A GPU-friendly Method for High Dynamic Range Texture Compression using Inverse Tone Mapping

Francesco Banterle [*] Kurt Debattista Patrick Ledda Alan Chalmers

Warwick Digital Laboratory
University of Warwick

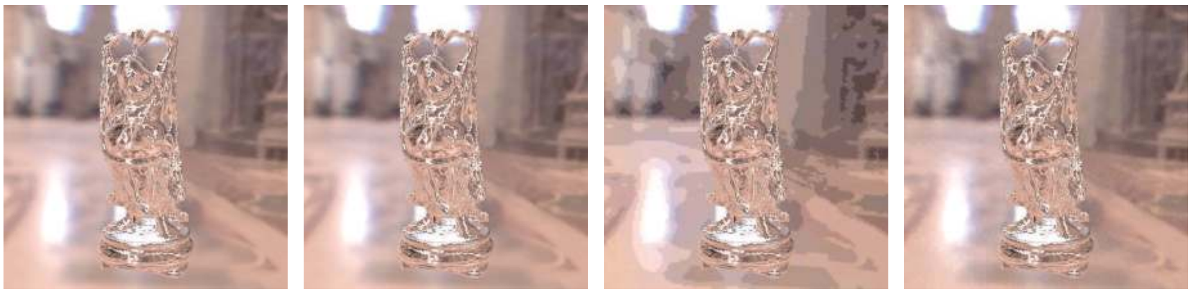


Figure 1: A simple real-time application showing Happy Buddha with emphasis on texturing operations for timing: on the left Ground truth HDR texture was encoded using 32bit floating point for each channel (96 bpp). On the center left RGBE encoding (32 bpp). On the center right Wang et al. [28] (16 bpp), note that in this case the splitting axis between LDR and HDR image minimizes RMSE error but appearance presents noticeable quantization artifacts. On the right our method using residuals (8 bpp).

Abstract

In recent years, High Dynamic Range Textures (HDRTs) have been frequently used in real-time applications and video-games to enhance realism. Unfortunately, HDRTs consume a considerable amount of memory, and efficient compression methods are not straightforward to implement on modern GPUs. We propose a framework for efficient HDRT compression using tone mapping and its dual, inverse tone mapping. In our method, encoding is performed by compressing the dynamic range using a tone mapping operator followed by a traditional encoding method for low dynamic range imaging. Our decoding method, decodes the low dynamic range image and expands its range with the inverse tone mapping operator. We present results using the Photographic Tone Reproduction tone mapping operator and its inverse encoded with S3TC running in real-time on current programmable GPU-hardware resulting in compressed HDRTs at $4-8$ bits per pixel (bpp), using a fast shader program for decoding. We show how our approach is favorable compared to other existing methods.

Index Terms: I.4.2 [Image Processing and Computer Vision]: Compression (Coding) Approximate methods— [I.3.1]: Computer Graphics—Architecture Graphics processors I.3.7 [Computer Graphics]: Three Dimensional Graphics and Realism Texture—

1 Introduction

High dynamic range (HDR) images [24] offer a more representative description of image-based digital content by storing data with a higher pixel depth than the more conventional low dynamic range (LDR) images. The increased depth presented by HDR image formats can account for the dynamic range visible by the human visual system. Not surprisingly, HDR imagery has been adapted for a large number of applications, for example, it served as the foundation of image-based lighting [5]. Interactive rendering applications, most notably video games, have been quick to adopt the format for lighting and general texture mapping methods [10, 19].

The use of texture mapping in real-time requires careful management of textures, due to limited texture memory and texture bandwidth which can negatively affect performance. The problem is further compounded with the use of HDR textures which can easily be twice the size or more of LDR textures (LDRTs). For LDRTs, texture compression algorithms have made it possible to pass compressed data to texture memory and have it decoded in real time upon texture access. A typical LDRT consisting of 24 bit-per-pixel (bpp) RGB data can be compressed into 4-8 bpp (depending on the format) saving texture memory and reducing texture bandwidth. Many of these compression formats require fixed-rate access to enable simple decoding in hardware. Similar methods are being developed for HDRTs, yet some of them are not automatically supported by current hardware/software and their compression rates are not yet comparable to those of LDRTs.

In this paper we present a new texture compression method for HDRTs which uses the concepts of tone mapping and inverse tone mapping. Tone mapping algorithms (see [24] for an overview) were developed to compress the dynamic range of HDR images to enable them to be displayed on conventional monitors and print which cannot support such high dynamic ranges. Some tone mapping algorithms use tone mapping operators (TMOs) which can be inverted [1]. These inverse TMOs (iTMOs) can be used to reconstruct the dynamic range of an image.

In this paper we present a framework for HDRT compression which encodes images using a TMO to reduce the dynamic range of an HDRT to that of an LDRT which can then be compressed using a standard encoding algorithm. The decoding process uses the standard decoding method to decode the LDRT and the iTMO to expand the LDRT back to an HDRT. We show our framework with respect to the Photographic Tone Reproduction Operator [23] and its inverse [1] using S3TC [9] compression for the LDRT, which enables us to achieve 4-8 bpp for HDRTs. We use a simple pixel

[*]e-mail: f.banterle@warwick.ac.uk

shader for the decoding which can run in real time on current graphics hardware using bilinear, and mip-mapping filtering. We present results of our implementation compared to other similar methods.

2 Related Work

2.1 Representation Format

An uncompressed HDR image typically uses 96 bpp for an RGB image, which is four times that of an uncompressed LDR image. This large amount of data needs to be compressed in order to be used efficiently. One of the first efficient HDR image representation was RGBE introduced by Ward in [29]. This format compresses a 96 bpp image in 32 bpp using an 8 bit mantissa for each channel and a shared 8 bit exponent. However, RGBE representation cannot cover the full visible color gamut since it does not allow negative values. To solve this problem two formats can be used: XYZE, the same encoding of RGBE in XYZ, and LogLuv format [12]. The latter format uses 32 bpp in which luminance is stored in 16 bit logarithmic space and 16 bits is used for colors. A 24 bpp alternative uses 10 bit for luminance and 14 bit for colors. Recently, a new format, OpenEXR, introduced by Industrial Light & Magic in [8] has become very popular. OpenEXR represents HDR data using 16 bit floating-point per channel for a total of 48 bpp. It is supported by modern graphics hardware. Finally, RGBS format presented in [19] is similar to RGBE, a shared exponent for red, green and blue channel. However, the exponent is not saved in logarithmic space so exponential operation is avoided during decoding.

These formats present some problems for texture mapping in real-time applications. RGBE needs expensive operations such as the exponential calculation to be programmed in the shader, and filtering needs to be performed on the shader to avoid artifacts using standard filtering on hardware (bilinear, mip-mapping, and anisotropic filtering). RGBS has the same problem for the filtering. LogLuv can be implemented only on new GPUs which have bit operators, and presents the same problems as RGBE. Finally, OpenEXR is not supported by all GPUs for filtering operations, and its 48 bpp mean that it is not memory efficient.

2.2 General HDR Compression

A representation format for a single pixel can help in reducing space, however high compression ratios may be achieved using information from neighboring pixels. In the last years several extensions to standard compression algorithms have been presented. Backwards-compatible HDR-JPEG presented in [30–32] extends JPEG standard keeping retro-compatibility. Firstly, an HDR image is tone mapped using PTR and stored as a normal JPEG. Secondly, a sub-band corresponding to HDR information is stored in the “application markers” of the standard for a maximum of 64 Kbytes, which is a constraint for encoding high resolution HDR images. HDR-JPEG2000 [33] is an extension to JPEG2000 that exploits the support of the standard for 16 bit integer data. So HDR data is transformed into the logarithmic domain, quantized into integers and compressed using standard JPEG2000 encoding. Finally, for HDR videos HDR-MPEG had been proposed in [16, 17] as an extension to MPEG-4. As in HDR-JPEG-backwards videos are tone mapped, and for each frame a reconstruction function is calculated for storing HDR data. To improve quality residuals of frames are saved in the video stream.

While these algorithms present high quality and high compression ratios, they are not ideally suitable for real-time applications since their lack of hardware support results in complex implementations, particularly due to the potentially complex fetching mechanisms required for decoding. For real-time applications, using fixed-rate bit compression simplifies the decoding process and makes it simpler and typically faster to implement on graphics hardware.

2.3 LDR Texture Compression

One of the first methods for rendering from compressed texture was Vector Quantization [2], which had a high compression ratio, but required access into a look-up table. The current standard de facto is S3TC [9] a blocking code scheme (see section 4.2 for more details). Recently, new methods have been proposed based on tiles. For example, in [6], for each tile two low resolution images are coded and magnified during decoding. In PackMan [27] and its evolution iPackMan [27], for each tile (4×4) two 4 bit base colors are stored with modifier values. The final color is calculated by adding a modifier value to the base color.

2.4 HDR Texture Compression

Three recent papers addressed HDRT compression [20, 25, 26], using new block based texture compression methods similar to S3TC. The main problem of these two new methods, despite the high quality in compression, is that they need special hardware that is not provided in the current target generation of graphics hardware. A different approach is adopted by Wang et al. [28], where HDR and LDR parts of the images were separated and quantized in two 8 bit textures compressed using S3TC with their residuals. The reconstruction was performed combining HDR and LDR part in a simple shader. The main disadvantage of their approach is that it takes two LDR textures (16 bpp). Recently a more general compression scheme was proposed [13]. This method relies on a hierarchical data structure that represents spatially coherent graphics data in an efficient way. While it achieves 5 bpp for HDRT maintaining high quality, it presents a complex shader (shader model 4) to decode the texture which is 20 times slower than a fetch to a compressed texture using S3TC. Finally hardware vendors presented hardware support for RGBE with filtering, EXT_texture_shared_exponent [11] or DXGI_FORMAT_R9G9B9E5_SHAREDEXP [3], using 9 bits for each channel and 5 bits for the shared exponent. However the main problem, low compression rate 32 bpp, still remains.

2.5 Inverse Tone Mapping for Compression

A multi-scale image processing technique for both tone mapping and *companding* (tone mapping followed by an expansion) was proposed in [14]. The main problem for the decoding in this algorithm is that the compressed LDR has to be decomposed into subbands, an operation that is not very efficient on current GPUs.

A compression method based on an inverse tone mapping operator and JPEG was presented in [21]. The TMO is based on the Hill function and parameters for inverse tone mapping are calculated using minimization techniques, and then encoded using JPEG. The residuals are calculated for increased quality and are compressed using wavelets. Wavelets and DCT decompression are computationally expensive to evaluate on GPU, and they do not provide a constant decompression time.

A compression scheme for still images and videos using a TMO based on a model of human cones was presented in [7]. This work neither presented a validation nor a study about a further compression stage of tone mapped LDR images or videos, such as applying JPEG, JPEG2000, or MPEG compression and the effect on the inversion. Also it does not provide the computational cost of the proposed TMO and iTMO.

3 Inverse Tone Mapping Compression Framework

In this section we present the general framework and underlying theory of our approach. Implementation details of our framework will be described in Section 4.

Our encoding and decoding processes can be seen in Figure 2. The entire framework is based on the concept of inverse tone mapping. Tone mappers generally use tone mapping operators (TMOs) to compress HDR images in order to visualize them on traditional displays. If a TMO is a monotonically increasing function it can

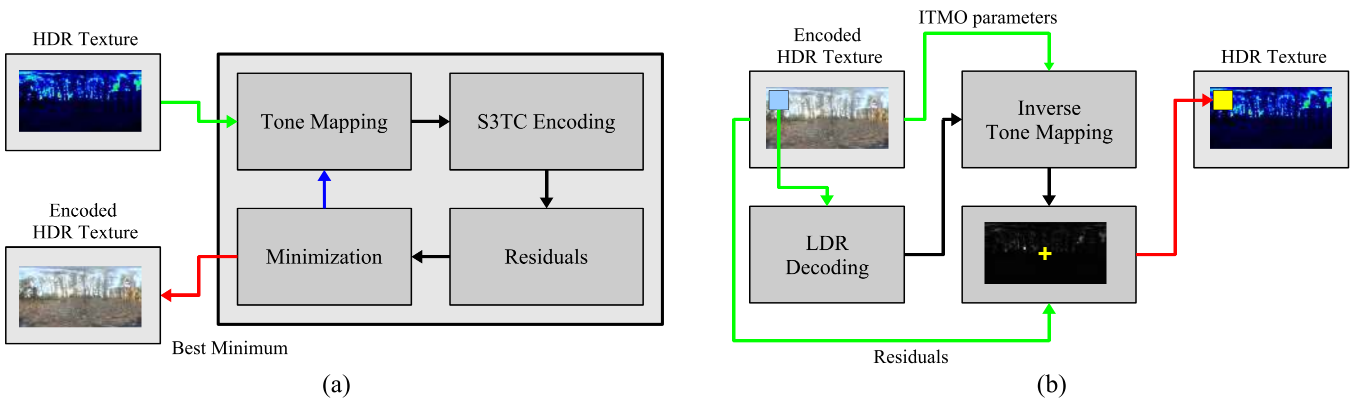


Figure 2: Our compression framework: a) The pipeline of the framework for enconding HDR textures: the process begins with an optimization step, a TMO is applied to it and the obtained LDRT is stored using a codec and residuals are calculated from this encoded LDRT. Finally we check if the error is small enough otherwise the process is iterated again until the best minimization is reached. b) The pipeline of the framework for decoding HDR textures: when a pixel needs to be evaluated LDR decoding is applied, S3TC/DXTC in our case. Then the luminance value of this pixel is expanded using inverse tone mapping using the iTMO parameters, which are calculated in the encoding phase, and the residuals.

be inverted in order to expand a low dynamic range image into a high dynamic range image [1]. In general a global TMO is usually invertible while a local operator is not, due to problems with deconvolution and since local TMOs are not usually monotonically increasing functions.

Using this knowledge, we compress HDRTs (96 bpp) using a TMO and store the result in an LDRT (24 bpp). We then compress this LDRT using a traditional LDR compression codec. When a texture is accessed we first decode the LDRT and then use the inverse of the tone mapping operator used in compression to expand the LDRT to an HDRT. Additive or multiplicative residuals can be added to increase the final quality.

3.1 An Inverse Tone Mapping Operator

An Inverse Tone Mapping Operator, $g(x)$, can be derived from a TMO when the TMO is a monotonically increasing function $f(x)$ that we can invert, such that $g = f^{-1}(x)$. As an example of how tone mapping and inverse tone mapping may be used in our framework we present our approach using the Global Photographic Tone Reproduction operator [24]. We chose this TMO because it is non-linear, which allows to define a good relationship between HDR and LDR values, easy to invert, and computationally cheap. This TMO compresses luminance without applying any particular operation on colors, it is defined as:

$$\begin{cases} f(L_{i,j}) = L'_{i,j} = \frac{\alpha L_{i,j}(\alpha L_{i,j} + L_{\text{H}} L^2_{\text{white}})}{L_{\text{H}} L^2_{\text{white}}(\alpha L_{i,j} + L_{\text{H}})} \\ [R'_{i,j}, G'_{i,j}, B'_{i,j}]^\top = \frac{L'_{i,j}}{L_{i,j}}[R_{i,j}, G_{i,j}, B_{i,j}]^\top \end{cases} \quad (1)$$

where R, G, B are respectively red, green and blue color channels, $'$ indicates compressed values, $L = 0.213R + 0.72G + 0.072B$ is the luminance channel of an HDRT, L_{H} is the harmonic mean, L_{white} is the maximum white point, and α is the scale factor, see [24] for a complete overview on these parameters. As in [1] we invert, the above TMO, for decoding the LDRT to an HDRT, by solving Equation 1 for $L_{i,j}$ obtaining $g = f^{-1}$:

$$\begin{cases} g(L'_{i,j}) = L_{i,j} = \frac{L^2_{\text{white}} L_{\text{H}}}{2\alpha}\left(L'_{i,j} - 1 + \sqrt{(1 - L'_{i,j})^2 + \frac{4L'_{i,j}}{L^2_{\text{white}}}}\right) \\ [R_{i,j}, G_{i,j}, B_{i,j}]^\top = \frac{L_{i,j}}{L'_{i,j}}[R'_{i,j}, G'_{i,j}, B'_{i,j}]^\top \end{cases} \quad (2)$$

At this point, we need to define how to set the parameters, α and L_{white}, for f and $g = f^{-1}$, because during the quantization to 8 bit per channel of the tone mapped texture, we need to fit the data in the bits as much as possible, see for example Figure 3. So we define a function that we need to minimize:

$$\varepsilon(L) = \sum_{i,j} \left\| \log(L_{i,j}) - \log(g([f(L_{i,j})]_0^{255})) \right\|^2 \quad (3)$$

where $[\ ]_0^{255}$ is the quantization in the range $[0, 255]$. ε is calculated in the logarithmic domain to be more robust to outliers, for example few pixels with high luminance levels. Minimization can be uniquely determinate as in [21], but for some non-linear TMOs cannot be inverted, since inversion implies finding zeros in a non-linear function. In order to keep our framework as general as possible we decided to use numerical methods such as Levenberg-Marquadt minimization. The parameters that are optimized during the minimization process are α and L_{white}, which are initialized using automatic estimation as presented in [22]. This approach worked well in practice.

The parameters α and L_{white} are used to tone map the HDRT. We then compress this LDRT using a standard traditional method. Other TMO and iTMO pairs can be used.

4 Implementation

In this section we describe the implementation of our framework and certain implementation-relevant design choices taken. We present two implementations, one based on DXT1 which compresses at 4 bpp and another one based on DXT5 which stores the residuals in the channel reserved for alpha values, for a total of 8 bpp.

4.1 Color Space Transform

A color transformation in compression algorithms is usually important because some properties can be exploited for assigning more bits to more important color channels. In certain HDRT compression methods, such as in [20, 25, 28], more bits are assigned to the luminance channel rather than the chroma channels. Similarly, we could have adopted the same approach in our implementation, but we use standard RGB color space for two reasons. Firstly, the TMO that we used compresses only luminance, and colors are scaled as can be seen from Equation 1. Note that is similar to the LUVW color space proposed in [28]. The second reason is the decoding

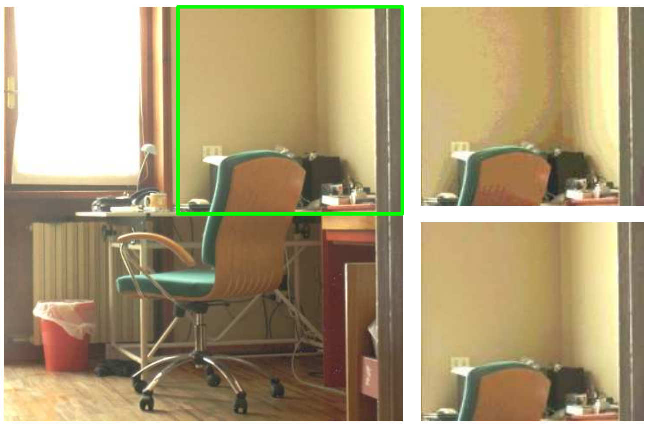


Figure 3: An example of wrong parameters selection: on the left the tone mapped image 15 from Figure 6 with a region of interest in green. On the right top a close-up of the region of interest after reconstruction using non estimated parameters for the (inverse) tone mapping operator. On the bottom right the region of interest after reconstruction using estimated parameters. As can be seen a non-automatic estimation of parameters may produce additional quantization artifacts because not all 8 bits are used.

efficiency, as we show in Section 4.4. If we were to apply for example Luv ([25]) or Yxy color space (or other color spaces which concentrate information in the luminance channel) the total number of operations would increase by at least four multiplications and two additions to calculate the final color of Luv.

4.2 S3TC for LDR Textures Encoding

The current de facto standard, as implemented on graphics hardware is S3TC/DXTC [9]. This is what we use as part of our own implementation. However, when a new standard will be available it can easily be updated in our system. DXTC works on 4×4 pixel tiles, where for each tile two 16 bit colors are calculated (5 bit for the red channel, 6 bit for the green channel and 5 bit for the blue channel), and 2 bits for each pixel are used for interpolating from these two colors. In our work we used two variants of DXTC: DXT1 and DXT5. DXT1 uses only a 1 bit alpha channel and is very compact (64 bits per 4×4 tile) resulting in 4 bpp. DXT5 uses an additional 64 bits for encoding an 8 bit alpha channel (128 bits per 4×4 tile) resulting in 8 bpp. In our implementations we use DXT1 for our simple implementation, and DXT5 to store residuals in the bits reserved for the alpha channel.

4.3 Residuals

For improved quality in one of our implementations we make use of the alpha channel to store residual information. As can be expected, there is a difference between the reconstructed HDRT and the original HDRT which can be expressed as:

$$\delta_a = L_{\text{ori}} - L_{\text{rec}} \qquad \delta_m = L_{\text{ori}} / L_{\text{rec}} \tag{4}$$

where δ_a is the additive difference, δ_m the multiplicative difference, L_{ori} the luminance of the original HDRT and L_{rec} the luminance of the reconstructed HDRT. We did not consider residuals for the colors because we would need another texture to store these. A TMO usually compresses only the luminance channel, keeping the color information so the errors in colors due to quantization during LDR encoding is lower than errors in the luminance channel. We use additive residuals, δ_a, since it has been shown that they do not introduce noticeable artifacts compared to multiplicative residuals δ_m [28]. We store δ_a quantized linearly into 8 bit in the alpha channel which is subsequently compressed by DXT5.

```
1  sampler2D texTMO;
2  sampler1D texITMO;
3  const float3 LUMINANCE= float3(0.21,0.72,0.07);
4
5  float3 tex2DITMO(float2 texCoords)
6  {
7      float4 tmoValue= tex2D(texTMO,texCoords);
8      float L= dot(tmoValue.rgb,LUMINANCE);
9      float Lw= tex1D(texITMO,L);
10     return tmoValue.rgb*(Lw+texTMO.a)/L;
11 };
```

Listing 1: tex2DITMO is the function which is used in a shader for decoding a compressed texture using our framework. This function takes 2 fetches to texture, 1 dot product, 1 multiplication, 1 division.

4.4 The Decoding Stage

For our decoding stage we implemented as a shader the iTMO from [1] described in Section 3.1. To speedup on-the fly calculations, we pre-compute, for each texture, Equation 2 into a 1D texture with 256 half floating point values (512 bytes). Our fragment shader program, `tex2DITMO`, is shown in Listing 1. When an HDRT lookup is called, our function firstly fetches the compressed tone mapped HDRT value, `texTMO` (line 8), and calculates the luminance, `L` (line 9). The luminance is then used as a lookup into the pre-calculated 1D texture, `texITMO` (line 10), and used to expand the value back into a high dynamic range (line 11). Finally, we add the residuals stored in the alpha channel.

4.5 Analysis of Error for Linear/Bi-Linear Interpolation

In our implementation we used bilinear, mip-mapping filtering performed on the `texTMO`. The used iTMO is not a linear function and this leads to errors during these filtering operations. However, the error introduced does not produce noticeable artifacts as in the case of interpolation in RGBE or RGBS. This is due to two reasons, the first is that we are not interpolating exponents, such as in RGBE, but we are interpolating only base values of a function. The second reason is that the iTMO is a monotonically increasing and continuous function. Therefore the interpolated value y, between two values a and b, will still be continuous without producing edges, see Figure 5. Finally residuals can be up-sampled and added without problems because they are additive, so they are linear.

For the one dimensional case, correct linear up-sampling of luminance is defined as:

$$L(a,b,x) = xg(a) + (1-x)g(b) \tag{5}$$

where g is the inverse tone mapping operator, a is the first value that we want to interpolate, b is the second one, and x is the in between factor of interpolation, where $\{a,b,x\} \in [0,1]$. In our implementation we firstly up-sample compressed luminance:

$$\hat{L}(a,b,x) = g(xa + (1-x)b) \tag{6}$$

So the error is given by:

$$\varepsilon(a,b,x) = \left\|L - \hat{L}\right\| = \left\|xg(a) + (1-x)g(b) - g(xa + (1-x)b)\right\| \tag{7}$$

Substituting g, Equation 2, in Equation 7 and simplifying it, we obtain:

$$\varepsilon(a,b,x) = \left\| \frac{L^2_{\text{white}} L_{\text{h}}}{2\alpha} \left(x\sqrt{(1-a)^2 + \frac{4a}{L^2_{\text{white}}}} + (1-x) \sqrt{(1-b)^2 + \frac{4b}{L^2_{\text{white}}}} - \sqrt{(1-xa-(1-x)b)^2 + \frac{4(xa+(1-x)b)}{L^2_{\text{white}}}} \right) \right\|$$

$\varepsilon(a,b,x)$ depends on the interval [a, b] that we want to interpolate and parameters which are specific for each image: L_h, L_{white}, and α. For an empirical analysis we took the worst case Image 3, in our image set Figure 6. We analyzed the error for two interpolation cases, an average interpolation interval in which the variation of the interval $b-a=0.3$ [0.3, 0.6], and the worst case, the interval [0, 1]. In Figure 4 the error function is plotted for these cases. As it can see the error is quite low in the average interval [0.3, 0.6]. On the other hand the error is quite high for the interval [0, 1], but as soon the interval is lowered a bit the error becomes again quite low, see Figure 4. Note that there are few pixels which are mapped to 1 which corresponds to the maximum value of the HDR texture.

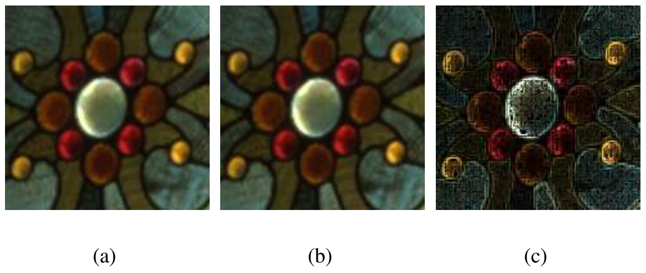


Figure 5: An example of bilinear up-sampling using our method for Image 11 from Figure 6: a) simulated bilinear filter in the shader b) up-sampling before applying inverse tone mapping function c) difference between a) and b) which is scaled 20 times.

5 Results

We implemented our compression framework in Matlab R2007b using the Levenberg-Marquadt function implemented in the Optimization toolbox. During the encoding stage, we always reached the local minimum using our set of images. Minimization (Equation 3) takes less than few seconds using a 748×1128 texture. The decoding function was integrated into an existing Direct3D9 application. This step is straightforward and can be easily integrated into similar real-time graphics applications. We tested our framework, compression and decompression, on an Intel Pentium M 1.76 Ghz equipped with 1.5 GB of memory and a GeForceGO 7300 with 256 MB.

5.1 Quality Metrics

In order to evaluate our work we use common metrics used to evaluate HDR images. We used HDR Visual Difference Predictor (VDP), a psychophysical metric, Root Mean Square Error (RMSE) in the $\log_2$[RGB] domain, and Multi-Exposure Peak Signal Noise Ratio (mPSNR), both of which are objective metrics.

The VDP [4] takes into account limitations in the human visual system rather than just physical values when comparing images. We use an extension to VDP, HDR-VDP which is specialized in comparing HDR images [15, 18]. HDR-VDP outputs an image showing per-pixel false coloring highlighting the perceptual difference between the images. This resulting color-coded image can be summarized the results using two values. The first is the percentage of different pixels detected with the probability of 75% ($P(X) \geq 0.75$). The second is the percentage of different pixels detected with the probability of 95% ($P(X) \geq 0.95$).

The RMSE in the $\log_2$[RGB] domain was proposed by Xu et al. [33], which is defined as follows:

$$RMSE(I,\hat{I}) = \sqrt{\frac{1}{n}\sum_{i,j}\left(\log_2\frac{R_{ij}}{\hat{R}_{ij}}\right)^2 + \left(\log_2\frac{G_{ij}}{\hat{G}_{ij}}\right)^2 + \left(\log_2\frac{B_{ij}}{\hat{B}_{ij}}\right)^2} \quad (8)$$

where I is the reference image and (R,G,B) its red, green and blue channels, $\hat{I}$ the comparison image and $(\hat{R},\hat{G},\hat{B})$ its channels, n the number of pixels of the two images. A small RMSE value means that image $\hat{I}$ is close to the reference, zero means that they are the same, while a high value means that they are very different.

The final metric is mPSNR, introduced in Munkberg et al. [20], this takes a series of exposures which are tone mapped using a simple gamma curve:

$$T(X,c) = \left[255(2^c X)^{\frac{1}{\gamma}}\right]_0^{255} \quad (9)$$

where c is the current f-stop, X is a color channel, and $\gamma = 2.2$. Then the classic Mean Square Error (MSE) is computed:

$$MSE(I,\hat{I}) = \frac{1}{w \times h \times p}\sum_{c=1}^{p}\sum_{i,j}\left(\Delta R_{ij,c}^2 + \Delta G_{ij,c}^2 + \Delta B_{ij,c}^2\right) \quad (10)$$

where p is the number of used exposures, n is the number of pixel in the image, $\Delta R_{ij,c} = T(R_{ij},c) - T(\hat{R}_{ij},c)$ for the red color channel, and so on for green and blue channels. Finally the PSNR is calculated using the standard formula:

$$mPSNR(I,\hat{I}) = 10\log_{10}\left(\frac{3\times 255^2}{MSE(I,\hat{I})}\right) \quad (11)$$

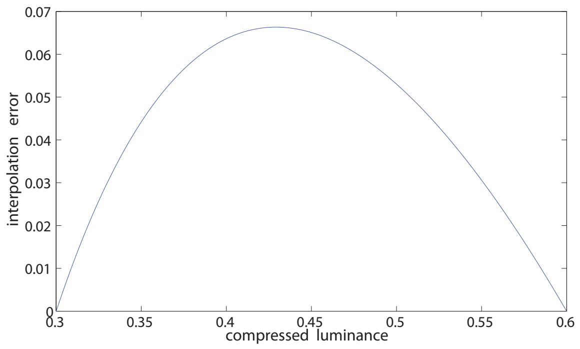
We determined automatically the p exposures. We used all exposures which generate an image that has a mean luminance less than or equal to 0.9, and greater than or equal to 0.1. These two thresholds were chosen after an examination of p values presented in Munkeberg et al. [20].

5.2 Comparisons

We compared our method with two other methods that are currently used in real-time applications, RGBE and Wang et al. [28]. In addition to these methods we compared our technique with other HDRT compression methods [20, 25] which are not currently supported by current OpenGL2/Direct3D9 and OpenGL3/Direct3D10 class hardware.

We compressed 22 HDRTs using our framework with and without residuals, these images were originally stored in the OpenEXR format (48 bpp). Figure 7 shows the values obtained for each picture using the metrics $\log_2$[RGB] RMSE, mPSNR and HDR-VDP, plotted as graphs. Table 1, summarizes these results with the average value for each of the metrics over the total number of pictures. Furthermore, Table 1, presents a second value which serves to highlight the quality as a function of the compression. These results must be interpreted as less is better for HDR-VDP and $\log_2$[RGB] RMSE, while more is better for mPSNR.

It is clear from the results that RGBE performs best in terms of quality in all metrics. However, when compared as a function of quality, it is best only in the case of the HDR-VDP metric. Furthermore, RGBE has problems when it comes to filtering and performance, only on G80 graphics cards RGBE filtering is implemented in hardware. The hardware methods [20, 25] perform better on average than our methods for straight out values, but less so as a function of quality. However, as mentioned earlier, these methods need special hardware to run. Our methods perform reasonably well, overall, they are better than the method of Wang et al. [28], more so when considering the compression as a function of quality. Our



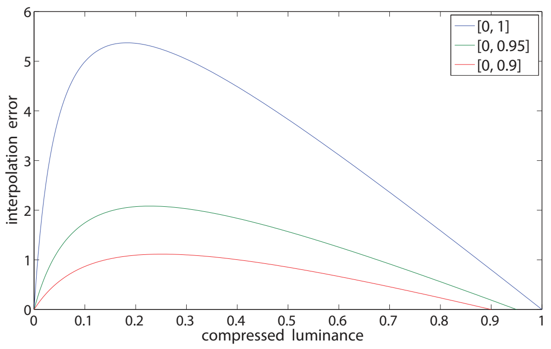


Figure 4: An example of evaluation of the error function for Image 3 from Figure 6: on the left hand side graph the error function for linear interpolation in the interval [0.3, 0.6], note the error is quite low. On the right hand side graph the error function for linear interpolation in the interval [0, 1], the error is quite high, but as soon as we lower down a bit the interval to [0, 0.95] or [0, 0.9] the error becomes acceptable.

	Roimela et al. [25]	Munkberg et al. [20]	Wang et al. [28]	Our method without residuals	Our method with residuals	RGBE*
Bits per pixel (Bpp)	8	8	16	4	8	32
Frames per seconds (FPS)	NA	NA	309	314	312	201
HDR-VDP average	0.0609	0.0014	1.333	0.815	0.1	0.0
HDR-VDP $\times$ Bpp	0.4872	0.0112	21.328	3.26	0.8	0.0
HDR-VDP / FPS	NA	NA	3.23e-3	2.59e-3	3.2e-4	0.0
$\log_2$[RGB] RMSE average	0.412	0.342	0.277	0.581	0.348	0.0623
$\log_2$[RGB] RMSE $\times$ Bpp	3.296	2.736	4.432	2.324	2.784	1.993
$\log_2$[RGB] RMSE / FPS	NA	NA	8.9e-4	1.85e-3	1.1e-3	3.1e-4
mPSNR average	84.43	89.31	81.12	77.12	83.11	99.03
mPSNR / Bpp	10.554	11.1638	5.07	19.28	10.3888	3.09
mPSNR $\times$ FPS	NA	NA	25066.1	24215.7	25930.32	19905.03

Table 1: Comparisons: this table shows the ratio of the average values of the used metrics and the bits per pixel, and as well for frames per second (FPS). We did not calculate FPS for Munkberg et al. [20] and Roimela et al. [25] because they can not be implemented using current hardware. * Note: on a G80 series RGBE is supported in hardware with filtering so it should scale with similar or better performances in fps compared to our method.

method without residuals seems to offer the best compression to quality ratio in the cases of mPSNR and $\log_2$[RGB] RMSE, mainly due to the high compression ratio of 4 bpp.

5.3 Real-time Graphics Application

Decoding speed is another important parameter that has to be taken into account for real-time applications. In modern applications such as games many texture fetches are needed to calculate post-processing filters. To show the efficiency of our implementation we timed the decoding time of all methods using a simple Direct3D9 application where bilinear filtering and mip-mapping were applied to textures. Real-time performance is summarized in Table 1. These values are timed from the simple application shown in Figure 1. Our methods obtained the fastest times, running without residuals at 314 fps, and with residuals at 312 fps. RGBE was the slowest, at 201 fps, due to the fact that bilinear filtering and mip-mapping has to be emulated in a shader because otherwise artifacts at the edges are present in results, because in these areas the exponent changes rapidly. Wang et al. [28] is nearly as fast as our methods, 309 fps, but it uses more operations: 3 multiplications, 3 additions, 1 dot product, 2 texture fetches compared to our method which uses 1 multiplication, 1 addition, 1 dot product, 1 division and 2 texture fetches.

An important issue is the caching during the second texture fetch in Listing 1, since it depends on the first texture lookup. Since our 1D texture is quite small (512 bytes) it remains in the GPU cache. To verify this we render a simple scene with 16 Happy Buddhas with one single texture and the same scene with 16 different textures. Performances were timed, and the results are 18.13 fps for the single texture application, and 17.91 fps for the application using 32 different textures. So the difference in terms of fps is only 5, which is quite low and shows that our approach does not harm caching.

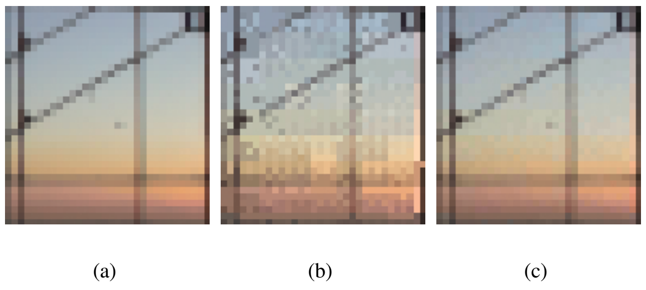


Figure 8: A close-up of the central window in image 12 in Figure 6. Note our method does unfortunately inherit color mixing artifacts from S3TC at edges and in a color gradient. a) Original HDR image. b) Compressed image without residuals, color mixing decreases the quality of luminance. c) Compressed image with residuals.

6 Conclusion and Future Work

In this paper we have presented a compression framework for HDR textures that can be implemented on current graphics hardware. Our framework is based on tone mapping for the encoding stage and inverse tone mapping for the decoding stage. In our implementation we used S3TC for compressing LDR textures resulting in a compression rate of 8 bpp for high quality HDRTs and 4 bpp for fairly good HDRTs. S3TC is the main limit to the quality, our methods

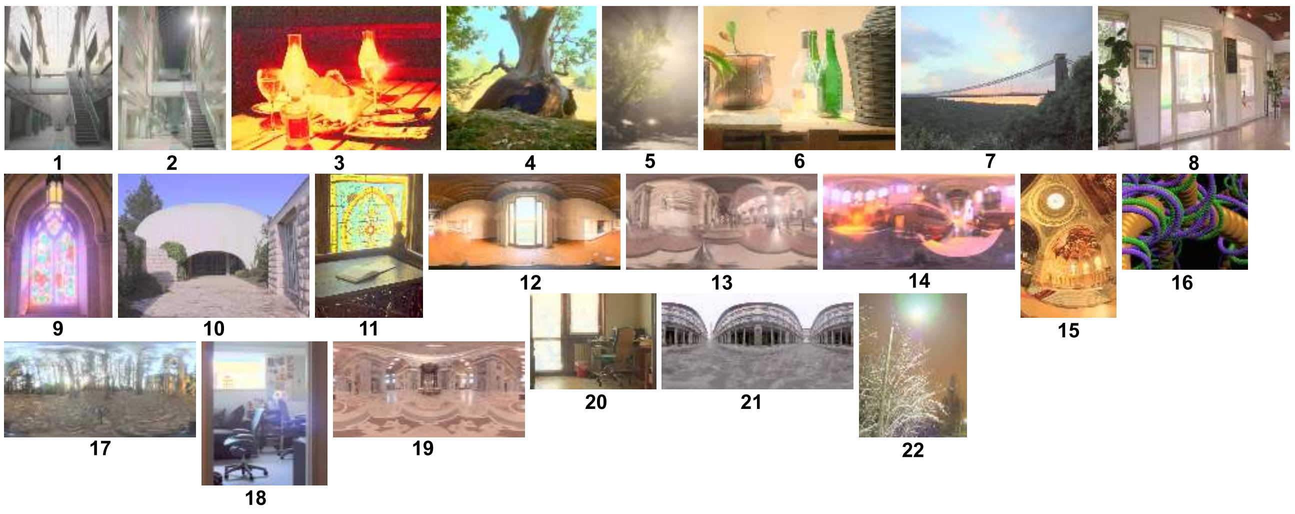


Figure 6: Tone mapped HDR textures used in our compression experiments.

are affected by classic problems of this compression scheme, see Figure 8. Our method has been designed for natural HDR textures, and tested with a maximum contrast ratio of $1.5 \times 10^8 : 1$ (in Image 11 from Figure 6). Images with contrast ratio over $10^9 : 1$ are a difficult scenario in which quantization artifacts would be noticeable.

We have shown that our method has a good quality compared to existing available solutions for real-time applications on current hardware, RGBE and Wang et al. [28], and state of art that needs special hardware Munkberg et al. [20] and Roimela et. al [25]. A further advantage of our method over the other compression methods such as RGBE is that it can handle filtering without visible artifacts (bilinear and mip-mapping) and it can be implemented efficiently on GPUs. Moreover, the decoding shader is easy to integrate into existing real-time applications. Compared to Wang et al. [28] we use half the memory and we can avoid quantization artifacts when the splitting axis can not find an optimal separation, as in the case of Saint Peter Basilica's lightprobe, see Figure 1.

For future work we would like to exploit new graphics hardware features to compress on the fly. This would be helpful for compressing the main rendering target which currently uses HD resolutions. This can take up a large amount of memory, especially for anti-aliasing or temporary post processing buffers.

Acknowledgements

We thank, Jacob Munkberg, Kimmo Roimela, Lvdi Wang and Peter Pike Sloan for sending to us their encoders, decoders and relative instructions for compressing images. We thank, Jacob Munkberg, Paul Debevec, Jack Tumblin, Greg Ward, Dani Linchiski, and Industrial Light and Magic for HDR images used in our paper. We thank anonymous reviewers and Alessandro Artusi for their useful comments. Finally we thank Stanford's Graphics Group for the Happy model from the Stanford 3D repository.

This work reported in this paper has formed part of EPSRC grant EP/D032148 whose funding and support is gratefully acknowledged.

References

[1] F. Banterle, P. Ledda, K. Debattista, and A. Chalmers. Inverse tone mapping. In *GRAPHITE '06*, pages 349–356, New York, NY, USA, 2006. ACM Press.

[2] A. C. Beers, M. Agrawala, and N. Chaddha. Rendering from compressed textures. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 373–378, New York, NY, USA, 1996. ACM Press.

[3] D. Blythe. The direct3d 10 system. *ACM Trans. Graph.*, 25(3):724–734, 2006.

[4] S. Daly. *The visible differences predictor: an algorithm for the assessment of image fidelity*. MIT Press, Cambridge, MA, USA, 1993.

[5] P. Debevec. Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 189–198, New York, NY, USA, 1998. ACM Press.

[6] S. Fenney. Texture compression using low-frequency signal modulation. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 84–91, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[7] J. H. V. Hateren. Encoding of high dynamic range video with a model of human cones. *ACM Trans. Graph.*, 25(4):1380–1399, 2006.

[8] Industrial Light & Magic. OpenEXR. *http://www.openexr.org*, 2002.

[9] K. Iourcha, K. Nayak, and Z. Hong. System and method for fixed-rate block-based im- age compression with inferred pixel values. *United States Patent 5,956,431*, 1997.

[10] J. R. Isidoro and P. V. Sander. Animated skybox rendering and lighting techniques. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, pages 19–22, New York, NY, USA, 2006. ACM Press.

[11] M. Kilgard, P. Brown, and J. Leech. GL_EXT_texture_shared_exponent. In *OpenGL Extension*. http://www.opengl.org/registry/specs/EXT/texture_shared_exponent.txt, 2007.

[12] G. W. Larson. Logluv encoding for full-gamut, high-dynamic range images. *Journal of Graphics Tools*, 3(1):15–31, 1998.

[13] S. Lefebvre and H. Hoppe. Compressed random-access trees for spatially coherent data. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)*. Eurographics, 2007.

[14] Y. Li, L. Sharan, and E. H. Adelson. Compressing and companding high dynamic range images with subband architectures. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 836–844, New York, NY, USA, 2005. ACM Press.

[15] R. Mantiuk, S. Daly, K. Myszkowski, and H.-P. Seidel. Predicting visible differences in high dynamic range images - model and its calibration. In B. E. Rogowitz, T. N. Pappas, and S. J. Daly, editors, *Human Vision and Electronic Imaging X, IS&T SPIE's 17th Annual Symposium on Electronic Imaging*, volume 5666, pages 204–214, 2005.

[16] R. Mantiuk, A. Efremov, K. Myszkowski, and H.-P. Seidel. Backward compatible high dynamic range mpeg video compression. *ACM Trans. Graph.*, 25(3):713–723, 2006.

[17] R. Mantiuk, G. Krawczyk, K. Myszkowski, and H.-P. Seidel. Perception-motivated high dynamic range video encoding. *ACM Trans. Graph.*, 23(3):733–741, 2004.

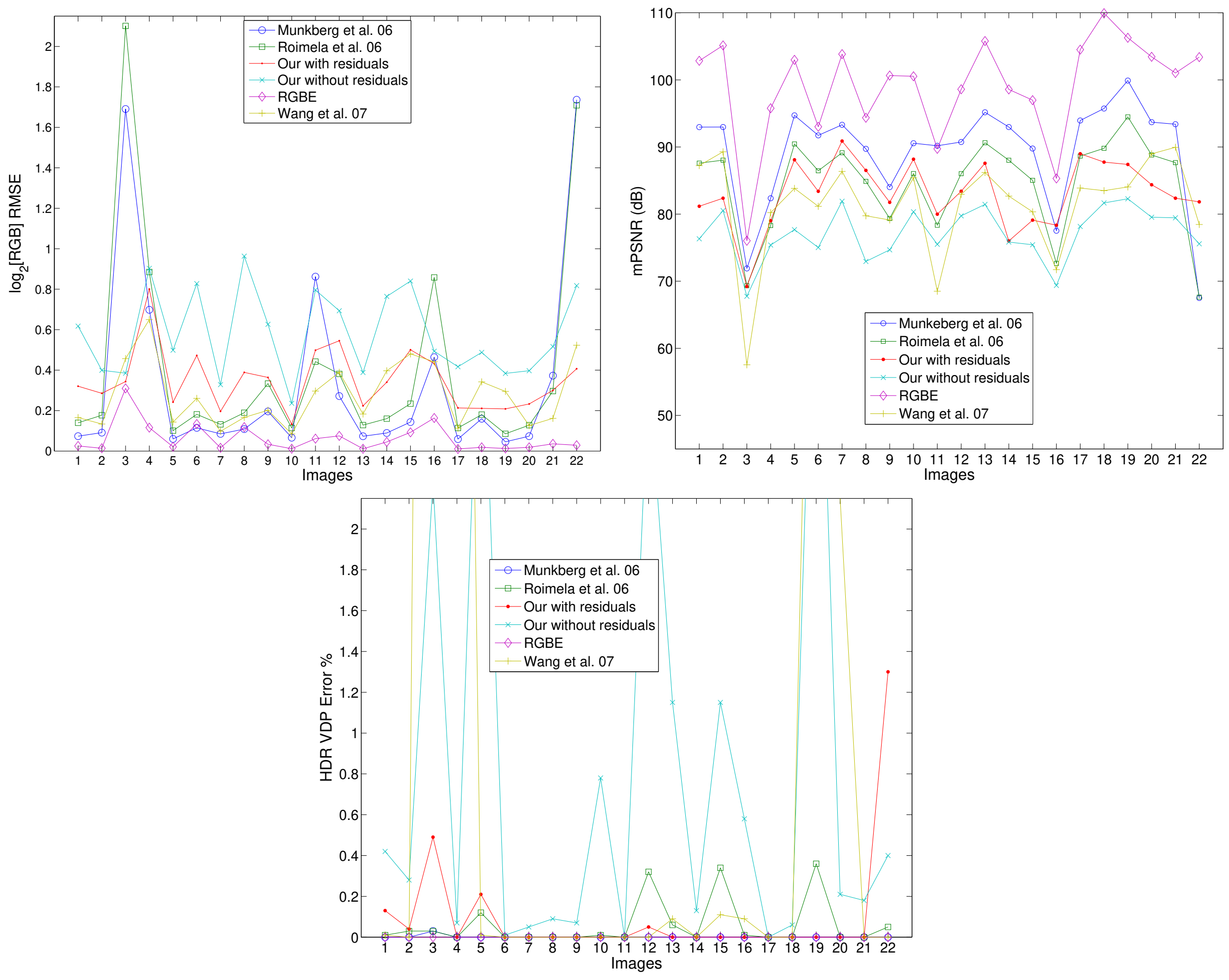


Figure 7: The results of the comparisons using the three metrics using the set in Figure 6: in the top left results for RMSE in $\log_2$[RGB], in the top right results for mPSNR, and in the bottom results for HDR-VDP with probability of detection equals to 0.75.

[18] R. Mantiuk, K. Myszkowski, and H.-P. Seidel. Visible difference predicator for high dynamic range images. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, pages 2763–2769, 2004.

[19] G. McTaggart, C. Green, and J. Mitchell. High dynamic range rendering in valve's source engine. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, page 7, New York, NY, USA, 2006. ACM Press.

[20] J. Munkberg, P. Clarberg, J. Hasselgren, and T. Akenine-Möller. High dynamic range texture compression for graphics hardware. *ACM Trans. Graph.*, 25(3):698–706, 2006.

[21] M. Okuda and N. Adami. Raw image encoding based on polynomial approximation. *IEEE International Conference on Acoustics, Speech and Signal Processing, 2007. ICASSP 2007*, pages 15–31, 2007.

[22] E. Reinhard. Parameter estimation for photographic tone reproduction. *J. Graph. Tools*, 7(1):45–52, 2002.

[23] E. Reinhard, M. Stark, P. Shirley, and J. Ferwerda. Photographic tone reproduction for digital images. *ACM Trans. Graph.*, 21(3):267–276, 2002.

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

[25] K. Roimela, T. Aarnio, and J. Itäranta. High dynamic range texture compression. *ACM Trans. Graph.*, 25(3):707–712, 2006.

[26] K. Roimela, T. Aarnio, and J. Itäranta. Efficient high dynamic range texture compression. In *SI3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games*, pages 207–214, New York, NY, USA, 2008. ACM.

[27] J. Ström and T. Akenine-Möller. ipackman: high-quality, low-complexity texture compression for mobile phones. In *HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, New York, NY, USA, 2005. ACM Press.

[28] L. Wang, X. Wang, P.-P. Sloan, L.-Y. Wei, X. Tong, and B. Guo. Rendering from compressed high dynamic range textures on programmable graphics hardware. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 17–24, New York, NY, USA, 2007. ACM Press.

[29] G. Ward. Real pixels. *Graphics Gems*, 2:15–31, 1991.

[30] G. Ward. JPEG-HDR: A backwards-compatible, high dynamic range extension to JPEG,. In *CIC 13th: Proceedings of the Thirteenth Color Imaging Conference*. The Society for Imaging Science and Technology, 2005.

[31] G. Ward. A general approach to backwards-compatible delivery of high dynamic range images and video. In *CIC 14th: Proceedings of the Fourteenth Color Imaging Conference*. The Society for Imaging Science and Technology, 2006.

[32] G. Ward and M. Simmons. Subband encoding of high dynamic range imagery. In *APGV '04: Proceedings of the 1st Symposium on Applied perception in graphics and visualization*, pages 83–90, New York, NY, USA, 2004. ACM Press.

[33] R. Xu, S. N. Pattanaik, and C. E. Hughes. High-dynamic-range still-image encoding in jpeg 2000. *IEEE Comput. Graph. Appl.*, 25(6), 2005.