Pacific Graphics 2014
J. Keyser, Y. J. Kim, and P. Wonka
(Guest Editors)

Parallel BTF Compression with Multi-Level Vector Quantization in OpenCL

P. Egert and V. Havran

Faculty of Eletrical Engineering
Czech Technical University, Czech Republic

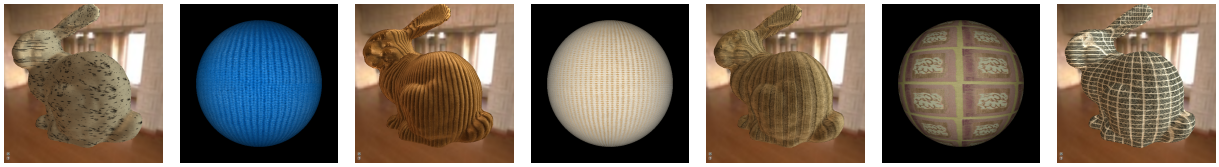


Figure 1: Example of BTF datasets used in rendering for bunny and sphere: ceiling, wool, corduroy, proposte, pulli, wallpaper, impalla. Environment map illumination (odd columns) and point light illumination (even columns) is used, average compression ratio 1:631, average compression time 3 hours, rate of decompression 127 million evaluations per second on a GPU.

Abstract

Bidirectional Texture Function (BTF) as an effective visual fidelity representation of surface appearance is becoming more and more widely used. In this paper we report on contributions to BTF data compression for multi-level vector quantization. We describe novel decompositions that improve the compression ratio by 15% in comparison with the original method, without loss of visual quality. Further, we show how for offline storage the compression ratio can be increased by 33% in total by Huffman coding. We also show that efficient parallelization of the vector quantization algorithm in OpenCL can reduce the compression time by factor of 9 on a GPU. The results for the new compression algorithm are shown on six low dynamic range BTFs and four high dynamic range publicly available BTF samples. Our method allows for real time synthesis on a GPU.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Shading, texture I.4.1 [Image Processing and Computer Vision]: Digitization and Image Capture—Quantization, Reflectance

1. Introduction

The Bidirectional Texture Function (BTF), introduced by Dana et al. [DvGNK99], is a practical and powerful image based technique for representing the visual appearance in the real world. It can be understood as a six-dimensional function for monochromatic data of fixed wavelength. In contrast to spatially varying BRDFs, BTF allows to capture visually complex phenomena such as subsurface scattering, self-shadowing, self-occlusion, and inter-reflections. The BTF data are acquired by taking thousands of photographs. The BTF data storage can be solved by using a suitable lossy fast compression algorithm with the highest possible compression ratio and with high speed of decompression on various computer hardware architectures.

In this paper we extend the compression method based on Multi-Level Vector Quantization (MLVQ) for BTF data proposed by Havran et al. [HFM10]. We propose alternate decomposition schemes that improve the compression ratio. We also show the use of Huffman coding to further improve the compression ratio. To decrease the compression time, we are first to show how MLVQ algorithm can be parallelized on contemporary GPUs using OpenCL [MGMG11]. We show that a parallelization of the BTF compression algorithm is possible and achieves a significant reduction in computation time.

2. Related Work

Bidirectional Texture Functions. The acquisition and compression of BTF has raised significant interest since its introduction by Dana et al. [DvGNK99]. A recent survey com-

paring acquisition approaches and compression methods for the first decade since introduction was presented by Filip and Haindl [FH09].

Several principles are used in lossy BTF data compression, including data fitting, factorization, tensor decomposition, inverse procedural modeling, etc. One of the most explored methods uses *fitting of analytical functions* such as Lafortune model lobes to the data in the method of McAllister [MLH02] and by Tsai and Shih [TS12] for real time rendering. Another set of methods is based on *Principal Component Analysis* (PCA) or its variant local PCA. Filip et al. [FCGH08] used a perceptual metric to reduce selectively some input data as the input to PCA compression. More recently, Havran et al. [HFM10] have proposed compressing BTF by MLVQ, taking BTF as a conditional probability density function. Our algorithm extends this method, so we describe it in greater detail in the next section.

Vector quantization on GPUs. Vector quantization [GG92] allows efficient mapping of an input set of vectors to a small representative set of vectors, referred to as a codebook, using a distance measure between two vectors. Its implementation on a GPU was used e.g. for volumetric compression [ZYX*11].

3. Multi-Level Vector Quantization for BTF data

MLVQ for BTF compression [HFM10] in fact extends the hierarchical vector quantization introduced by Gersho and Shoham [GS84] by indices and scales for improved adaptive modeling of the input data. A description of the algorithm and its settings can be found out in the original paper [HFM10] and we due to the lack of space need cannot describe it in more detail, it almost corresponds to the decomposition used here is shown in Figure 2b.

Several key components are needed to make the algorithm efficient: suitable parameterization over a hemisphere allowing such a hierarchical decomposition, resampling algorithm for BTF data, and error metric required by vector quantization (*Structural Similarity Index Metric* (SSIM) [WBSS04] that utilizes the neighborhood of a pixel compared against the reference, and can to some extent model such effects such as visual masking, luminance, and contrast measures). In the MLVQ algorithm, the SSIM is computed for a currently encoded vector over already existing vectors for all the codebooks along the pipeline. It achieves about a ten times higher compression ratio than a single level vector quantization. The similarity of vectors in a codebook and a candidate vector is also given up to a scale factor. Another difference is that ordinary vector quantization is applied for monochromatic data, but the BTF data are trichromatic.

Compression is then computed relatively simply by data pruning [GS84] in a hierarchical way starting from the 4D codebook and ending in 1D codebook. The lowest dimensional codebook stores the data in a simple table representing the data vectors themselves.

BTF decompression given a particular 6D coordinate is computed by chained indexing from the existing codebooks, with necessary interpolation among data vectors to allow for visually pleasing results. When implemented naively, so that the scale factors are represented by floating point values stored as 4 Bytes and indices among codebooks by 4 other Bytes (denoted as C.R.[1] in Table 1), has a mild compression ratio of the order of 1:200. Without deteriorating the visual quality, the scale factors can be represented by 8 bits for LDR data and by 16 bits for HDR data (scalar quantization), and the indices among codebooks can be represented by only necessary number of bits (denoted as C.R.[4] in Table 1). For this variant of the algorithm the compression ratio was reported to be between 1:233 and 1:2267, with an average of 1:764 for twenty BTF samples in the original paper. Some of the used BTF samples have not been made publicly available, so reproducibility cannot be verified.

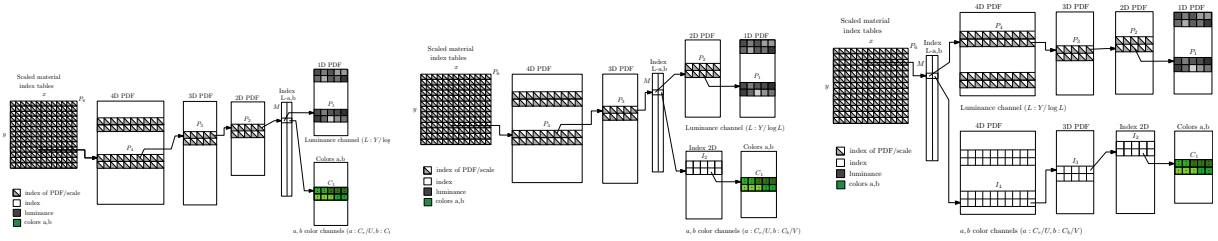
4. New BTF Compression Algorithm

In this section, we describe briefly two new modifications to the compression scheme as described in the previous section and in detail in [HFM10]. We also described how to address the relatively high compression time (average 20 hours for one BTF data 256×256 texels) by a parallel algorithm.

4.1. Adaptive Multi-Level Vector Quantization

There is an interesting but unjustified choice in the original algorithm related to the decomposition shown in Figure 2b. A common problem of all BTF compression algorithms is due to the necessity to represent (at least) trichromatic BTF data as three separate channels. There is obviously some redundancy as most real world BTF samples exhibit only slightly varying chromaticity for a single BTF texel. In MLVQ algorithm, this is handled by separation to luminance and chromaticity in codebook M, which merges together the two-dimensional luminance data (one channel) given by codebook P_2 and the two-dimensional chromaticity data (two channels) given by codebook I_2.

It is not obvious, where to separate the data of the luminance and chromaticity channels. The original paper seems to make an ad-hoc decision to separate them between codebook P_2 and P_3. It is unclear when we want to get the highest compression ratio, if the vector data in VQ are to be shared, including luminance and chromaticity, for many vectors so the separation should come as late as possible. However, it can be argued that they should be separated as soon as possible, since they are not sufficiently similar. We designed a total of four decomposition schemes, where one corresponds to the original scheme [HFM10] with only a small difference. The three of four implemented and tested decomposition schemes are shown in Figure 2, where the separation



(a) Luminance-chromaticity separation in 1D. (b) Luminance-chromaticity separation in 2D. (c) Luminance-chromaticity separation in 4D.

Figure 2: Three of four implemented and tested BTF decomposition schemes for compression using MLVQ algorithm with luminance and chromaticity is separated in one, two, and four dimensions by the codebook M representing indices.

of luminance and chromaticity is either at 1D, 2D, or 4D dimension, using a common index codebook M. This common index codebook M in fact implements the color model conversion from sRGB space (3 channels) to luminance (1 channel) and chromaticity (2 channels).

4.2. Optional Huffman Coding of Indices

The indices stored in any individual codebook pointing to a lower dimensional codebook present another possible potential redundancy that can be utilized to improve the compression ratio. We show below how we can apply simple Huffman coding [Huf52] as a method for lossless compression based on the entropy of individual symbols. This well-known algorithm first computes the static frequency of the symbols in the data, and its time complexity is $O(N.\log N)$ for N symbols, knowing the probability of the individual symbols. As symbols we take the indices for each codebook that point from one codebook to another lower dimensional codebook. We calculate the size needed to represent both the binary tree of Huffman coding and the size of the encoded stream representing the indices. We choose such a representation that requires less memory. This step is applied as post-processing after the whole set of codebooks is constructed.

4.3. Parallelization of Compression with OpenCL

The compression pipeline functionality is designed so that parallelism is exploited at several levels. First, decompression to the reconstruction caches is executed in parallel in all the codebooks. For 4D PDF only one vector is decompressed, while for 3D PDF all necessary vectors are decompressed that correspond to 2D PDFs, etc. Second, the computation of SSIM errors is computed in parallel for all data vectors in the codebooks. Third, the conversion from sRGB to the selected color model is computed in parallel. Fourth, when the vectors are inserted into the codebook, it is done by a single thread and can be done in parallel by more than one thread simultaneously. We used OpenCL [MGMG11] as the parallel API to keep the parallel algorithm platform independent.

To implement parallelization efficiently we designed a new modular task-driven scheme for the compression pipeline. This uses several functional nodes that work independently for compression and for decompression without knowledge of the rest of the pipeline. These nodes have basically three obligatory parts: the input data, the output data, optionally compare unit, and the settings to describe their functionality, This design of parallelization allows high flexibility of the compression pipeline, specified in an XML file.

5. Results

We implemented the proposed algorithm on a PC running MS Windows 7, x64 architecture, and GPU NVidia GTX780 Ti (GK110). We used the same discretization over the hemisphere as in the original paper ($16 \times 7 \times 11 \times 11$), and all the data were compressed for 256×256 texels to allow for meaningful comparison. A summary of the results for 6 LDR and 4 HDR BTF samples from the Bonn BTF repository is shown in Table 1. The time for compressing a single BTF sample of size 256×256 texels varies, and is on an average 3 hours (about 9 times faster than the compression time than in [HFM10]) for BTF spatial resolution 256×256 texels. The smaller the compression ratio, the higher the compression time, as larger codebooks require more time for searching.

5.1. Discussion

Table 1 shows that it is not possible to select one of the decomposition schemes for all the BTF samples to get the highest compression ratio and the same visual quality for a single BTF sample. The resulting compression ratios differ while visual quality is the same. To get the highest compression ratio we need to compress the BTF material in all four decompositions and select the one that has the highest compression ratio. In comparison with the original paper [HFM10], we achieve a higher compression ratio by 15% on an average for C.R.[4] than in the original paper, and also slightly smaller error for SSIM. The compression ratio when using Huffman coding is improved in total by 33%. The speed of decompression is even higher than for the original paper, as one codebook is omitted (for storing individual colors) and reaches a frame rate over 150 FPS (and about 600

BTF	T [h] *Ha10*	T [h] *our*	$C.R.^1$ *Ha10*	$C.R.^4$ *Ha10*	$C.R.^1$ 1D *our*	$C.R.^1$ 2D *our*	$C.R.^1$ 3D *our*	$C.R.^1$ 4D *our*	$C.R.^4$ 1D *our*	$C.R.^4$ 2D *our*	$C.R.^4$ 3D *our*	$C.R.^4$ 4D *our*	$C.R.^4$ best *our*	$C.R.^4$ HC *our*	*ratio* $[\frac{Best}{2D}]$ *our*	*ratio* $[\frac{HC}{2D}]$ *our*	MSSIM *Ha10*	MSSIM *our*
UBO BTF database (LDR)																		
corduroy	19.2	1.1	1:128	1:418	1:170	1:142	1:142	1:141	**1:485**	1:422	1:420	1:417	1:485	1:561	+15%	+33%	0.748	0.731
impalla	21.8	2.0	1:162	1:522	1:184	1:177	1:169	1:172	**1:536**	1:512	1:491	1:501	1:536	1:614	+5%	+20%	0.730	0.854
proposte	18.0	1.1	1:236	1:806	1:284	1:306	1:305	1:309	1:838	1:931	1:939	**1:954**	1:954	1:1107	+2%	+19%	0.710	0.786
pulli	27.1	10.6	1:87	1:264	1:55	1:58	1:60	1:62	1:143	1:157	1:163	**1:170**	1:170	1:208	+8%	+32%	0.699	0.770
wallpaper	28.8	1.1	1:222	1:728	1:170	1:195	1:227	1:245	1:481	1:593	1:707	**1:767**	1:767	1:875	+29%	+48%	0.776	0.770
wool	50.2	7.3	1:77	1:233	1:83	1:87	1:86	1:85	1:220	**1:239**	1:235	1:232	1:239	1:278	+0%	+16%	0.684	0.763
UBO BTF database (HDR)																		
ceiling$^\diamond$	20.1	1.6	1:235	1:780	1:318	1:291	1:227	1:203	**1:855**	1:733	1:611	1:532	1:855	1:984	+16%	+34%	0.711	0.839
floortile$^\diamond$	28.7	3.2	1:136	1:360	1:216	1:198	1:186	1:195	**1:567**	1:533	1:506	1:529	1:567	1:663	+6%	+24%	0.772	0.893
pinktile$^\diamond$	15.6	0.7	1:711	1:2267	1:483	1:389	1:278	1:220	**1:1286**	1:968	1:691	1:551	1:1286	1:1455	+32%	+50%	0.961	0.932
walkway$^\diamond$	37.4	21.3	1:102	1:257	1:123	1:138	1:148	1:162	1:305	1:370	1:421	**1:446**	1:446	1:563	+20%	+52%	0.884	0.891
Avg(LDR)	27.5	3.2	1:152	1:495	1:158	1:161	1:165	1:169	1:451	1:476	1:493	1:507	1:525	1:607	+10%	+28%	0.725	0.779
Avg(HDR)	25.5	2.8	1:296	1:967	1:285	1:254	1:187	1:195	1:753	1:651	1:557	1:515	1:789	1:916	+21%	+41%	0.832	0.889
Avg	26.7	3.0	1:210	1:664	1:209	1:198	1:172	1:179	1:572	1:546	1:518	1:510	1:631	1:731	+15%	+33%	0.768	0.823

$^\diamond$: HDR sample, 12 bits per channel, unmarked samples are LDR, 8 bits per channel. 256×256 texels are used for all samples.

Table 1: The compression time, ratio, and error with respect to the original data using MSSIM [WBSS04] over the rendered images. The reference method *Ha10* [HFM10] is shown for two settings. $C.R.^4$ HC refers to the use of Huffman coding over the scheme that has the highest compression ratio. Values marked in bold are the best for $C.R.^4$. The averages are reported.

FPS for GLSL) when the whole screen of size 800×600 is covered by BTF material rendered and illuminated by a single point light. We also analyzed the memory pattern that is handled by OpenCL itself. The 3 GBytes GPU memory is fully exploited during compression, while part of the data is in the main memory and is transferred to and from as needed.

6. Conclusions

We have proposed a novel BTF compression algorithm that uses new decomposition schemes adaptively for MLVQ. The compression ratio is increased by 15%, on an average, improved with the Huffman coding of the indices in total by up to 33%, on an average. We have parallelized the compression algorithm on a GPU in OpenCL, where we achieve a significant time reduction by a factor of 9 against single threaded implementation on a CPU.

Acknowledgments

We would like to acknowledge the use of BTF data from UBO. This research was partially supported by the Czech Science Foundation under projects No. P202/12/2413 and GA14-19213S, and by the Grant Agency of the Czech Technical University, grant No. SGS13/214/OHK3/3T/13.

References

[DvGNK99] DANA K., VAN GINNEKEN B., NAYAR S., KOENDERINK J.: Reflectance and Texture of Real-World Surfaces. *ACM Transactions on Graphics 18*, 1 (1999), 1–34. 1

[FCGH08] FILIP J., CHANTLER M. J., GREEN P. R., HAINDL M.: A Psychophysically Validated Metric for Bidirectional Texture Data Reduction. *ACM Trans. Graph. 27*, 5 (Dec. 2008), 138:1–138:11. 2

[FH09] FILIP J., HAINDL M.: Bidirectional Texture Function Modeling: A State of the Art Survey. *PAMI, IEEE Transactions on 31*, 11 (Nov 2009), 1921–1940. 2

[GG92] GERSHO A., GRAY R. M.: *Vector Quantization and Signal Compression*. Communications and Information Theory. Kluwer Academic Publishers, Norwell, MA, USA, 1992. 2

[GS84] GERSHO A., SHOHAM Y.: Hierarchical Vector Quantization of Speech with Dynamic Codebook Allocation. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '84.* (Mar 1984), vol. 9, pp. 416–419. 2

[HFM10] HAVRAN V., FILIP J., MYSZKOWSKI K.: Bidirectional Texture Function Compression Based on Multi-Level Vector Quantization. *Computer Graphics Forum 29*, 1 (jan 2010), 175–190. 1, 2, 3, 4

[Huf52] HUFFMAN D.: A Method for the Construction of Minimum-Redundancy Codes. In *Proc. of the IRE 40*, 9 (Sept 1952), 1098–1101. 3

[MGMG11] MUNSHI A., GASTER B., MATTSON T., GINSBURG D.: *OpenCL Programming Guide*. OpenGL Series. Addision Wesley, 2011. 1, 3

[MLH02] MCALLISTER D. K., LASTRA A., HEIDRICH W.: Efficient Rendering of Spatial Bi-directional Reflectance Distribution Functions. In *Proc. of conference on Graphics Hardware*, HWWS '02, pp. 79–88. 2

[TS12] TSAI Y.-T., SHIH Z.-C.: K-clustered Tensor Approximation: A Sparse Multilinear Model for Real-time Rendering. *ACM Trans. Graph. 31*, 3 (June 2012), 19:1–19:17. 2

[WBSS04] WANG Z., BOVIK A., SHEIKH H., SIMONCELLI E.: Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (April 2004), 600–612. 2, 4

[ZYX*11] ZHAO L., YUE G., XIAO D., ZHOU X., YU X., YU F.: A Content-based Classified Hierarchical Vector Quantization Algorithm for Volume Compression. *JSW 6*, 2 (2011), 322–330. 2