Real-Time High-Dynamic Range Texture Compression Based on Local Fractal Transform

Jerzy Stachera*
Warsaw University of Technology
Faculty of Electronics and Information Technology
The Institute of Computer Science

Przemysław Rokita†
Warsaw University of Technology
Faculty of Electronics and Information Technology
The Institute of Computer Science

Abstract

In this paper we propose a new HDR texture compression algorithm for real-time rendering. This algorithm is based on our local fractal and wavelet transform for chrominance and luminance encoding, respectively. The proposed algorithm is a block based fixed length coding algorithm and operates on a 4x4 blocks. The blocks are compressed to 8bpp. As the absolute difference between maximum and minimum value for most blocks fits into the LDR range, we can map the HDR values into the integer domain. Then, the block values are encoded by the modified LDR algorithm. In the case of luminance, it is required to store minimum and maximum value per block and to remap the values in the last decompression step.

CR Categories: I.3.7 [Computer Graphics]: Color, shading, shadowing, and texture, I.4.2 [Compression (Coding)]: Approximate methods

Keywords: Compression, Coding, Texture, High Dynamic Range, Shading

1 Introduction

High dynamic range (HDR) textures are becoming increasingly popular in the real-time computer graphics [McTaggart et al. 2006]. This is especially seen in the computer games where they are used for the realistic light modeling. The HDR texture may store the luminance levels acquired from the real environment. Thus, the HDR texture require a bit depth which is usually twice the one used for the low dynamic range (LDR) textures. At the same time, the demands for storage and memory bandwidth for HDR textures are increased accordingly. Moreover, the application of texture filtering rise the demands for efficient memory handling even further.

With the hardware support for HDR texture processing, it becomes possible to rise the level of realism of real-time computer graphics. Up to that moment only LDR textures were available. Low dynamic range textures are represented by 8-bit RGB format. This format can only accurately represent the diffuse surfaces and is not sufficient for storing the luminance range presented in the real environments. To tackle the problem of limited range a number of lossless HDR formats was introduced [Reinhard et al. 2005, chapter 3]. Basically, the HDR format may require 16-bits per component. In this case, the uncompressed RGB texture needs 48bpp. This format requires twice the memory space and bandwidth as compared to LDR textures.

*e-mail: J.Stachera@ii.pw.edu.pl
†e-mail: P.Rokita@ii.pw.edu.pl

A general trend in computer graphics architectures is that the computing power growth is much faster than the corresponding improvement in memory technologies [Owens 2005][Buck 2006]. Recent experimental studies of game workload revealed that the memory bandwidth is mostly occupied by access to texture data and z-buffer [Roca et al. 2006]. Thus, the memory is the main bottleneck in the real-time rasterization based graphics hardware and should be used with great care. The problem with HDR textures is that they drastically consume memory bandwidth and in the end may lead to severe reduction in the rendering performance. This problem is even more critical in the context of filtering methods, where one filtered texture sample requires several access to memory.

One of the solutions to the problem of texture memory bandwidth and space is hardware texture compression introduced by Beers [Beers et al. 1996], Knittel [Knittel et al. 1996] and Torborg [Torborg et al. 1996]. Generally, the idea is to apply lossy compression for the textures and store the compressed representation of textures in memory. When the rendering engine access the texture, the texture is transferred in compressed form over the bus and decompressed on-the-fly as needed. Thus, texture bandwidth and memory can be significantly reduced. Additionally, if the texture decompression is done just before filtering, then the texture data can be stored in the cache in compressed form. It means better cache utilization [Hakura and Gupta 1997][Igehy et al. 1999]. At the same time the texture cache can be made smaller by the factor comparable to the texture compression ratio, therefore leaving space for more processing units.

2 Problem definition

Although a texture can be regarded as a digital image, most of the classical image compression algorithms cannot be applied to textures. According to Beers [Beers et al. 1996] and others [Fenney 2003][Ström and Möller 2005] when designing a texture compression algorithm the following aspects must be taken into account:

Decoding speed. The access to the data is critical in texture mapping application. Therefore, the decompression speed is of paramount importance. The decompression algorithm should be of low complexity to enable hardware implementation. If some sort of filtering is required then a number of parallel decompression units are needed to process single texel [Fenney 2003], e.g. to deliver a single texel per clock in trilinear filtering it is required to have eight decompression units.

Random access. Since the texture mapping procedure introduces discontinuous texture access in texture space, it is difficult to know in advance how the texture will be

accessed. Thus, the algorithms which are considered for fast random access are based on fixed length codes.

Compression rate and visual quality. The difference between textures and images is that images are viewed on their own in static content, while the texture are the part of the scene which usually changes dynamically. Moreover, textures are subject to filtering, blending and lighting calculations which change the way they are perceived. In contrast to LDR textures, when rendering from HDR texture, it is not possible to know in advance which part of the dynamic range will be used. Thus, the compression algorithm must deliver the same quality for wide range of luminance levels.

Encoding speed. Texture compression is an asymmetric process, in which the decompression speed is crucial and the encoding speed is useful for some application as for example real-time compression of dynamically generated environment maps but in most cases is done off-line.

3 Previous work

The lossless representation of HDR images has been actively researched for some time with radiance RGBE being one of the first HDR formats [Ward 1992]. This format uses 24-bits for RGB data and 8-bits for exponent which is shared by all three components. The format bit allocation is considered as not being compact due to the typically unused wide dynamic range. The LogLuv format represents an improved encoding with separated luminance and chrominance components [Ward 1998]. The logarithm of luminance is stored in 16-bits, while the chrominance occupies another 16-bits for 32bpp compression ratio. The LogLuv format has also a variant which is characterized by 24bpp compression ratio. Another improvement upon RGBE is OpenEXR format [Kainz et al. 2003]. This format encodes the image in 16-bit floating-point RGB pixels for a total of 48bpp. This representation is directly supported by many graphics cards vendors. The lossy HDR compression algorithms appeared recently with the introduction of Ward backward compatible HDR JPEG algorithm [Ward and Maryann 2005]. In this algorithm HDR image is represented as a tone-mapped image and luminance ratio image. There are two possible scenarios in the decompression stage depending on the decoder type. In the case of unaware decoder, the ratio image is ignored and only the tone-mapped image is decompressed. For HDR decoder, both images are decompressed in order to reconstruct high dynamic range image. Another JPEG-based algorithm is proposed by Xu et al. [2005]. This algorithm uses a pre-processing step to compress the HDR image by utilizing the JPEG 2000 algorithm. In the pre-processing step, the HDR RGB data is subject to logarithm function and then is converted to the integer representation which is fed to JPEG 2000 encoder. This, allows to achieve high compression ratios while retaining high visual quality. Another group of algorithms constitutes tone-mapping operators [Reinhard et al. 2005, chapter 6, 7]. Since, the tone-mapping reduces the dynamic range to displayable range it is not possible to restore the lost values. Thus, tone mapping has different goal and is not directly applicable in textures compression.

Generally, HDR image formats were designed with the aim of providing high image quality and compact storage. None of them fulfils the requirements for texture compression enumerated in section 2. The RGBE, LogLuv and OpenEXR format do not use compression, therefore they are expensive in terms of memory storage and bandwidth. The JPEG formats are characterized by the variable length coding, thus they do not support random access to individual texels.

Real-time HDR texture compression was introduced by Munkberg et al. [2006] and Roimela et al. [2006] and later work in this field was carried by Wang et al. [2007]. All the algorithms have some common steps: conversion to custom luminance/chrominance color space, independent encoding of 4x4 blocks to fixed length code. In Munkberg algorithm [2006][2007], the luminance is represented by minimum and maximum block values and additionally fourteen in-between values are found. Two quantization modes are proposed with uniform and non-uniform quantization steps. The chrominance is sub-sampled either horizontally or vertically. To encode resulting chrominance values they use a set of shapes each with four points for better fit to chrominance distribution. The compression ratio of Munkberg algorithm is 8bpp. Completely different approach was taken by Roimela et al. [2006]. They make use of 16-bit floating point integer bit pattern to represent the luminance data more compactly. The chrominance data is sub-sampled two times in each direction and the values are quantized to seven bits. The compression ratio of Roimela algorithm is 8bpp. Wang et al. propose an algorithm targeted for DirectX [Blythe 2006] platform which utilize DXTC compression and native filtering. The luminance data is divided into two ranges which are stored in alpha component of two RGBA textures. Remaining RGB components in both textures store chrominance and luminance residual values which are the difference between original and reconstructed luminance. These two textures are encoded by DXT5 resulting in 16bpp.

4 New algorithm

The new algorithm was designed with the texture compression requirements outlined in section 2 taken into account. It is a block based fixed length coding compression algorithm. Each block of size 4x4 texels is compressed to 8bpp. Our aim was to support the same dynamic range as OpenEXR which is equal to 10^9. The whole scheme representing compression process of our algorithm is depicted on figure 1. We decided to use color space conversion to separate luminance and chrominance. The conversion outcome is HDR luminance and LDR chrominance component respectively. To compress the luminance component we employ our local fractal transform [Stachera and Rokita 2006]. For chrominance we use Pereberin approach based on Haar wavelet transform [1999]. Since, the human visual system response to wide range of illumination is close to logarithmic function, therefore in our algorithm the compression is applied to logarithmic luminance. Further, to reduce the luminance dynamic range and thus to better fit the quantization levels, we store minimum and maximum value for each block. Those values are use for luminance float-integer mapping. The logarithmic transform and float-integer mapping are similar to HDR JPEG 2000 algorithm proposed by Xu et al. [2005]. But as opposed to this algorithm we utilize luminance/chrominance representation and we store per-block scaling factors to reduce the dynamic range. The

next sections discuss the design process of our algorithm which is related to successive compression steps (Figure 1).

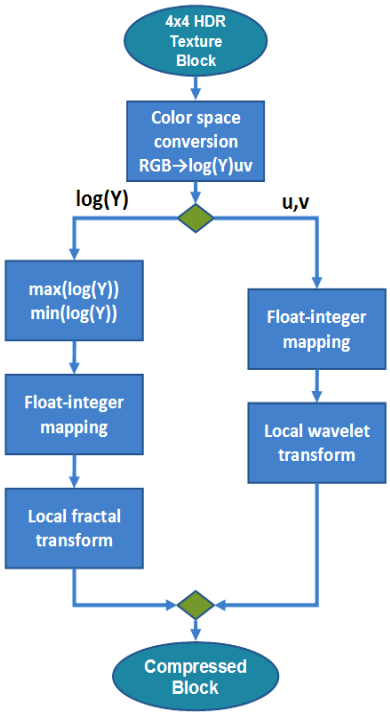


Figure 1. Compression scheme of proposed algorithm.

Color space conversion.

The RGB representation is not optimal for encoding, since each component requires the same accuracy. As most of the HDR algorithms, our aim was to concentrate all the HDR information into one component. It is possible by converting the RGB values to color space with separate luminance and chrominance components. Thus, the problem of HDR encoding is restricted to the luminance component. The chrominance component has low dynamic range. Moreover, low sensitivity of human visual system to the chrominance as compared to the luminance allows for coarse representation of the chrominance component.

With the introduction of real-time HDR texture compression, there were proposed different color spaces. Generally, they represent the data by the luminance and chrominance. Munkberg logYuv [2006] color space transform computes the logarithmic luminance from RGB components using weights (0.299, 0.587, 0.114) as according to Rec. 601 [Poynton 2003]. Roimela simplifies the color transform with the goal of efficient decoding and to compute the luminance he uses hardware friendly weights (0.25, 0.5, 0.25). Wang [2007] employs four component color space called LUVW. In this space the luminance component L is computed as euclidian distance of RGB components. The chrominance components UVW are computed from RGB components by division by L, (R/L, G/L, B/L). The exception to the separate luminance coding is log[RGB] color space being a part of HDR JPEG 2000 algorithm proposed by Xu [2005]. In this space the logarithm function is applied to the RGB components.

Munkberg logYuv color space is not optimal for hardware implementation due to the luminance weights. Wang uses four component LUVW color space which is most advantageous in case of his algorithm, but generally more compact representations are based on three components. Xu log[RGB] color space has potential disadvantage of representing all three RGB channels with the same accuracy. We chose Roimela color space transform for our algorithm. We found that it provides better decorrelation as compared to Munkberg logYuv for chosen HDR test image set (Figure 5) and at the same time it provides simplified hardware implementation. Additionally, in our algorithm we apply logarithmic function to the luminance. The forward and backward transform for modified Roimela color space are shown below:

$$(\bar{Y},u,v) = (\log(Y), R/4Y, B/4Y)$$
$$(R,G,B) = 2^{\bar{Y}}(4u, 2(1-u-v), 4v)$$
$$Y = (R+2G+B)/4$$

Float-integer mapping

In our algorithm we introduced an additional step called float-integer mapping to reduce the luminance block dynamic range and to make use of our local fractal transform [Stachera and Rokita 2006]. The motivation for using this mapping is that for most 4x4 HDR image blocks the difference between maximum and minimum value fits into the low dynamic range. This can be seen on figure 2 which represents the differences for HDR test image set (Figure 5). In this case around 99% of the blocks fits into the low dynamic range and can be represented on 8-bits. The rest of blocks fits into 9-bits precision and above.

After applying the color space transform, the logarithmic luminance log(Y) values are in the range [-16, 16], and the chrominance values (u, v) are in the range [0,1]. For the luminance data, the mapping is preceded by finding the minimum and maximum value in the block as follows:

$$x_{\min} = \min(\log(Y)), \quad x_{\max} = \max(\log(Y))$$

These values are stored in the block code on 5-bits as 1-bit sign and 4-bits module, where module represents the absolute value. Those values are used during the decompression process for inverse mapping.

The logarithmic luminance block values are uniformly quantized to *b*-bits using the following equation:

$$\left[(x - x_{\min})/(x_{\max} - x_{\min}) * (2^b - 1)\right]$$

The inverse mapping is done as follows:

$$\left[x/(2^b - 1) * (x_{\max} - x_{\min}) + x_{\min}\right]$$

Since, the chrominance values are in the range [0,1] we do not have to store the minimum and maximum value. Therefore, the chrominance data are directly mapped to the range [0, 255]. The resulting luminance and chrominance values are subject to local fractal and wavelet transform respectively.

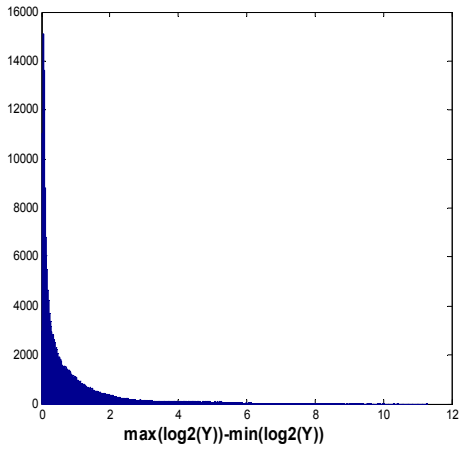
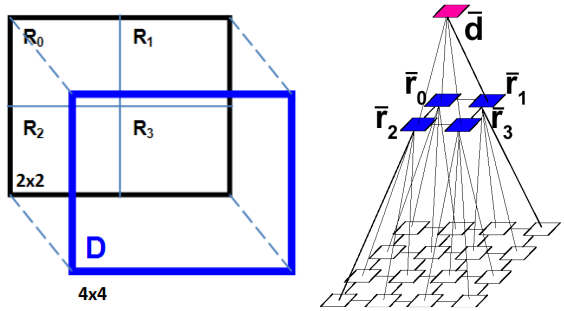


Figure 2. Histogram of differences between maximum and minimum value for logarithmic luminance for all 4x4 blocks from the HDR test image set (Figure 5).

Local fractal transform

For luminance compression we employ the local fractal transform which details can be found in [Stachera and Rokita 2006]. The introduction to the fractal compression is presented in [Lu 1997] and in [Fisher 1995]. The local fractal transform is based on modified local IFS (Iterated Function System) [Lu 1997]. The transform approximates the block by its rescaled version (Figure 3). Formally, the block is uniformly divided into non-overlapping blocks which are called range blocks *R* (2x2). Additionally, there is defined a domain block *D* (4x4) which constitutes the whole block. The relation between the domain block and the range blocks is as follows: $D = R_0 \cup R_1 \cup R_2 \cup R_3$ (Figure 3a). Thus, the domain block *D* is the union of range blocks *R*. For each range block *R* there is defined a contractive transform *w* which maps on it domain block D: $w : D \to R$. Due to the compact representation the contractive transform is of the form: $w(D) = s\varphi(D - \bar{d}) + \bar{r}$, where: $\bar{d}$ is the domain block mean, *s* is scaling factor, $\bar{r}$ is the range mean and φ is the geometric function. This transform uses domain block normalization as opposed to the one presented in [Stachera and Rokita 2006]. In compression process for each range block *R* we are computing the optimal parameters of the transform *w* [Fisher 1995]. Thus, the block luminance fractal code is represented by contractive transform parameters for each of four range blocks *R*: $\{s_i, \bar{r}_i, \varphi_i\}_{i=0}^{3}$. We use the following bit allocation for transform parameters: scale factor – 6-bits, range mean – 11-bits, iso – 3-bits. Iso parameter represents the geometric function φ and as will be explained later, it essentially represents one of eight isometric operations.

Applying the transform *w* (where $w(D) = s\varphi(D - \bar{d}) + \bar{r}$ - inverse local fractal transform) for given range block *R* can be done in two phases. In the first phase, the domain block *D* is scaled to the size of range block *R* usually by averaging texel values. Since, the rescaled values of domain block are part of the fractal code in the form of range block means $\bar{r}$, this step is omitted (Figure 3b). Then, the resulting values are normalized by subtracting the domain mean $\bar{d}$, where $\bar{d} = (\bar{r}_o + \bar{r}_1 + \bar{r}_2 + \bar{r}_3)/4$ (Figure 3b). Finally, they are subject to geometric function which maps them on range block. The mapping can be done in eight different ways which comprise isometric operations such as rotations and reflections [Lu 1997]. In second phase, the resulting values are linearly processed on the basis of scaling factor *s* and range mean $\bar{r}$.



a) b)

Figure 3. Local fractal transform.

Local wavelet transform

For the chrominance compression we utilize two levels of Haar wavelet decomposition as proposed by Pereberin [1999]. This transform was chosen due to its compact representation and low computational complexity. The low resolution level is represented by one mean value for each chrominance component. The medium resolution level is approximated by one detail coefficient *d* which is multiplied by one of eight refining matrices ***RM*** [Pereberin 1999]. In the decompression process the mean *m* value is added to the detail coefficient multiplied by the refine matrix: *m+d*RM[i]* (inverse local wavelet transform), where *i* is the refine matrix index. The mean value *m* and the detail coefficient *d* are stored on 8-bits respectively [Pereberin 1999]. Three bits are used for the refine matrix index *i*. Thus, one chrominance component is represented by 19-bits.

Decompression

The decompression process is depicted on figure 4. For luminance data we first apply the inverse local fractal transform (section 4.3). Then we expand the compressed range on the basis of block minimum and maximum value to the range [-16, 16] (section 4.2). For the chrominance component we apply the inverse local wavelet transform (section 4.4). The resulting values are divided by 255 resulting in the range [0,1]. Next, the floating point component data is subject to color space conversion (section 4.1) to obtain HDR RGB data. The decompression process is done on per-texel basis giving access to individual block texels.

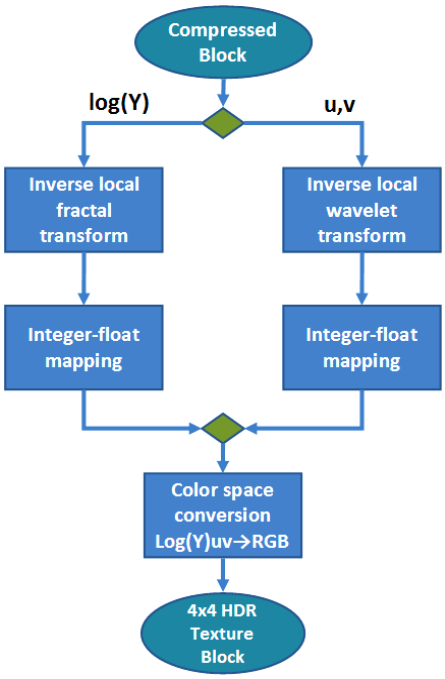


Figure 4. Decompression scheme of proposed algorithm.

5 Experimental results

We chose twelve HDR images for our textures test set (Figure 5). The set is consisted of real photographs (Figure 5a-j) and computer generated images (Figure 5k, 5l). The dynamic range for the set spans the values from 2 to 9 in log10 domain (Figure 6). The problem with evaluation of HDR texture compression algorithms is that it is not possible to predict which part of the dynamic range will be used during rendering. Therefore, the compression algorithm must deliver the same quality for wide range of luminance levels. Using LDR image metrics for evaluation of HDR algorithms may be inappropriate. Thus, we adopted Log[RGB] RMSE [Xu et al. 2005] and mPSNR [Munkberg et al. 2006] metrics.

The mPSNR metric is computed on the basis of tone mapped images. As a tone mapping operator, it is used a gamma-adjustment after exposure compensation:

$$T(I) = [255(2^c I)^{1/\gamma}]_0^{255}$$

Where I is the HDR image, c is the exposure compensation in f-stops, γ is the display gamma and $[\cdot]_0^{255}$ indicates clamping to integer interval [0,255]. For a set of exposures covering the HDR image range there is computed MSE error. The values of all MSE errors are averaged and used for PSNR error computation [Munkberg et al. 2006]. We used simplified version of this metric by separately computing PSNR values for chosen set of exposures. This means that after applying the tone mapping operator we calculate the standard PSNR error as used for RGB LDR images. The results with reconstructed images in full resolution (electronic version only) are shown on figure 9. Additionally, the error image for memorial and dani belgium (Figure 5i, 5d) is shown on figure 10.

The log[RGB] RMSE metrics [Xu et al. 2005] is computed according to the following formula:

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

The values (r,g,b) are from the original texture and the values $(\hat{r},\hat{g},\hat{b})$ are from the reconstructed texture, where N is the number of texels in the texture. The motivation of measuring the error in logarithmic domain is that the values in this domain are more perceptually uniform.

To better show the performance of our algorithm for different range of luminance levels, we computed the mean log[RGB] RMSE values for sets of blocks having the same dynamic range (Figure 7). The dynamic range for a block is computed as a difference between the maximum and minimum value in the block in log2 domain (see section 4.2). The values from the chart on figure 7 should be evaluated along with the histogram on figure 2 which shows the blocks distribution depending on theirs dynamic range.

The result of comparison of our algorithm with the state of the art algorithms can be found in table 1. In this case, the log[RGB] RMSE metric is computed for whole image. Our algorithms shows better results as compared to Romeila [2006] and Wang [2007] on Desk image. This may be due to the high variation of chrominance in this image, which is better capture by our algorithm. Visual evaluation was not possible due to the lack of publicly available encoders.

Figure 5. HDR test image set.

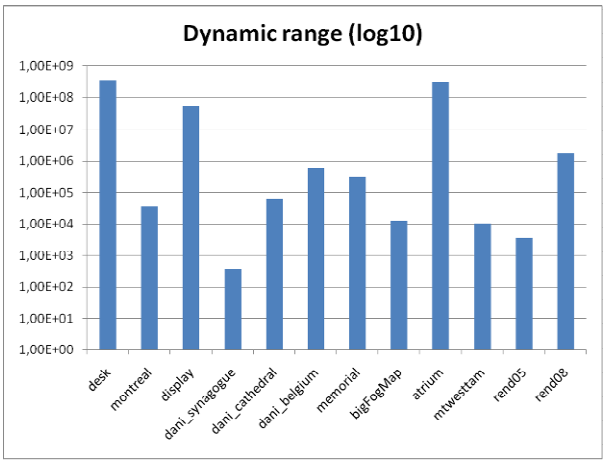


Figure 6. HDR test image set (Figure 5) dynamic rage.

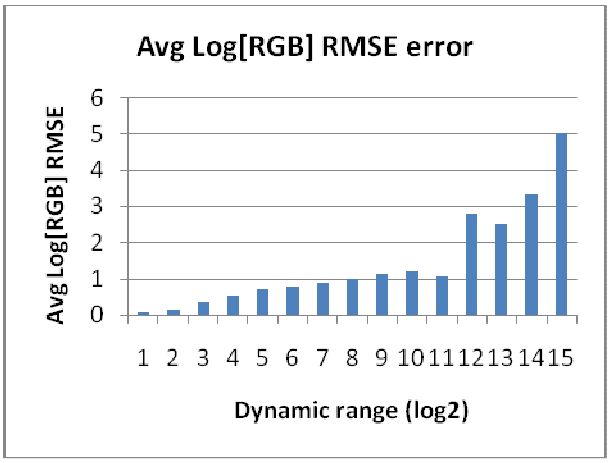


Figure 7. Average Log[RGB] RMSE for block dynamic ranges (Figure 5).

Images	Our	Wang*	Romeila*	Munkberg*
big Fog Map	0,14	0,14	0,1	0,06
memorial	0,34	0,69	0,26	0,14
desk	0,51	2,92	1,14	0,25
atrium night	0,32	-	-	-
dani belgium	0,25	-	-	-
dani cathedral	0,36	-	-	-
dani synagogue	0,12	-	-	-
display1000	0,4	-	-	-
montreal	0,12	-	-	-
MtTamWest	0,26	-	-	-
rend05	0,12	-	-	-
rend08	0,22	-	-	-

Table 1. Log[RGB] RMSE error for test image set (Figure 5). (*) The first three results for Wang [2007], Roimela [2006] and Munkberg [2006] are available in [Munkberg et al. 2007].

6 Conclusion

We have presented a new HDR texture compression algorithm. Our algorithm is able to effectively reduce the dynamic range. We prove that it is possible due to the low variation of block values. This allow us to compress the dynamic range by storing maximum and minimum value per block and applying the float to integer mapping. Thus, in most the cases the problem of HDR encoding can be simplified to LDR encoding. In the present algorithm we use our local fractal transform, but we plan to investigate other low complexity transform as well. Comparison with the state of the art methods shows that our algorithm gives comparable and better results to recent algorithms. Further improvement of image quality would be carried with more investigation into the local fractal transform parameters coding. At the moment we use uniform quantization but other form of non-uniform quantization can be useful for high dynamic range blocks.

References

BEERS, A. C., AGRAWALA, M., CHADDHA, N. 1996. Rendering from compressed textures, In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, July 1996, 373 – 378.

BLYTHE, D. 2006. The Direct3D 10 system, *ACM SIGGRAPH 2006 Papers*, 724 – 734.

BUCK, I. 2006. *Stream Computing on Graphics Hardware*, Ph.D. Thesis, Stanford University, September 2006.

FENNEY, S. 2003. Texture Compression using Low-Frequency Signal Modulation. In *Graphics Hardware* (2003), ACM Press, 84 – 91.

FISHER, Y. (ed) 1995. *Fractal Image Compression: Theory and Application*, Springer Verlag, New York.

HAKURA, Z. S., GUPTA, A. 1997. The Design and Analysis of a Cache Architecture for Texture Mapping, In *24th International Symposium of Computer Architecture* (June 1997), ACM/IEEE, 108 – 120.

IGEHY, H., ELDRIDGE, M., HANRAHAN, P. 1999. Parallel Texture Caching, In *Graphics Hardware* (1999), ACM Press, 95 – 106.

KAINZ, F.,, BOGART, R., HESS, D. 2003. The OpenEXR image file format, *SIGGRAPH 2003 Technical Sketches*.

KNITTEL, G., SCHILLING, A., KUGLER A., STRASSER, W. 1996. *Hardware for Superior Texture Performance. Computers & Graphics* 20, 4 (July 1996), 475 – 481.

LU, N. 1997. *Fractal Imaging*, Academic Press.

MCTAGGART, G., GREEN, C., MITCHELL, J. 2006. High dynamic range rendering in valve's source engine, July 2006, *Siggraph 2006: ACM SIGGRAPH 2006 Courses*.

MUNKBERG, J., CLARBERG, P., HASSELGREN, J., AKENINE-MÖLLER, T. 2006. High Dynamic Range Texure Compression For Graphics Hardware, *ACM Transactions on Graphics,* (*Proceedings of ACM SIGGRAPH 2006*), vol. 25, no. 3, 698 – 706.

MUNKBERG, J., CLARBERG, P., HASSELGREN, J., AKENINE-MÖLLER, T. 2007. Practical HDR Texture Compression, Technical Report, ISSN 1404-1200, Report 92, August 2007 - Submitted to CGF.

OWENS. J., 2005. Streaming Architectures and Technology Trends, In *GPU Gems 2*, Matt Pharr editor, chapter 29, 457 – 470. Addison Wesley, March 2005.

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

POYNTON, C. A. 2003. *Digital Video and HDTV*: Algorithms and Interfaces, Morgan–Kaufmann.

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

ROCA, J., MOYA, V., GONZÁLEZ, C., SOLIS, C., FERNÁNDEZ, A., ESPASA, R. 2006. Workload Characterization of 3D Games, IEEE *International Symposium on Workload Characterization* (IISWC-2006), 17 – 26.

ROIMELA, K., AARNIO, T., IT, J., Efficient high dynamic range texture compression, *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, 2008, ACM, 207 – 214.

ROIMELA, K., AARNIO, T., ITÄRANTA, J. 2006. High Dynamic Range Texture Compression, *ACM Transactions on Graphics* (*Proceedings of ACM SIGGRAPH 2006*), vol. 25, no. 3, 707 – 712.

STACHERA, J., ROKITA, P. 2006. Fractal-based hierarchical mip-pyramid texture compression. *International Conference on Computer Vision and Graphics ICCVG 2006*, Book of Abstracts, September 25-27, 2006, Warsaw, Poland.

STRÖM, J., AKENINE-MÖLLER T. 2005. iPACKMAN: High-Quality, Low-Complexity Texture Compression for Mobile Phones, *Graphics Hardware 2005*, 63 – 70.

TORBORG, J., KAJIYA, J. 1996. Talisman: Commodity Realtime 3D Graphics for the PC, *SIGGRAPH '96*, ACM, New York, NY, 1996, 353 – 363.

WANG, L., WANG, X., SLOAN, P., WEI, L., TONG, X., GUO, B. 2007. Rendering from Compressed High Dynamic Range Textures on Programmable Graphics Hardware, *SIGGRAPH Symposium on Interactive 3D Graphics and Games 2007*, 17 – 24.

WARD, G. 1992. Real Pixels, *Graphics Gems II*, Academic Press, 80 – 83.

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

WARD, G., MARYANN S. 2005. JPEG-HDR: A Backwards-Compatible, High Dynamic Range Extension to JPEG, In *Proceedings of the Thirteenth Color Imaging Conference*, November 2005.

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

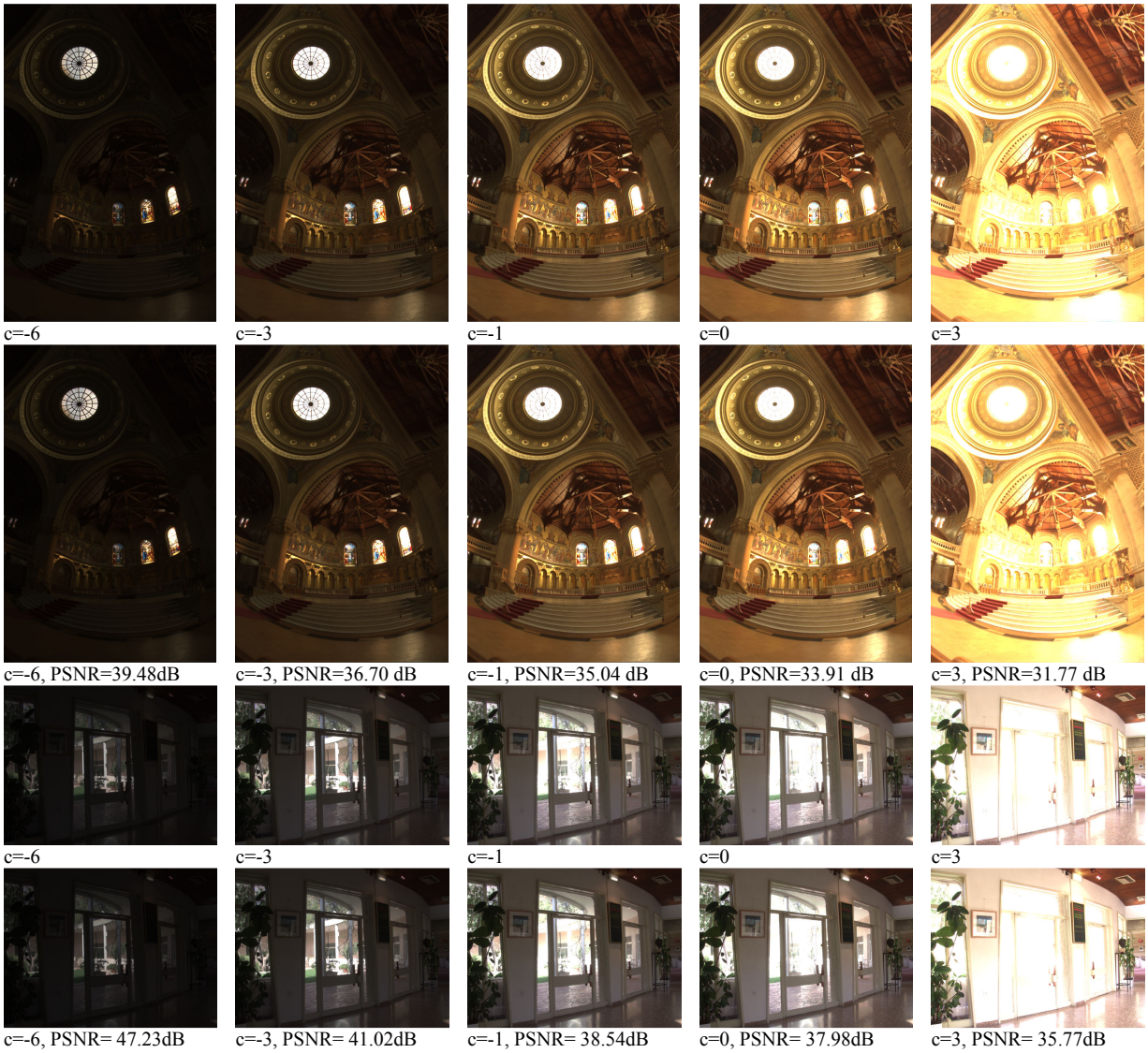


Figure 9. HDR compression result for proposed algorithm for images: memorial (Figure 5i) and dani belgium (Figure 5d). The top row shows the original image while the bottom row shows our reconstruction with PSNR error. Each column represent images tone-mapped based on exposure c (see section 5).

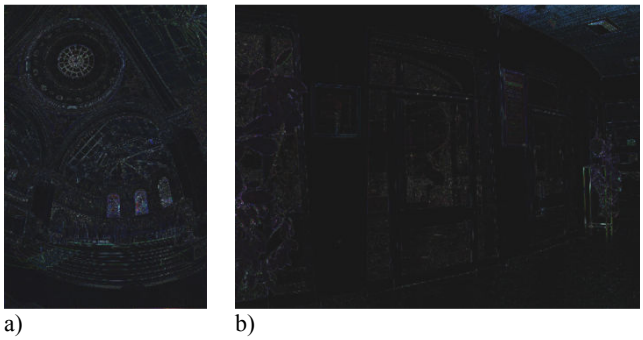


Figure 10. Gamma corrected error images of square log differences: a) memorial (Figure 5i, 6), b) dani belgium (Figure 5d, 6).