Using Daala Intra Frames for Still Picture Coding

Nathan E. Egge, Jean-Marc Valin, Timothy B. Terriberry, Thomas Daede, Christopher Montgomery
Mozilla
Mountain View, CA, USA

***Abstract*—Recent advances in video codec technology have been successfully applied to the field of still picture coding. Daala is a new royalty-free video codec under active development that contains several novel coding technologies. We consider using Daala intra frames for still picture coding and show that it is competitive with other video-derived image formats. A finished version of Daala could be used to create an excellent, royalty-free image format.**

I. Introduction

The Daala video codec is a joint research effort between Xiph.Org and Mozilla to develop a next-generation video codec that is both (1) competitive performance with the state-of-the-art and (2) distributable on a royalty free basis. In working to produce a royalty free video codec, Daala introduces new coding techniques to replace those used in the traditional block-based video codec pipeline. These techniques, while not completely finished, already demonstrate that it is possible to deliver a video codec that meets these goals.

All video codecs have, in some form, the ability to code a still frame without prior information. As the bitstreams of other video codecs have stabilized, new standards in still picture coding have been proposed most notably with WebP based on VP8 [1] and BPG based on H.265 [2].

This paper outlines the techniques used in Daala to code just these still-image 'intra' frames. It includes a brief experimental study comparing still image performance against other lossy still picture codecs, e.g., JPEG, WebP, and BPG, using two quality metrics: PSNR and Fast MS-SSIM[3]. In the future work section we describe what features are missing from Daala that would be required by an effort to standardize Daala intra frames as a stand alone still picture format.

II. Coding Techniques

Coding of intra frames in Daala is similar to other block based image codecs. An input image is broken into *super blocks* of 32x32 pixels which are processed left-to-right, top-to-bottom. Based on content, each *super block* can be split into quadrants down to blocks as small as 4x4 pixels. Each block goes through familiar stages: transform, prediction, quantization, and coefficient coding. However, many traditional picture coding approaches are inapplicable to Daala, and new techniques need to be invented.

For example, Daala's use of an invertible lapped transform (see Section II-A) means that spatial information from neighboring blocks is not available for use in block prediction, as shown in Figure 1. On the other hand, use of gain-shape quantization (see Section II-B) allows for new intra-prediction techniques. Unsignaled horizontal and vertical prediction (see Section II-D), and low complexity prediction of chroma coefficients from their spatially coincident luma coefficients (see Section II-E) are two examples.

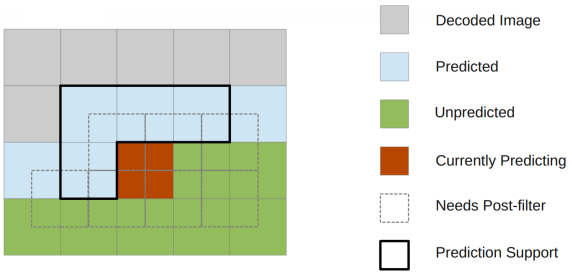


Fig. 1. State of blocks within the decode pipeline of a codec using lapped transforms. Immediate neighbors of the target block (bold lines) cannot be used for spatial prediction as they still require post-filtering (dotted lines).

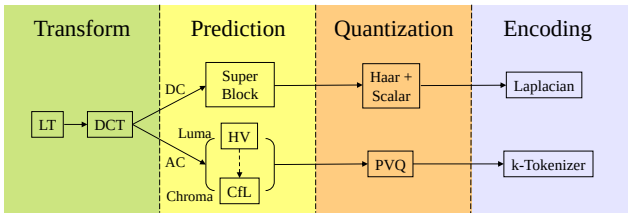


Fig. 2. High-level block diagram of Daala encode pipeline.

A. Time-Domain Lapped Transforms

Lapped transforms were proposed for video coding as early as 1989 [4]. Like the loop filters more commonly found in recent video coding standards, Time-Domain Lapped Transforms (TDLTs) use a post-processing filter that runs between block edges to reduce or eliminate blocking artifacts. Unlike a loop filter, the TDLT filter is invertible, allowing the encoder to run the inverse filter on the input video [5].

The Time-Domain Lapped Transform can be viewed as a set of pre- and post- filters to an existing block-based DCT transform.

Subset 1 γ_c	4x4	8x8	16x16
KLT	12.47dB	13.62dB	14.12dB
DCT	12.42dB	13.55dB	14.05dB
CDF(9/7)	13.14dB	13.82dB	14.01dB
LappedKLT	13.35dB	14.13dB	14.40dB
LappedDCT	13.33dB	14.12dB	14.40dB

Fig. 4. Coding gain figures for several 2-D transforms collected over Xiph.Org's standard image coding test subset 1.

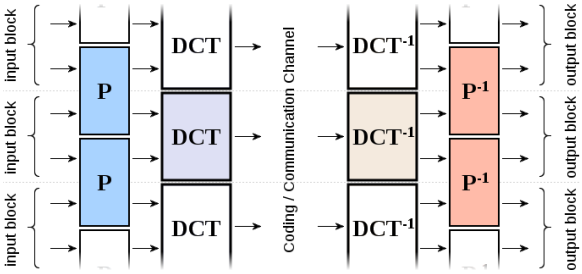


Fig. 3. Illustration of time-domain lapped transforms consisting of the DCT with pre-filters P and post-filters P^{-1} straddling block boundaries.

The pre-filter P operates in the time domain, processing block boundaries and removing inter-block correlation. The blocks are then transformed by the DCT into the frequency domain, where the resulting coefficients are quantized and encoded. When decoding, the inverse operator P^{-1} is applied as a post-filter to the output of the inverse DCT. This has two benefits:

1) Quantization errors are spread over adjacent blocks via the post-filter P^{-1}, reducing blocking artifacts without the need for a separate deblocking filter.
2) The increased support region of the transform allows it to take advantage of inter-block correlation to achieve a higher coding gain than a non-overlapped DCT. This allows it to more effectively code both smooth and textured regions (see Figure 4.)

Daala implements the TDLT as a reversible lifting filter with perfect reconstruction, allowing lossy or fully lossless operation at up to 12-bits of input depth [6].

B. Gain-Shape Quantization

The Daala video codec uses gain-shape quantization, a form of vector quantization in which the vector of coefficients $\mathbf{x}$ is separated into two intuitive components: a magnitude (*gain*) and its direction (*shape*). The gain $g = \|\mathbf{x}\|$ represents how much energy is contained in the block, and the shape $\mathbf{u} = \mathbf{x}/\|\mathbf{x}\|$ indicates where that energy is distributed among the coefficients. The gain is then quantized using scalar quantization, while the shape is quantized by finding the nearest VQ-codeword in an algebraically defined codebook. The codec does not explicitly store the VQ-codebook and the encoder need search only a small set of VQ-codewords around the input vector.

Given the gain quantization index γ_g, the shape vector quantization index γ_u and an implicitly defined VQ-codebook

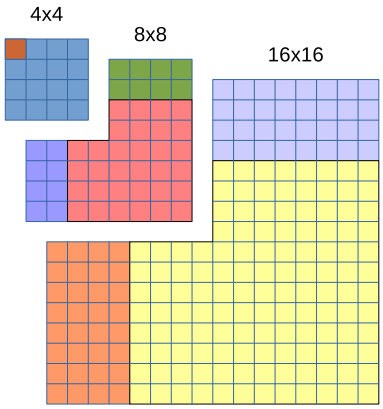


Fig. 5. Bands used for PVQ. 32x32 and larger blocks follow a similar pattern.

CB, the reconstructed gain $\hat{g}$ and shape $\hat{\mathbf{u}}$ can be found by

$$\hat{g} = \gamma_g \cdot Q \tag{1}$$
$$\hat{\mathbf{u}} = CB[\gamma_u] \tag{2}$$

and reconstructed coefficients $\hat{\mathbf{x}}$ are thus

$$\hat{\mathbf{x}} = \hat{g} \cdot \hat{\mathbf{u}} \tag{3}$$

Each block is subdivided into bands, as shown in Figure 5. Each band is separately coded with a gain and shape. By explicitly signaling the amount of energy in a block, and roughly where that energy is located, gain-shape quantization is texture preserving.

The algebraic codebook used in Daala is based on the pyramid vector quantizer described by Fisher [7]; we refer to our extended technique as Perceptual Vector Quantization (PVQ). A complete description of PVQ usage in Daala and its other advantages over scalar quantization is outside the scope of this paper and is described in detail by Valin [8].

C. Prediction with PVQ

In block based codecs, intra prediction can often construct a very good predictor for the block that will be decoded next. In the encoder, this predicted block is typically subtracted from the input image and the residual is transformed to the frequency domain, quantized and entropy coded. When the transform is linear, as is the case with codecs based on lapped transforms, this is equivalent to transforming the predictor and computing the difference in the frequency domain. However, if one were to simply quantize the frequency domain residual using PVQ, the texture preservation property described in the previous section would be lost; the energy is no longer that of the block being coded, but rather the difference from its predictor. In Daala, this is avoided by explicitly *not* computing a residual, but instead extracting another intuitive parameter in gain-shape quantization.

We can reduce the entropy of the symbols we code and retain the energy preserving properties of PVQ by using a Householder reflection. Considering the predictor as another n-dimensional vector, a reflection plane is computed that

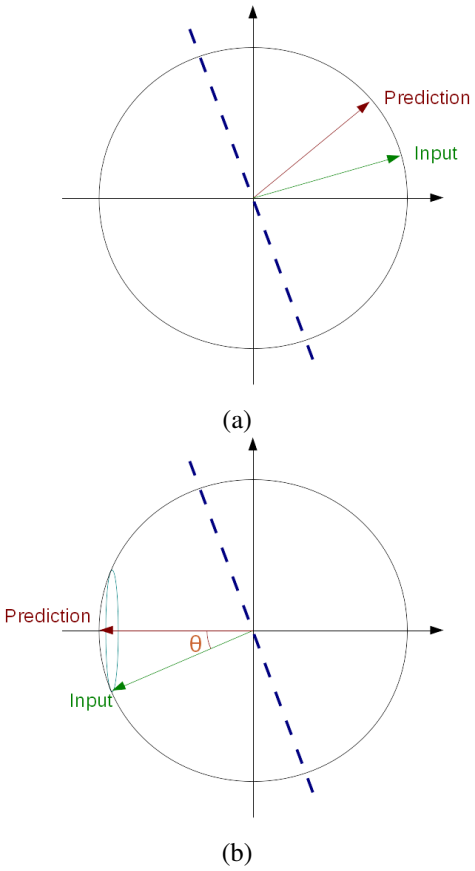


Fig. 6. (a) A Householder reflection plane is computed that aligns the prediction vector so that its largest component is along an axis. (b) The input vector is reflected across the plane and the angle θ is computed and coded using scalar quantization. The axis on which the predictor lies is eliminated, the remaining $n-1$ dimensions are coded using PVQ.

aligns the predictor with one of the axes in our n-dimensional vector space making all but one of the components in the predictor equal zero. The encoder can then reflect the input vector $\mathbf{x}$ across this reflection plane in a way that is perfectly reproducible in the decoder, see Figure 6.

Let $\mathbf{r}$ be the n-dimensional vector of predictor coefficients. Then the normal to the reflection plane can be computed as

$$\mathbf{v} = \frac{\mathbf{r}}{\|\mathbf{r}\|} + s \cdot \mathbf{e}_m \tag{4}$$

where $s \cdot \mathbf{e}_m$ is the signed unit vector in the direction of the axis we would like to reflect $\mathbf{r}$ onto. The input vector $\mathbf{x}$ can then be reflected across this plane by computing

$$\mathbf{z} = \mathbf{x} - 2\frac{\mathbf{v}^T\mathbf{x}}{\mathbf{v}^T\mathbf{v}}\mathbf{v} \tag{5}$$

We can measure how well the predictor $\mathbf{r}$ matches our input vector $\mathbf{x}$ by computing the cosine of the angle θ between them as

$$\cos\theta = \frac{\mathbf{x}^T\mathbf{r}}{\|\mathbf{x}\|\,\|\mathbf{r}\|} = \frac{\mathbf{z}^T\mathbf{r}}{\|\mathbf{z}\|\,\|\mathbf{r}\|} = -s\frac{z_m}{\|\mathbf{z}\|} \tag{6}$$

We are free to choose any axis in our n-dimensional space and we select $\mathbf{e}_m$ to be the dimension of the largest component of our prediction vector $\mathbf{r}$ and $s = \mathrm{sgn}(r_m)$. Thus the largest component lies on the m-axis after reflection. When the predictor is good, we expect that the largest component of $\mathbf{z}$ will also be in the $\mathbf{e}_m$ direction and θ will be small. If we code $\hat{\theta}$ using scalar quantization, we can remove the largest dimension of $\mathbf{z}$ and reduce the coding of $\mathbf{x}$ to a gain-shape quantization of the remaining $n-1$ coefficients where the gain has been reduced to $\sin\theta \cdot g$. Given a predictor $\mathbf{r}$, the reconstructed coefficients $\hat{\mathbf{x}}$ are computed as

$$\hat{\mathbf{x}} = \hat{g}\big(-s \cdot \cos\hat{\theta} \cdot \mathbf{e}_m + \sin\hat{\theta} \cdot \hat{\mathbf{u}}\big) \tag{7}$$

When the predictor is poor, θ will be large and the reflection is unlikely to make things easier to code. Thus when θ is greater than $90°$ we code a flag and use PVQ with no predictor. Conversely when $\mathbf{r}$ is exact, $\hat{\theta}$ is zero and no additional information needs to be coded. In addition, because we expect $\mathbf{r}$ to have roughly the same amount of energy as $\mathbf{x}$, we can get additional compression performance by using $\|\mathbf{r}\|$ as a predictor for g:

$$\hat{g} = \gamma_g \cdot Q + \|\mathbf{r}\| \tag{8}$$

D. Horizontal & Vertical Intra Prediction

Traditional directional intra prediction, as seen in video codecs like H.264, is not usable for lapped transform based codecs. The final spatial reconstruction of the neighboring blocks, necessary for prediction, is not available until after prediction and decode, as shown in Figure 1. However, the complete frequency-domain coefficients are available.

A simple horizontal and vertical predictor is implemented by copying the first row of coefficients of the horizontal bands from the block above, and the first column of horizontal coefficients from the block to the left (see Figure 5 for the band layout). No signaling is done at this step - rather, PVQ can signal on a per-band basis whether to use a band's prediction or not. This is more flexible as well, as there are multiple horizontal and vertical bands on larger blocks.

The first band is nondirectional and a special case - whether to copy from the top or left is decided based on which has a greater variance in the row or column to be copied.

E. Chroma from Luma Prediction

In spatial-domain chroma-from-luma [9], the key observation is that the local correlation between luminance and chrominance can be exploited using a linear prediction model. For the target block, the chroma values can be estimated from the reconstructed luma values as

$$C(u,v) = \alpha \cdot L(u,v) + \beta \tag{9}$$

where the model parameters α and β are computed as a linear least-squares regression using N pairs of spatially coincident luma and chroma pixels.

When α and β are sent explicitly in the bitstream, the pairs are taken from the original, unmodified image. However, the decoder can also compute the same linear regression using its previously decoded neighbors and thus α and β can be derived *implicitly* from the bitstream.

In codecs that use lapped transforms, the reconstructed pixel data is not available. However the transform coefficients in the lapped frequency domain are the product of two linear transforms: the linear pre-filter followed by the linear forward DCT. Thus the same assumption of a linear correlation between luma and chroma coefficients holds. In addition, we can take advantage of the fact that we are in the frequency domain to use only a small subset of coefficients when computing our model.

The chroma values can then be estimated using frequency-domain chroma-from-luma (FD-CfL) [10]:

$$C_{DC} = \alpha_{DC} \cdot L_{DC} + \beta_{DC} \tag{10}$$

$$C_{AC}(u,v) = \alpha_{AC} \cdot L_{AC}(u,v) \tag{11}$$

where the α_{DC} and β_{DC} are computed using a linear regression with the DC coefficients of the three neighboring blocks: up, left and up-left. When estimating $C_{AC}(u,v)$ we can omit the constant offset β_{AC} as we expect the AC coefficients to be zero mean. Additionally, we do not include all of the AC coefficients from the same three neighboring blocks when computing α_{AC}.

Consider what happens when the frequency-domain chroma-from-luma (FD-CfL) algorithm it is used with gain-shape quantization. As an example, consider a 4x4 chroma block where the 15 AC coefficients are coded using gain-shape quantization with the FD-CfL predictor from Equation 11. The 15-dimensional predictor $\mathbf{r}$ is simply a linearly scaled vector of the coincident reconstructed luma coefficients:

$$C_{AC}(u,v) = \alpha_{AC} \cdot L_{AC}(u,v) \implies \mathbf{r} = \alpha_{AC} \cdot \hat{\mathbf{x}}_L \tag{12}$$

Thus the shape of the chroma predictor $\mathbf{r}$ is exactly that of the reconstructed luma coefficients $\hat{\mathbf{x}}_L$ with one exception:

$$\frac{\mathbf{r}}{\|\mathbf{r}\|} = \frac{\alpha_{AC} \cdot \hat{\mathbf{x}}_L}{\|\alpha_{AC} \cdot \hat{\mathbf{x}}_L\|} = \mathrm{sgn}(\alpha_{AC}) \frac{\hat{\mathbf{x}}_L}{\|\hat{\mathbf{x}}_L\|} \tag{13}$$

Because the chroma coefficients are sometimes inversely correlated with the coincident luma coefficients, the linear term α_{AC} can be negative. In these instances the *shape* of $\hat{\mathbf{x}}_L$ points in exactly the wrong direction and must be flipped.

Moreover, consider what happens to the gain of $\mathbf{x}_C$ when it is predicted from $\mathbf{r}$. The PVQ prediction technique assumes that $\|\mathbf{r}\| = \alpha_{AC} \cdot \|\hat{\mathbf{x}}_L\|$ is a good predictor of $g_C = \|\mathbf{x}_C\|$. Because α_{AC} for a block is learned from its previously decoded neighbors, often it is based on highly quantized or even zeroed coefficients. When this happens, $\alpha_{AC} \cdot \|\hat{\mathbf{x}}_L\|$ is no longer a good predictor of g_C and the cost to code $\|\mathbf{x}_C\| - \alpha_{AC} \cdot \|\hat{\mathbf{x}}_L\|$ using scalar quantization is actually greater than the cost of just coding g_C alone.

Thus we present a modified version of PVQ prediction in Section II-C that is used just for chroma-from-luma intra prediction called PVQ-CfL. For each set of chroma coefficients coded by PVQ, the prediction vector $\mathbf{r}$ is exactly the coincident luma coefficients. We determine if we need to flip the predictor by computing the sign of the cosine of the angle between $\hat{\mathbf{x}}_L$ and $\mathbf{x}_C$:

$$f = \mathrm{sgn}(\hat{\mathbf{x}}_L^T \mathbf{x}_C) \tag{14}$$

A negative sign means the angle between the two is greater than $90°$ and flipping $\hat{\mathbf{x}}_L$ is guaranteed to make the angle less than $90°$.

We then code f using a single bit[3], and the gain $\hat{g}_C$ using scalar quantization with no predictor. The shape quantization algorithm for $\mathbf{x}_C$ is unchanged except that $\mathbf{r} = f \cdot \hat{\mathbf{x}}_L$. This algorithm has the advantage over FD-CfL of being both lower complexity (neither the encoder nor decoder need to compute a linear regression per block) and providing better compression (the chroma gain g_C is never incorrectly predicted).

F. Paint Deringing Filter

The paint deringing filter is a post-processing step to attenuate ringing artifacts. The idea is to turn the reconstructed image into a *painting*, that is an image where pixels in each block follow a single direction. In regions where the original image is also directional, such as edges, the painted image looks better than the coded image, especially at low bitrate. By using the painted image only in regions where it improves quality, we can reduce the amount of ringing without causing additional artifacts.

The painting algorithm works in three steps:

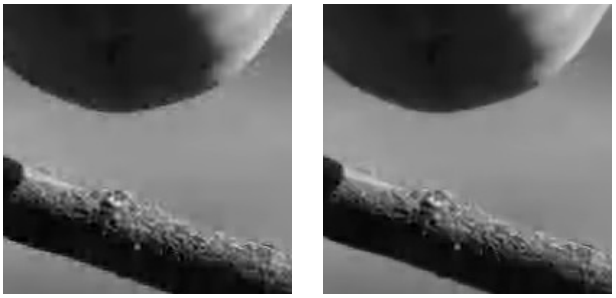
1) **Direction search**: Determine which direction best matches the pattern in each 8x8 block of the quantized image. The direction does not need to be signaled since the decoder also computes it on the quantized image.
2) **Boundary pixels**: Determine the pixel values at block boundaries that optimally match the image using the directions found in step 1. Again, no signaling is required.
3) **Painting**: For each block, use all four boundaries as well as the direction to paint the pixels inside the block. The boundary pixels computed in step 2 are used. Block discontinuities are avoided by blending within blocks based on the distance to each boundary.

Chosing which pixels should be replaced with the painted pixels and which ones are best unmodified is done using a gain derived from a Wiener filter. Intuitively, we see that in regions of the image that have clear directional patterns, the painted image should be much closer to the decoded image than in regions with unpredictable texture. Also, knowing the quality at which the image was coded, we can estimate the amount of quantization noise that was introduced, which is also the magnitude of the difference we can expect between decoded image and its painted version. The choice is made using the following equation for the optimal weight to apply to the painted image:

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

where Q is the quantization step size, σ^2 is the mean squared distance between decoded image and the painted

[3]It is not strictly necessary to code a bit for f. Instead the parameter α_{AC} could be found using least-squares regression and the sign extracted. However, using a single bit to code f is (1) better overall than relying on least-squares regression which can be wrong and (2) reduces the complexity significantly.



(a) (b)

Fig. 7. Daala luma plane (a) before and (b) after applying paint deringing.

image, and α is a tunable parameter between 0 and 1. For each boundary pixel, we compute a weight along the direction of each adjacent block. This gives us more control than computing the weight at the block level. There are still cases where we might be wrong and where our deringing would hurt image quality by blurring texture. Because of this, there is an additional per-superblock gain α transmitted to the decoder. It is the only information transmitted for the deringing filter.

III. Experimental Evaluation

To compare the Daala intra frame coding to other still image formats a small experimental study was conducted using the 8 mandatory images provided as part of the Image Compression Evaluation feature session at PCS 2015 [11]. The original 2 mega-pixel 8-bit RGB images were converted to 8-bit YUV and downsampled to 4:2:0. The resulting images were fed through Daala and 3 other lossy image codecs at a wide variety of quality factors. A graph of the rate-distortion curves for luma PSNR and luma Fast MS-SSIM can be found in Figures 8 and 9 respectively.

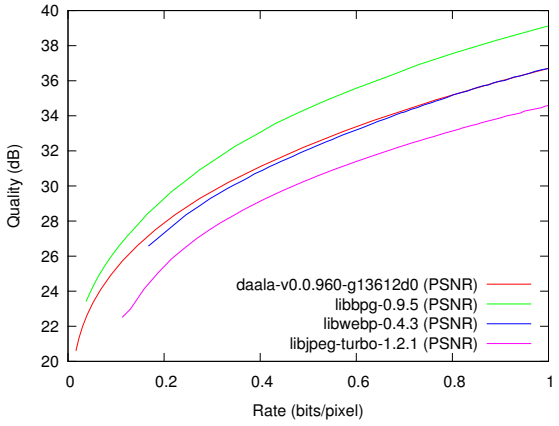


Fig. 8. Rate-Distortion comparison using luma PSNR.

IV. Future Work

The Daala video codec is under active development [12] and there are clearly features missing that are needed to produce a competitive still picture format. Most obviously is the lack of

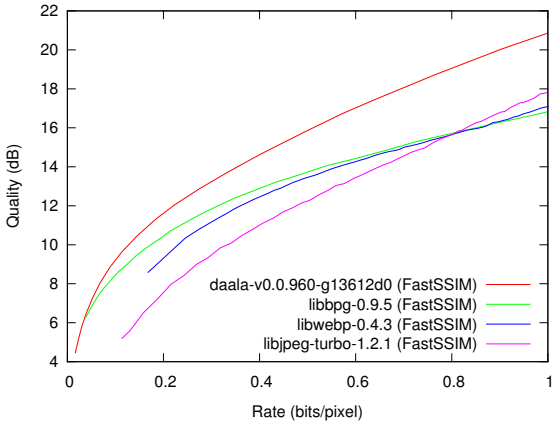


Fig. 9. Rate-distortion comparison using luma Fast MS-SSIM.

support for color encoding schemes and metadata. The Daala roadmap [13] includes adding support for 4:2:2 subsampling, monochrome (luminance only) input, alpha plane support and for lossless RGB using YCgCo.

Other highly requested features would require more fundamental changes to Daala. The lapped transforms only allow up to 12-bit input per channel and a separate design would be needed for higher bit-depth images. There is no support for depth channels, no support for high dynamic range content, no way to specify a region of interest for rate allocation, and no spatial scalability.

Daala is intended for use in real-time and streaming video applications and some design choices, e.g., memory footprint, encoder/decoder performance, etc. are not optimal for single frame encoding. However, many of these requirements can be relaxed for still image coding, which could yield better efficiency at the cost of speed.

References

[1] "WebP website," https://developers.google.com/speed/webp/.
[2] "BPG website," http://bellard.org/bpg/.
[3] M.-J. Chen and A. C. Bovik, "Fast structural similarity index algorithm," in *Proc. ICASSP*, march 2010, pp. 994–997.
[4] H. S. Malvar and D. H. Staelin, "The LOT: Transform coding without blocking effects." *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 37, no. 4, pp. 553–559, 1989.
[5] T. D. Tran, J. Liang, and C. Tu, "Lapped transform via time-domain pre- and post-filtering," *IEEE Transactions on Signal Processing*, vol. 51, no. 6, pp. 1557–1571, 2003.
[6] "Time domain lapped transforms for video coding," https://tools.ietf.org/html/draft-egge-videocodec-tdlt.
[7] T. R. Fischer, "A pyramid vector quantizer," *IEEE Trans. on Information Theory*, vol. 32, pp. 568–583, 1986.
[8] J.-M. Valin and T. B. Terriberry, "Perceptual vector quantization for video coding," vol. 9410, 2015.
[9] J. Kim, S. Park, Y. Choi, Y. Jeon, and B. Jeon, "New intra chroma prediction using inter-channel correlation," no. JCTVC-B021. Geneva, Switzerland, 2nd meeting: Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T VCEG and ISO/IEC MPEG, July 2010.
[10] N. E. Egge and J.-M. Valin, "Predicting chroma from luma with frequency domain intra prediction," vol. 9410, 2015, pp. 941 008–941 008–10.
[11] "Picture Coding Symposium (2015) website," http://www.pcs2015.org/.
[12] "Daala git repository," https://git.xiph.org/daala.git.
[13] "Daala roadmap," https://wiki.xiph.org/DaalaRoadmap.