Volume 0 (1981), Number 0 pp. 1–11

Practical HDR Texture Compression

Jacob Munkberg Petrik Clarberg Jon Hasselgren Tomas Akenine-Möller

Lund University

Abstract

The use of high dynamic range (HDR) textures in real-time graphics applications can increase realism and provide a more vivid experience. However, the increased bandwidth and storage requirements for uncompressed HDR data can become a major bottleneck. Hence, several recent algorithms for HDR texture compression have been proposed. In this paper, we discuss several practical issues one has to confront in order to develop and implement HDR texture compression schemes. These include improved texture filtering and efficient offline compression. For compression, we describe how Procrustes analysis can be used to quickly match a predefined template shape against chrominance data. To reduce the cost of HDR texture filtering, we perform filtering prior to the color transformation, and use a simple trick to reduce the incurred errors. We also introduce a number of novel compression modes, which can be combined with existing compression schemes, or used on their own.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Texture; I.4.2 [Image Processing and Computer Vision]: Compression (Coding); E.4 [Coding and Information Theory]: Data compaction and compression

1. Introduction

A general trend in computer architecture is that computing power growth is much faster than the corresponding growth in memory access speeds [Owe05]. This implies that the available memory bandwidth should be regarded as a scarce resource and be exploited as best as possible. One way is to use different *compression* techniques to reduce the required memory bandwidth. For graphics processing units (GPUs), there are many types of compression, such as buffer compression (e.g., depth, color, and stencil), vertex compression, and texture compression.

The focus in this paper is on *texture compression* (TC), where a texture is simply a read-only image. A texture is stored in compressed form in memory, and when the GPU requests access to a small part of the texture, the desired part is sent in compressed form over the bus. The GPU then decompresses the data. The pixels in a texture can be accessed in any order, any number of times, and hence it is of uttermost importance to provide random access in constant time. This has a number of implications, including that the majority of TC schemes operate on blocks of pixels (e.g., 4×4), and compress such a block to a fixed number of bits (e.g., 4 bits per pixel). In general, this also means that TC schemes are *lossy*.

High dynamic range (HDR) images [RWPD05] have had a big impact on the computer graphics community. Lately, these are used as textures in real-time rendering as well, and therefore, methods for HDR texture compression [MCHAM06, RAI06, WWS*07] have been developed. Roimela et al. [RAI06] try to minimize the hardware cost, and propose a simplified color space. They subsample chrominance, and use a quick floating-point trick to convert to approximately logarithmic luminance. In a recent paper [RAI08], they extend the algorithm with a more compact chrominance encoding and higher luminance fidelity. Munkberg et al. [MCHAM06] also develop a new compression algorithm, which uses an S3TC-like [INH99] encoding for the luminance, and introduce *shape transforms* for flexible chrominance encoding. All of these methods compress 4×4 pixel blocks down to eight bits per pixel (bpp). Wang et al. [WWS*07] propose an algorithm based on existing texture compression hardware, rather than targeting new hardware mechanisms, but only achieve a compression rate of 16 bpp. Some details of these schemes will be discussed in later sections.

In this paper, we bring to light several practical issues when dealing with HDR texture compression. To evaluate the quality of a compression algorithm, an error metric has to be used. However, standard error metrics are not suitable for HDR data due to the larger dynamic range and higher precision of floating-point numbers. A few HDR error metrics have been proposed, although more work needs to be done in this area. Section 2 gives a brief review of existing metrics. Similarly, color spaces must be defined with HDR data in mind so that efficient compression is achieved, which often means that a non-linear color space is preferred. In Section 3, we review the color spaces currently used for HDR texture compression. The use of non-linear color spaces complicates texture filtering, as each fragment must be converted to RGB-space prior to filtering for correct results. In Section 4, we introduce a novel algorithm for texture filtering in non-linear color spaces. This is important as it reduces the cost of hardware filtering.

To improve the compression quality, Section 5 introduces a new algorithm with better quality than previous algorithms at 8 bpp. Our algorithm builds on the method of Munkberg et al. [MCHAM06], but adds a novel compression mode focused on improving the chrominance precision. In addition, several other ideas on HDR texture compression algorithms are described in Section 6. The implementation of our codec is discussed in Section 7, where the details of how we use Procrustes analysis and clustering are presented for the first time. Furthermore, we use a non-linear optimization trick for improved chrominance quality and we analyze the desired precision for luminance. We compare all existing algorithms, including our new HDR TC scheme in Section 8, and finally we offer some conclusions in Section 9.

2. HDR Error Metrics

When you cannot foresee or predict the usage of an HDR texture, you need to make sure the accuracy of every value, regardless of its absolute magnitude, is preserved as well as possible. For example, a very dark region can become bright and details can appear after tone-mapping. Similarly, the details of a very bright region can become visible if the exposure is low. Standard LDR error metrics are therefore not suitable. In this paper, we use three HDR error metrics: the logarithmic error, multi-exposure PSNR, and HDR-VDP.

Xu et al. [XPH05] compute the *root mean square error* (RMSE) of the compressed image in the log[RGB] color space. More formally, if $(\hat{r},\hat{g},\hat{b})$ denotes the compressed texel color and (r,g,b) is the original color, the error is defined as:

$$\sqrt{\frac{1}{N}\sum\left(\log_2\left(\frac{\hat{r}}{r}\right)\right)^2+\left(\log_2\left(\frac{\hat{g}}{g}\right)\right)^2+\left(\log_2\left(\frac{\hat{b}}{b}\right)\right)^2}, \tag{1}$$

where N is the number of pixels in the texture.

The *multi-exposure PSNR* (mPSNR) error measure computes the standard mean square error for a range of tone-mapped exposures of the HDR image, and averages them together. Then the standard formula for computing peak signal-to-noise ratio (PSNR) is applied. For details, see Munkberg et al. [MCHAM06]. Roimela et al. [RAI08] compute the PSNR on the L^* and a^*b^* channels in the CIE 1976 $L^*a^*b^*$ color space with the motivation that it is perceptually linear.

HDR-VDP [MDMS05] is an extension of the *visual difference predictor*, which finds perceived differences over the dynamic range of the image. The current implementation only works on the luminance channel, so perceived chrominance artifacts are not detected.

3. HDR Color Spaces

In this section, we will present the color spaces used in previous HDR texture compression schemes, and shortly discuss their characteristics.

3.1. LogYuv

Munkberg et al. [MCHAM06] use a *logYuv* color space with logarithmic luminance and two chrominance channels:

$$\begin{aligned} Y &= w_r R + w_g G + w_b B \\ (\bar{Y},\bar{u},\bar{v}) &= \left(\log_2 Y, w_b\frac{B}{Y}, w_r\frac{R}{Y}\right) \\ (R,G,B) &= \left(\frac{1}{w_r}\bar{v}2^{\bar{Y}}, \frac{1}{w_g}(1-\bar{u}-\bar{v})2^{\bar{Y}}, \frac{1}{w_b}\bar{u}2^{\bar{Y}}\right) \end{aligned} \tag{2}$$

The luminance channel is encoded as a weighted combination of the RGB channels, with weights according to the Rec. 601 [Poy03] standard. By storing log-luminance, $\bar{Y}=\log_2 Y$, the maximum *relative* luminance error can be effectively bounded over the entire dynamic range. The two chrominance channels are constructed to be in the $[0,1]$ interval. This gives a decorrelated and compact representation of HDR texture data, with HDR information concentrated to the luminance channel. This color space is used in the proposed format in Section 5 of this paper.

A minor issue with logarithm-based luminance encodings is that entirely black pixels, RGB=(0,0,0), need to be treated with care, as $\log x \to -\infty$ when $x \to 0^+$ A simple solution is to clamp the log-luminance to the smallest representable value, $\bar{Y}_{\min}$. However, in some cases it may be desirable to have a true $(0,0,0)$ black level. A possible solution, which we use, is to define the inverse luminance transform as:

$$Y = \begin{cases} 0, & \text{if } \bar{Y}=\bar{Y}_{\min}, \\ 2^{\bar{Y}}, & \text{otherwise.} \end{cases} \tag{3}$$

However, it should be noted that true black pixels rarely occur in natural images, and only sometimes in artificial images.

3.2. Roimela et al.

Roimela et al. [RAI06] use a different transform in order to provide very efficient decoding. The forward and inverse transforms are shown below:

$$\begin{aligned} \tilde{Y} &= \frac{R+2G+B}{4} \\ (\tilde{Y},\tilde{u},\tilde{v}) &= \left(\tilde{Y},\frac{R}{4\tilde{Y}},\frac{B}{4\tilde{Y}}\right) \\ (R,G,B) &= \tilde{Y}(4\tilde{u},2(1-\tilde{u}-\tilde{v}),4\tilde{v}) \end{aligned} \tag{4}$$

Compared with logYuv of Munkberg et al. [MCHAM06], the luminance is here encoded with simplified weights with hardware-friendly constants. Again, the color space decorrelates the HDR data in one luminance and two chrominance channels.

3.3. LUVW

Wang et al. [WWS*07] use a color space called *LUVW* with four channels, where the luminance information is concentrated to the L channel as follows:

$$\begin{aligned} L &= \sqrt{R^2+G^2+B^2} \\ (U,V,W) &= \left(\frac{R}{L},\frac{G}{L},\frac{B}{L}\right) \end{aligned} \tag{5}$$

The L channel is the length of the RGB vector, and the UVW channels are divided by L to get three values in the range $[0,1]$. This is not as compact as the previous approaches, as four channels are stored, but allows for simplified texture filtering on existing hardware.

3.4. Log[RGB]

The log[RGB] color space [XPH05] is simply defined as the logarithm of the R, G and B components:

$$(R',G',B') = (\log_2 R, \log_2 G, \log_2 B) \tag{6}$$

The motivation is that quantization gives a constant, or nearly constant, relative error over the entire dynamic range. This makes it a good choice for measuring the error in each color channel. However, this transform often fails to decorrelate the image data, so all channels needs to be stored with equal resolution, which makes it less suitable for compression algorithms.

4. Texture Filtering

Texture filtering is crucial in real-time graphics, and all modern GPUs support fast bilinear and trilinear filtering. As the most widely used color space on GPUs is RGB, a filtered texture lookup is expected to perform a linear weighting of each of the R, G, and B channels. In the one-dimensional case, we have:

$$r = (1-\alpha)r_1 + \alpha r_2, \tag{7}$$

for linear interpolation between the red components, r_1 and r_2, of two texels (similarly for green and blue).

For compressed HDR textures using a non-RGB space, texture filtering can be performed before *or* after converting to RGB. In post-conversion filtering, we have to perform multiple color transforms for each filtered lookup, e.g., four $\bar{Y}\bar{u}\bar{v} \rightarrow RGB$ transforms in a bilinear two-dimensional lookup, which may harm performance, or increase the cost in terms of extra hardware for duplicated color space transform units. However, filtering before conversion gives deviating results, as the currently used HDR color spaces (Section 3) involve a non-linear transform, i.e., division by the luminance. According to Wang et al. [WWS*07], this is a minor issue as most blocks have a small dynamic range. However, in areas with sharp luminance transitions, we have observed clearly visible artifacts (color shifts). See Figure 2 for an example. We propose a simple way to reduce the problem.

Consider the logYuv color space in Section 3.1. The red channel is reconstructed as $r = \bar{v}Y/w_r$, and correct post-transform filtering yields:

$$r = (1-\alpha)r_1 + \alpha r_2 = \frac{1}{w_r}\left[(1-\alpha)\bar{v}_1 Y_1 + \alpha\bar{v}_2 Y_2\right]. \tag{8}$$

If we instead interpolate the Y, $\bar{u}$, and $\bar{v}$ components prior to the color transform, i.e., $Y = (1-\alpha)Y_1 + \alpha Y_2$ and similarly for $\bar{u}$ and $\bar{v}$, we get:

$$\begin{aligned} r = \frac{1}{w_r}\bar{v}Y &= \frac{1}{w_r}\left((1-\alpha)\bar{v}_1 + \alpha\bar{v}_2\right)\left((1-\alpha)Y_1 + \alpha Y_2\right) \\ &= \frac{1}{w_r}\left[(1-\alpha)\bar{v}_1 Y_1 + \alpha\bar{v}_2 Y_2\right] + \varepsilon, \end{aligned} \tag{9}$$

where the error term, ε, depends on the difference between the two luminance values, $\Delta Y = Y_2 - Y_1$, as follows:

$$\varepsilon = \frac{1}{w_r}\alpha(\alpha-1)(\bar{v}_2 - \bar{v}_1)\Delta Y. \tag{10}$$

This error is due to the non-linearity of the color transform. However, we note that the error goes to zero as ΔY gets smaller. We can exploit this to improve the interpolation quality by normalizing the luminance values so that $Y_1 \approx Y_2$. By shifting the bits of Y_2 to make it roughly the same magnitude as Y_1, we drastically reduce the error. At the same time, we must also shift the bits of $\bar{v}_2$ in the opposite direction to keep the term $\bar{v}_2 Y_2$ constant. The net effect is a change of variables to $Y_2' = Y_2/c$ and $\bar{v}_2' = c\bar{v}_2$, where $c = 2^k$. The number of bits to shift, k, is chosen by comparing the magnitudes of Y_1 and Y_2.

Consider the example of interpolating between two texels with $\bar{v}_1 = 0.25$, $\bar{v}_2 = 0.1$, $Y_1 = 1$, and Y_2 varying between 0.1 and 10. The absolute error in the red component with α arbitrarily set to 0.5, grows from roughly -0.1 to over 1.1 as the luminance Y_2 increases. This is shown as the blue line in Figure 1. With our luminance normalization, the error is much smaller (red line).

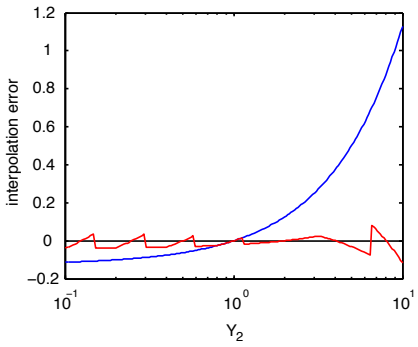

Figure 1: *Example of the interpolation error (blue line) introduced by performing texture filtering in the log-luminance color space prior to color transformation. By normalizing the luminance values with simple bit shifts, we effectively reduce the error (red line).*

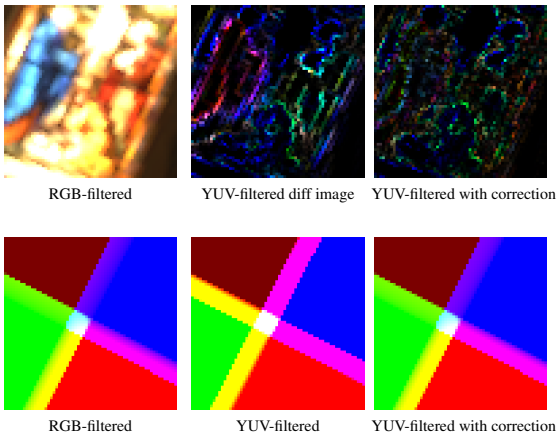

Figure 2: *Bilinear filtering examples. The upper row shows a filtered cutout from the 'memorial' image with difference images to highlight the errors. In natural images, pixel-sharp edges are rare, and filtering artifacts are not very visually disturbing. In the example, there are no clearly visible differences, so we present only difference images ($\times 10$) for this example. The lower row shows a worst-case bilinear filtering scenario (after tone-mapping) where there are sharp color and luminance transitions. Here the artifacts are clearly visible. The four colors are (0.1,0,0), (0,10,0), (0,0,1000) and (1,0,0). As can be seen, our texture filtering correction reduces the errors for both examples.*

With our color transform (Section 3.1), the green component is computed by subtraction of the two chrominance channels: $G = (1 - \bar{u} - \bar{v})Y/w_g$. This causes a minor difficulty, as this formula is no longer valid if we rescale Y, $\bar{u}$, and $\bar{v}$. One solution is to replace the constant 1 with the bit shift factor, $c = 2^k$, in the transform, i.e., $G = (c - \bar{u}' - \bar{v}')Y'/w_g$. Note, as c may differ for the two pixels, we need to interpolate its value: $c = (1-\alpha)c_1 + \alpha c_2$. This adds a small cost, but the savings compared to performing a full color transform for all pixels prior to filtering should be significant. In this discussion, we have studied the one-dimensional case, but the theory extends naturally to bilinear and trilinear filtering, as well as higher dimensions. We show comparison images with and without this pre-filter correction in Figure 2. These images were first compressed using our algorithm and then bilinearly filtered with and without the texture filtering correction discussed above.

5. Improved Shape Transform Compression

In this section, we extend the HDR compression algorithm based on shape transforms [MCHAM06] (see also Section 7.1) to allow for more freedom in the chrominance representation. The original algorithm quantizes the chrominance information aggressively to allocate enough space for a high quality luminance encoding. The ratio between bits spent on luminance and chrominance is 5 : 3. In most cases, this is a preferred bit layout, as luminance artifacts are more visually disturbing. However, in regions with smaller luminance range but large chrominance variations, a different bit layout may be more appropriate.

In the original encoder, each luminance value was encoded with 4 bits selecting a value linearly interpolated between two 8-bit end points, resulting in 80 bits for a 4×4 block of pixels ($16 \times 4 + 2 \times 8 = 80$). Reducing the luminance resolution to 3 bits, we can encode the luminance for a 4×4 block in $16 \times 3 + 16 = 64$ bits. On a 128-bit budget, this leaves 64 bits to encode chrominance, resulting in a ratio 1 : 1 between luminance and chrominance. Figure 3 shows the bit layout of the two modes. In the original paper, one source of artifacts is the chrominance subsampling, forcing nearby texels to have the same encoded chrominance. With an increased chrominance bit budget, we avoid subsampling and allow for higher chrominance detail.

By combining these two modes, we have an algorithm that is more flexible and handles difficult chrominance regions better than before. In practice, we need one bit for indicating which of these two modes is used per block. We remove the option of non-linear luminance distribution from the format [MCHAM06], and use that bit. The hardware cost of our algorithm is very modest as many parts of the decoder can be shared between the two modes. It is mostly a matter of reallocating the existing resources required for the original algorithm. Our new format is also more suitable for LDR textures, as they have a limited luminance range. A standard test suite of images is evaluated in Figure 10.

A remaining difficult case for the combined algorithm is slow chrominance gradients. Each block has only four chrominance values to choose from, resulting in a limited color resolution. It can be argued that most gradients in natural images are due to luminance changes from

shadow/occlusion and these are covered by the high luminance resolution of the algorithm. However, one can construct examples where the limited chrominance resolution is obvious. To handle these cases, one approach is to include a high resolution variant of S3TC, using a line in RGB or logYuv space with higher end point resolution and more values along the line. This is further described below, but this mode is not included in our results presented in Section 8.

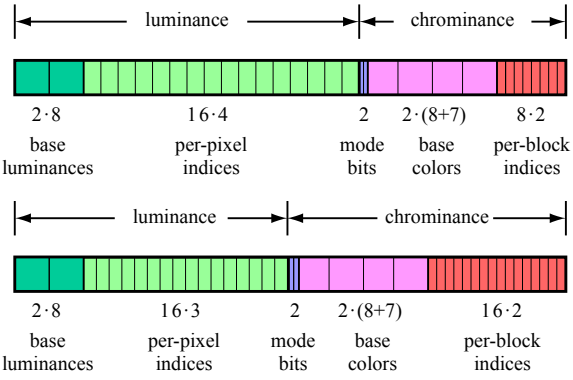

Figure 3: *The bit allocation for the two modes included in our new format. The upper figure shows the bit layout used by Munkberg et al. [MCHAM06], and the lower shows the bit layout for our new mode, allocating more bits for chrominance.*

6. Rejected Compression Modes

Our algorithm presented in Section 5 includes two compression modes based on shape transforms [MCHAM06], with different bit allocations between the luminance and chrominance channels. If more resources are available, it may be beneficial to include other compression modes in the format. In this section, we discuss a number of alternative compression modes that were tried during the development of Munkberg et al.'s format. Although not included in our current format, we believe these new modes present many valueable insights and may be of use to other people in the field.

During the encoding of a block, each of the compression modes is tested, and the one with the smallest error in the chosen metric (we use log[RGB] error) is selected. Hence, the inclusion of alternative compression modes can ideally only improve the result, never increase the error. This assumes we have one or more spare bits to indicate which compression mode a block uses. If no extra bits are available, we have to "steal" them from the encoded data, e.g., by quantizing harder, which may reduce the quality.

An illustration of the usage frequency of the presented additional modes is given in Figure 4. In total, the new modes are only used in on average 17% of the blocks. Thus, we do not believe the added hardware complexity is motivated by a large enough increase in quality.

6.1. Extended S3TC

It is easy to extend the S3 texture compression (S3TC) format [INH99] to handle HDR images. Traditional S3TC compresses a 4×4 pixel block by storing two reference colors with 16 bits (RGB565) each, and creates a color palette from the reference colors and two additional colors computed through linear interpolation. Each pixel in the block is given a 2-bit index, which is used to select one color from the palette. The format thus requires $16 \times 2 + 4 \times 4 \times 2 = 64$ bits per block.

We extend S3TC by using 2×24 bits (RGB888) for the reference colors and a color palette with 32 entries, computed using linear interpolation between the base colors just like traditional S3TC. As a consequence, a 5-bit index is needed for each pixel, resulting in $2 \times 24 + 16 \times 5 = 128$ bits per block, which was our target bit-rate. We also experimented with a palette of 16 colors and 2×32 bits for the reference colors.

Our extended S3TC works remarkably well for blocks with smooth gradients, and blocks with mostly luminance features. However, the linear approach fails for blocks with three or more distinct colors. This often shows as block artifacts in regions with complex chrominance.

6.2. Fixed-rate DCT-based Compression

The *discrete cosine transform* (DCT) is an energy compaction transform that is popular in image and video compression. It is, for example, used in the JPEG and MPEG standards [Poy03]. These formats use a variable bit rate to allow for a higher precision in important transform coefficients.

In our application, where we require a fixed rate and no global per-texture data, the best we can do is to design an algorithm that works well on average. We work in the logYuv space, and allocate 64 bits to the luminance channel and 32 bits to each of the chrominance channels. To select the bit allocation within each channel, we estimated the variance of each DCT coefficient over all 4×4 blocks in a set of 18 HDR images of both natural and synthetic origin. These estimated variances were used to find a bit allocation table that minimizes the average reconstruction error using Lagrange minimization [Den69, Say96].

The quantization method plays a vital role in the performance of the algorithm. For the DC-components, we used uniform quantization over the range of possible values, as we do not want to favor any particular luminance or chrominance range. Consistent with previous work [RG83, SR96], we found that the AC-components approximately follow a Laplacian distribution. Therefore, we applied non-uniform midtread quantization [Say96] optimized for the Laplacian distributions given by the estimated variances. The additional cost of using non-uniform quantization as opposed to

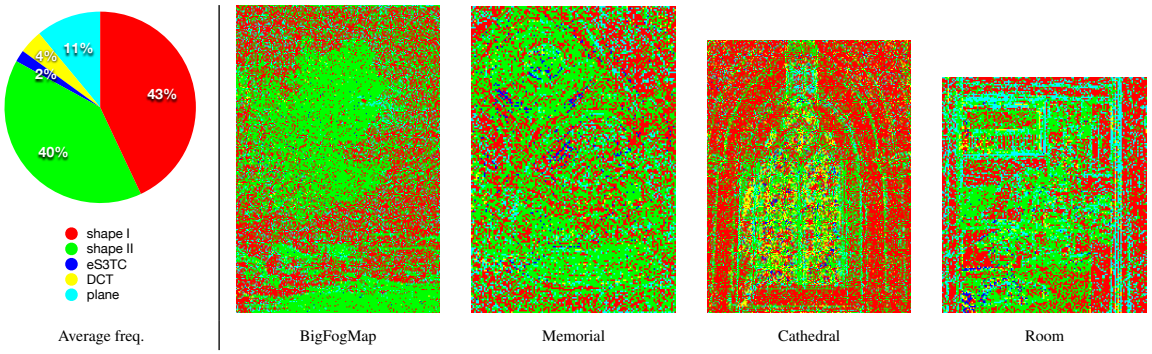

Figure 4: *The leftmost diagram shows the average usage frequency of each of our presented algorithms on our suite of test images.* ***shape I*** *refers to the new mode (Section 5) with increased chrominance precision,* ***shape II*** *to Munkberg et al. [MCHAM06],* ***S3TC*** *is an S3TC line with 32 levels in RGB space,* ***DCT*** *a fixed-rate DCT codec, and* ***plane*** *encodes luminance with two planes and chrominance as in the shape I mode. The color-coded images to the right illustrate the usage of the algorithms in different image regions. As seen in the figures, the proposed format (shape I + shape II) is used in more than 80% of the blocks, with the two modes complementing each other well.*

uniform quantization was well motivated by the increase in quality. In hardware, the reconstruction translates to simple table lookups.

The fixed-rate DCT-based approach works well for a large number of blocks. However, it has a number of drawbacks. First, block artifacts between adjacent blocks are relatively common. These look like typical JPEG-artifacts, and are rather disturbing. To some extent, this could be remedied by taking a larger neighborhood into account when computing the quantization levels. For blocks with a large dynamic range, it proved difficult to find quantization levels that work well. In addition, the decompression is relatively expensive. In summary, the best option is probably to limit the use of DCT-based compression to blocks with smooth gradients and other low-frequency features.

6.3. Plane Encoders for Luminance

The luminance values in a block can be seen as a height-field, $z = f(x,y)$, where x,y is the pixel coordinate of a point, and the z-value is the luminance. One option for encoding luminance is to store the equation of the plane that approximates the z-values as closely as possible: $f(x,y) = z_0 + x \cdot \Delta x + y \cdot \Delta y$, where z_0 is a constant offset, and Δx and Δy are the slopes. Finding the optimal plane parameters is a simple linear optimization problem. However, an encoder based on a single plane is rather restricted, as only linear luminance gradients can be represented.

A more general approach is to store *two* (or more) plane equations for each block, and a per-pixel index to choose between them. If more bits are available, it is also possible to add intermediate levels in between the planes. For example, a 2-bit per-pixel index can be used to select between the two planes and two additional levels at $1/3$ and $2/3$ between the planes, similar to the work by Fenney [Fen03].

We have implemented a simple plane encoder for luminance information, storing two plane equations with a 10-bit offset and two 7-bit deltas each. A 16-bit mask was used to select which of the two planes each pixel belongs to. This gives a total bit count of $2 \times (10+7+7)+16 = 64$ for the luminance encoder. To find the plane equations, we used exhaustive search and regression, but more intelligent methods can be developed. Our luminance plane encoder was combined with the chrominance encoding using shape transforms as described in Section 5.

Our experiments with the plane encoders yielded promising results. Many blocks contain relatively smooth gradients, and the two-plane encoder is good at handling the case of two partially overlapping surfaces of different luminance. However, areas with complex high-frequency features are poorly represented. The main drawback of the method is that the restriction of the luminance to linear planes can lead to visually noticeable block artifacts. Hence, the method must be combined with other, more flexible, encodings.

7. Implementation

In this section, we describe the details of the compression algorithms based on shape transforms (Section 5).

7.1. Shape Transforms

Shape transforms [MCHAM06] is a compact way of encoding the 2D chrominance points of a block. Each block stores a scaled and rotated template shape chosen out of the set of pre-defined shapes shown in Figure 5, and a 2-bit index is used to indicate the nearest chrominance point for each pixel or group of pixels. The transform parameters, i.e., scale and rotation, are implicitly encoded by storing the location of two base colors, marked as black dots in the figure.

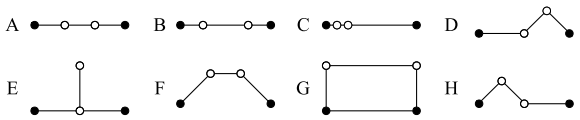

Figure 5: *The set of template shapes used for representing chrominance information of a block.*

To find the transform parameters that give the best match for the chrominance information, we have to minimize the shape fitting error. We use the squared distance between the original and the compressed chrominance points:

$$E = \sum_{i=1}^{16} \left[(\bar{u}_i - \bar{u}_i')^2 + (\bar{v}_i - \bar{v}_i')^2 \right], \tag{11}$$

where $(\bar{u}_i, \bar{v}_i)$ is the original chrominance point for pixel i, and $(\bar{u}_i', \bar{v}_i')$ is the corresponding point after compression. The error is evaluated for the best fit of each template shape, and we select the shape with the smallest overall error.

In practice, minimizing the shape fitting error (Equation 11) is a relatively hard problem. The error function has four degrees of freedom (the location of two base colors), and 16 different $(\bar{u}_i, \bar{v}_i)$-pairs for a 4×4 block, that each snaps to the nearest compressed chrominance point. Exhaustive search is possible, but impractical. If the base colors are quantized to 8 bits per component, there are 2^{32} combinations to try for *each* block. For a quick approximate solution, it is possible to use *Procrustes analysis*, which is further described in the next section. Other alternatives we have tried include *simulated annealing* and *exhaustive search* (see below).

After shape fitting, the position of the two base colors are encoded as two fixed-point values. The limited precision introduces some additional compression error. To further improve the solution, we search in a small radius of quantized values around each base point to minimize the mean square error for the reconstructed chrominance block.

Procrustes Analysis and Clustering

A landmark is a specific feature of an object, in our case represented as 2D coordinates. The idea behind Procrustes analysis [DM98] is to compare the shapes of objects, represented as sets of landmarks, by removing translation, rotation and scaling. More formally, this analysis finds the similarity transformations to be applied to one set of landmarks, X_1 (template shape coordinates), which minimize its Euclidean distance from a second set, X_2 (chrominance values). These are: b (uniform scaling), $\mathbf{R}$ (rotation) and $\mathbf{v}$ (translation), which minimize the functional:

$$\|X_2 - bX_1\mathbf{R} - \mathbf{1}_k\mathbf{v}^T\|^2. \tag{12}$$

The problem of finding the parameters that minimize this functional has an exact, fast solution: First, center X_1 and X_2 by subtracting the average from each coordinate. $\mathbf{v}$ is given as the average of X_2 prior to centering. Form the matrix $\mathbf{A} = {X_2}^T X_1$, and apply a singular value decomposition $\mathbf{A} = \mathbf{VSU}^T$. The transform parameters that minimize the functional above are given by (where trace is the sum of the diagonal elements of a square matrix):

$$\mathbf{R} = \mathbf{UV}^T,$$
$$b = \frac{\text{trace}(\mathbf{AR})}{\text{trace}({X_1}^T X_1)}. \tag{13}$$

With 2D points, the matrix $\mathbf{A}$ is of size 2×2, so the SVD decomposition is lightweight.

We use blocks of 4×4 texels, containing 16 chrominance points, so the problem is to fit a shape with four landmarks to 16 chrominance points $(\bar{u}, \bar{v}) \in [0,1]^2$. To set up the point correspondences, we create up to four clusters of the chrominance points using *pnn*-clustering and the *k-means* algorithm [Say96]. Clustering the points is an approximation of the optimal solution to the fitting problem, but it allows us to compare blocks against template shapes in constant time.

Procrustes analysis needs consistent ordering of the two sets of landmark points. Therefore, each cluster is linked to a point on the template shape. With four landmarks per shape, the number of unique mappings is $4! = 24$, and we test all combinations. It is worth noting that for a problem with a larger set of landmarks per shape, this is obviously not a feasible approach. Numbering schemes based on the template shape geometry can be developed to avoid this brute-force solution.

Once the point correspondences are set up, each cluster $k \in \{1 \ldots 4\}$, is assigned a gravity point and a weight $w_k = n_k/16$, where n_k is the number of points in cluster k. In order to take the number of points per cluster into account, we multiply the functional above with a diagonal matrix:

$$W = \begin{bmatrix} w_1 & 0 & 0 & 0 \\ 0 & w_2 & 0 & 0 \\ 0 & 0 & w_3 & 0 \\ 0 & 0 & 0 & w_4 \end{bmatrix}, \tag{14}$$

containing the cluster weights w_k. Our new functional is:

$$\|\mathbf{W}(X_2 - bX_1\mathbf{R} - \mathbf{1}_k\mathbf{v}^T)\|^2, \tag{15}$$

which favors solutions with close fits for clusters containing many points. Another possibility is to duplicate template points according to the number of points in the matching cluster, avoiding the need of cluster weights altogether. We have evaluated both approaches, and they give equal quality in our tests. We use the former approach in our implementation as it is slightly faster.

The shape fitting routines were implemented in C++. The Procrustes step is fast, as we are only interested in a 2D fit, and the most complex step is the singular value decomposition of a 2×2 matrix. As previously discussed, we need to set up point correspondences in order to use Procrustes

analysis, and we tested both ordering schemes and a brute-force solution. The latter was selected as the many special cases of the former made it more error-prone and inflexible when adding new template shapes. A 1024×1024 image is encoded in about a minute, using non-optimized code. The same image is decompressed in a fraction of a second using our software decompressor.

Simulated Annealing and Exhaustive Search

Our shape fitting algorithm based on Procrustes analysis and clustering works very well in practice, but it should be noted that it is a heuristic and is not guaranteed to yield optimal results. We have also examined two other approaches for finding shape parameters, based on simulated annealing and exhaustive search.

Simulated annealing (SA) [KGV83] is a probabilistic optimization algorithm that finds some minimum (not necessarily the global minimum) through a series of small random steps of decreasing length. Unlike greedier methods, spurious "uphill" moves are allowed, which makes the algorithm less prone to getting stuck at local minima. In our case, we wish to minimize the four-dimensional function $f(u_1, v_1, u_2, v_2)$, which takes the two base colors used to represent a shape as input and computes the error according to Equation 11.

When compared to Procrustes analysis, we found that SA is prone to generate a few bad blocks per image due to its probabilistic nature. This problem can be reduced by running more iterations for blocks with higher errors, but it is very hard to reach the same level of stability and performance as we got with Procrustes analysis. SA can potentially generate better shape fits, but only if a very large number of iterations are used, leading to excessive compression times.

Our exhaustive search used the same functional representation as for simulated annealing, but here we redefine the problem using interval arithmetic [Moo66]. That is, the function takes intervals as parameters and computes interval bounds of the error. We search for the global minimum by evaluating the function over the entire search space, and then recursively split the search space into two halves. The traversal of splits are sorted by error intervals, and when we find a solution, it can be used to cull further traversal. This search is exhaustive since we continue the recursion until we reach the resolution used to store the base colors.

Although exhaustive search provides optimal results, it is not practically useful due to the extreme compression times. Even images of moderate sizes take over a day to compress. However, exhaustive search can be used as a reference solution, against which other optimization algorithms can be compared.

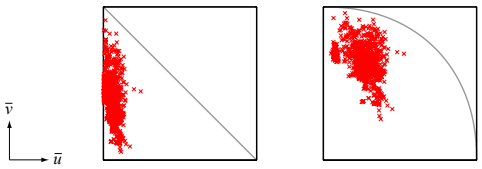

Figure 6: *The chrominance information in the test image 'desk'. The left image shows all $(\bar{u}, \bar{v})$ pairs in the image, and the right shows the same points after stretching the $(\bar{u}, \bar{v})$-space.*

7.2. Optimizing with a Non-linear Error Function

The shape transform algorithm enables a compact representation of chrominance information. However, the quantization of the base colors and the limited set of shapes introduce small errors. We have noticed that the $(\bar{u}, \bar{v})$ values in natural images are typically concentrated to a small region close to the $\bar{u}$ and $\bar{v}$ axes. Figure 6 (left) shows an example of this. Therefore, it is often better to use a non-linear error function that gives more weight to small chrominance values.

We employ a transform, $f(x) = x^{\alpha}$, to the $(\bar{u}, \bar{v})$ points prior to measuring the shape fitting error. By choosing the constant α in the range $[0, 1]$, we effectively "stretch" the $(\bar{u}, \bar{v})$ space, as illustrated in Figure 6 (right). The drawback is that errors in the green component, $1 - \bar{u} - \bar{v}$, get slightly smaller weights. We use a value $\alpha = 0.455$, which is inspired by the gamma-adjustment step typically included in tone-mapping operators. This works well in practice, but it should be noted that other transforms may perform better, and we have not performed extensive experiments with this.

Note that this trick of using a non-linear error function when minimizing the shape fitting error is only applied during the compression step. Hence, the shapes are stored exactly as before, and no modifications to the decompression hardware are needed. The drawback is slightly longer compression times. As seen in Table 1, the log[RGB] error decreases by 20% on average, compared to performing shape optimization using the original linear error function (Equation 11).

Image	Non-linear	Linear	Improvement (%)
bigFogMap	0.06	0.07	12
cathedral	0.17	0.21	18
memorial	0.13	0.18	27
room	0.08	0.09	11
desk	0.22	0.53	59
tubes	0.28	0.31	10

Table 1: *log[RGB] error comparison with non-linear* vs *linear error functions for the chrominance points.*

7.3. Luminance Precision

In this section, we discuss the necessary precision for encoding logarithmic luminance, as used in the color space defined in Section 3.1. Assume we work with values in the range $\bar{Y} \in [\bar{Y}_{\min}, \bar{Y}_{\max}]$, and uniformly quantize $\bar{Y}$ using k bits. Then, the maximum absolute quantization error, $|\Delta\bar{Y}|$, is:

$$|\Delta\bar{Y}| = \frac{1}{2}\frac{\bar{Y}_{\max} - \bar{Y}_{\min}}{2^k}. \tag{16}$$

In linear luminance values, this quantization error translates to a maximum relative luminance error, $|\Delta Y|$, equal to:

$$|\Delta Y| = \max\left|\frac{2^{\bar{Y}\pm|\Delta\bar{Y}|}}{2^{\bar{Y}}} - 1\right| = 2^{|\Delta\bar{Y}|} - 1 \tag{17}$$

As an example, consider the dynamic range supported by the 16-bit `half` type, which is approximately $[2^{-16}, 2^{16}]$. The maximum relative error after quantization to $k = 6 \ldots 16$ bits are presented in Figure 7. It is widely accepted that a relative luminance error of about 1% is the smallest visually detectable difference [Wan95]. Accordingly, a log-luminance precision of 10 bits (i.e., 1.09% relative error) should be sufficient. Mantiuk et al. [MKMS04] came to the same conclusion after a similar reasoning based on measured luminance threshold curves of the human visual system. Note that they use a perceptual quantization of luminance, so the results are not directly comparable.

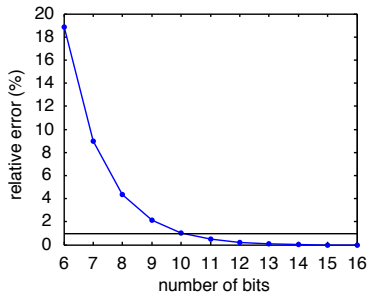

Figure 7: *Maximum relative luminance error with uniform quantization of the log-luminance.*

Munkberg et al. [MCHAM06] encode luminance values by quantizing the minimum and maximum log-luminance over a 4×4 block to 8 bits precision. Then, a 4-bit per-pixel index is used to choose between 16 intermediate luminance levels, uniformly distributed between the block's minimum and maximum. In practice, this gives a variable luminance precision, where blocks with a small dynamic range will have the best precision.

Figure 8 shows a histogram over the difference between the maximum and minimum log-luminance over all blocks in our test images (Figure 11). With our luminance encoding, the histogram shows that we get 10 bits precision or better for 51.0% of the blocks (maximum difference 0.5),

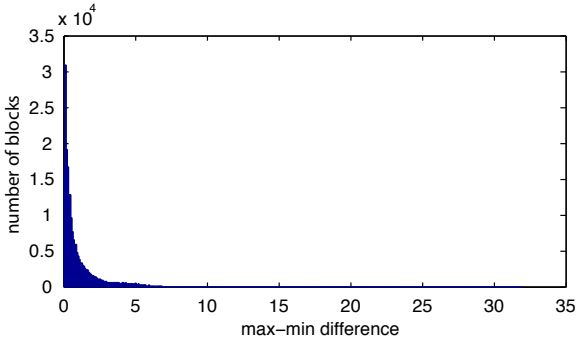

Figure 8: *Histogram over the difference between the smallest and the largest log-luminance values over all 4×4 blocks in our suite of test images (Figure 11).*

84.7% falls within 8 bits precision, and 99.8% within 6 bits. However, as described by Munkberg et al. [MCHAM06], we search in a small neighborhood around the end-points of the luminance range to minimize the error. In many cases, it is possible to find a combination that better fits the data, as illustrated in Figure 9.

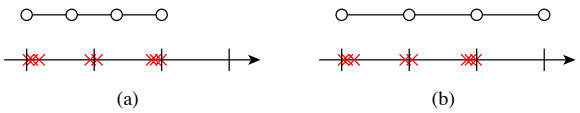

Figure 9: *By searching around the end-points of the luminance range, we can often find a better match (b) for the luminance values in a block, rather than just picking the min and max as in (a). The example shows a quantization to 4 distinct levels for clarity, although our format supports 16 different levels.*

8. Results

Here, we compare our combined compression mode (Section 5) against state-of-the-art HDR texture compression schemes. We also show that our algorithm works well on regular LDR textures.

8.1. Comparison with Other Approaches

We compare our algorithm with the recently published HDR texture compression formats [MCHAM06, RAI06, WWS*07]. We would like to point out that the approaches have different goals: Roimela et al.'s algorithm is very fast and designed for a simple hardware implementation, while Wang et al. present a format that can be directly implemented in DX9/10 without any hardware changes. Munkberg et al. focus on image quality, while keeping the

decompression hardware simple. Our algorithm extends this by including a chrominance mode without subsampling.

A set of six test images was used, as shown in Figure 11. One image ('tubes') is artificial, while the others are natural images commonly used in the research community. Table 2 shows the log[RGB] error (Section 2) for the test images. Our combined mode shows slightly better or equal results for all images. Not surprisingly, the largest improvement is in the 'tubes' image, which contains sharp chrominance transitions. The combined mode handles this better by avoiding chrominance subsampling. Table 3 shows that the mPSNR error (Section 2) follows a similar pattern.

Table 4 shows the HDR-VDP (Section 2) error at 75% detection probability at an adaptation luminance manually adjusted per image so that it is close to 300 cd/m^2. We found that the implementation of the HDR-VDP we used (v1.6) had problems with true zeros in images, indicating errors even in totally black areas, e.g., in the 'tubes' image. The results for this image are therefore overly conservative. Our algorithm has one mode with lower luminance resolution, and as the encoder selects the best mode based on the log[RGB] error, the HDR-VDP scores are somewhat higher than with Munkberg et al.'s algorithm. This is because HDR-VDP only measures perceived luminance, so chrominance artifacts are not captured.

image	**Our** 8 bpp	**Munkberg** 8 bpp	**Roimela** 8 bpp	**Wang** 16 bpp
bigFogMap	0.06	0.06	0.10	0.14
cathedral	0.17	0.20	0.33	0.36
memorial	0.13	0.14	0.26	0.69
room	0.08	0.09	0.23	0.71
desk	0.22	0.25	1.14	2.92
tubes	0.28	0.43	0.85	0.81

Table 2: *Log[RGB] error (smaller is better).*

image	**Our** 8 bpp	**Munkberg** 8 bpp	**Roimela** 8 bpp	**Wang** 16 bpp
bigFogMap	51.9	51.7	47.1	46.3
cathedral	40.0	38.9	34.2	35.8
memorial	46.5	46.1	41.3	38.0
room	48.6	48.1	41.6	34.1
desk	40.3	39.7	31.1	21.3
tubes	35.7	32.2	26.6	29.1

Table 3: *Multi-exposure PSNR (larger is better).*

A visual comparison is presented in Figure 11, where the compressed images are diagonally split, showing the squared log differences in the upper left triangle, and the compressed result in the lower right. As can be seen, the images obtained using Wang et al.'s algorithm (fourth column) often (1st, 2nd, 4th, and 5th row) have relatively large errors in the luminance channel. This can be seen in that the error images contain gray regions. In addition, there are also often larger chrominance errors than for the other algorithms (except for the last row, where the method of Roimela et al. seems to produce the largest errors). Our algorithm and that of Roimela et al. reproduce the luminance quite accurately. However, Roimela et al.'s subsampling strategy for the chrominance gives a higher error than our algorithm in all test images. We believe it is clear from these images that our algorithm is more robust and accurate than previous methods.

image	**Our** 8 bpp	**Munkberg** 8 bpp	**Roimela** 8 bpp	**Wang** 16 bpp
bigFogMap	0.00	0.00	0.01	7.18
cathedral	0.02	0.00	0.19	0.04
memorial	0.01	0.01	0.15	15.4
room	0.02	0.02	0.64	26.4
desk	0.03	0.00	2.58	4.34
tubes	1.59	0.66	3.35	2.00

Table 4: *HDR-VDP error with 75% detection probability at an adaptation luminance of 300 cd/m^2 (smaller is better). Note that the HDR-VDP only measures luminance errors, while our format improves the chrominance precision. Hence the higher scores.*

8.2. LDR Measures

We have also compared the quality of our algorithm on a set of standard low dynamic range (RGB888) images. Here we have used a standard RGB to YUV color transform [Poy03] instead of the HDR color spaces discussed earlier. All other parts of the algorithm were left unchanged. Figure 10 shows the results for a set of standard test images, and it is clear that our format performs significantly better than the industry standard (S3TC), though at a higher bit-rate.

9. Conclusion

The HDR texture compression algorithm by Wang et al. [WWS*07] focuses on reusing existing hardware for texture compression, and therefore arrive at an algorithm using 16 bits per pixel with rather low image quality. However, the algorithm can be used today on all DX9 hardware, which is a major advantage. The algorithm by Roimela et al. [RAI06] is a proposal for new hardware, and their focus was to provide very simple decompression hardware, and still the image quality is rather high.

In contrast, our focus has been to increase the image quality as much as possible, as we think this is very important for content creators. We have introduced a new mode for blocks of pixels with difficult chrominance, and combined that with the algorithm of Munkberg et al. [MCHAM06]. To make this usable, we provide an inexpensive texture filtering method.

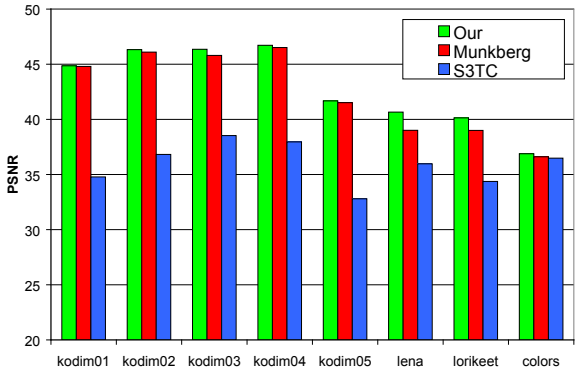

Figure 10: *PSNR for a set of standard 24-bit LDR images. We compare our algorithm against S3TC and Munkberg et al.'s algorithm. Our new format gives up to 10 dB improvement over S3TC, but note that S3TC uses only 4 bpp (compared to 8 bpp for the other formats). The 'colors' image consists entirely of color gradients along directed lines, which is well captured by S3TC.*

In addition, the details of our compressor using Procrustes analysis and k-means clustering have been described. We hope that all this information will be useful to many when developing new HDR TC schemes.

Acknowledgements

We acknowledge support from the Swedish Foundation for Strategic Research and Vetenskapsrådet. Thanks to Rafal Mantiuk for sharing the HDR-VDP implementation, Kimmo Roimela and Xi Wang for sharing their respective codecs.

References

[Den69] DENN M. M.: *Optimization by Variational Methods*. McGraw-Hill, 1969.

[DM98] DRYDEN I., MARDIA K.: *Statistical Shape Analysis*. Wiley, 1998.

[Fen03] FENNEY S.: Texture Compression using Low-Frequency Signal Modulation. In *Graphics Hardware* (2003), pp. 84–91.

[INH99] IOURCHA K., NAYAK K., HONG Z.: System and Method for Fixed-Rate Block-Based Image Compression with Inferred Pixel Values. US Patent 5,956,431, 1999.

[KGV83] KIRKPATRICK S., GELATT C. D., VECCHI M. P.: Optimization by Simulated Annealing. *Science 220, 4598* (1983), 671–680.

[MCHAM06] MUNKBERG J., CLARBERG P., HASSELGREN J., AKENINE-MÖLLER T.: High Dynamic Range Texture Compression for Graphics Hardware. *ACM Transactions on Graphics, 25*, 3 (2006), 698–706.

[MDMS05] MANTIUK R., DALY S., MYSZKOWSKI K., SEIDEL H.-P.: Predicting Visible Differences in High Dynamic Range Images – Model and its Calibration. In *Human Vision and Electronic Imaging X* (2005), pp. 204–214.

[MKMS04] MANTIUK R., KRAWCZYK G., MYSZKOWSKI K., SEIDEL H.-P.: Perception-Motivated High Dynamic Range Video Encoding. *ACM Transactions on Graphics, 23*, 3 (2004), 733–741.

[Moo66] MOORE R. E.: *Interval Analysis*. Prentice-Hall, 1966.

[Owe05] OWENS J. D.: Streaming Architectures and Technology Trends. In *GPU Gems 2*. Addison-Wesley, 2005, pp. 457–470.

[Poy03] POYNTON C.: *Digital Video and HDTV Algorithms and Interfaces*. Morgan Kaufmann, 2003.

[RAI06] ROIMELA K., AARNIO T., ITÄRANTA J.: High Dynamic Range Texture Compression. *ACM Transactions on Graphics, 25*, 3 (2006), 707–712.

[RAI08] ROIMELA K., AARNIO T., ITÄRANTA J.: Efficient High Dynamic Range Texture Compression. In *Proceedings of I3D* (2008), pp. 207–214.

[RG83] REININGER R. C., GIBSON J. D.: Distrubutions of the Two-Dimensional DCT Coefficients for Images. *IEEE Transactions on Communications 31*, 6 (1983), 835–839.

[RWPD05] REINHARD E., WARD G., PATTANAIK S., DEBEVEC P.: *High Dynamic Range Imaging: Acquisition, Display and Image-Based Lighting*. Morgan Kaufmann, 2005.

[Say96] SAYOOD K.: *Introduction to Data Compression*. Morgan Kaufmann, 1996.

[SR96] SMOOT S., ROWE L.: Study of DCT Coefficient Distributions. In *Proceedings of the SPIE Symposium on Electronic Imaging* (1996), vol. 2657, pp. 403–441.

[Wan95] WANDELL B.: *Foundations of Vision*. Sinauer Associates, 1995.

[WWS*07] WANG L., WANG X., SLOAN P.-P., WEI L.-Y., TONG X., GUO B.: Rendering from Compressed High Dynamic Range Textures on Programmable Graphics Hardware. In *Proceedings of I3D* (2007), pp. 17–24.

[XPH05] XU R., PATTANAIK S. N., HUGHES C. E.: High-Dynamic-Range Still-Image Encoding in JPEG 2000. *IEEE Computer Graphics and Applications, 25*, 6 (2005), 57–64.

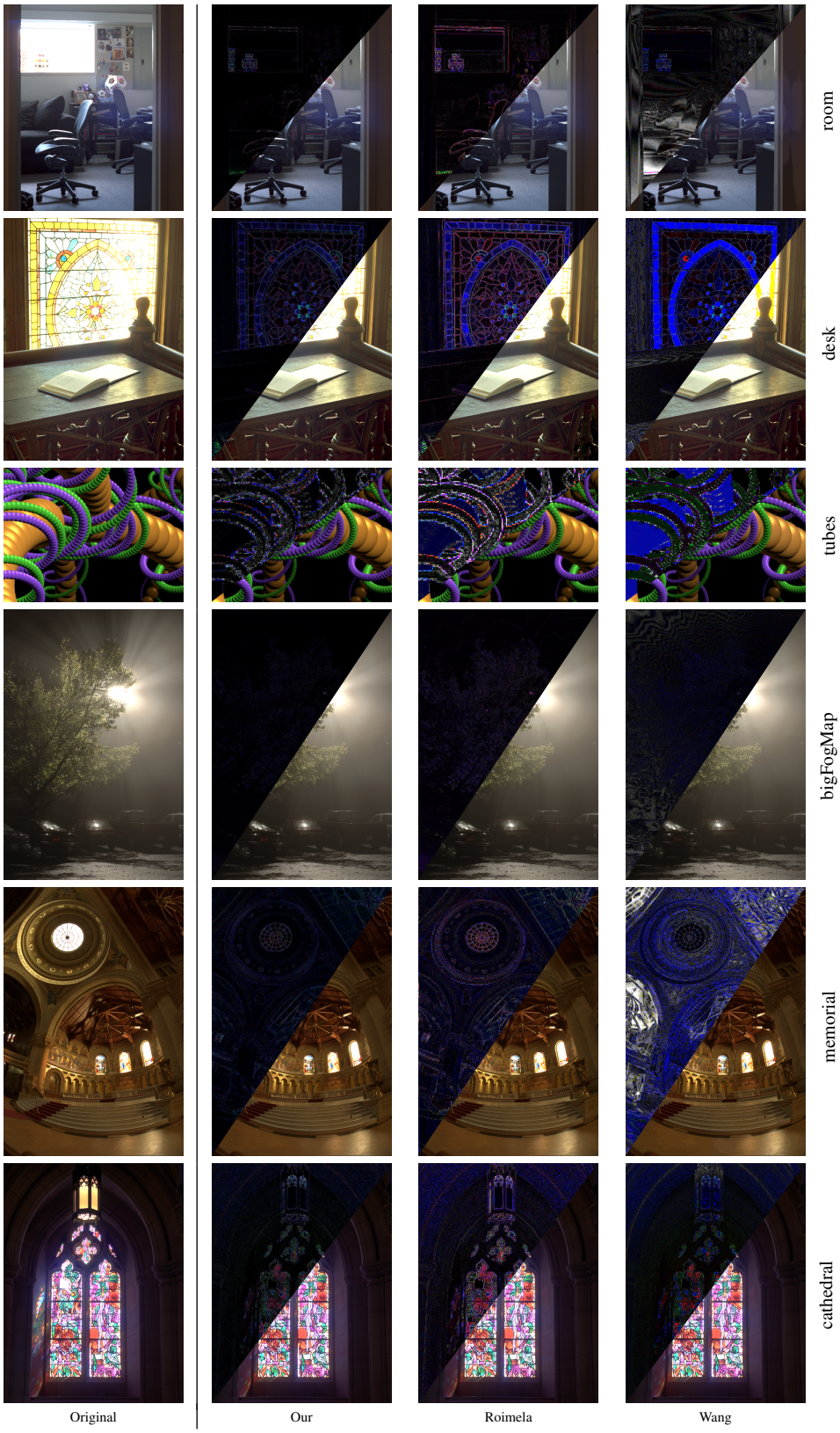

Figure 11: *Image comparison. The upper left triangles in the compressed images show the squared log differences.*