Graphics Hardware (2008)
David Luebke and John D. Owens (Editors)

DHTC: An Effective DXTC-based HDR Texture Compression Scheme

Wen Sun[1†] Yan Lu[2] Feng Wu[2] Shipeng Li[2]

[1]University of Science and Technology of China [2]Microsoft Research Asia

Abstract

In this paper, we propose a simple yet effective scheme (named DHTC) for HDR (high dynamic range) texture compression based on the popular LDR (low dynamic range) texture compression scheme – S3TC/DXTC. In the proposed scheme, the original HDR texture is first pre-processed with adaptive color transform and local dynamic range reduction. Then a color distribution linearization process and a joint-channel texture coding process are applied iteratively to generate the compressed HDR texture at 8 bpp. These techniques lead to near lossless visual quality comparable to and even better than the state-of-the-art HDR texture compression schemes. Since DHTC is built upon the ubiquitous DXTC, dedicated DHTC hardware design only needs moderate extension on current hardware. Furthermore, the proposed DHTC format is not only suitable for HDR textures, but also LDR textures with alpha channels. We believe this paper provides a unique solution to meet all the practical requirements for HDR and LDR texture compression.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Texture

1. Introduction

HDR imaging technologies are bringing us to a new era of recording and reproducing the real world. While traditional low dynamic range (LDR) images only contain device-referred pixels in a very limited color gamut, HDR images provide the real radiance values of natural scenes. By using HDR textures, more accurate lighting and post-processing are enabled, resulting in unprecedented reality in rendering [War94, Deb98, CTHD01]. Thus, supporting HDR textures has become the trend in designing both graphics hardware and APIs. At the same time, LDR textures continue to be indispensable to efficiently support the decal maps that do not usually require an expanded dynamic range.

The main problem of using textures is their large size. LDR textures in typical 24 bpp raw RGB format just consume too much storage and bandwidth. The situation is even worse for HDR textures which are usually in half-floating or floating point format and cost 2 or 4 times more space than the raw LDR textures. Huge texture size limits the frame rate and scene complexity in real-time rendering. Texture compression (TC) techniques can effectively reduce the memory storage and bandwidth requirements at some computational cost. For LDR textures, many compression schemes have been devised, some of which, such as the de facto standard DXTC (S3TC), have been widely supported by commodity graphics hardware. The common idea behind these practical LDR TC schemes is to use color (joint-channel) quantization for the texels within a small block. Thus the correlations among different color channels can be efficiently utilized, and the decoding process is also very fast.

On the other hand, HDR TC is a relatively new research topic with only a limited number of contributions so far [MCHAM06, MCHAM07, RAI06, RAI08, WWS*07]. In general, the existing HDR TC schemes, aiming at either future hardware implementation or current hardware accommodation, achieve good quality at a compressed rate of 8 bpp or 16 bpp, and make real-time HDR rendering affordable for commodity rendering systems. However, all these schemes share the common idea to compress luminance and chrominance information independently. The separate-channel coding framework, which divides the limited bit budget into luminance bits and chrominance bits statically and leaves the

† This work has been done while the author was with Mirosoft Research Asia as an intern.

cross-channel correlation unused, constrains them to achieve higher compression ratio or better visual quality.

A novel HDR TC scheme is proposed in this paper. Our basic idea is to convert the HDR compression problem into an LDR-like compression problem, by utilizing the locality property within each texture block. More specifically, we carefully design some pre-processing units to handle special issues of HDR textures, such as the expanded dynamic range, non-linear texel color distribution, etc., to make LDR TC techniques (specifically, the DXTC linear fitting approach in this paper) an effective compression kernel. To differentiate from other schemes, we name the proposed scheme *DXTC-based HDR Texture Compression* or *DHTC* in short.

In the following, we will first describe our compression framework, which enable us to make a joint-channel bit allocation and remove cross-channel correlations. Under the framework, we present the proposed DHTC texture format. HDR textures in DHTC format are of near lossless visual quality at a compressed rate of 8 bpp. In particular, DHTC preserves much higher fidelity than the state-of-the-art HDR TC schemes in tricky areas, especially the areas with rich colors. Since DHTC is built upon the ubiquitous DXTC, dedicated DHTC hardware design only needs moderate extension on current hardware. Furthermore, the proposed DHTC format also support compression of LDR textures with alpha channels. Thus we arrive at a unified compressed format for HDR textures, LDR textures and alpha maps, using the same set of decoding logic based on the existing DXTC hardware.

2. Related Work

2.1. LDR Texture Compression

LDR TC was introduced to rendering systems to reduce storage and bandwidth consumption [BAC96, KSKS96, TK96]. Beers et al. [BAC96] address a few key issues in TC algorithm design, including random accessibility, decoding speed, and the tradeoff between compression rate and visual quality. A simple vector quantization (VQ) method with a compression ratio of 35:1 is also proposed. However, in the VQ scheme, two memory accesses are needed to decode a single texel, which doubles the memory bandwidth cost.

Compared to VQ, the local palette approach provides more practical LDR TC methods, e.g., the de facto standard DXTC (S3TC) [INH99]. In DXTC, textures are divided into 4x4 independent blocks. Each block stores two 16-bit base colors (both in RGB565 format) and sixteen 2-bit color indices. The base colors and other two interpolated colors form a block palette based on which a best matched color index is selected for each texel.

Methods to further improve LDR TC performance have been also attempted. In PVR-TC [Fen03], a high frequency but low precision modulation signal blends with two low-pass signals of the original texture to generate pertexel colors. Due to the continuity of the low frequency signals, block artifacts can be effectively alleviated. iPACKMAN [SAM05], as part of the OpenGL ES API, divides each 4x4 block into two 2x4 or 4x2 sub-blocks, each with one base color. And the luminance values are refined with modifiers in a modifier table pointed by the per-texel indices. Thanks to the fine granularity block division, iPACKMAN preserves better luminance detail than DXTC.

In summary, the local palette based LDR TC schemes provide highly compact textures with good enough quality for most graphics applications. They apply joint-channel compression to small texture blocks to efficiently exploit the local similarity. Although these schemes cannot be directly applied in HDR TC, the experience gained and techniques developed are instrumental in HDR TC scheme design.

2.2. HDR Texture Compression

The first usable HDR texture formats come from fixed rate pixel-wise HDR image encoding. A detailed overview of these formats is given by Ward [War05]. However, pixel-wise encoding methods cannot leverage the cross-pixel correlation and still have to spend tens of bits per pixel. Later, advanced HDR image [WS04, XPH05] and video [MKMS04, MEMS06] compression techniques with much higher compression ratio are developed. But the lack of random accessibility makes them unsuitable for HDR textures.

Analogous to LDR TC, block-wise compression methods have been proposed with a typical block size of 4x4. Wang et al. [WWS*07] present a 16 bpp scheme aiming at current generation GPUs. When encoding one block, they first convert texels in RGB space to LUVW space. And then data in LUVW channels are quantized and packed into two DXT5 blocks. While their scheme enables HDR rendering and native texture filtering on existing programmable graphics hardware, it only provides a relatively low compression ratio.

A simple 8 bpp solution is proposed by Roimela et al. [RAI06]. After color transform, luminance and chrominance signals are separately encoded. Luminance blocks in high dynamic range are compressed via predictive coding, leveraging the bit pattern property of floating point numbers. The chrominance values, which fall into the range of [0, 1], are down sampled and uniformly quantized. In their later work [RAI08], Roimela et al. apply predictive coding for chrominance channels and achieve even better quality while still keeping the decoding very simple.

Munkberg et al. [MCHAM06] give another 8 bpp HDR TC scheme. They transform RGB channels to LogYuv color space. Adaptive quantization is used in luminance encoding in either uniform mode or non-uniform mode for each block. For chrominance channels, horizontal or vertical subsampling is applied at first. Then some pre-defined shape patterns are used to fit the block chrominance distribution in

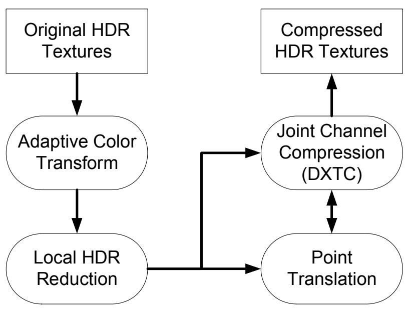

Figure 1: *Our HDR texture compression framework.*

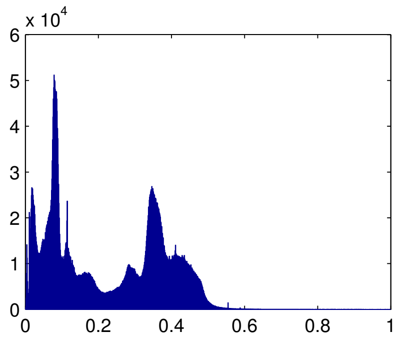

Figure 2: *Chrominance value distribution in [0, 1].*

the 2D chrominance plane. Based on the above scheme, a new compression mode is worked out to enhance the quality of textures with rich colors [MCHAM07]. The new mode allocates more bits to chrominance channels and avoids chrominance down-sampling. However, multiple modes with different bit layouts in a single format may complicate the decoding logic design.

The above 8 bpp block-based schemes are considered to be the state of the art in HDR TC. In general, they have achieved very good quality at a relatively low compressed rate. But they also have some disadvantages. First, they take no consideration on the compatibility with the existing LDR texture formats and hardware. Second, their separate-channel compression can hardly lead to a proper luminance/chrominance bit allocation suitable for every block, and leaves the cross-channel correlation unused.

3. Framework

To extend DXTC to HDR texture compression, we face two main challenges. First, DXTC is designed for 8-bit RGB channels and cannot directly handle HDR textures with the expanded bit depth. Second, the linear fitting approach used in DXTC is based on the local linearity assumption in LDR RGB space. But this assumption does not always hold for HDR contents during the compression process.

In previous work, researchers also tried to adapt DXTC to HDR textures [RAI06, MCHAM06, WWS*07]. But all these efforts fail to generate satisfactory results. It is mainly because the HDR contents are not properly processed for DXTC. We demonstrate a carefully designed DXTC-based HDR TC scheme can achieve comparable and even better performance than the state of the arts. Figure 1 depicts the proposed HDR TC framework, where HDR textures are processed block by block independently. First, we adaptively transform RGB texels into luminance and chrominance space. Then the transformed values in floating point format are quantized to integer levels with local dynamic range reduction. Finally, joint channel compression coupled with point translation helps to generate the compressed HDR textures.

3.1. Adaptive Color Transform

In HDR TC, a color transform is usually performed to aggregate the high dynamic range values into one component. A typical HDR color transform is as follows [MCHAM06]:

$$\begin{aligned} Y &= \sum_{t\in\{r,g,b\}} w_t C_t \\ S_t &= \frac{w_t C_t}{Y},\ t \in \{r,g,b\} \end{aligned} \tag{1}$$

Here Y is the luminance channel. $\{S_t\}$ are chrominance channels corresponding to R, G and B. $\{w_t\}$ are constant weights. Note that only the luminance channel as well as two out of the three chrominance channels has to be stored in the compressed textures to avoid redundancy. For instance, S_b does not need to be stored but RGB values still can be recovered via the inverse transform:

$$\begin{aligned} R &= S_r \times Y / w_r \\ G &= S_g \times Y / w_g \\ B &= (Y - w_r R - w_g G) \times Y / w_b \end{aligned} \tag{2}$$

However, for a lossy compression scheme, this exposes channel B to accumulated quantization errors, and sometimes these errors can be relatively very large. We define an *error accumulative channel* as one of the R, G, and B channels whose corresponding chrominance channel is not stored. Ideally, to best control the relative error, we should always assign the channel with largest value as the error accumulative channel, which can be indicated by a per-pixel color transform mode (***Ch_mode***) as:

$$Ch\_mode \equiv m = \underset{t\in\{r,g,b\}}{\operatorname{argmax}} \{S_t\} \tag{3}$$

Here the dominant chrominance channel S_m will not be encoded. This will guarantee the values of the two encoded chrominance channels fall in the range of [0, 0.5] and the relative accumulative error can be well controlled.

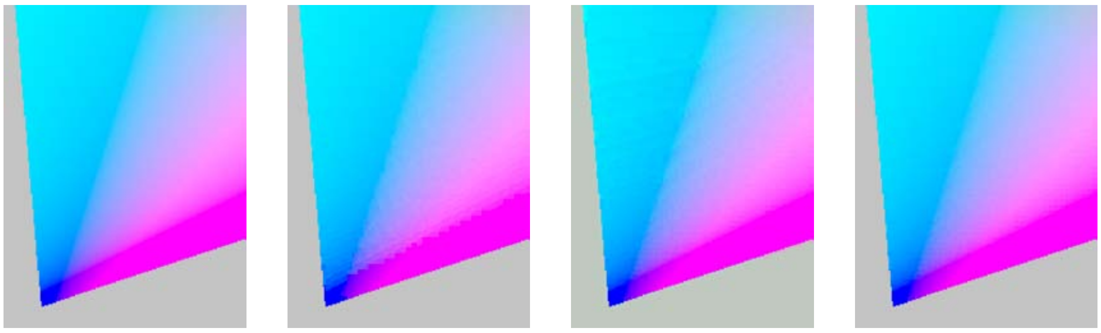

Figure 3: *An example comparing different UV coding methods in our scheme. From left to right: Original; Linear UV encoding; Log UV encoding; Adaptive linear/log UV encoding. Note that we avoid most banding artifacts in the rightmost result.*

In practice, we cannot afford the per-pixel ***Ch_mode*** overhead. Thus in the proposed scheme, we use a per-block ***Ch_mode*** instead as below:

$$Ch\_mode = \underset{t\in\{r,g,b\}}{\mathrm{argmax}} \{ \sum_{i\in\{all\ texels\ in\ a\ block\}} S_t^{(i)} \} \quad (4)$$

The per-block mode still capture the main idea of adaptive color transform, with the help of which the inverse color transform can be performed correctly, in a similar form as (2). Since the texels in a block share the same ***Ch_mode***, the data range of the two encoded chrominance channels (noted as U and V) becomes [0, 1]. But in fact, because of the local similarity, very few texels are of UV values larger than 0.5 after adaptive color transform. Figure 2 shows the UV distribution in our statistics.

To facilitate the following compression steps (especially the log encoding), we limit the transformed YUV values within tractable ranges. Channel Y is clamped into the range $[2^{-15}, 2^{16}]$, while channels UV are clamped into the range $[2^{-11}, 1]$. The clamping helps us avoid negative and near zero values as the input, and involves little visual artifact.

3.2. Local Dynamic Range Reduction

After adaptive color transform, values in luminance and chrominance channels are still in floating point format which are much harder to compress than integers. Hence in this stage we would like to quantize the floating point values into tractable integer levels. Similar floating point to integer conversion exists in the well studied tone mapping operations. However, after tone mapping, the dynamic range of the original HDR images is shrunk irreversibly. In order to preserve the original dynamic range and keep the accuracy as much as possible, we leverage the locality property within each texture block, and map floating point data to integers with local dynamic range reduction.

Luminance channel Y contains most of the HDR data. In HDR images, a luminance dynamic range of 4 orders of magnitude is common [War05]. We refer this as the frame (global) dynamic range. However, in texture compression, we do not manipulate texels in the whole frame all together, but divide and conquer textures block by block. We find the luminance dynamic range within a small block (i.e. the local dynamic range) seldom exceeds 2 orders. To reduce the luminance dynamic range, we first uniformly quantize the global luminance (ranging from 2^{-15} to 2^{16}) into 32 levels in log2 space. These levels serve as the anchor points, from which we select the luminance upper and lower bounds for each block. The bounds help to map absolute luminance in global high dynamic range into relative values in local low dynamic range. Then we can use 8-bit log encoding (i.e. linear quantization in log2 space) to handle the local dynamic range with desired accuracy.

As for the chrominance channels UV, adaptive color transform has significantly reduced the local dynamic range. Given the distribution in Figure 2, we have to find a suitable method to quantize UV values. Linear encoding minimizes absolute error and accounts for the error visibility, but may introduce huge relative error to texels with highly saturated colors. Considering HDR textures may be tone mapped at high exposures or used for lighting calculation, such artifact is unacceptable. In contrast, log encoding can best control the relative errors in all R, G, and B channels, and the quantization steps become coarser and coarser towards 1.0, which accords to the data distribution. Although log encoding generates good results for most cases, it may suffer from banding artifacts in some regions with smooth color variation. In our scheme, we adopt both linear encoding and log encoding for chrominance channels to avoid the above worst cases. For each block, 8-bit linear encoding and log encoding are alternatively selected by a UV coding mode, to minimize the reconstructed relative error. Figure 3 gives an example comparing different UV coding methods in our scheme.

3.3. Joint Channel Compression

After local luminance and chrominance dynamic range reduction, now we have 8-bit-depth data in all channels, which

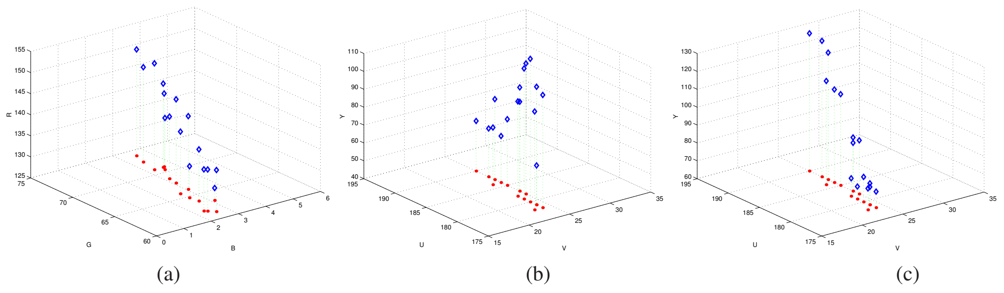

Figure 4: *An example of point translation. (a) Texel distribution of a typical HDR texture block in RGB space; (b) The distribution after color transform and local dynamic range reduction in local YUV space; (c) The reshaped distribution after point translation. Blue diamonds denote texels in the 3D spaces, and red points are their projection in the bottom plane.*

M_idx \ T_idx	0.	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.	15.
0.	1	1	1	1	2	2	2	2	4	4	4	4	8	8	8	8
1.	-1	-1	-1	-1	-2	-2	-2	-2	-4	-4	-4	-4	-8	-8	-8	-8
2.	2	3	4	5	4	6	8	10	8	12	16	20	16	24	32	40
3.	-2	-3	-4	-5	-4	-6	-8	-10	-8	-12	-16	-20	-16	-24	-32	-40
4.	3	5	7	9	6	10	14	18	12	20	28	36	24	40	56	72
5.	-3	-5	-7	-9	-6	-10	-14	-18	-12	-20	-28	-36	-24	-40	-56	-72
6.	4	7	10	13	8	14	20	26	16	28	40	52	32	56	80	104
7.	-4	-7	-10	-13	-8	-14	-20	-26	-16	-28	-40	-52	-32	-56	-80	-104

Table 1: *The modifier table used in DHTC.*

is very similar to the input data in LDR TC. Intuitively, we can directly apply DXTC here to further compress the integer levels. However, this will result in large distortion due to the non-linear pre-processing operations mentioned above. As we know, HDR textures also possess a local linearity property in RGB space that meets the underlying assumption of DXTC [WWS*07]. However, after the adaptive color transform and local dynamic range reduction, texels in the local YUV color space do not hold the linearity property any more, as shown in Figure 4 (a) and (b).

In the proposed scheme, we want to take advantage of DXTC, while avoiding the linear approximation of the non-linearly distributed texels. To solve this problem, we introduce a novel technique named *point translation* to help reshape the texel distribution (see Section 3.4). Then in the joint channel compression stage, we can simply apply DXTC to further compress the 8-bit integer levels. For each block, we select two base colors (***X0Y0Z0***, ***X1Y1Z1***) that can be seen as two end points of a line segment in the local YUV space. The end points and the trisection points of the line segment form a local palette. The texels within the block are mapped to the nearest palette colors (indicated by 2-bit color indices ***C_idx***) as their approximations.

3.4. Point Translation

Obviously, if we can force the texels within a block to redistribute along a line, DXTC scheme should work effectively even after the previous non-linear pre-processing steps. Our basic idea is to shift the points (i.e. texels) in the local YUV space, until they are closely along a line. Note that we cannot afford to indicate a free translation in the 3D space, which will cost too many bits. Considering the variation of block chrominance is usually very small, we can assume linear fitting still works for the chrominance channels. Hence, we just need to translate the points along Y axis to get a desired distribution, as shown in Figure 4 (c).

Our solution is to use a modifier table to perform the point translation. We offline design a constant modifier table containing 16 table entries, in which each entry contains 8 modifiers, as shown in Table 1. In the compressed HDR textures, we only need to store per-block table entry indices (***T_idx***) and per-texel modifier indices (***M_idx***), which address the proper modifier for each texel:

$$\textit{modifier} = \mathbf{mod\_table}[T\_idx][M\_idx] \tag{5}$$

In the table design, we consider both the performance and the implementation efficiency. First, we calculate the typical translation ranges of local texture blocks based on a large set of HDR images. These ranges are clustered into 16 categories, corresponding to 16 table entries (***T_idx***). Second, given a fixed translation range within each entry, we uniformly set the modifiers, corresponding to 8 entry items (***M_idx***). Thus, the modifier table can be formulized to be a function of ***T_idx*** and ***M_idx*** as below:

$$\begin{aligned} \textit{modifier} = (-1)^{(M\_idx \& 1)} \times 2^{(T\_idx >> 2)} \times \\ [1 + (T\_idx \& 3 + 1) \times (M\_idx >> 1)] \end{aligned} \tag{6}$$

In our testing, the compression performance is not sensitive to the modifier table definition as long as it covers all translation ranges.

The modifiers actually serve as the translation vectors. It can be seen that the modifier table covers a broad range translations, from short distances to long distances. And the point translation is performed as follows:

$$Y_{trans} = Clamp_{[0,255]}\{Y_{int} + \textit{modifier}\} \tag{7}$$

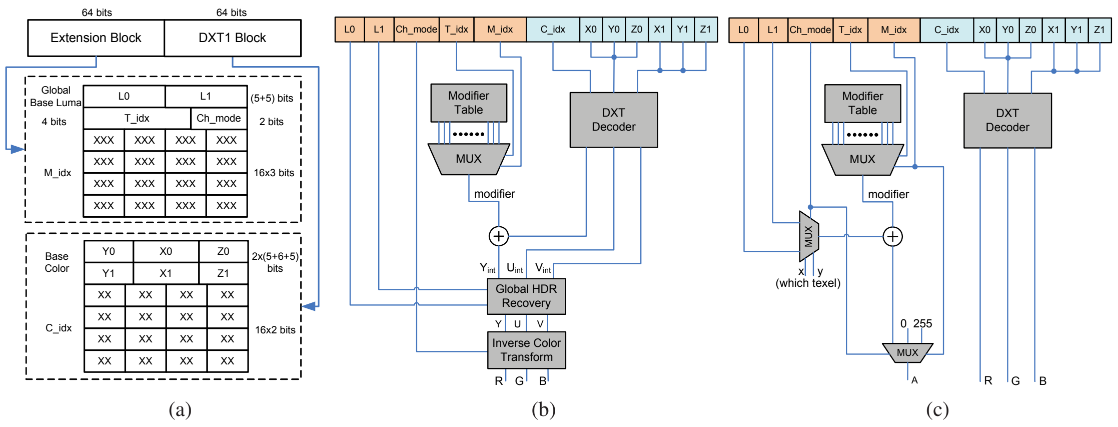

Figure 5: *DHTC texture format and decoding logics. (a) Bit layout of DHTC format; (b) DHTC decoding logic for HDR textures; (c) DHTC decoding logic for LDR textures with alpha channels.*

Y_{int} is the original luminance value, while Y_{trans} is the luminance value after point translation. Both are 8-bit integers.

In summary, the proposed HDR TC framework is composed of all the above modules as shown in Figure 1. First, the adaptive color transform not only aggregates most HDR information into the luminance channel but also reduces the accumulative errors. Then the transformed floating point texel values are mapped to integer levels without losing the high dynamic range by using local dynamic range reduction. Finally, the integer values are efficiently compressed via joint channel compression along with point translation, resulting in a very compact representation of the original textures.

4. Implementation

Under the proposed framework, we design a novel compressed texture format (DHTC format) suitable for both HDR and LDR contents. In the following we first present the bit layout of DHTC format, and then describe our HDR texture encoding and decoding process, as well as the accommodation for LDR textures with alpha channels.

4.1. Bit Layout

In DHTC format, textures are divided into 4x4 blocks that are compressed and decompressed independently. There are 128 bits in each block, resulting in a compressed rate of 8 bpp. One compressed texture block comprises of two parts, a DXT1 block and an extension block, each with a 64-bit budget. The bit layout is depicted in Figure 5 (a).

4.2. HDR Texture Compressor

The HDR texture encoder follows exactly the flowchart in Figure 1. The first two stages are straightforward. First of all, ***Ch_mode*** is determined for each block per equation (4), and the adaptive color transform is performed per equation (1). In the local dynamic range reduction module, the block luminance bounds are quantized to 5-bit global base lumas, ***L0*** and ***L1***, and log encoding is carried out in channel Y. For chrominance channels UV, either log encoding or linear encoding is performed to minimize the reconstruction error. Note that the 1-bit UV coding mode is implicit in the mutual order of ***L0*** and ***L1*** as shown in Figure 5 (a).

The joint channel compression and point translation stages take most of the encoding time. Although we have reduced the point translation from 3D space to 1D along Y axis, enumerating all the possible translations would cost too much time. In our implementation, we first project the texels to the UV plane, where the DXTC linear fitting is performed and the base chrominance values (***Y0Z0*** and ***Y1Z1***) as well as the color indices (***C_idx***) are figured out. The fixed color indices specify the relative position of each texel and much simplify the calculation followed. We then enumerate modifier table entries (***T_idx***), and find out the best base luminance values (***X0***, ***X1***) and modifiers (***M_idx***) with minimal square error.

Since the encoding process is performed offline, its speed is not much an issue as that of decoding. When considering joint channel compression and point translation together, we find it is a complex optimization problem. A full search algorithm or more sophisticated approximation methods will definitely generate better results. In this paper, we just show that a simple encoder described above has already demonstrated very good performance. Further encoder optimization will be investigated in the future work.

4.3. HDR Texture Decompressor

Figure 5 (b) shows a possible HDR texture decoding logic for DHTC format. First of all, the DXT1 block is sent to a

Textures	mPSNR (dB)					Log[RGB] RMSE					HDR-VDP >75% error		
	Our	Munkberg 2007	Munkberg 2006	Roimela 2008	Wang 2007	Our	Munkberg 2007	Munkberg 2006	Roimela 2008	Wang 2007	Our	Munkberg 2007	Roimela 2008
BigFogMap	51.0	51.9	51.7	50.4	46.3	0.06	0.06	0.06	0.07	0.14	0.00	0.00	0.00
Cathedral	39.7	40.0	38.9	34.3	35.8	0.17	0.17	0.20	0.35	0.36	0.10	0.02	0.03
Memorial	46.8	46.5	46.1	41.7	38.0	0.14	0.13	0.14	0.31	0.69	0.01	0.00	0.00
Room	48.1	48.6	48.1	44.0	34.1	0.09	0.08	0.09	0.15	0.71	0.01	0.01	0.00
Desk	41.5	40.3	39.7	28.4	21.3	0.17	0.22	0.25	1.26	2.92	0.03	0.01	0.00
Tubes	35.7	35.7	32.2	27.0	29.1	0.32	0.28	0.43	0.81	0.81	0.87	1.25	1.20
Average	**43.8**	**43.8**	**42.8**	**37.6**	**34.1**	**0.16**	**0.16**	**0.20**	**0.49**	**0.94**	**0.17**	**0.22**	**0.21**

Table 2: *Objective quality comparisons. Left: mPSNR results, at the same exposure ranges as Munkberg 2007 [MCHAM07]; Middle: Log[RGB] RMSE results; Right: HDR-VDP above 75% probability visible artifacts (texels in percentages), under the global adaptation luminance at about 10 cd/m^2. Note that HDR-VDP does not take chrominance distortions into account.*

PSNR	Map1	Map2	Map3	Map4	Map5	Map6	Map7	Map8	Map9	Map10	Map11	Map12	**Average**
Our	49.2	53.6	54.4	56.4	46.3	47.5	40.2	47.1	47.3	44.9	45.0	41.9	**47.8**
DXT5	48.6	54.8	57.1	61.2	45.3	45.2	39.0	46.1	44.2	42.9	43.8	40.2	**47.4**

Table 3: *Alpha map compression results (in PSNR). We use the DXT5 codec in ATI's Compressonator.*

DXT decoder, and the 8-bit integer levels of the three channels are recovered. At the same time, the table entry index and modifier index are used to look up the desired modifier in the modifier table. The modifier is then added to channel Y to compensate point translation. In the next, the inverse process of local dynamic range reduction is carried out in the *global HDR recovery* module, including luminance log decoding and chrominance log or linear decoding. Note that log decoding is a combination of linear decoding and exp2 operation. Finally, based on the color transform mode, the inverse color transform is carried out and the RGB values of the desired texel is reconstructed.

Apparently, a hardware DHTC HDR texture decoder only needs moderate extension based on the DXT decoder. The inverse color transform involves one addition, two tri-channel multiplications, and a MUX to reorder signals based on ***Ch_mode***. The global HDR recovery module includes one tri-channel uniform dequantization and one tri-channel exponentiation operation at most. We try our best to make the three channels share the same operations at the same time to further accelerate the decoding process. In summary, the decoder is simple and the decoding should be very fast.

4.4. LDR Texture Codec

The proposed DHTC format can also be used in compressing LDR textures with alpha channels. Given a raw RGBA LDR texture, we first compress RGB channels with DXTC and pack them into the DXT1 block in DHTC format. And then, the alpha channel is compressed into the extension block by using the following alpha coding method similar as the LDR TC scheme iPACKMAN [SAM05].

First, we divide each 4x4 alpha block into two sub-blocks, either 4x2 or 4x2, indicated by one bit of ***Ch_mode***. In each sub-block, a 5-bit base alpha (***L0*** or ***L1***) plus a proper modifier approximates the original alpha value. Per-block ***T_idx*** and per-texel ***M_idx*** still indicate the desired modifier in the modifier table (Table 1). The other bit of ***Ch_mode*** determines whether the alpha block use 8 modifiers or 6 modifiers. For the 8-modifier block, all the modifiers in the selected modifier table entry are valid, while for the 6-modifier block, only the first 6 modifiers are valid, plus two special alpha values 0 and 255. This trick is similar as DXT5 alpha coding.

A possible DHTC decoding logic for LDR RGBA textures is shown in Figure 5 (c). Most functional modules are shared with the HDR decoding logic in Figure 5 (b), and we only need to add two MUXs to select the base alpha value and handle the 8-modifier/6-modifier cases.

5. Results

In this section, we demonstrate that DHTC, no matter whether used for HDR TC or LDR TC, is superior to the state of the arts. Thus, we arrive at a unified compressed format suitable for HDR textures, LDR textures and alpha maps.

5.1. HDR TC Quality Comparison

We compare DHTC with existing HDR TC methods on the same set of test images used by Munkberg et al. [MCHAM07]. The objective comparison results are listed in Table 2. According to the three error measurements, it is obvious that Munkberg et al.'s method in 2007 and our scheme are of much better quality than others'. Recall that Munkberg et al. define two compression modes with different bit layouts, which makes the decoder design more complicated.

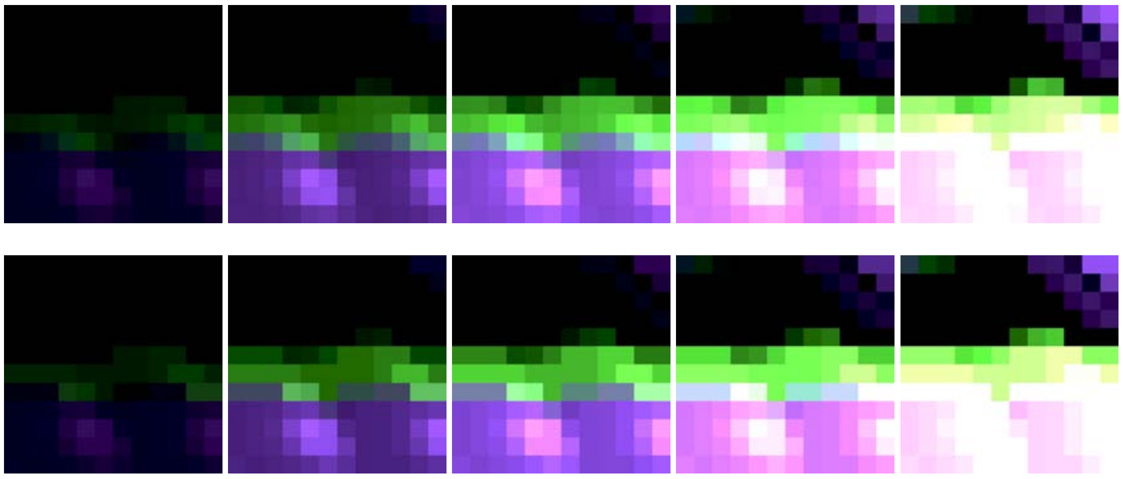

Figure 6: *Coding high contrast blocks. Top row: Original at different exposures. Bottom row: Our compressed at the same exposures. (Part of texture "Tubes")*

Based on our observation, reconstructed textures with mPSNR more than 40 dB are visually near lossless. In general, Munkberg et al.'s methods as well as ours have obtained high enough visual fidelity. However, recall that in all the previous HDR TC schemes, the limited bit budget is statically divided into luminance bits and chrominance bits, and thus they can hardly guarantee an optimal bit allocation for each block. Most likely visible artifact will occur in regions with rich colors. In contrast, our method is able to adaptively allocate bits among different blocks and color channels, thus it avoids large distortions. Two examples are shown in Figure 8. The example textures are part of texture "Desk" and "Cathedral". Clearly, our method provides much better visual quality. Note that the other methods [MCHAM07, RAI08] preserve the luminance information very well for both textures (under HDR-VDP metric in Table 2), which implies more bits should be spent in the chrominance channels.

There also exist some hard cases for DHTC. Recall that we assume the local dynamic range is usually within 2 orders so that 8-bit encoding is sufficient within a block. Although only about 0.7% blocks are of local luminance contrast higher than 100:1 in our statistics, DHTC cannot provide fine precision for these blocks. An extreme case with local dynamic range more than 30000:1 is shown in Figure 6. In addition, DHTC still use linear fitting to handle chrominance channels, and provide only 4 chrominance values per block. This simple approach works well for most HDR textures, but may result in visible artifact in texture regions with frequent color changes (See Figure 3 for example).

5.2. Alpha Channel Coding

Inherited from the embedded DXT1 block, DHTC format for HDR textures may contain 1-bit alpha information to support HDR alpha blending. The mutual order of the both base colors indicates whether or not a single transparent color exists in the block palette. A blending example is shown in Figure 7.

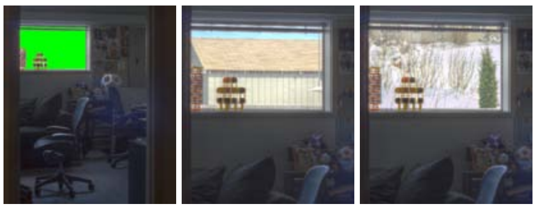

Figure 7: *HDR alpha blending. Left: compressed HDR texture with 1-bit alpha; Middle & right: blended scenes.*

As described in Section 4.4, DHTC format for LDR textures can also support 8-bit alpha channel coding. We select 12 alpha maps from an Xbox360 game as a test suite, and compare DHTC alpha coding with the de facto standard DXT5 alpha coding. The comparison results are listed in Table 3. For most testing cases, DHTC provides superior quality.

6. Conclusions

In this work, we have presented a new scheme for HDR texture compression, namely DHTC, based on the ubiquitously adopted DXTC technique. We introduce some techniques, such as adaptive color transform, local dynamic range reduction, and point translation, to fully take advantages of DXTC to achieve higher compression efficiency. When evaluated under different error metrics, our scheme is always comparable to or outperforms the state of the arts. More importantly, hardware implementation of a DHTC decoder only needs moderate extension on existing hardware. Further, we demonstrate that the proposed DHTC format is able to handle LDR textures with alpha channels, with no need to re-design special LDR decoding pipeline. We believe this paper provides a unique solution to meet all the practical requirements for HDR and LDR texture compression, and many HDR related applications can benefit from our research.

Acknowledgement

We would like to thank Kimmo Roimela from Nokia Research Center and Jacob Munkberg from Lund University for their great helps with the comparison results part. Test images used in the paper are courtesy of Jack Tumblin, Dani Lischinski, Paul Debevec, and ILM OpenEXR test suite. Many thanks to anonymous reviewers for their valuable comments and suggestions.

References

[BAC96] BEERS A. C., AGRAWALA M., CHADDHA N.: Rendering from compressed textures. In *Proceedings of SIGGRAPH 96* (Aug. 1996), Computer Graphics Proceedings, Annual Conference Series, pp. 373–378.

[CTHD01] COHEN J., TCHOU C., HAWKINS T., DEBEVEC P.: Real-time high-dynamic range texture mapping. In *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering* (June 2001), pp. 313–320.

[Deb98] DEBEVEC P.: Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proceedings of SIGGRAPH 98* (July 1998), Computer Graphics Proceedings, Annual Conference Series, pp. 189–198.

[Fen03] FENNEY S.: Texture compression using low-frequency signal modulation. In *Graphics Hardware 2003* (July 2003), pp. 84–91.

[INH99] IOURCHA K. I., NAYAK K. S., HONG Z.: System and method for fixed-rate block-based image compression with inferred pixel values. *US Patent 5,956,431* (1999).

[KSKS96] KNITTEL G., SCHILLING A., KUGLER A., STRASSER W.: Hardware for superior texture performance. *Computers & Graphics 20*, 4 (July 1996), 475–481.

[MCHAM06] MUNKBERG J., CLARBERG P., HASSELGREN J., AKENINE-MÖLLER T.: High dynamic range texture compression for graphics hardware. *ACM Transactions on Graphics 25*, 3 (July 2006), 698–706.

[MCHAM07] MUNKBERG J., CLARBERG P., HASSELGREN J., AKENINE-MÖLLER T.: *Practical HDR texture compression*. Tech. rep., Lund University, 2007.

[MEMS06] MANTIUK R., EFREMOV A., MYSZKOWSKI K., SEIDEL H.-P.: Backward compatible high dynamic range MPEG video compression. *ACM Transactions on Graphics 25*, 3 (July 2006), 713–723.

[MKMS04] MANTIUK R., KRAWCZYK G., MYSZKOWSKI K., SEIDEL H.-P.: Perception-motivated high dynamic range video encoding. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 733–741.

[RAI06] ROIMELA K., AARNIO T., ITÄRANTA J.: High dynamic range texture compression. *ACM Transactions on Graphics 25*, 3 (July 2006), 707–712.

[RAI08] ROIMELA K., AARNIO T., ITÄRANTA J.: Efficient high dynamic range texture compression. In *SI3D '08: Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games* (2008), pp. 207–214.

[SAM05] STRÖM J., AKENINE-MÖLLER T.: iPACKMAN: High-quality, low-complexity texture compression for mobile phones. In *Graphics Hardware 2005* (July 2005), pp. 63–70.

[TK96] TORBORG J., KAJIYA J.: Talisman: Commodity real-time 3D graphics for the PC. In *Proceedings of SIGGRAPH 96* (Aug. 1996), Computer Graphics Proceedings, Annual Conference Series, pp. 353–364.

[War94] WARD G. J.: The RADIANCE lighting simulation and rendering system. In *Proceedings of SIGGRAPH 94* (July 1994), Computer Graphics Proceedings, Annual Conference Series, pp. 459–472.

[War05] WARD G.: High dynamic range image encodings. http://www.anyhere.com/gward/hdrenc/hdr_encodings.html.

[WS04] WARD G., SIMMONS M.: Subband encoding of high dynamic range imagery. In *APGV 2004* (Aug. 2004), pp. 83–90.

[WWS*07] WANG L., WANG X., SLOAN P.-P., WEI L.-Y., TONG X., GUO B.: Rendering from compressed high dynamic range textures on programmable graphics hardware. In *SI3D '07: Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games* (2007), pp. 17–24.

[XPH05] XU R., PATTANAIK S. N., HUGHES C. E.: High-dynamic-range still-image encoding in JPEG 2000. *IEEE Computer Graphics and Applications 25*, 6 (2005), 57–64.

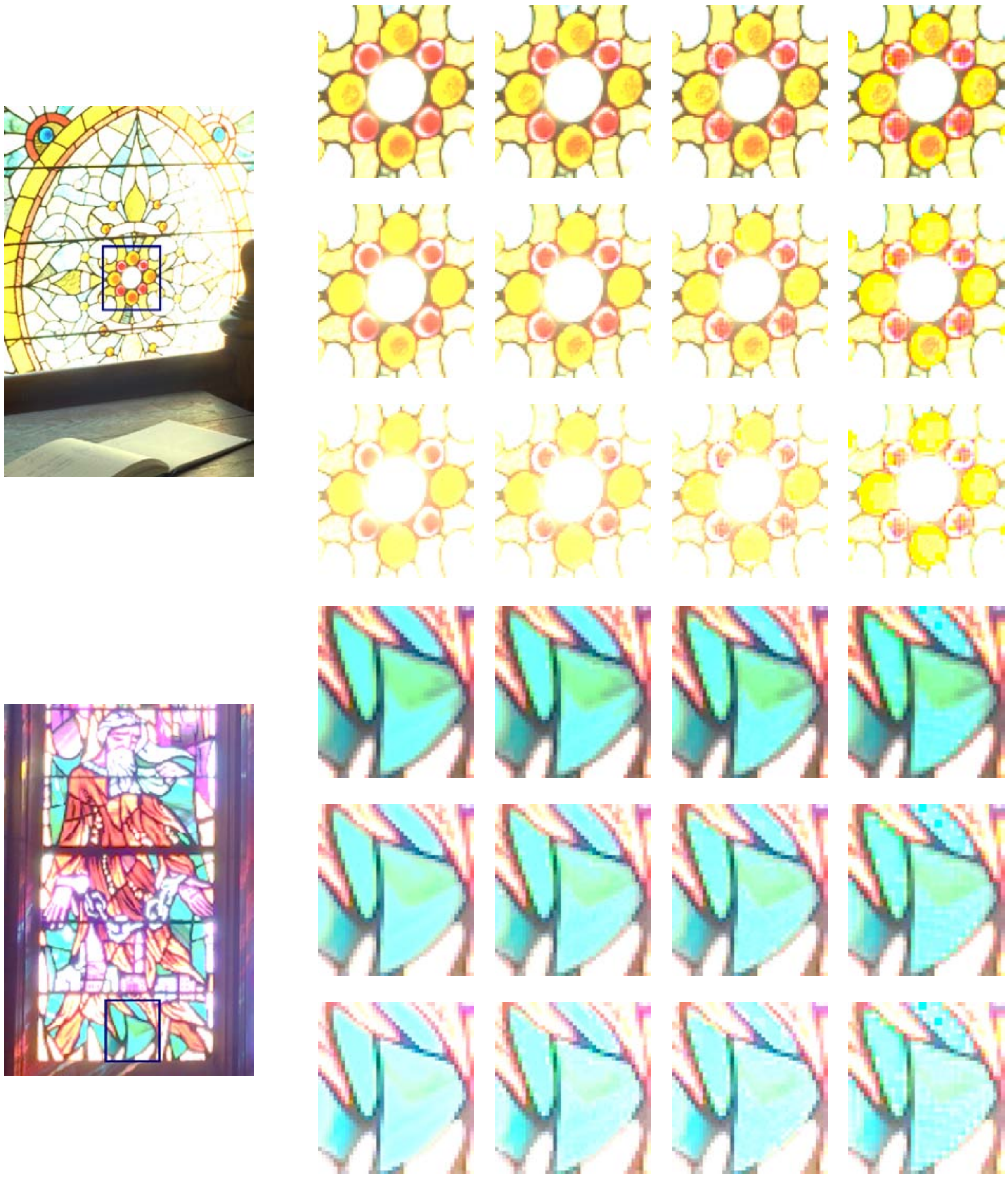

Figure 8: *Visual comparison at different exposures. We show the close-up of two HDR texture regions with rich colors. Left column: Original; Middle left column: DHTC; Middle right column: Munkberg 2007; Right column: Roimela 2008.*