High-Quality Wavelet Compressed Textures for Real-time Rendering

Nico Grund
Philipps-Universität Marburg
Hans-Meerwein-Str.
35032 Marburg, Germany
ngrund@informatik.uni-marburg.de

Nicolas Menzel
Philipps-Universität Marburg
Hans-Meerwein-Str.
35032 Marburg, Germany
menzel@informatik.uni-marburg.de

Michael Guthe
Philipps-Universität Marburg
Hans-Meerwein-Str.
35032 Marburg, Germany
guthe@informatik.uni-marburg.de

ABSTRACT

Although modern graphics hardware provides up to 1.5 gigabytes of memory, methods for effective texture compression are still required since there is always demand for more detailed and realistic images. In this paper, we present a method for the effective compression of large images and textures based on a quadratic B-Spline wavelet. The transformation is followed by a tree-compaction algorithm, which achieves high compression ratio at good image quality.

Keywords

Texture Compression, Wavelets.

1 INTRODUCTION

Compression of large textures and images is of crucial interest in many fields in computer graphics. The programmability of modern GPU allows texture and image compression and decompression algorithms to exploit the full parallel processing power and streaming capability. However, one considerable obstacle is yet the limited support and capacity for general purpose data storage on the graphics card: Though provided with up to 1.5 gigabytes of memory, modern GPU's texture size is still limited to currently 8192x8192 pixels.

Wavelet encoding has proven to be an appropriate tool for image compression, as in JPEG2000 [15]. Advantages are that it is easily implemented in software and can be adapted to hardware for improved performance [16]. There are several benefits arising from wavelet compression. First, the encoding itself leads to a straightforward lossy compression scheme by quantizing the coefficients. By encoding the wavelet coefficients into a quadtree, some memory can be saved by removing subtrees containing only zero coefficients after quantization. This means that textures will require less memory for storage, allowing them to fit into the limited texture size of the graphics card without the use of tiling. This way a shader program can be used for decompression and filtering.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Based on these observations, we present a compact tree-coding algorithm for the efficient high-quality compression of two-dimensional image data and a real-time random access decompression algorithm running on the GPU. Our approach is easily extensible to multidimensional data and to non-linear HDR data.

2 RELATED WORK

The S3 Inc. introduced five simple lossy block-decomposition-based compression schemes with compression rates of 4:1 and 8:1 [10] for 8-Bit RGBA images, which have been adopted by the Microsoft DirectX framework. Based on the observation that large textures, as required for terrain rendering, are not supported by graphics cards, Tanner et al. [14] proposed the clipmapping algorithm, which subdivides a huge texture into small tiles which fit into the texture memory.

For the compression of images the JPEG2000 standard [15, 2] supports the use of the LeGall and the Cohen-Daubechies-Feauveau $7/5$ wavelet, superseding the discrete cosine transform used in regular JPEG compression. Wavelet-based compression schemes have proven to be more flexible, providing higher compression rates while yielding higher quality. Compared to other algorithms, they demand a higher decompression complexity. Therefore, recent work aimed at the use of modern graphics-hardware to yield interactive frame rates.

Beers et al. [1] introduced a vector-quantization-based technique that uses a precomputed codebook and stores a smaller texture of indices into this codebook. The size of the codebook determines the level of compression. More recently, Fenney [4] described a way to store a compressed texture so that decompres-

sion needs one lookup per sample only. Schneider et al. [12] introduced a compression scheme for static and time-varying volumetric datasets. This algorithm is based on a vector quantization with a fixed bit-rate. To initialize the compression they use a codebook, which is obtained using a splitting technique. The number of generated entries is confined to the bit-rate of the quantization. The compression rate is nearly 20:1.

Shaprio [13] presents the embedded zerotree wavelet algorithm (EZW), which is based on a discrete wavelet transform and a zerotree coding to store a compact multiresolution representation of significance maps, which contains the positions of the significant wavelet coefficients. This method can provide good performance with very low complexity. The disadvantage of the EZW procedure is, that all values are classifed by an certain threshold. Coefficients below this threshold are simply omitted. As an result of this it will remove noise in uniform regions but also it generates blurry artifacts in the reconstructed image. DiVerdi et al. [3], proposed a method to implement the EZW algorithm for decoding on graphics hardware using the Haar wavelet. The wavelet coefficients are arranged in a tree with a zero node, where all child pointers of the leaves and nodes, which contain coefficients equal to zero, point to the zero node. While they achieve good compression rates, noisy images are problematic since too few wavelet coefficients are sufficiently close to zero for an imperceivable difference.

3 WAVELET-TRANSFORMATION

Wavelet transformation in general has been well studied in literature so we will not discuss it in detail. The most important property of the wavelet transformation is that it decomposes the image into perceptually meaningful subbands that can afterwards be compressed more efficiently than the original image.

Before the wavelet transformation the gamma correction is applied for linearization of the intensity values. This step needs to be replaced by a log-mapping in the case of HDR data. In both cases, the RGB color space is converted to the $Y'P_bP_r$ color space, where a luminance value and two differential color values are stored to consider the human visual system, which is more sensitive to changes in luminance than in color.

The choice of the wavelet basis is crucial for the wavelet compression. The two major characteristics of a basis are the width of support and the compression it can provide. A wider support yields better compression results, but is computational more expensive. We implemented three different wavelet bases in order to compare their benefits and disadvantages.

Our first implementation is the Haar wavelet, which has the most compact support. This simplicity makes it optimal for decoding performance. Disadvantages are, however, that it is neither continuous nor differentiable. This results in highly visible block artifacts in the compressed image.

The LeGall biorthogonal wavelet, which is also described in the JPEG2000 specification [15, 2], is continuous, but not differentiable. Compared to the Haar wavelet, it represents local changes in frequency smoother and thus produces more appealing compression results. As shown in Figure 1, its support is three times as wide as the Haar wavelet.

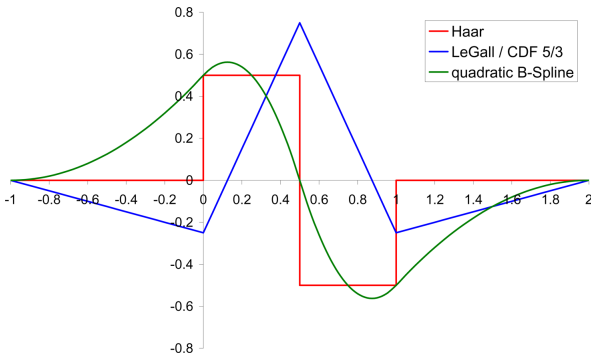

Figure 1: Haar, LeGall and quadratic B-Spline mother wavelet.

The quadratic B-Spline wavelet [11] is both continuous and differentiable. It therefore should achieve the best compression results compared to the two previous bases. It has the same support width as the Le Gall wavelet so the decompression performance is equivalent. The associated coefficients of the analysis and synthesis filter are shown in Table 1.

4 TREE-BASED COMPRESSION

Unfortunately, an entropy-based coding of the quantized wavelet coefficients as in image compressions algorithms like JPEG2000 is not suitable for real-time decompression on the GPU. Instead we first build a tree data structure from the wavelet decomposed image and then exploit redundancy in this tree by converting it into a general directed graph. In this procedure, identical or similar nodes are iteratively combined into a single node until a desired compression ratio is achieved.

	Analysis Filter Coefficients		**Synthesis Filter Coefficients**	
i	Lowpass Filter	Highpass Filter	Lowpass Filter	Highpass Filter
-1	1/4	1/4	-1/4	-1/4
0	3/4	3/4	3/4	3/4
1	3/4	-3/4	3/4	-3/4
2	1/4	-1/4	-1/4	1/4

Table 1: Coefficients of the quadratic B-Spline analysis and synthesis filter.

4.1 Wavelet tree

Based on the dyadic decomposition a natural tree structure for the wavelet coefficients is to store the LH, HL, and HH coefficients of a single pixel in the current level together with four pointers to the next finer level. The LL coefficient for the root level then needs to be stored outside the tree. This way the coefficients required to reconstruct a single pixel can be collected by traversing the tree from the root node to the leaf containing the highest resolution coefficients for that pixel. The major drawback is that for storing the quantized coefficients only 9 bytes are required, while the pointers require 12 bytes, when using up to 24 Bits which allows up to 16 MB for the compressed representation.

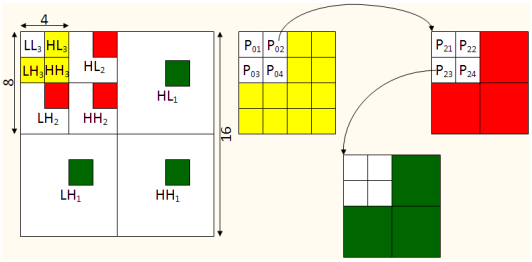

Figure 2: Dyadic decomposition and derived tree data structure.

In our approach we reduce the pointer overhead by grouping the wavelet coefficients of a two by two pixel block on each level. This way, only four pointers are required per 12 quantized YCC coefficients. Thus, the overhead is only 12 bytes per 36 bytes of data or in other words roughly 33%. Since the lowest resolution level only contains one coefficient of each type, four coefficients need to be stored outside the tree instead of only one. As these must be considered seperately anyways, we stop the wavelet decomposition at two by two pixels and store all of them as LL coefficients. Figure 2 shows the dyadic decomposition and the resulting tree data structure.

4.2 Tree compression

After the tree data structure is generated, redundant nodes are iteratively removed. Since combining two nodes also joins their subtrees, only nodes with the same children are candidates for such a collapse operation. The final data structure now is a general directed graph with the addition of a specifically marked root node (see Figure 3). As a collapse operation might introduce an approximation error the ordering of collapses as well as the choice which two nodes are collapsed at each step determine the quality of the decompressed result.

We use a priority queue to perform the node collapse operations in an optimal order on the directed graph. To minimize the total mean square error (MSE), the key

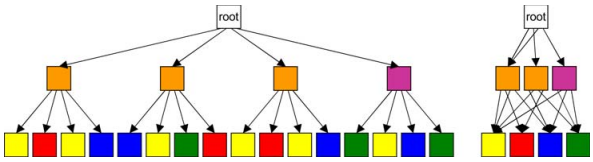

Figure 3: Uncompressed (left) and compressed (right) directed graph data structure. The colors depict identical coefficients stored in the tree nodes.

by which the operations are sorted needs to be proportional to the sum of squared differences (SSD) of all nodes collapsed together by this operation, i.e. all original nodes collapsed into the two candidates i and j. As the coefficients of each node are the average coefficients of all contained original nodes, this error $\varepsilon(i,j)$ can be computed by summing up the SSD of both nodes (ε_i and ε_j) with the appropriately weighted SSD when collapsing the coefficients of the two candidates:

$$\varepsilon(i,j) = \varepsilon_i + \varepsilon_j + d^2(i,j)\frac{w_i w_j}{w_i + w_j},$$

where $d^2(i,j)$ is the sum of squared differences of the coefficients of node i and j and $w_{i/j}$ is the sum of the weights of all original nodes collapsed to i and j, respectively. For the computation of the new coefficients, those of node i and j are simply multiplied by the weight stored in each of these nodes and divided by the new weight which is the sum w_i and w_j.

Since each node is always collapsed with the one for which the collapse has the lowest cost, we only need to find the closest node c_i for each node i and store this pair in the priority queue. This problem is similar to the nearest neighbor search in high dimensional spaces as our coefficient vector has a dimensionality of 36. The only exception is that the distance beween two nodes with different child nodes must be set to infinite to prevent collapsing them. Section 4.2 discusses the nearest neighbor search algorithm we use in more detail.

When a collapse is performed, some of the queue entries become invalid and need to be recomputed. Assuming that the new nearest neighbor of those nodes introduces a higher SSD we can postpone the recomputation until that collapse is fetched from the priority queue. The only nodes for which we need to immediately find the nearest neighbor are the newly constructed node and all nodes that had one of the two collapsed nodes as immediate children. The latter is necessary as these nodes might now have a closer neighbor than the one that was previously found. Another property we used to speed up the initial filling of the priority queue is that inner nodes cannot be collapsed before the first few leaf nodes were removed since they cannot initially have the same child nodes.

Zerotree coding In addition to the optimizations described above we can also remove all leaf nodes for

which their coefficients are all quantized to zero by introducing a zero node similar to [3]. Since the zerotree coding does not need a nearest neighbor search or a priority queue, those parts of the wavelet tree that do not contain any information can be quickly removed. In contrast to [3] we do, however, not collapse nodes containing near-zero coefficients although these might be collapsed with the zero node at a later time if the introduced SSD is the lowest one.

As this step reduces the total number of nodes before the first neighbor search and the maximum number of collapse operations in the priority queue it can significantly reduce the total runtime. This is especially important for images that required a padding before the wavelet transformation.

Nearest neighbor search As mentioned above the coefficient vector for which we need to find the nearest collapse candidate is 36-dimensional. Thus we require an efficient method to search the nearest neighbor in this 36-dimensional space. Since each collapse implies removing two and adding one point to the candidate set, a spatial acceleration data structure like the r-tree [6] cannot be used and we need to restrict ourselves to a linear ordering based on some sort of key value.

Fortunately, we can exploit the fact that most coefficient vectors will be centered around the origin with a more or less gaussian distribution. Therefore, we chose our hash function to be the distance to the origin and only need to search those node with a similar distance. As soon as we find the first candidate, we can thus efficiently stop searching in one of the two directions if points farther away or closer to zero cannot introduce a lower error.

5 IMPLEMENTATION

To achieve real-time decompression, we had to meet some contraints that are given by the graphics hardware. First the coefficient values have to be quantized to the range $0-255$ using a global scaling to the range $[0,1]$ when storing them in a 24 bit RGB texture. After this, all values are simply scaled by the factor 255. The quantization also restricts us to textures of size 256 in each dimension since the color value is to be directly used as texture coordinate. Therefore, we use a three-dimensional texture to encode the tree. The size of this texture is $256^2 \times 2^n$, where $0 \leq n \leq 8$ and each pixel uses 24 bit in the regular RGB format.

As shown in figure 5 we encode blocks in pairs of 4×4 pixels. In each block, the upper left four pixels contain pointers to the children of the current node. In each pointer pixel, the color values contain the texture coordinates of the child nodes upper left pixel. With this scheme, we can encode $64 \times 64 = 4096$ nodes in each layer of the texture so we can store up to one million nodes or 36 million unique coefficients.

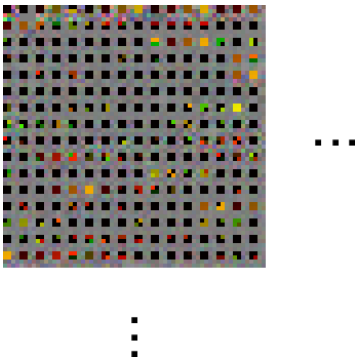

Figure 4: Part of the compressed wavelet data stored in the 3D-texture. Since the image is taken from depth 0, the root node is visible in the upper left corner.

5.1 Parallel decompression

For parallel decompression on the GPU, all wavelet functions contributing to the current pixel need to be evaluated and multiplied with the corresponding coefficients. The number of coefficients per pixel depends on the width of the mother wavelet and is one for Haar and three for LeGall und quadratic B-Spline in each dimension. This yields a total of 3 or 27 coefficients per level for Haar and LeGall/quadractic B-Spline, respectively. To extract these coefficients, one (Haar) or four nodes (both others) have to be visited per level. This sums up to 5 texture lookups per level for the Haar wavelet and 31 for the other two wavelets. Note, that since the child nodes are not queried at the leaf level, the total number of lookups if $4l-1$ for the Haar wavelet and $31l-4$ for both others, where l is the number of levels in the coefficient tree.

Although the number of lookups for the more complex wavelets might seem rather high, the Haar wavelet only allows nearest neighbor interpolation and thus produces inferior quality when zooming. To achieve bilinear interpolation the number of lookups for the Haar wavelet is quadrupled. This yields a total of $16l-4$ which is approximately half than that of the other two wavelets. Due to the smoother wavelet functions however, these produce better quality images at the same compression rate and thus the higher number of texture lookups is tolerable.

6 RESULTS

To evaluate our proposed algorithm and compare it to existing approaches, we mainly used images from the image compression benchmark [5] (Figure 6 and 7).

A quality comparison of the three implemented wavelet transformations is shown in Figure 5. The Haar wavelet shows significant block artifacts, which neither appear using the LeGall nor the quadratic B-Spline wavelet. Since the LeGall scaling function is a linear filter, it tends to produce star-shaped artifacts. The quadratic B-Spline wavelet reproduces sligthly

Figure 5: From left to right: Haar-, LeGall- and B-Spline wavelet.

more detail than the LeGall wavelet. In addition, the biquadratic interpolation that comes for free with the B-Spline wavelet generates smoother results when magnifying the image. The PSNR is similar for all three wavelets, where the Haar wavelet has the lowest (41.7 dB) and LeGall (45.4 dB) and quadratic B-Spline (42.9 dB) are slightly better.

Figure 6: Compression results with embedded zero tree coding (left) and with our approach (right).

Figure 6 shows the differences between embedded zero tree coding [13] (35.4 dB) and our method (37.6 dB). Both were compressed at a rate of 23:1. One disadvantage of the zero tree coding is, that all values below a certain threshold are simply omitted. While this removes noise in uniform regions, it cannot compress data in images containing high frequencies. In these cases the threshold needs to be significantly increased to achieve a desired compression ratio and thus the quality of the reconstructed image is degraded. In contrast to this, our clustering approach can also exploit similarities in high frequency regions and thus much fewer nodes need to be collapsed with the zero node. This greatly improves the visual quality when compressing this type of images.

Figure 7: Comparison between S3TC (middle) and our approach with same compression ratio (top, no visual difference to original) and same quality (bottom).

In Figure 7 a comparison between S3TC (DXT1) and our approach is shown. The upper two images are both encoded with a compression ratio of 6:1 (8:1 for RGBA images) at 55.1 dB. Note, that our approach yields a significant higher quality (60.7 dB) at slightly smaller texture size (1.5 MB compared to 1.6 for DXT1). With four times the compression rate (26:1) the visual quality of our method (still 57.1 dB) is equivalent to S3TC, as shown in the lower image. Figure 8 shows an aerial image with a resolution of 3000×3000 compressed at a rate of 34:1 with 36.4 dB. Despite the high compression rate, important features are still preserved.

The decoding was performed in a pixel shader running on an nVidia GeForce GTX 295 in real-time. The performance for a 4096×4096 texture is approximately 400 Mpixels per second using the Haar wavelet and nearest neighbor filtering (100 Mpixels with bilinear filtering) and roughly 55 Mpixels per second with the LeGall and quadratic B-Spline wavelet. For smaller or larger images, the runtime is almost linear in the number of levels of the wavelet decomposition. E.g. for a

Figure 8: Drastic compression (34:1) of a 3000×3000 pixel aerial image. The marked area is magnified below the full image.

16k$\times$16k texture we still achieve 46 Mpixels per second.

7 CONCLUSION AND LIMITATIONS

We presented an effective method for the compression of large textures and images based on the linear LeGall and quadratic B-Spline wavelet. With our tree-compaction algorithm, we achieve high compression ratios while still preserving high visual quality. The decompression is implemented as a pixel shader on a GPU and runs in real-time on current graphics hardware. Our approach has shown to be superior to simple zero-tree removal.

In the future we want to improve the compression time, which is currently 30 minutes for a 67 Mpixel image (8192×8192 pixel) and thus still rather slow. In addition, we want to extend our method to high dynamic range images and multi-dimensional datasets.

REFERENCES

[1] Andrew C. Beers, M.Agrawala, and N.Chaddha, Rendering from compressed textures In *SIGGRAPH '96: Proceedings of the 23rd annual conference on computer graphics and interactive techniques,* pp.373-378, 1996.

[2] C.Christopoulos, A.Skodras, T.Ebrahimi, The JPEG2000 Still Image Coding System: An Overview In *IEEE Transactions on Consumer Electronics, Vol.46, No.4,* pp.1103-1127, 2000.

[3] S.DiVerdi, N.Candussi, T.Höllerer, Real-time Rendering with Wavelet-Compressed Multi-Dimensional Datasts on the GPU In *Technical Report UCSB//CSD-05-05,* 2005.

[4] S.Fenney, Texture compression using low-frequency signal modulation In *HWWS'03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware,* pp.84-91, 2003.

[5] S.Garg, Image Compression Benchmark. *http:www.imagecompression.infotest_images*.

[6] A.Guttman, R-trees: a dynamic index structure for spatial searching In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data,* pp.47-57, 1984.

[7] P.Lalonde, A.Fournier, A wavelet representation of reflectance functions *IEEE Transactions on Visualization and Computer Graphics,* pp.329-336, 1997.

[8] P.Lalonde, A.Fournier, Interactive rendering of wavelet projected light fields In *Proceedings of the 1999 conference on Graphics interface '99,* pp.107-114, 1999.

[9] D.Le Gall, A.Tabatabai, Subband Coding of Digital Images Using Symmetric Short Kernel Filters and Arithmetic Coding Techniques In *Proceedings of the ICASSP 1988*, pp. 761-765.

[10] S3TC DirectX 6.0 Standard Texture Compression *S3 Inc,* 1998.

[11] F.F.Samavati, R.H.Bartels, Local Filters of B-spline Wavelets In *Proceedings of International Workshop on Biometric Technologies 2004,* pp.105-110.

[12] J.Schneider, R.Westermann, Compression Domain Volume Rendering In *Proceedings of the 14th IEEE Visualization 2003,* pp.39-47.

[13] J.M.Shapiro, Embedded Image Coding Using Zerotrees of Wavelet Coefficients In *IEEE Transactions on Signal Processing, Vol.41 No.12, 1993,* pp.3445-3462

[14] C.C.Tanner, C.J.Migdal, M.T.Jones, The Clipmap: A Virtual Mipmap In *Proceedings of SIGGRAPH 98,* pp.151-158.

[15] D.S.Taubman, M.W.Marcellin, JPEG2000: Image Compression Fundamentals, Standards and Practice *Kluwer Academic Publishers,* 2001.

[16] J.Wang, T.-T.Wong, P.-A.Heng, and C.-S.Leung, Discrete wavelet transform on GPU In *Proceedings of ACM Workshop on General Purpose Computing on Graphic Processors,* pp. C-41, 2004.