^{3} w_k \left[ \text{thresh} \left( x(i,j) - x \left( i + \left\lfloor kd_y \right\rfloor , j + \left\lfloor kd_x \right\rfloor \right), T_d \right) \\
+ \text{thresh} \left( x(i,j) - x \left( i - \left\lceil kd_y \right\rceil , j - \left\lceil kd_x \right\rceil \right), T_d \right) \right]
\]

(10)

where \( d_x \) and \( d_y \) are defined in Table 3(b) and \( T_d \) is the threshold for the directional filter stage. Since the direction is constant over \( 8 \times 8 \) blocks, all operations in this filter are directly vectorizable.

We choose the weights \( w_k \) to be \( w = [3 \ 2 \ 2] \) with \( W = 16 \). Although ringing is roughly proportional to the quantization step size, \( Q \), as the quantizer increases the error grows less than linearly because the unquantized coefficients become very small compared to \( Q \). We start with a power model of the form

\[ T_0 = \alpha_1 Q^\beta , \]

(11)

with \( \beta = 0.842 \) and \( \alpha_1 = 1 \), which were chosen by manually testing thresholds for \( Q = 5 \) and \( Q = 400 \) (on a linear scale). We can use a stronger filter on more directional blocks, both because they have more ringing, and because there is less chance of blurring non-directional features. Blocks that are less directional require a weaker filter. We estimate the degree of directionality, \( \delta \), as the difference between the optimal variance and the variance along the orthogonal direction:

\[ \delta = |\sigma_{d_{\text{opt}}}^2 - \sigma_{(d_{\text{opt}}+4) \mod 8}^2| = s_{d_{\text{opt}}} - s_{(d_{\text{opt}}+4) \mod 8} . \]

(12)

The final threshold is then

\[ T_d = T_0 \cdot \max \left( \frac{1}{2}, \min \left( 3, \alpha_2 (\delta \cdot \delta_{sb})^{0.16} \right) \right) , \]

(13)
where $\delta_{sb}$ is the average of $\delta$ over a $32 \times 32$ superblock and $\alpha_2 = 1.02$.

The second filter stage is always horizontal or vertical, and is chosen to operate across the direction lines used in the first filter:

$$z(i, j) = y(i, j) + \frac{1}{W} \sum_{k=1}^{2} w_k \left[ \text{thresh} \left( y(i, j) - y(i + \lfloor kd_y \rfloor, j + \lfloor kd_x \rfloor), T_d \right) 
+ \text{thresh} \left( y(i, j) - y(i - \lfloor kd_y \rfloor, j - \lfloor kd_x \rfloor), T_2(i, j) \right) \right]$$

(14)

where $d_x$ and $d_y$ are defined in Table 3(c) and $T_2(i, j)$ is a position-dependent threshold for the second stage. Since the second filter risks blurring edges, and its input has considerably less ringing than the first, it only has 5 taps and we choose $T_2(i, j)$ more conservatively:

$$T_2(i, j) = \min \left( T_d, \frac{T_d}{3} + |y(i, j) - x(i, j)| \right).$$

(15)

We choose the filter weights to be $w = [3 \quad 3]$ with $W = 16$.

If a superblock was skipped and is not in an intra frame, it is never deringed. Otherwise, an entropy-coded flag enables or disables deringing for a superblock. Even when deringing is enabled on a superblock, we do not dering $8 \times 8$ blocks that were skipped and whose surrounding $4 \times 4$ neighbors were also skipped (taken into account because of lapping). The deringing process sometimes reads pixels that lie outside the current superblock. When these pixels belong to another superblock, the filtering always uses the unfiltered pixel values—even for the second stage filter—to avoid adding a dependency between superblocks. This makes it possible to filter all superblocks in parallel. The threshold function $\text{thresh}(d, T)$ always returns 0 for pixels outside of the image.

### Subjective Results

Daala participated in the 2015 Picture Coding Symposium evaluation of existing and future still image coding technologies [19]. This included both an objective evaluation using metrics different from the ones used during development, and a subjective evaluation with test subjects drawn from a pool of multimedia quality experts. In the subjective evaluation, Daala was compared with five other state of the art video and still image codecs including BPG [20] and VP9 [21]. In Figure 4 we show the subjective results for two of the six still images tested.

The image where Daala had the best results was **woman**, a close up portrait of a woman wearing a knit shirt. About two thirds of this image are hair and skin, which contain texture that is well coded using PVQ even at very low rates. Quantization errors in the hair are hidden by activity masking where other codecs end up removing detail by zeroing the high frequencies.

The image where Daala had the worst result was **bike**, a picture of a tennis racket leaned against a bike wheel. This picture contains a lot of strong directional edges which are very well predicted by directional intra-prediction. Because Daala only
Figure 4: Subjective still-image comparisons of Daala and several competing codec implementations. Shown here are results from the images that gave the (a) best and (b) worst results for Daala.

Figure 5: Metric comparisons between Daala, libvpx-vp9 1.4.0, and x265 1.6.

contains a limited set of horizontal and vertical predictors, it is forced to spend bits coding regions that are well predicted in other codecs. In addition, the version of Daala used for the competition was using an earlier prototype of the deringing filter that was less effective.

**Objective Results**

We computed objective results using standard metrics, using implementations in the Daala repository. Daala is compared against x265 1.6 and libvpx-vp9 1.4.0. The comparisons are made automatically by the AreWeCompressedYet tool, and the source code is available online [16]. Two perception-based metrics are used, PSNR-HVS-M and Fast MS-SSIM [14] [13]. In addition, PSNR metric results are also shown. Daala does much better than the other two codecs at high bitrates, though there is still room for improvement at lower rates. As expected, Daala does much better at the perceptual metrics than PSNR, an effect of its perceptually based coding methods.
References


