High Dynamic Range Texture Compression for Graphics Hardware

Jacob Munkberg* Petrik Clarberg Jon Hasselgren Tomas Akenine-Möller

Lund University

Abstract

In this paper, we break new ground by presenting algorithms for fixed-rate compression of high dynamic range textures at low bit rates. First, the S3TC low dynamic range texture compression scheme is extended in order to enable compression of HDR data. Second, we introduce a novel robust algorithm that offers superior image quality. Our algorithm can be efficiently implemented in hardware, and supports textures with a dynamic range of over 10^9:1. At a fixed rate of 8 bits per pixel, we obtain results virtually indistinguishable from uncompressed HDR textures at 48 bits per pixel. Our research can have a big impact on graphics hardware and real-time rendering, since HDR texturing suddenly becomes affordable.

CR Categories: I.3.1 [Computer Graphics]: Hardware Architecture—Graphics processors; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Texture; E.4 [Data]: Coding and Information Theory—Data compaction and compression;

Keywords: texture compression, image compression, high dynamic range images, graphics hardware

1 Introduction

The use of high dynamic range (HDR) images in rendering [Ward 1994; Debevec and Malik 1997; Debevec 1998; Reinhard et al. 2005] has changed computer graphics forever. Prior to this, only low dynamic range (LDR) images were used, usually storing 8 bits per color component, i.e., 24 bits per pixel (bpp) for RGB. Such images can only represent a limited amount of the information present in real scenes, where luminance values spanning many orders of magnitude are common. To accurately represent the full dynamic range of an HDR image, each color component can be stored as a 16-bit floating-point number. In this case, an uncompressed HDR RGB image needs 48 bpp.

In 2001, HDR images were first used in real-time rendering [Cohen et al. 2001], and over the past years, we have observed a rapidly increasing use of HDR images in this context. Game developers have embraced this relatively new technique, and several recent games use HDR images as textures. Examples include Unreal Engine 3, Far Cry, Project Gotham Racing 3, and Half-Life 2: Lost Coast.

*e-mail: {jacob | petrik | jon | tam}@cs.lth.se

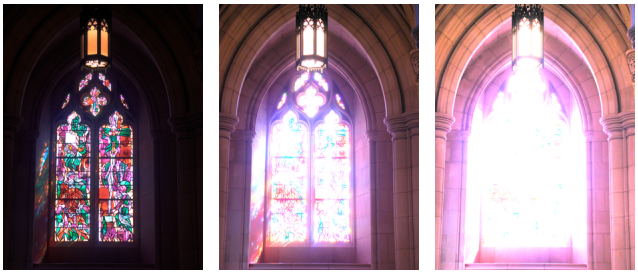


Figure 1: Example of a high dynamic range image, here shown at three different exposures, compressed with our algorithm to a fixed rate of 8 bits per pixel. Our algorithm gives excellent image quality over a large dynamic range, and is fast to decompress in hardware.

The disadvantage of using HDR textures in real-time graphics is that the texture bandwidth usage increases dramatically, which can easily limit performance. With anisotropic filtering or complex pixel shaders, it can become even worse. A common approach to reduce the problem is *texture compression*, introduced in 1996 [Knittel et al. 1996; Beers et al. 1996; Torborg and Kajiya 1996]. By storing textures in compressed form in external memory, and sending compressed data over the bus, the bandwidth is significantly reduced. The data is decompressed in real time using special-purpose hardware when it is accessed by the pixel shader. Several formats use as little as 4 bpp. Compared to 24 bpp RGB, such techniques can potentially reduce the texture bandwidth to only 16.7% of the original.

Texels in textures can be accessed in any order during rasterization. A fixed-rate texture compression (TC) system is desirable, as it allows random addressing without complex lookup mechanisms. Hence, JPEG and similar algorithms do not immediately qualify as reasonable alternatives for TC, since they use an adaptive bit rate over the image. The fixed bit rate also implies that all realistic TC algorithms are lossy. Other characteristics of a TC system are that the decompression should preferably be fast and relatively inexpensive to implement in hardware. However, we expect that increasingly complex decompression schemes can be accepted by the graphics hardware industry, since the available bandwidth grows at a much slower pace than the computing power [Owens 2005]. A difference between LDR and HDR TC is that for HDR images, we do not know in advance what range of luminance values will be displayed. Hence, the image quality must remain high over a much larger range of luminance.

We present novel HDR TC schemes, which are inexpensive to implement in hardware. Our algorithms compresses tiles of 4×4 pixels to only 8 bpp. The compressed images are of very high quality over the entire range of input values, and are essentially indistinguishable from uncompressed 48 bpp data. An example of a compressed image is shown in Figure 1.

2 Related Work

Here, we first present research in LDR texture compression (TC) for graphics hardware that is relevant to our work. For a more complete overview, consult Fenney's paper [2003]. Second, some attention is given to existing HDR compression systems.

LDR Texture Compression Vector quantization (VQ) techniques have been used by Beers et al. [1996] for TC. They presented compression ratios as low as one or two bpp. However, VQ requires an access in a look-up table, which is not desirable in a graphics hardware pipeline. The S3TC texture compression scheme [Iourcha et al. 1999] has become a de facto standard for real-time rendering. Since we build upon this scheme, it is described in more detail in Section 4.

Fenney [2003] presents a system where two low-resolution images are stored for each tile. During decompression, these images are bilinearly magnified and the color of a pixel is obtained as a linear blend between the magnified images. Another TC scheme assumes that the whole mipmap pyramid is to be compressed [Pereberin 1999]. Box filtering is used, and the luminance of a 4×4 tile is decomposed using Haar wavelets. The chrominance is subsampled, and then compressed. The compression ratio is 4.6 bpp.

In a TC system called iPACKMAN [Ström and Akenine-Möller 2005], 4×4 tiles of pixels are used, and each tile encodes two base colors in RGB444 and a choice of modifier values. The color of a pixel is determined from one of the base colors by adding a modifier value.

HDR Image and Video Compression To store an HDR image in the RGBE format, Ward [1991] uses 32 bits per pixel, where 24 bits are used for RGB, and the remaining 8 bits for an exponent, E, shared by all three color components. A straightforward extension would be to compress the RGB channels using S3TC to 4 bits per pixel, and store the exponent uncompressed as a separate 8 bpp texture, resulting in a 12 bits per pixel format supported by current graphics hardware. However, RGBE has a dynamic range of 76 orders of magnitude and is not a compact representation of HDR data known to reside in a limited range. Furthermore, as both the RGB and the exponent channel contain luminance information, chrominance and luminance transitions are not separated, and artifacts similar to the ones in Figure 13 are likely to occur. Ward also developed the LogLuv format [1998], where the RGB input is separated into luminance and chrominance information. The logarithm of the luminance is stored in 16 bits, while the chrominance is stored in another 16 bits, resulting in 32 bits per pixel. A variant using 24 bpp was also presented.

Ward and Simmons [2004] use the possibility of storing an extra 64 kB in the JPEG image format. The file contains a tone mapped image, which can be decompressed using standard JPEG decompressors. In the 64 kB of data, a ratio image of the luminance in the original and the tone mapped image is stored. A loader incapable of handling the format will display a tone mapped image, while capable loaders will obtain the full dynamic range. Xu et al. [2005] use the wavelet transform and adaptive arithmetic coding of JPEG 2000 to compress HDR images. They first compute the logarithm of the RGB values, and then use existing JPEG 2000 algorithms to encode the image. Impressive compression ratios and a high quality is obtained. Mantiuk et al. [2004] present an algorithm for compression of HDR video. They quantize the luminance using a non-linear function in order to distribute the error according to the luminance response curve of the human visual system. Then, they use an MPEG4-based coder, which is augmented to suppress errors around sharp edges. These three algorithms use adaptive bit rates, and thus cannot provide random access easily.

There is a wide range of tone mapping operators (cf. [Reinhard et al. 2005]), which perform a type of compression. However, the dynamic range is irretrievably lost in the HDR to LDR conversion, and these algorithms are therefore not directly applicable for TC. Li et al. [2005] developed a technique called companding, where a tone mapped image can be reconstructed back into an HDR image with high quality. This technique is not suitable for TC since it applies a global transform to the entire image, which makes random access extremely slow, if at all feasible. Still, inspiration can be obtained from these sources.

In the spirit of Torborg and Kajiya [1996], we implemented a fixed-rate HDR DCT encoder, but on hardware-friendly 4×4 tiles at 8 bits per pixel. The resulting images showed severe ringing artifacts near sharp luminance edges and moderate error values. The decompressor is also substantially more complex than the algorithms we present below.

3 Color Spaces and Error Measures

In this section, we discuss different color spaces, and develop a small variation of an existing color space, which is advantageous in terms of hardware decompression and image quality. Furthermore, we discuss error metrics in Section 3.2, where we also suggest a new simple error metric.

3.1 Color Spaces

The main difficulty in compressing HDR textures is that the dynamic range of the color components can be very large. In natural images, a dynamic range of 100,000:1 is not uncommon, i.e., a factor 10^5 difference between the brightest and the darkest pixels. A 24-bit RGB image, on the other hand, has a maximum range of 255:1. Our goal is to support about the same dynamic range as the OpenEXR format [Bogart et al. 2003], which is based on the hardware-supported `half` data type, i.e., 16-bit floating-point numbers. The range of representable numbers with full precision is roughly $6.1 \cdot 10^{-5}$ to $6.5 \cdot 10^4$, giving a dynamic range of 10^9:1. We aim to support this range directly in our format, as a texture may undergo complex image operations where lower precision is not sufficient. Furthermore, the dynamic range of the test images used in this paper is between $10^{2.6}$ and $10^{7.3}$. An alternative is to use a tighter range and a per-texture scaling factor. This is a trivial extension, which would increase the quality for images with lower dynamic ranges. However, this requires global per-texture data, which we have opted to avoid. We leave this for future work.

To get consistently good quality over the large range, we need a color space that provides a more compact representation of HDR data than the standard RGB space. Taking the logarithm of the RGB values gives a nearly constant relative error over the entire range of exposures [Ward 2005]. Assume we want to encode a range of 10^9:1 in 1% steps. In this $\log[RGB]$ space, we would need $k = 2083$ steps, given by $1.01^k = 10^9$, or roughly 11 bits precision per color channel. Because of the high correlation between the RGB color components [Sangwine and Horne 1998], we need to store all three with high accuracy. As we will see in Section 4, we found it difficult to reach the desired image quality and robustness when using the $\log[RGB]$ space.

In image and video compression, it is common to decorrelate the color channels by transforming the RGB values into a luminance/chrominance-based color space [Poynton 2003]. The motivation is that the luminance is perceptually more important than the chrominance, or *chroma* for short, and more effort can be spent on encoding the luminance accurately. Similar techniques have been proposed for HDR image compression. For example, the LogLuv encoding [Ward 1998] stores a log representation of the luminance and CIE (u', v') chrominance. Xu et al. [2005] apply the same transform as in JPEG, which is designed for LDR data, but on the logarithm of the RGB components. The OpenEXR format supports a simple form of compression based on a luminance/chroma space with the luminance computed as:

$$Y = w_r R + w_g G + w_b B, \tag{1}$$

and two chroma channels, U and V, defined as:

$$U = \frac{R - Y}{Y}, \qquad V = \frac{B - Y}{Y}. \tag{2}$$

Lossy compression is obtained by subsampling the chroma components by a factor two horizontally and vertically.

Inspired by previous work, we define a simple color space denoted $\log Y \bar{u} \bar{v}$, based on log-luminance and two chroma components. Given the luminance Y computed using Equation 1, the transform from RGB is given by:

$$(\bar{Y}, \bar{u}, \bar{v}) = \left(\log_2 Y, w_b \frac{B}{Y}, w_r \frac{R}{Y} \right). \tag{3}$$

We use the Rec. 601 [Poynton 2003] weights (0.299, 0.587, 0.114) for w_r, w_g and w_b. With non-zero, positive input RGB values in the range $[2^{-16}, 2^{16}]$, the log-luminance $\bar{Y}$ is in the range $[-16, 16]$, and the chroma components are in the range $[0, 1]$ with $\bar{u} + \bar{v} \leq 1$.

In our color space, the HDR luminance information is concentrated to the $\bar{Y}$ component, which needs to be accurately represented, while the $(\bar{u}, \bar{v})$ components only contain chrominance information normalized for luminance. These can be represented with significantly less accuracy.

3.2 Error Measures

In order to evaluate the performance of various compression algorithms, we need an image quality metric that provides a meaningful indication of image fidelity. For LDR images, a vast amount of research in such metrics has been conducted [Chalmers et al. 2000]. Perceptually-based metrics have been developed, which attempt to predict the observed image quality by modeling the response of the human visual system (HVS). The prime example is the *visible differences predictor* (VDP) introduced by Daly [1993].

Error measures for HDR images are not as thoroughly researched, and there is no well-established metric. The image must be tone-mapped before VDP or any other standard image quality metric, designed for LDR data, can be applied. The choice of tone mapping operator will bias the result, which is unfortunate. In our application, another difficulty is that we do not know how the HDR textures will be used or what the display conditions will be like. For example, a texture in a 3D engine can undergo a number of complex operations, such as lighting, blending and multi-texturing, which change its appearance.

Xu et al. [2005] compute the *root-mean-square error* (RMSE) of the compressed image in the log[RGB] color space. Their motivation is that the logarithm is a conservative approximation of the HVS luminance response curve. However, we argue that this error measure can be misleading in terms of visual quality. The reason is that an error in a small component tends to over-amplify the error measure even if the small component's contribution to the final pixel color is small. For example, consider a mostly red pixel, $\mathbf{r} = (1000, 1, 1)$, which is compressed to $\mathbf{r}^* = (1000, 1, 8)$. The log[$RGB$] RMSE is then $\log_2 8 - \log_2 1 = 3$, but the log-luminance RMSE, to which the HVS is most sensitive, is only 0.004. Still, we include the log[RGB] RMSE error because it reflects the relative, per-component error of the compressed image. It is therefore well suited to describe the expected error of the aforementioned image operations: blending, lighting etc.

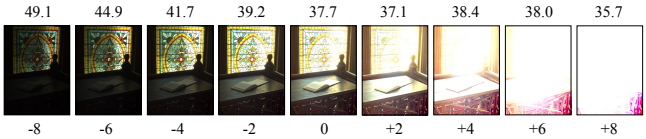


Figure 2: In our *multi-exposure PSNR* error measure, the image is tone mapped to a number of different exposures to account for all normal viewing conditions, and the PSNR is computed from the average MSE. In this case, the mPSNR is 39 dB. The top row shows the standard PSNRs, and the bottom row shows the exposure compensation, c.

To account for all normal viewing conditions, we propose a simple error metric, which we call *multi-exposure peak-signal-to-noise ratio*, or *mPSNR* for short. The HDR image is tone mapped to a number of different exposures, uniformly distributed over the dynamic range of the image. See Figure 2 for an example. For each exposure, we compute the *mean square error* (MSE) on the resulting LDR image, and then compute the peak-signal-to-noise ratio (PSNR) using the mean of all MSEs. As a tone mapping operator, we use a simple gamma-adjustment after exposure compensation. The tone mapped LDR image, $T(I)$, of the HDR image, I, is given by:

$$T(I) = \left[255 \left(2^c I \right)^{1/\gamma} \right]_0^{255}, \tag{4}$$

where c is the exposure compensation in f-stops, γ is the display gamma, and $[\cdot]_0^{255}$ indicates clamping to the integer interval $[0, 255]$. The mean square error over all exposures and over all pixels is computed as:

$$\text{MSE} = \frac{1}{n \times w \times h} \sum_c \sum_{x,y} \left(\Delta R_{xy}^2 + \Delta G_{xy}^2 + \Delta B_{xy}^2 \right), \tag{5}$$

where n is the number of exposures, c, and $w \times h$ is the image resolution. The error in the red component (similar for green and blue) at pixel (x, y) is $\Delta R_{xy} = T_R(I) - T_R(C)$, where I is the original image, and C is the compressed image. Finally, mPSNR is computed as:

$$\text{mPSNR} = 10 \log_{10} \left(\frac{3 \times 255^2}{\text{MSE}} \right). \tag{6}$$

The obtained mPSNR over all exposures gives us a prediction of the error in the compressed HDR image. The PSNR measure has traditionally been popular for evaluating the performance of TC schemes, and although no other HDR texture compression techniques exist, the use of mPSNR makes our results more easily interpreted.

Recently, Mantiuk et al. [2005] have presented a number of modifications to the visual differences predictor, making

it possible to predict the perceived differences over the entire dynamic range in real scenes. This novel HDR VDP takes into account a number of complex effects such as the non-linear response and local adaptation of the HVS. However, their current implementation only works on the luminance, and does not take the chroma error into account.

As there is no established standard for evaluating HDR image quality, we have chosen to use a variety of error metrics. We present results for our algorithm using the mPSNR, the $\log[RGB]$ root-mean-square error, and the HDR VDP.

4 HDR S3 Texture Compression

The S3 texture compression (S3TC) method [Iourcha et al. 1999] is probably the most popular scheme for compressing LDR textures. It is used in DirectX and there are extensions for it in OpenGL as well. S3TC uses tiles of 4×4 pixels that are compressed to 64 bits, giving a compression rate of 4 bpp. Two base colors are stored in 16 bits (RGB565) each, and every pixel stores a two-bit index into a local color map consisting of the two base colors and two additional colors in between the base colors. This means that all colors lie on a straight line in RGB space.

A natural suggestion for an HDR TC scheme is to adapt the existing S3TC algorithm to handle HDR data. Due to the increased amount of information, we double the rate to 8 bpp. We also apply the following changes. First, we transform the linear RGB data into a more compact color space. Second, we raise the quantization resolution and the number of per-pixel index bits. In graphics hardware, the memory is accessed in bursts of 2^n bits, e.g., 256 bits. To simplify addressing, it is desirable to fetch 2^m pixels per burst, which gives 2^{n-m} bits per pixel (e.g., 4,8,16,...). Hence, keeping a tile size of 4×4 pixels is a reasonable choice, as one tile fits nicely into 128 bits on an 8 bpp budget. In addition, a small tile size limits the variance across the tile and keeps the complexity of the decompressor low.

The input data consists of three floating-point values per pixel. Performing the compression directly in linear RGB space, or in linear YUV space, produces extremely poor results. This is due to the large dynamic range. Better results are obtained in the $\log[RGB]$ and the $\log Y\bar{u}\bar{v}$ color spaces (Section 3.1). Our tests show that 4-bit per-pixel indices are needed to accurately capture the luminance variations. We call the resulting algorithms *S3TC RGB* (using $\log[RGB]$), and *S3TC YUV* (using $\log Y\bar{u}\bar{v}$). The following bit allocations performed best in our tests:

Color space	Base colors	Per-pixel indices
$\log[RGB]$	$2 \times (11{+}11{+}10) = 64$	$16 \times 4 = 64$
$\log Y\bar{u}\bar{v}$	$2 \times (12{+}10{+}10) = 64$	$16 \times 4 = 64$

Even though these S3TC-based approaches produce usable results in some cases, they lack the robustness needed for a general HDR TC format. Some of the shortcomings of S3TC RGB and S3TC YUV are clearly illustrated in Figure 13. As can be seen in the enlarged images, both algorithms produce serious block artifacts, and blurring of some edges. This tends to happen where there is a chroma and a luminance transition in the same tile, and there is little or no correlation between these. The reason is that all colors must be located on a straight line in the respective 3D color space for the algorithms to perform well. In Figure 13, we also show the results of our new HDR texture compression scheme. As can be seen, the image quality is much higher. More importantly, our algorithm is more robust, and rarely generates tiles of poor quality.

5 New HDR Texture Compression Scheme

In the previous section, we have seen that building a per-tile color map from a straight line in some 3D color space does not produce acceptable results for S3TC-based algorithms. To deal with the artifacts, we decouple the luminance from the chrominance and encode them separately in the $\log Y\bar{u}\bar{v}$ space defined in Equation 3. By doing this, difficult tiles can be handled much better. In the following, we describe how the luminance and chrominance can be accurately represented on an 8 bpp budget, i.e., 128 bits per tile.

5.1 Luminance Encoding

In the $\log Y\bar{u}\bar{v}$ color space, the log-luminance values $\bar{Y}$ are in the range $[-16, 16]$. First, we find the minimum and maximum values, $\bar{Y}_{\min}$ and $\bar{Y}_{\max}$, in a tile. Inspired by S3TC, we then quantize these linearly and store per-pixel indices indicating which luminance step between $\bar{Y}_{\min}$ and $\bar{Y}_{\max}$ that is to be used for each pixel.

As we have seen, we need approximately 16 steps, i.e., 4-bit per-pixel indices, for an accurate representation of HDR luminance data. If we use 12-bit quantization of $\bar{Y}_{\min}$ and $\bar{Y}_{\max}$ as in S3TC YUV, a total of $2 \times 12 + 16 \times 4 = 88$ bits are consumed, and only 40 bits are left for the chroma encoding. This is not enough. By searching in a range around the quantized base values, it is very often possible to find a combination that gives a significantly reduced error. Thus, we manage to encode the base luminances with only 8 bits each without any noticeable artifacts, even on slow gradients.

Another approach would be to use spatial subsampling of the luminance. Recent work on HDR displays by Seetzen et al. [2003; 2004] suggests that the human eye's spatial HDR resolution is lower than its LDR resolution. However, the techniques developed for direct display of HDR images are not directly applicable to our problem as they require high-precision per-pixel LDR data to modulate the subsampled HDR luminance. We have tried various hierarchical schemes, but the low bit budget made it difficult to obtain the required per-pixel precision. Second, our compression scheme is designed for textures, hence we cannot make any assumptions on how the images will be displayed on screen. The quality should be reasonable even for close-up zooms. Therefore, we opted for the straightforward solution of storing per-pixel HDR luminance.

The most difficult tiles contain sharp edges, e.g., the edge around the sun in an outdoor photograph. Such tiles can have a very large dynamic range, but at the same time, both the darker and the brighter areas must be represented accurately. For this, a uniform quantization between the min/max luminances is not ideal. To better handle such tiles, we add a mode using non-uniform steps between the $\bar{Y}_{\min}$ and $\bar{Y}_{\max}$ values. Smaller quantization steps are used near the base luminances, and larger steps in the middle. Thus, two different luminance ranges that are far apart can be accurately represented in the same tile. In our test images (Figure 10), the non-uniform mode is used for 11% of the tiles, and for these tiles, the log-luminance RMSE is decreased by 12.0% on average. The two quantization modes are illustrated in Figure 3.

To choose between the two modes, we use the mutual ordering[1] of $\bar{Y}_{\min}$ and $\bar{Y}_{\max}$. In decoding, if $\bar{Y}_{\min} \leq \bar{Y}_{\max}$, then the uniform mode is used. Otherwise, $\bar{Y}_{\min}$ and $\bar{Y}_{\max}$ are reversed, and we use the non-uniform mode. Hence, no

[1] Similar ordering techniques are used in the S3TC LDR texture compression format.

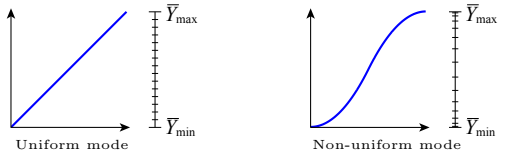


Figure 3: The two luminance quantization modes. The non-uniform mode is used for better handling tiles with sharp luminance transitions, such as edges.

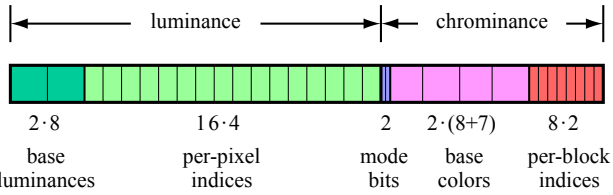


Figure 4: The bit allocation we use for encoding the luminance and chrominance of a 4×4 tile in 128 bits (8 bpp).

additional mode bit is necessary, and the luminance encoding uses a total of $2 \times 8 + 16 \times 4 = 80$ bits, leaving 48 bits for the chrominance. The bit allocation is illustrated in Figure 4.

5.2 Chrominance Line

Our first approach to chrominance compression on a 48-bit budget, is to use a line in the $(\bar{u}, \bar{v})$ chroma plane. Similar to the luminance encoding, each tile stores a number of indices to points uniformly distributed along the line.

In order to fit the chroma line in only 48 bits, we sub-sample the chrominance by a factor two, either horizontally or vertically, similar to what is used in DV encoding [Poynton 2003]. The sub-sampling mode that minimizes the error is chosen. To simplify the following description, we define a *block* as being either 1×2 (horizontal) or 2×1 (vertical) sub-sampled pixels. The start and end points of the chroma line, each with 2×8 bits resolution, and eight 2-bit per-block indices, gives a total cost of $4 \times 8 + 8 \times 2 = 48$ bits per tile.

In our color space, the normalized chroma points $(\bar{u}_i, \bar{v}_i)$, $i \in [0, 7]$, are always located in the lower triangle of the chroma plane. If we restrict the encoding of the end points of the line to this area, we can get two extra bits by a *flip trick* described in Figure 5. If a chrominance value $\mathbf{c} = (\bar{u}_i, \bar{v}_i)$ is in the upper (invalid) triangle, this indicates that the extra bit is set to one, and the true chroma value is given by $\mathbf{c}' = (1 - \bar{u}_i, 1 - \bar{v}_i)$, otherwise the bit is set to zero. We can use one of these extra bits to indicate whether to use horizontal or vertical sub-sampling. The other bit is left unused.

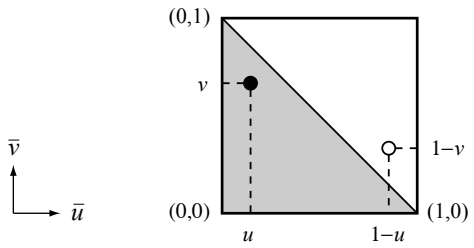


Figure 5: Illustration of the *flip trick*. By mirroring the coordinates for a base color, we exploit our triangular chrominance space in order to obtain another bit.

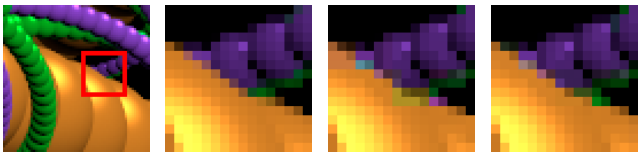


Figure 6: A region where a line in chroma space is not sufficient for capturing the complex color. With shape transforms, we get a much closer resemblance to the true color, although minor imperfections exist due to the sub-sampling.

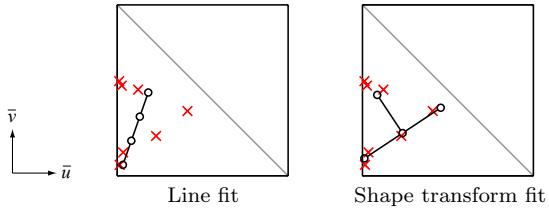


Figure 7: In tiles with difficult chrominance, such as in this example taken from Figure 6, a line in chroma space has difficulties representing the $(\bar{u}, \bar{v})$ points accurately (left). Our algorithm based on *shape transforms* generates superior chroma representations, as it is often possible to find a shape that closely matches the chrominance points (right).

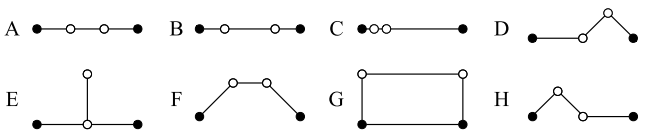
5.3 Chroma Shape Transforms

A line in chroma space can only represent two chrominances and the gradient between them. This approximation fails for tiles with complex chroma variations or sharp color edges. Figure 6 shows an example of such a case. One solution would be to encode $\bar{u}$ and $\bar{v}$ separately, but this does not easily fit in 48 bits.

To better handle difficult tiles, we introduce *shape transforms*; a set of shapes, each with four discrete points, designed to capture chroma information. In the classic game *Tetris*, the optimal placement of a shape in a grid is found by rotating and translating it in 2D. The same idea is applied to the chrominances of a tile. We allow arbitrary rotation, translation, and uniform scaling of our shapes to make them match the eight sub-sampled chrominance values of a tile as closely as possible. The transformation of the shape can be retrieved by storing only two points.

During compression, we select the shape that most closely covers the chroma information of the tile, and store its index along with two base chrominances (start & end) and per-block indices. This allows each block in the tile to select one of the discrete positions of the shape. The shape fitting is illustrated in Figure 7 on one of the difficult tiles from the image in Figure 6. Using one of the transformed shapes, we get much closer to the actual chroma information.

The space of possible shapes is very large. In order to find a selection of shapes that perform well, we have analyzed the chrominance content in a set of images (a total of 500,000 tiles), different from our test images. First, clustering was done to reduce the chroma values of a tile to four chroma points. Second, we normalized the chroma points for scale and rotation, and then iteratively merged the two closest candidates until the desired number of shapes remained. Figure 8 shows our selection of shapes after a slight manual adjustment of the positions. See Appendix B for the exact coordinates. Shapes A through C handle simple color gradients, while D–H are optimized for tiles with complex chrominance. Also, by including the uniform line (shape A), we make the chrominance line algorithm (Section 5.2) a sub-



Shape	A	B	C	D	E	F	G	H
Freq(%)	5.8	10.8	6.3	18.3	17.4	16.8	6.6	18.0

Figure 8: The set of shapes we use for the *shape transform* algorithm and their frequencies for our test images. The points corresponding to base colors are illustrated with solid black circles.

set of the shape transforms approach. Note that the set of shapes is fixed, so no global per-texture data is needed.

Compared to the chrominance line, shape transforms need three bits per tile to indicate which of the eight shapes to use. We exploit the unused bit from the chrominance line, and the other two extra bits are taken from the quantization of the start and end points. We lower the $(\bar{u}, \bar{v})$ quantization from $8+8$ bits to $8+7$ bits. Recall from Section 3.1, that in the $\log Y\bar{u}\bar{v}$ to RGB transform, the R and B components are given as $R = \bar{v}Y/w_r$ and $B = \bar{u}Y/w_b$, with $w_r = 0.299$ and $w_b = 0.114$. As w_b is about three times smaller than w_r, it makes the reconstructed color more sensitive for quantization errors in the $\bar{u}$ component, and therefore more bits are spent there.

With these modifications, shape transforms with eight shapes fit in precisely 48 bits. The total bit allocation for luminance and chrominance is illustrated in Figure 4. We evaluated both the chrominance line and the shape transform approach, combined with the luminance encoding of Section 5.1, on our test images. On average, the mPSNR was about 0.5 dB higher using shape transforms, and the resulting images are more visually pleasing, especially in areas with difficult chrominance.

The shape fitting step of our new algorithm is implemented by first clustering the eight sub-sampled input points to four groups. Then we apply Procrustes analysis [Dryden and Mardia 1998] to find the best orientation of each shape to the clustered data set, and the shape with the lowest error is chosen. This is an approximate, but robust and efficient approach. We achieved somewhat lower errors by using an extensive search for the optimal shape transform, but this is computationally much more expensive.

6 Hardware Decompressor

In this section, we present a decompressor unit for hardware implementation of our algorithm. We first describe how the chrominance, $(\bar{u}, \bar{v})$, is decompressed for a single pixel. In the second part of this section, we describe the color space transformation back to RGB-space. A presentation of log-luminance decompression is omitted, since it is very similar to S3TC LDR decompression. The differences are that the log-luminance is one-dimensional (instead of three-dimensional), and more bits are used for the quantized base values and per-pixel indices. In addition, we also have the non-uniform quantization, but this only amounts to using different constants in the interpolation.

The decompression of chrominance is more complex than for luminance, and in Figure 9, one possible implementation is shown. To use shape transforms, a coordinate frame must be derived from the chroma endpoints, $(\bar{u}_0, \bar{v}_0)$ and $(\bar{u}_1, \bar{v}_1)$, of the shape. In our case, the first axis is defined by $\mathbf{d} = (d_u, d_v) = (\bar{u}_1 - \bar{u}_0, \bar{v}_1 - \bar{v}_0)$. The other axis is

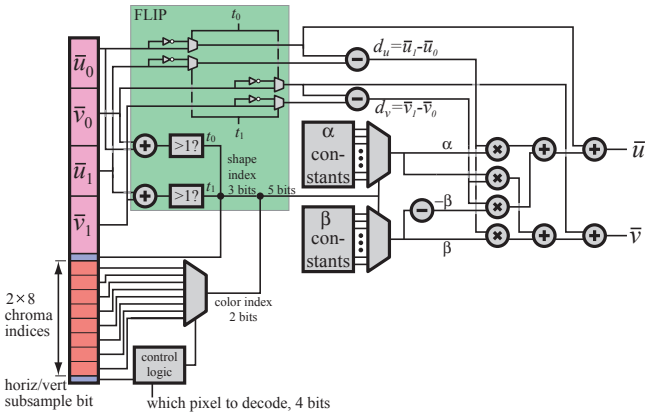


Figure 9: Decompression of chrominance, $(\bar{u}, \bar{v})$, for a single pixel. The indata is the 48 bits for chrominance (left part of the figure), and a 4-bit value indicating which pixel in a tile to decode. The outdata is $(\bar{u}, \bar{v})$ for one pixel. The green box contains the logic to implement the *flip trick*, where inverters have been used to compute the $1 - x$ terms.

$\mathbf{d}^\perp = (-d_v, d_u)$, which is orthogonal to $\mathbf{d}$. The coordinates of a point in a shape are described by two values α and β, which are both fixed-point numbers in the interval $[0, 1]$, using only five bits each. The chrominance of a point, with coordinates α and β, is derived as:

$$\begin{pmatrix}\bar{u}\\ \bar{v}\end{pmatrix} = \alpha\mathbf{d} + \beta\mathbf{d}^\perp + \begin{pmatrix}\bar{u}_0\\ \bar{v}_0\end{pmatrix} = \begin{pmatrix}\alpha d_u - \beta d_v + \bar{u}_0\\ \beta d_u + \alpha d_v + \bar{v}_0\end{pmatrix}. \quad (7)$$

The diagram in Figure 9 implements the equation above. As can be seen, the hardware is relatively simple. The α and β constants units contain only the constants which define the different chrominance shapes. Only five bits per value are used, and hence the four multipliers compute the product of a 5-bit value and an 8 or 9-bit value. Note that $(\bar{u}, \bar{v})$ are represented using fixed-points numbers, so integer arithmetic can be used for the entire chroma decompressor.

At this point, we assume that $\bar{Y}$, $\bar{u}$ and $\bar{v}$ have been computed for a certain pixel in a tile. Next, we describe our transform back to linear RGB space. This is done by first computing the floating-point luminance: $Y = 2^{\bar{Y}}$. After that, the red, green, and blue components can be derived from Equation 3 as:

$$(R, G, B) = \left(\frac{1}{w_r}\bar{v}Y, \frac{1}{w_g}(1 - \bar{u} - \bar{v})Y, \frac{1}{w_b}\bar{u}Y\right). \quad (8)$$

Since the weights, $(w_r, w_g, w_b) = (0.299, 0.587, 0.114)$, are constant, their reciprocals can be precomputed. We propose using the hardware-friendly constants: $(1/w'_r, 1/w'_g, 1/w'_b) = \frac{1}{16}(54, 27, 144)$. This corresponds to $(w'_r, w'_g, w'_b) \approx (0.296, 0.593, 0.111)$. Using our alternative weights makes the multiplications much simpler. This comes with a non-noticeable degradation in image quality.

In summary, our color space transform involves one power function, two fixed-point additions, three fixed-point multiplications, and three floating-point times fixed-point multiplications. The majority of color space transforms include at least a 3×3 matrix/vector multiplication, and in the case of HDR data, we have seen that between 1–3 power functions are also used. Ward's LogLuv involves even more operations. Our transform involves significantly fewer arithmetic operations compared to other color space transforms, and this is a major advantage of our decompressor.

The implementation shown above can be considered quite inexpensive, at least when compared to using other color spaces. Still, when compared to popular LDR TC schemes [Iourcha et al. 1999; Ström and Akenine-Möller 2005], our decompressor is rather complex. However, we have attempted to make it simpler by designing a hardware-friendly color space, avoided using too many complex arithmetic operations, and simplified constants. In addition, we believe that in the near future, graphics hardware designers will have to look into more complex circuitry in order to reduce bandwidth, and the technological trend shows that this is the way to go [Owens 2005].

7 Results

To evaluate the algorithms, we use a collection of both synthetic images and real photographs, as shown in Figure 10. Many of these are well-known and widely used in HDR research. In the following, we refer to *our algorithm* as the combination of our luminance encoding (Section 5.1) and shape transforms (Section 5.3). The results using mPSNR, log[*RGB*] RMSE, and HDR VDP are presented in Figure 11. After that follows visual comparisons.

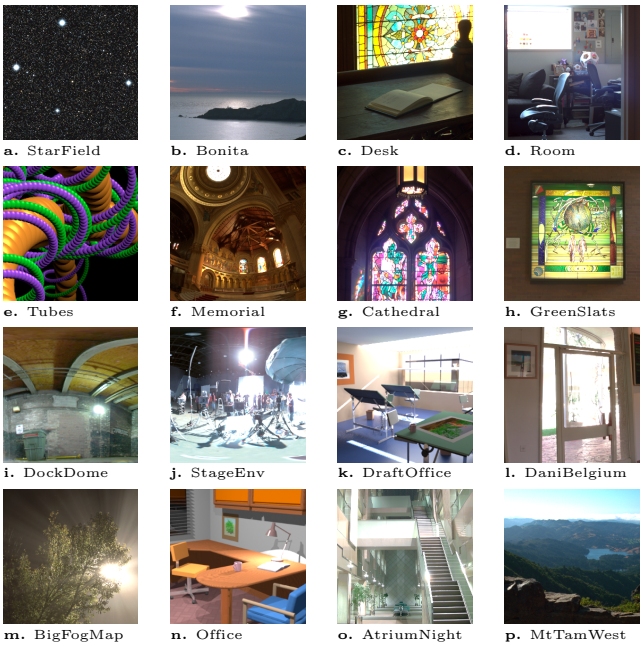


Figure 10: The HDR test images we use for evaluating our algorithms. The figure shows cropped versions of the actual test images. **(e)**, **(k)**, and **(n)** are synthetic.

In terms of the mPSNR measure, our algorithm performs substantially better over the entire set of images, with an average improvement of about 3 dB over the S3TC-based approaches. The mPSNR measure simulates the most common use of an HDR image in a real-time application, where the image is tone mapped and displayed under various exposures, either as a decal or as an environment map. The range of exposures used in mPSNR is automatically determined by computing the min and max luminance over each image, and mapping this range to exposures that give a nearly black, and a nearly white LDR image respectively. See Appendix A for the exact numbers.

The log[*RGB*] RMSE measures the relative error per component over the entire dynamic range of the image. In this

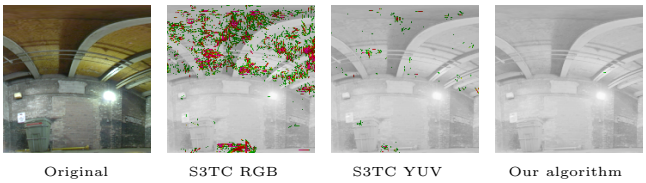


Figure 12: HDR VDP difference images. Green indicates areas with 75% chance of detecting an artifact, and red indicates areas with 95% detection probability.

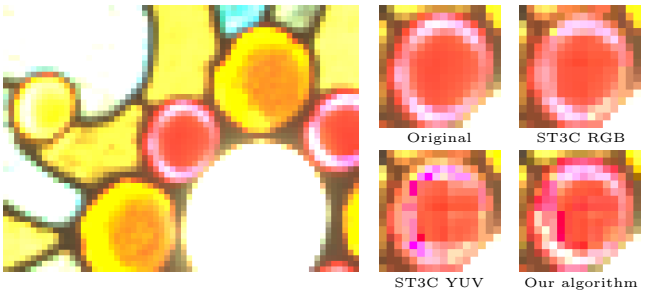


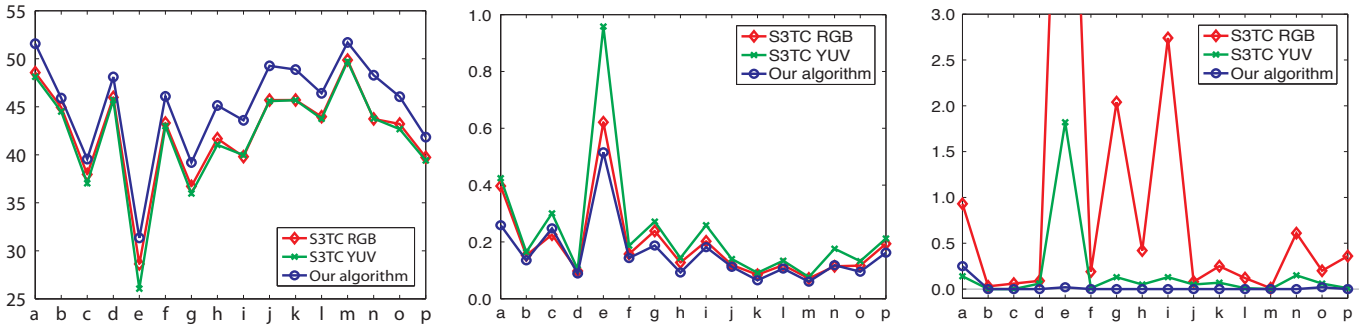
Figure 14: A difficult scenario for our algorithm is an overexposed image **(c)** with sharp color transitions. S3TC YUV does, however, handle this case even worse. Surprisingly, S3TC RGB performs very well here.

metric, the differences between the algorithms are not as obvious. However, our algorithm gives slightly lower error on average.

The HDR VDP chart in Figure 11 shows just noticeable luminance differences. The 75%-value indicates that an artifact will be visible with a probability of 0.75, and the value presented in the chart is the percentage of pixels above this threshold. Our test suite consists of a variety of both luminance-calibrated and relative luminance HDR images. To compare them, we multiply each image with a luminance factor so that all HDR VDP tests are performed for a global adaptation level of approximately 300 cd/m^2. Although we quantize the base luminances to only 8 bits, our algorithm shows near-optimal results, except for image **(a)**. VDP difference images for image **(i)** are shown in Figure 12. Increasing the global adaptation level increases the detection rates slightly, but the relationship between the algorithms remains.

Our algorithm is the clear winner both in terms of robustness and perceived visual quality, as can be seen in Figure 13. In general, luminance artifacts are more easily detectable, and both S3TC YUV and our algorithm handle these cases better due to the luminance focus of the log $Y\bar{u}\bar{v}$ color space. However, the limitation of S3TC YUV to a line in 3D makes it unstable in many cases. The only errors we have seen using our algorithm are slight chrominance leaks due to the subsampling, and artifacts in some images with high exposures, originating from the quantization of the base chrominances. Such a scenario is illustrated in Figure 14. Overall, our algorithm is much more robust since it generates significantly fewer tiles with visible errors, and this is a major strength.

It is very important that a TC format developed for real-time graphics handle mipmapping well. Figure 15 shows the average error from all our test images for the first 7 mipmap levels. The average errors grow at smaller mipmap levels due to the concentration of information in each tile, but our algorithm is still very robust and compares favorably to the S3TC-based techniques. In both error measures, our approach is consistently much better.



mPSNR (dB) log[*RGB*] RMSE HDR VDP 75% error

Figure 11: These diagrams show the performance of our algorithm compared to the S3TC RGB and S3TC YUV algorithms for each of the images **(a)**–**(p)** in Figure 10. The mPSNR measure gives consistently better values (left), while the log[*RGB*] RMSE (middle) is lower on nearly all of the test images. The HDR VDP luminance measure (right) indicates a perceivable error very close to 0.0% for most images with our algorithm, clearly superior to both S3TC-based algorithms.

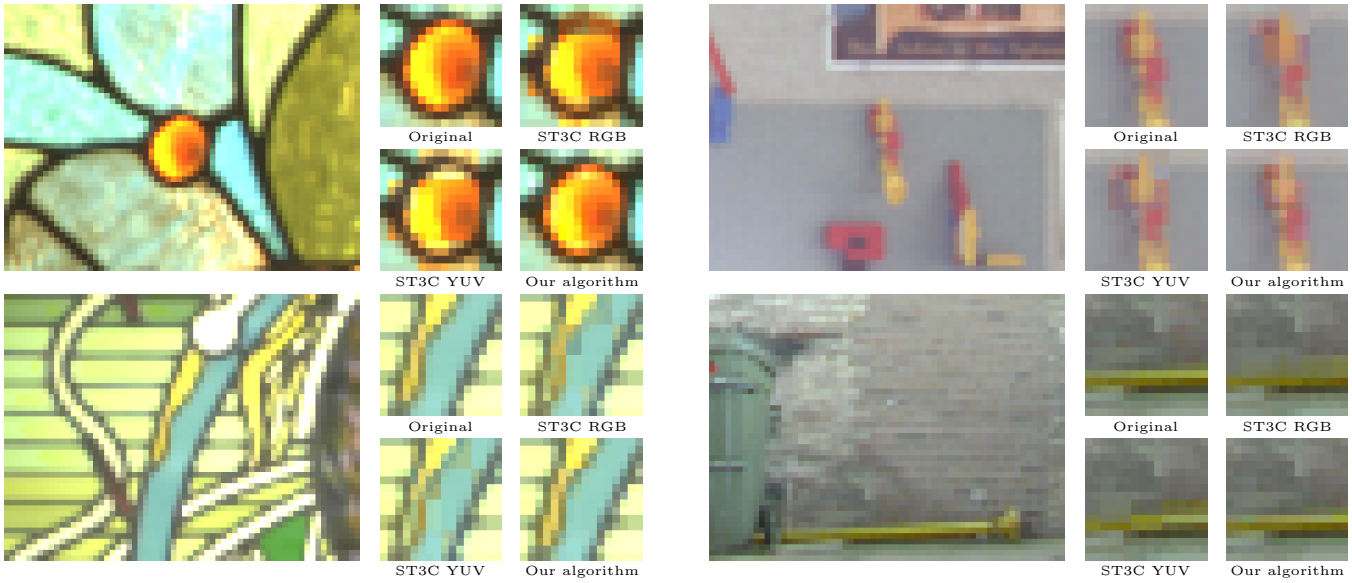


Figure 13: This figure shows magnified parts of the test images **(c)**, **(d)**, **(h)**, and **(i)**, compressed with our algorithm and the two S3TC-based methods. In difficult parts of the images, the S3TC algorithms sometimes produce quite obvious artifacts, while this rarely happens with our algorithm due to its separated encoding of luminance and chrominance.

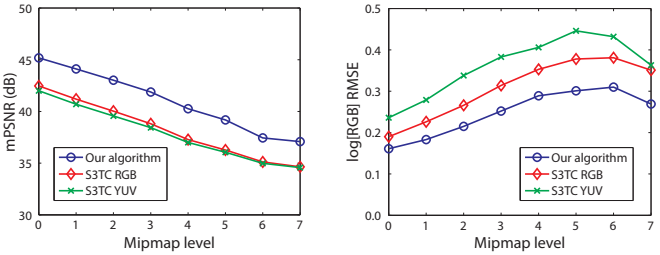


Figure 15: Here, the average mPSNR and log[*RGB*] RMSE values are presented for various mipmap levels of our test images, where 0 is the original image and higher numbers are sub-sampled versions.

8 Conclusions

In this work, we have presented the first low bit rate HDR texture compression system suitable for graphics hardware. In order to accurately represent the wide dynamic range in HDR images, we opted for a fixed rate of 8 bpp. Although it would have been desirable to further reduce the bandwidth, we found it hard to achieve an acceptable image quality at 4 bpp, while preserving the full dynamic range. For future work, it would be interesting to incorporate an alpha channel in the 8 bpp budget.

Our algorithm performs very well, both visually and in the chosen error metrics. However, more research in meaningful error measures for HDR images is needed. The HDR VDP by Mantiuk et al. [2005] is a promising approach, and it would be interesting to extend it to handle chroma as well. We hope that our work will further accelerate the use of HDR images in real-time rendering, and provide a basis for future research in HDR texture compression.

Acknowledgements

We acknowledge support from the Swedish Foundation for Strategic Research and Vetenskapsrådet. Thanks to Calle Lejdfors & Christina Dege for proof-reading, Rafal Mantiuk for letting us use his HDR VDP program, and Apple for the temporary Shake license. Image (**f**) is courtesy of Paul Debevec. (**g**) and (**l**) are courtesy of Dani Lischinski. (**h**) was borrowed from the RIT MCSL High Dynamic Range Image Database. (**i**) was created using HDRI data courtesy

of HDRIMaps (`www.hdrimaps.com`) from the LightWorks HDRI Starter Collection (`www.lightworkdesign.com`). (**k**) and (**n**) are courtesy of Greg Ward. (**m**) is courtesy of Jack Tumblin, Northwestern University. (**o**) is courtesy of Karol Myszkowski. The remaining images were taken from the OpenEXR test suite.

References

Beers, A., Agrawala, M., and Chadda, N. 1996. Rendering from Compressed Textures. In *Proceedings of ACM SIGGRAPH 96*, 373–378.

Bogart, R., Kainz, F., and Hess, D. 2003. OpenEXR Image File Format. In *ACM SIGGRAPH Sketches & Applications*.

Chalmers, A., McNamara, A., Daly, S., Myszkowski, K., and Troscianko, T. 2000. Image Quality Metrics. In *ACM SIGGRAPH Course Notes*.

Cohen, J., Tchou, C., Hawkins, T., and Debevec, P. 2001. Real-Time High Dynamic Range Texture Mapping. In *Eurographics Workshop on Rendering*, 313–320.

Daly, S. 1993. The Visible Differences Predictor: An Algorithm for the Assessment of Image Fidelity. In *Digital Images and Human Vision*. MIT Press, 179–206.

Debevec, P. E., and Malik, J. 1997. Recovering High Dynamic Range Radiance Maps from Photographs. In *Proceedings of ACM SIGGRAPH 97*, 369–378.

Debevec, P. E. 1998. Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-Based Graphics with Global Illumination and High Dynamic Range Photography. In *Proceedings of ACM SIGGRAPH 98*, 189–198.

Dryden, I., and Mardia, K. 1998. *Statistical Shape Analysis*. Wiley.

Fenney, S. 2003. Texture Compression using Low-Frequency Signal Modulation. In *Graphics Hardware*, 84–91.

Iourcha, K., Nayak, K., and Hong, Z., 1999. System and Method for Fixed-Rate Block-Based Image Compression with Inferred Pixel Values. US Patent 5,956,431.

Knittel, G., Schilling, A. G., Kugler, A., and Strasser, W. 1996. Hardware for Superior Texture Performance. *Computers & Graphics, 20*, 4, 475–481.

Li, Y., Sharan, L., and Adelson, E. H. 2005. Compressing and Companding High Dynamic Range Images with Subband Architectures. *ACM Transactions on Graphics, 24*, 3, 836–844.

Mantiuk, R., Krawczyk, G., Myszkowski, K., and Seidel, H.-P. 2004. Perception-Motivated High Dynamic Range Video Encoding. *ACM Transactions on Graphics, 23*, 3, 733–741.

Mantiuk, R., Daly, S., Myszkowski, K., and Seidel, H.-P. 2005. Predicting Visible Differences in High Dynamic Range Images – Model and its Calibration. In *Human Vision and Electronic Imaging X*, 204–214.

Owens, J. D. 2005. Streaming Architectures and Technology Trends. In *GPU Gems 2*. Addison-Wesley, 457–470.

Pereberin, A. 1999. Hierarchical Approach for Texture Compression. In *Proceedings of GraphiCon '99*, 195–199.

Poynton, C. 2003. *Digital Video and HDTV Algorithms and Interfaces*. Morgan Kaufmann Publishers.

Reinhard, E., Ward, G., Pattanaik, S., and Debevec, P. 2005. *High Dynamic Range Imaging: Acquisition, Display and Image-Based Lighting*. Morgan Kaufmann Publishers.

Sangwine, S. J., and Horne, R. E. N., Eds. 1998. *The Colour Image Processing Handbook*. Chapman and Hill.

Seetzen, H., Whitehead, L. A., and Ward, G. 2003. A High Dynamic Range Display Using Low and High Resolution Modulators. *Society for Information Display Internatiational Symposium Digest of Technical Papers*, 1450–1453.

Seetzen, H., Heidrich, W., Stuerzlinger, W., Ward, G., Whitehead, L., Trentacoste, M., Ghosh, A., and Vorozcovs, A. 2004. High Dynamic Range Display Systems. *ACM Transactions on Graphics 23*, 3, 760–768.

Ström, J., and Akenine-Möller, T. 2005. iPACKMAN: High-Quality, Low-Complexity Texture Compression for Mobile Phones. In *Graphics Hardware*, 63–70.

Torborg, J., and Kajiya, J. 1996. Talisman: Commodity Realtime 3D Graphics for the PC. In *Proceedings of SIGGRAPH*, 353–364.

Ward, G., and Simmons, M. 2004. Subband Encoding of High Dynamic Range Imagery. In *Proceedings of APGV '04*, 83–90.

Ward, G. 1991. Real Pixels. In *Graphics Gems II*. Academic Press, 80–83.

Ward, G. J. 1994. The RADIANCE Lighting Simulation and Rendering System. In *Proceedings of ACM SIGGRAPH 94*, 459–472.

Ward, G. L. 1998. LogLuv Encoding for Full Gamut High Dynamic Range Images. *Journal of Graphics Tools, 3*, 1, 15–31.

Ward, G., 2005. High Dynamic Range Image Encodings, `http://www.anyhere.com/`.

Xu, R., Pattanaik, S. N., and Hughes, C. E. 2005. High-Dynamic-Range Still-Image Encoding in JPEG 2000. *IEEE Computer Graphics and Applications, 25*, 6, 57–64.

A mPSNR Parameters

In this appendix, we summarize the parameters used when computing the mPSNR quality measure for the images in Figure 10 (**a**–**p**). For all mPSNR computations, we have computed the mean square error (MSE) only for the integers between the start and stop exposures (as shown in the table below). For example, if the start exposure is -10, and the stop exposure is $+5$, then we compute the MSE for all exposures in the set: $\{-10, -9, \ldots, +4, +5\}$.

Test image	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**
Start exposure	-9	-8	-8	-9	-3	-9	-4	0
Stop exposure	+4	+2	+5	+3	+7	+3	+7	8
Test image	**i**	**j**	**k**	**l**	**m**	**n**	**o**	**p**
Start exposure	-6	-12	-7	-6	-8	-6	-12	-4
Stop exposure	+3	+1	+2	+5	+2	+3	+1	+5

B Shape Transform Coordinates

Below we present coordinates, (α, β), for each of the template shapes in Figure 8.

Shape	p1	p2	p3	p4
A	(0, 0)	(11/32, 0)	(21/32, 0)	(1, 0)
B	(0, 0)	(1/4, 0)	(3/4, 0)	(1, 0)
C	(0, 0)	(1/8, 0)	(1/4, 0)	(1, 0)
D	(0, 0)	(1/2, 0)	(3/4, 1/4)	(1, 0)
E	(0, 0)	(1/2, 0)	(1/2, 1/2)	(1, 0)
F	(0, 0)	(11/32, 11/32)	(21/32, 11/32)	(1, 0)
G	(0, 0)	(0, 1/2)	(1, 1/2)	(1, 0)
H	(0, 0)	(1/4, 1/4)	(1/2, 0)	(1, 0)