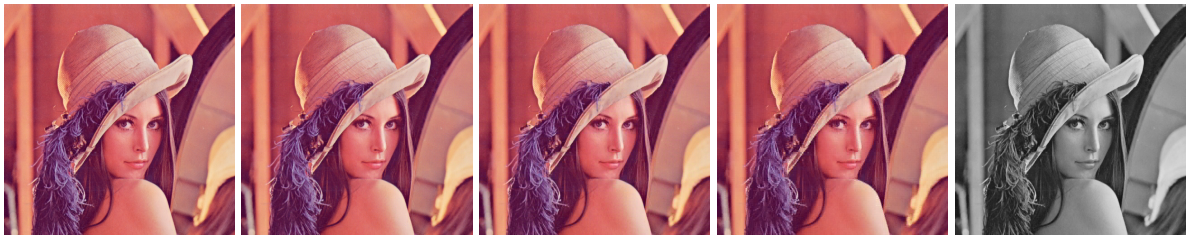
Pacific Graphics 2012
C. Bregler, P. Sander, and M. Wimmer
(Guest Editors)

Volume 31 (2012), Number 7

Texture Compression using Wavelet Decomposition

Pavlos Mavridis and Georgios Papaioannou

Department of Informatics, Athens University of Economics & Business



5 bpp 3 bpp 2.25 bpp 1.75 bpp 2 bpp

Figure 1: *Our method encodes grayscale and color images at various bitrates, improving the flexibility of existing texture compression formats.*

Abstract

In this paper we introduce a new fixed-rate texture compression scheme based on the energy compaction properties of a modified Haar transform. The coefficients of this transform are quantized and stored using standard block compression methods, such as DXTC and BC7, ensuring simple implementation and very fast decoding speeds. Furthermore, coefficients with the highest contribution to the final image are quantized with higher accuracy, improving the overall compression quality. The proposed modifications to the standard Haar transform, along with a number of additional optimizations, improve the coefficient quantization and reduce the compression error. The resulting method offers more flexibility than the currently available texture compression formats, providing a variety of additional low bitrate encoding modes for the compression of grayscale and color textures.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

Texture mapping is used excessively in computer graphics in order to increase the realism of the rendered scenes by adding visual detail to geometric objects. In today's real-time applications a tremendous number of high resolution textures are used to encode not only the surface colors, but also to specify surface properties like displacement, glossiness and small-scale details with normal or bump maps. All these textures consume large amounts of memory and bandwidth, causing storage and performance problems on commodity graphics hardware, where the available memory resources are often limited. To counter these problems various texture compression methods have been developed, reducing both the storage and the bandwidth requirements.

As noted by Beers et al. [BAC96] in one of the earliest works in the field, such a method should provide very fast random access to the compressed texels, while some loss of fidelity is acceptable, since the important issue is the quality of the rendered scene rather than the quality of individual textures. The requirement for fast random access excludes the application of traditional image coding methods, such as JPEG, or the more recent wavelet-based JPEG 2000.

Traditional transform coding methods encode images using a transformation with energy compaction properties, such as the *Discrete Cosine Transform (DCT)* or the *Discrete Wavelet Transform (DWT)*. After the application of such a transform, most of the spectral energy of the original image is compacted into a small

set of coefficients, which are then quantized taking into account perceptual metrics. Finally, the coefficients are reordered in order to group similar values together and they are compressed and stored using an entropy encoding method. This scheme can result in state-of-the-art coding performance, but as noted before, the variable per-pixel bit-rate and the inherently serial nature of entropy encoding makes it unsuitable for texture compression, where fast random access to the compressed data is critical.

In this paper we introduce a flexible wavelet-based compression scheme for low bitrate encoding of single-channel and color textures. We first describe a unique way to combine transform coding concepts with standard fixed-rate block compression, such as the industry standard DXT5 and BC7. We identify the challenges in this area and we propose a solution where, instead of entropy coding, the quantization and storage of the transform coefficients is performed using the industry standard DXT compression formats, ensuring efficient implementation using the already existing decompression hardware. Coefficients with higher contribution to the final image are stored at higher accuracy, resulting in good image quality even when low bitrates are used. To maximize quality and minimize the quantization error, an optimization framework scales the wavelet coefficients before the compression.

The goal of our method is not to replace the existing DXT compression formats, but to provide more flexibility in terms of compression rates. In particular, for grayscale and color images DXT offers only two compression rates at 4bpp and 8bpp (bits per pixel), while our encoding offers additional bitrates starting from 2bpp for grayscale and 2.25bpp for color textures, providing increased control on the trade-off between image quality and compression rate. For completeness, in Figure 1 we have also included a 1.75bpp variation of our format which uses two levels of wavelet decomposition, but it has a higher performance overhead (see Section 3.4).

2. Related Work

We first review general texture compression schemes, followed by a brief overview of DXTC/BC7 compression, and some software methods that improve or extend hardware texture compression and are related to our work.

2.1. General Texture Compression

Texture compression methods can be classified as fixed-rate, where the number of bits per pixel is constant throughout the image, and variable-rate, where the bit rate varies from pixel to pixel.

Fixed Rate Encoding. Beers et al. [BAC96] proposed a Vector Quantization (VQ) method for texture compression, where blocks of texels are replaced with indices to an optimized code book. Reasonable quality and fast random access is achieved, but the required random indirection to the memory and the size of the code books makes a fast implementation with caching impractical. A variation of this method was used in the Dreamcast game console.

Block truncation coding (BTC) [DM79] avoids the indirection of VQ. A gray scale image is subdivided into non-overlapping 4×4 blocks and each block is encoded independently. Two representative gray scale values are stored per block (8 bits each) and one bit per pixel is used to select between these values, thus giving an overall rate of 2 bpp. Color cell compression (CCC) [CDF*86] extends BTC by storing indices to two colors, instead of gray scale values, thus compressing color images at the same bit rate. However, the method suffers from visible color banding and artifacts at block boundaries. These artifacts are addressed in DXTC/S3TC [INH99] and later BPTC [BPT09]. Since our method is designed to take advantage of these two formats, we provide a brief overview of them in Section 2.2.

PVRTC [Fen03] encodes a texture using two low-frequency signals of the original texture and a high frequency but low precision modulation signal. The low frequency signals are bi-linearly up-scaled to yield two colors for each texel, which are then interpolated by the modulation signal, consisting of 2 bits per texel. The author presents both a 4 bpp and a 2 bpp version of the codec. The usage of the continuous low frequency signals helps to avoid blocking artifacts, often present in other block methods. PVRTC is used on Apple's iPhone and other mobile devices, but it is not available on desktops.

ETC1 [SAM05] divides each 4×4 block in two 4×2 or 2×4 sub-blocks, each with one base color. The luminance of each texel is refined using a modifier from a table, indexed using 2 bits per texel. Overall, the format uses 4 bpp. ETC2 [SP07] further improves ETC by providing three additional modes using some invalid bit combinations in the original format. Thanks to the sub-block division, ETC and ETC2 preserve more texture detail than S3TC.

ASTC [NLO11] introduces a new coding scheme which allows efficient bit allocation among different types of data. Similar to our method, different bitrates are supported by adjusting the block size. As noted by the authors, this format is still under development, but it is the only other texture compression format that offers similar flexibility to our method.

Variable Rate Encoding. The energy compaction properties of the DCT and DWT are widely known and have been used in state of the art image coders, but as noted in the introduction, such codecs are impractical for texture compression. Nevertheless, variations of these codecs are used in texture streaming solutions [vW06] [OBGB11], where the entropy encoded textures are decompressed to the graphics memory before the actual rendering. The application of wavelets to texture compression has been explored before in [DCH05] and [Bou08], where the typical entropy encoding step is skipped and the most important wavelet coefficients

are stored in a tree for fast random access, but the method does not produce significantly better results than S3TC, and requires a tree traversal for each texel fetch.

Software Methods. Van Waveren et al. [vWC07] present an 8 bpp high quality compression scheme which performs the compression in the YCoCg color space. Luminance is stored in the alpha channel of a DXT5 texture to improve quality. However the bandwidth and storage requirements are doubled compared to 4 bpp methods.

Kaplanyan [Kap10] proposes a method to improve the quality of the DXT1 format by normalizing the color range of 16-bit input textures before compression. Some aspects of the optimization framework proposed in this paper for quantization of the wavelet coefficients are inspired by this work.

2.2. DXT Compression

DXT1 encodes RGB data at 4 bpp. The image is subdivided in non-overlapping 4×4 blocks and the colors of each block are quantized into points on a line through color space. The endpoints of the line are defined using two 16-bit $R_5G_6B_5$ colors, and two additional points are defined using interpolation. For each pixel in the block, a 2-bit index is pointing to one of the four points in the line. The format also supports 1-bit transparency using a slightly different encoding, but this mode is not used in our method.

DXT5 encodes RGBA data at 8 bpp. Colors are encoded at 4 bpp in the same way as DXT1, and additional 4 bpp are used for the alpha channel. A line in the alpha space is defined using two 8-bit representative values, plus six interpolated points. For each pixel in the block, a 3-bit index is pointing to one of the eight points in the line. A very important observation, exploited by our method, is that the alpha channel in this format is encoded at higher accuracy than the color data. A texture format that stores only the alpha channel of DXT5 is also available, called DXT5/A.

The BPTC/BC7 format encodes both RGB and RGBA data at 8bpp. As with DXT1, the colors of each 4×4 block are quantized into points on a line through color space, but the quality is improved by allowing up to three endpoint pairs (lines) per block, effectively subdividing each block into partitions. Furthermore, the encoder can select between eight different encoding modes for each block. Each mode is tailored to different types of content, by offering different trade-offs between the number of partitions, accuracy of endpoints and the number of intermediate points. Four modes are available for RGB data and another four for RGBA data.

3. Wavelet Texture Compression

In this section we first introduce a wavelet-based compression scheme for single channel (grayscale) data, and then we demonstrate how this scheme can efficiently compress color images.

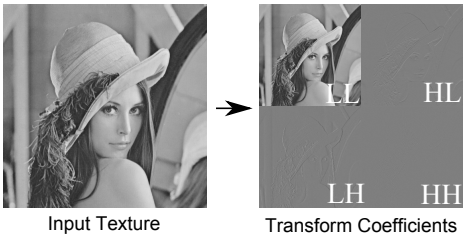


Figure 2: *The Haar transform of the standard lena image.*

3.1. Decomposition

The encoding process is performed off-line by the CPU. The first step in our encoding framework is to decompose the input image into four coefficient bands. This decomposition is based on the Discrete Wavelet Transform and in particular, a simple modification of the well known *Haar* Transform. A formal introduction to wavelets is beyond the scope of this paper. The interested reader is referred to [SDS95].

For each block of 2×2 texels in the input image, the Haar transform is defined by the following equation:

$$\begin{pmatrix} LL & HL \\ LH & HH \end{pmatrix} = \frac{1}{2} \begin{pmatrix} a+b+c+d & a-b+c-d \\ a+b-c-d & a-b-c+d \end{pmatrix} \quad (1)$$

where LL, LH, HL, HH are the resulting transform coefficients and a, b, c, d are the pixels in the original block of data. LL are referred to as *scale coefficients* and the rest are the *wavelet coefficients*. Coefficients of the same type can be grouped together in order to form four coefficient bands (sub-images), as shown in Figure 2.

After the wavelet transform, most of the spectral energy of the input image will be concentrated in the scale coefficients, while a large portion of the wavelet coefficients will be close to zero. This is demonstrated in Figure 2, where the input texture appears scaled in the LL coefficient band, while the three wavelet bands (LH, HL, HH) appear mostly grey, which is the color that corresponds to zero (this mapping is used because the wavelet coefficients are signed).

The number of the initial pixels and the number of the resulting coefficients after the wavelet transform is the same, thus the transform alone does not result in any storage space gains. The actual data compression in our method is achieved through the quantization of these coefficients using standard block compression methods, like DXT5 and BC7. A good compression strategy is to encode the LL coefficients with more precision, since they have the largest contribution to the final image. Contrary, HH have the least contribution, both because the energy of this band on typical images is rather low and also because the human visual system is known to be less sensitive to high frequency details, a fact also exploited by the JPEG quantization matrices [Wal91].

The first step towards a compression scheme that meets

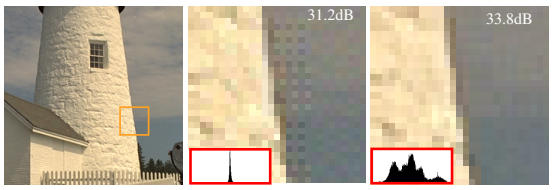


Figure 3: *Partially Inverted Haar (Sec. 3.1.1) and coefficient optimization (Sec. 3.2) eliminate the artifacts caused by inefficient DXTC encoding of the standard Haar coefficients. Inset: Histograms of the wavelet coefficients before and after our optimizations.*

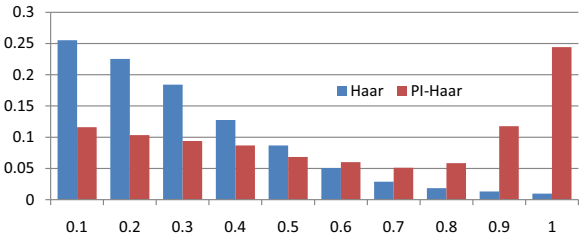


Figure 4: *Histogram counting the absolute Peasron's correlation of the 4×4 blocks encoding the walet coefficients of Haar and PI-Haar with w=0.4. PI-Haar improves the compressibility of the coefficients by increasing their correlation.*

these requirements is to encode the *LL* subband in the alpha channel of a DXT5 texture, taking advantage of the higher precision of this channel, and the three wavelet coefficients in the RGB channels. This effectively packs 2*x*2 blocks of the original texture in the RGBA texels of a DXT5 texture, thus each 128bit DXT5 block effectively encodes 8*x*8 texels, giving a 2bpp encoding rate for a single channel of data, which is half the bitrate of the native DXT5/A format. This rather naive approach, results in visible artifacts as shown in Figure 3(center). In the remainder of this section, we will investigate the reasons behind this failure and will propose ways to address it.

In DXT5 alpha is encoded independently but the three RGB channels are encoded together; the same per pixel 2-bit index is used to control the values in all of them. This works rather well on data that correlate rather well, which is usually the case for the colors of an image. On the other hand, wavelet coefficients normally would exhibit very little correlation, since the wavelet transform has well known decorrelation properties. We can verify this by counting the pairwise correlation coefficient of the 4*x*4 blocks being encoded. For the luminance channel of the standard Lena image we get the histogram in Figure 4, showing that very few blocks have high correlation (only 0.4% have a correlation coefficient higher than 0.8, with 1.0 being perfect correlation / anticorrelation). Therefore, as shown in the measurements of Table 1, when trying to compress the three wavelet coefficients together, the error can become quite high. In fact, compressing the HL channel alone by zeroing-out the other channels gives a rather low compression error, 4.6 MSE, but when the LH and HH channels are added back, the error in HL rises to 31.1 MSE, more than 6 times higher.

3.1.1. Partially Inverted Haar

To address this issue we modify the Haar transform in order to produce wavelet coefficients with a higher degree of correlation. We keep the *LL* band unmodified, but we define three new wavelet coefficients HL',HL' and HH' as a linear

	R	G	B	MSE_R
Haar	HL	0	0	4.6
	HL	LH	HH	31.1
PI-Haar	HL'	0	0	4.3
	HL'	LH'	HH'	11.0

Table 1: *Compression error of the R channel (MSE_R) when compressing a single coefficient subband vs three of them in the RGB channels of a DXT1 tetxure. Data for the Lena image. PI-Harr demonstrates better compressibility.*

combination of the traditional Haar wavelet coefficients

$$\begin{pmatrix} HL' \\ LH' \\ HH' \end{pmatrix} = \begin{pmatrix} 1 & 1 & w \\ -1 & 1 & -w \\ 1 & -1 & -w \end{pmatrix} \begin{pmatrix} HL \\ LH \\ HH \end{pmatrix} \quad (2)$$

where $0 \leq w \leq 1$ controls how much the HH subband influences the new coefficients. Since HH gives the worst correlation measurements among the other bands and also the spectral energy of this band is usually the lowest, a reasonable choice is to reduce the influence of this signal on the new coefficients. Our measurements indicate that the new coefficients exhibit higher correlation (Fig. 4) and hence better compressibility (Table 1). The error when compressing the new coefficients together does not increase so dramatically, as in the previous case. The new coefficients are actually the terms that appear in the calculation of the inverse Haar transform. The general idea is to partially invert the wavelet transform, thus adding some redundancy back to the coefficients. It is worth noting that a full inversion (ie skipping the transform) is not desirable since we will lose the advantage of energy compaction, something that results in a significant loss of quality.

To further improve the results, we can take advantage of the BC7 format, where each 4×4 block is subdivided into smaller partitions, and the RGB points of each partition are approximated by an independent line in the 3D color space (Sec. 2.2). This capability of BC7 is very important when encoding wavelet coefficients, because the correlation requirement for the three RGB values is only true for the smaller

area inside each sub-block and not the entire 4×4 block. In particular, discontinuities in the wavelet coefficients will usually happen along the edges of the original image. A good strategy is to partition the texels along the edge in a separate sub-block, since the coefficients over the edge will probably have similar values. In practice, the BC7 compression algorithm performs an exhaustive search on all partitionings and encoding modes for each 4×4 block, in order to find the combination with the minimum error. This makes the encoding time much slower than the DXT5 format. While encoding times are not important for our method, we should mention that encoding a 512×512 texture in the BC7 format requires roughly 5 minutes on a 2GHz Core2 CPU, while DXT5 encoding is instantaneous.

BC7 supports both RGB and RGBA data, but most RGBA modes do not take advantage of sub-block partitioning. Thus, the transform coefficients should be encoded using only three channels. To this end, perhaps the only reasonable choice is to completely drop the HH subband, since it is the least significant perceptually, as noted before. Substituting $w = 0$ in Equation 2, gives the wavelet coefficients $HL' = HL + LH$ and $LH' = HL - LH$. Along with LL, we store them in the RGB channels. The quantization weights during the BC7 compression are set at 0.6 for the LL band and 0.2 for HL' and LH', since the scale coefficients have a higher contribution to the final image. It is worth noting that the encoder, on a per-block basis, can decide to independently encode the LL channel in the alpha, using internal swizzling [BPT09]. With this packing, each texel of the BC7 texture encodes a $2x2$ block of texels from the original texture, thus a full 128bit BC7 block encodes $8x8$ texels, giving a bitrate of 2bpp.

Although, we have completely dropped the HH coefficients, this encoding mode outperforms the one based on the DXT5 format for most images and eliminates most of the compression artifacts. A simple strategy for the off-line encoder to determine the optimal mode for a particular texture is to perform the encoding using both modes and then keep the mode with the smallest error.

3.2. Coefficient Optimization

A quick inspection of the histogram of a wavelet coefficient subband, shown in Fig. 3(inset), reveals that most of the coefficient values are clustered/biased towards zero(the middle), while the edges of the spectrum are empty or very sparsely occupied. The range of the wavelet coefficients has been expanded relatively to the input domain, thus these values need to be quantized to 8 bits in order to be stored as regular fixed point textures. During this scalar quantization process, many values at the middle of the spectrum will be mapped at the same point. On the other hand, many quantization points at the edges will remain unused, thus reducing the efficiency of our method and creating unwanted artifacts.

A straightforward way to address this issue is to calculate the exact range of the wavelet coefficients and use it to perform the quantization, effectively normalizing their color range. Nevertheless, the existence of some coefficients near the edges of the spectrum, called *outliers*, can limit the gain from this step. Therefore, a number of outliers must be clamped, in order to improve the quantization of the coefficients near zero. The resulting range (minimum and maximum) is stored along with the compressed texture and is used during the decompression, in order to scale the coefficients back to their original range.

Furthermore, it's clear that a logarithmic quantizer, that allocates more quantization points at the middle of the spectrum could perform better than a linear one, where the quantization intervals are evenly spaced. Instead of using a logarithmic quantizer, we perform an exponential scale to the data in order to redistribute the values more evenly in the spectrum. This exponential scale corresponds to a change in the color space. During decoding, the original color space should be restored. It is worth noting that the gain from this exponential mapping is rather limited (less than 0.15dB on most images), thus a hardware implementation that aims to minimize the total logic gates can skip this step. On the other hand, an implementation aiming at the maximum possible quality can perform the required exponentiation using the already available shader units of the hardware, at the expense of additional computation time.

An optimization method (brute force search or hill climbing) is then used to determine both the optimal amount of outliers to be clamped and the optimal color space for the wavelet coefficients. In practice, we have found that for most images the optimization process is converging around the same point in the search space, clamping 26% of the outliers and encoding the wavelet subbands with a gamma value of 0.86. This consistency in the results can be explained by the fact that the wavelet coefficients of most images have roughly the same statistical properties.

3.3. Decoding

Decoding is straightforward and extremely efficient. With a single texture fetch, the transform coefficients are fetched and de-quantized via the dedicated texture hardware. Then, the coefficients are scaled back to their original range and the inverse Haar transform is computed, either using a few lines of pixel shader code or in dedicated hardware.

The transform in Equation 1 is the inverse of itself. First we calculate the original Haar coefficients by inverting Equation 2. Furthermore, we make the observation that the decoded texel C is always given by the weighted sum of the transform coefficients (Eq.1), where the weights are either 1 or -1 depending on the position of the texel in the block. Therefore, we construct the following equation

$$C(u,v) = LL - f((u\&1) \wedge (v\&1))HH - f(v\&1)LH - f(u\&1)HL \quad (3)$$

Original - 8bpp 1 level - 2 bpp 2 levels - 1.5 bpp

Figure 5: *Wavelet compression using one and two levels of decomposition. For both quality and performance reasons, only one level is used in our format.*

bpp	Y	CoCg	CoCg samples
5	dxt5/a	wlt	2:1
3	wlt	wlt	2:1
2.25	wlt	wlt	4:1
2	wlt	-	-

Table 2: *The bitrates of our method for grayscale anc color textures. (wlt denotes our 2bbp wavelet based encoding).*

where $f(x) = 2x - 1$ for $x = 0, 1$. The coefficient factors in this equation and the function f are carefully chosen in order to give the correct weights depending on the coordinates of the decoded texel. Equation 3 can be used to randomly access any texel in the encoded texture without using any conditional expressions. It's worth noting that *LL* corresponds to the next mip-map level, thus a single fetch can actually decode two successive texture mip-maps, something very useful when implementing trilinear filtering.

3.4. Multi-level Decomposition

The scale coefficients are essentially a scaled-down version of the original texture, therefore the same encoding process can be applied recursively to them, up to a level, in order to get better compression rates at the expense of quality. The coefficients of the last level should be stored as a DXT5 or BC7 texture, as described before, while previous levels are stored in the DXT1 format.

A two level decomposition yields a bitrate of just 1.5 bpp for a grayscale image. In the general case, the bit-rate B_N after N levels of decomposition is $B_N = 1 + B_{N-1}/4$, with $B_1 = 2$. This gives a maximum asymptotic compression ratio of $6:1$, or 1.33bpp. In other words, at the limit all the wavelet coefficients will be encoded using the DXT1 format.

Even though a multilevel decomposition is impractical from a performance or bandwidth point of view, since the data would be scattered in memory, it's very interesting and educational to examine the limits of such a method. When a single decomposition level is used, 25% of the total transform coefficients are scale coefficients, encoded at higher precision. For an additional decomposition level, this percentage drops to 6.25%, resulting in a noticeable loss of quality, as shown in Figure 5. Therefore, in practice we only use a single level of decomposition in our format.

3.5. Color Textures

In our method color textures are encoded using chrominance subsampling. The human visual system is much more sensitive to variations of brightness than color. Using this knowledge, encoders can be designed to compress color images more efficiently, by encoding chrominance data at lower accuracy. This has been also exploited in JPEG and many other analog and digital image coders.

The input texture is first decomposed to luminance and chrominance components using the RGB to YCoCg-R transform [MS03]. The two chrominance channels are downsampled by a factor of two or four and then encoded in separate textures using the method discussed in the previous section. Since the bitrate of the single channel encoding is 2bpp, this method will encode color textures at 3bpp and 2.25bpp respectively. A higher quality 5bpp encoding is also possible by encoding the luminance channel using the DXT5/A format. The exact combinations are shown in Table 2.

The YCoCg-R transform has very similar properties to those of YCoCg, a transform that was first introduced in H.264 compression and has been shown to have better coding performance than YCbCr or similar transforms [MS03]. However, the original YCoCg transform generates components that require two additional bits of precision to represent compared to the input RGB data, while YCoCg-R improves on this, requiring only one additional bit for the two CoCg channels. Therefore, when operating with 8-bit textures on the GPU, which is the focus of this paper, YCoCg-R preserves one more bit of chrominance precision compared to YCoCg, resulting in a 0.1dB gain over YCoCg, as shown in Table 3, and a rather significant gain over YCbCr. Furthermore, a hardware implementation of the YCoCg-R transform is rather simple and practical, since the decoder needs just four additions and two bit-wise shifts to convert the colors back to the RGB color space, as shown below:

$$t = Y - (C_g \gg 1);\ G = Cg + t;\ B = t - (C_o \gg 1);\ R = B + Co;$$

One interesting observation is that not all images make use of the expanded CoCg range. In fact, from the set of 24 images in the Kodak suite, only two images have a dynamic range higher than 255 for the Cg channel and only ten for the Co channel. It is easy to verify that these statistics hold true for a wider range of images. Therefore, in more than half of the cases YCoCg-R can operate entirely in 8-bits without any loss of precision. For the rest of the cases, one bit of chrominance precision will be lost, something that is generally not perceivable by the human visual system.

Wavalet	PSNR	Color Space	PSNR
Haar	33.29	YCoCg-R	33.86
PLHaar	32.04	YCoCg	33.76
S-Transform	33.04	YCbCr	33.42
PI-Haar	33.86		

Table 3: *Comparison of color spaces and wavelet transforms for the Lena image. The combination of PI-Haar and YCoCg-R gives the best coding performance.*

	Color			Gray
	5bpp	3bpp	2.25bpp	2bpp
kod01	38.7	30.7 / +1.0	30.4 / +2.3	31.1 / +2.9
kod02	39.2	35.5 / +1.0	33.9 / +0.7	36.4 / +1.9
kod03	40.8	36.7 / +0.5	35.7 / +1.0	37.8 / +2.3
Kodak	39.3	33.4 / +0.6	32.6 / +1.4	34.0 / +2.2
Quake	38.7	33.0	32.1	33.6

Table 4: *Average PSNR of our method on the Kodak suite and on a set of 200 textures from the Quake 3 game. For the complete list of results, consult the suppl. material. (Error: PSNR / Gain over the scaled DXTC format).*

4. Results

4.1. Quality Evaluation

A direct comparison of our method with DXT1 and DXT5/A is not possible, because our method does not provide a 4bpp mode. After all, the purpose of this research is not to replace the existing block compression formats, but to increase their flexibility by offering additional bit-rates. Nevertheless, a reasonable comparison can be performed by scaling down the DXTC encoded textures by the appropriate factors in order to match the bitrates of our method. The scaled down textures are then decompressed and scaled back to their original size using the hardware bilinear filter, a process that is equivalent to rendering with lower resolution textures. This comparison is very important, since any sensible texture compression algorithm should perform better than just scaling down the textures. The downscaling is performed using the Lanczos filter with a footprint of 16 pixels. Compared to other resampling filters, this one provided sharper results and better image quality. The scaling factors used are 71% in both dimensions for the grayscale 2bpp format, 75% for the 2.25bpp RGB format and 86% for the 3bpp one.

Format	PSNR	Format	PSNR
PVRTC 2bpp	31.7	ASTC 1.28bpp	31.7
PVRTC 4bpp	36.1	ASTC 2.0bpp	33.9
DXT1 4bpp	36.6	ASTC 3.56bpp	38.0

Table 5: *Average PSNR of other low-bitrate methods on the Kodak suite.*

The reference DXT1 images are created using the Nvidia Texture Tools (v. 2.0) with default weights. The same encoder was used to quantize the wavelet coefficients in our method. The BC7 packing mode requires the specification of weights for each channel, something not supported by the original version of the encoder. Therefore, we have modified the source code and we provide it in the web page of this paper, in order to help researchers reproduce our results. It is worth noting that the coding performance of our method greatly depends on the quality of the DXTC/BC7 quantizer.

For the quality evaluation we have used the well-known Kodak Lossless True Color Image Suite, a standard benchmark for image coding algorithms. This allows quick and easy comparisons with other present and future relevant techniques. Table 4 shows the PSNR of our compression scheme at various bit-rates, truncated to the first decimal point. All the measurements are performed on the original full resolution images. Grayscale modes use the luminance of the image (as defined in the YCoCg space).

The results show that for grayscale textures, our 2bpp encoding is up to 3.9dB (2.2db on average) better than DXT5/A when scaled at the same bitrate. For color images, the 5bpp mode outperforms DXT1 by up to 4dB (2.9dB average), although the advantage is lower on images with strong chrominance transitions, due to the chrominance subsampling in our scheme. The 3bpp and 2.25bpp modes perform better than DXT1 when the last is scaled at the same bitrate, confirming that using our method is preferable over using lower resolution textures. Table 5 shows the average PSNR of other low-bitrate encoding formats on the Kodak dataset. Our 2.25bpp mode outperforms the 2bpp PVRTC format by 0.9dB, indicating that the additional modes we have added to DXT1 are very competitive with other low-bitrate formats in the industry. For completeness, we also present PSNR measurements for the upcoming ASTC format. Please note that ASTC is still under development and the measurements shown here are from a brief sketch [NLO11] from the authors of the technique. The ASTC format provides slightly better PSNR, but unlike our method, it requires a completely new hardware implementation, thus it cannot be used on existing GPUs. On the other hand, as shown in Sec. 4.4, our method can be efficiently implemented using the programmable shaders of a commodity GPU.

Interestingly, ASTC is the only other texture compression format that provides a similar level of flexibility to our method. Unlike our method, which achieves this flexibility using a combination of transform coding concepts and chroma subsampling, ASTC encodes each block using a new low-level coding scheme that allows bits to be allocated among different types of information in a very flexible way, without sacrificing coding efficiency. The two methods are orthogonal and nothing prevents this new coding scheme to be combined with our technique. Therefore, in the future it would be very interesting to investigate whether our ap-

Original - 24 bpp DXT1 - 4 bpp Ours - 4 bpp

Figure 6: *Close-up to the lighting calculated from a normalmap (inset). Our method eliminates the banding artifacts that are visible with the DXT1 format.*

proach can be used to incorporate some transform coding concepts or other perceptual optimizations, like chroma subsampling, in the ASTC format.

In order to test whether the results from the Kodak set generalize to typical textures, we have also measured the PSNR when compressing a set of 200 textures from the Quake 3 game. An older game was used, since the textures on newer games are already DXT compressed. The average error on this dataset is similar to the one in the Kodak suite.

Figure 7 demonstrates the visual quality on kodim19 and kodim06 from the Kodak set and a more typical texture. When observed without any magnification, all the formats provide an image quality very close to the original uncompressed data, without any visible artifacts. Nevertheless, it's important to study the quality under extreme magnification, since textures can be magnified when used in 3D graphics applications. The results are generally artifact free, especially for the 5bpp mode, while the 2.25bpp mode exhibits some occasional chrominance distortion, due to the aggressive chrominance subsampling. The interested reader is highly encouraged to examine the high resolution images and the additional examples in the supplemental material.

To better demonstrate the properties of our format, we have encoded a normalmap, where the lighting is computed based on the normals fetched from this texture, instead of the geometric normals, as shown in Figure 6. The DXT1 format has difficulty encoding smoothly varying data and the resulting lighting exhibits several banding artifacts, because in this format each 4×4 block can only encode 4 possible values. On the other hand, when our method is used (encoding the XY coordinates in the texture and reconstructing Z in the shader), the banding is eliminated, since in this case the final texel color is given by a combination of 4 coefficients, thus the total possible values are greatly increased.

A disadvantage of the PSNR metric is that the human perception is not taken into account. For this reason in Figure 7 we also report the perceptual difference of the compressed images from the original ones, using the perceptual metric by Yee et al. [YPG01]. The reported number is the fraction of the perceptually different pixels over the total number of pixels (0 being the best), measured with the pdiff utility using the default settings. Many aspects of our compression scheme are designed over human perception, such as the chrominance subsampling and the omission of the high frequency details in the wavelet coefficients. For this reason, we observe that this metric favors our codec over DXT1, which does not incorporate such design considerations. In a sense, our method can be seen as a way to incorporate some perceptual optimizations in the DXT1/BC7 encoding and similar block compression formats. Nevertheless, since there is not a commonly agreed perceptual metric, we still base our analysis on the PSNR results.

YCoCg-R	Chroma	PI-Haar	Drop HH	BC7
48.0	41.9(2:1)	41.1	39.3	36.3
48.0	38.4(4:1)	38.0	36.6	34.4
-	-	47.9	42.3	38.0

Table 6: *Breakdown of the error (PSNR) after each successive stage when encoding kodim23 from the kodak suite. 1st row: 2:1 chroma sampling, 2nd row: 4:1 chroma sampling, 3nd row: grayscale encoding.*

4.2. Error Breakdown

Our method uses several individual steps to compress an image, so it's very interesting and educational to measure the amount of error introduced on each step, as shown in Table 6. The error from the YCoCg-R transform is due to the chrominance range expansion, as discussed in Sec. 3.5. We also observe that on images with very strong chrominance components, such as kodim23, the error from the chrominance subsampling step is high.

Interestingly, a significant amount of error is introduced when the wavelet coefficients are converted to 8-bit integers, in order to be stored as textures. Just by performing the wavelet transform on the grayscale version of kodim23 and storing the results we get 47.9db of PSNR, without any DXTC quantization. This is because the Haar transform, and its derivatives in Sec. 3.1, expand the input domain, thus additional bits of precision are required to store the resulting coefficients without any loss. To address this issue, we have also experimented with two integer-to-integer wavelet transforms, PLHaar [SLDJ05] and S-transform [CDSY97]. PLHaar provides a lossless Haar-like transform which operates only with 8-bits of fixed-point precision, while S-transform requires 9-bits. Our experiments (Table 3) demonstrate that when using these transforms the total coding performance degrades, because the new coefficients have a higher DXT quantization error, even when applying the optimization techniques discussed previously in this paper.

It is worth noting that our method is more efficient when compressing grayscale images, since in this case the error introduced by the YCoCg transform and the chrominance subsampling is completely avoided. Nevertheless, our method provides competitive performance for color images too.

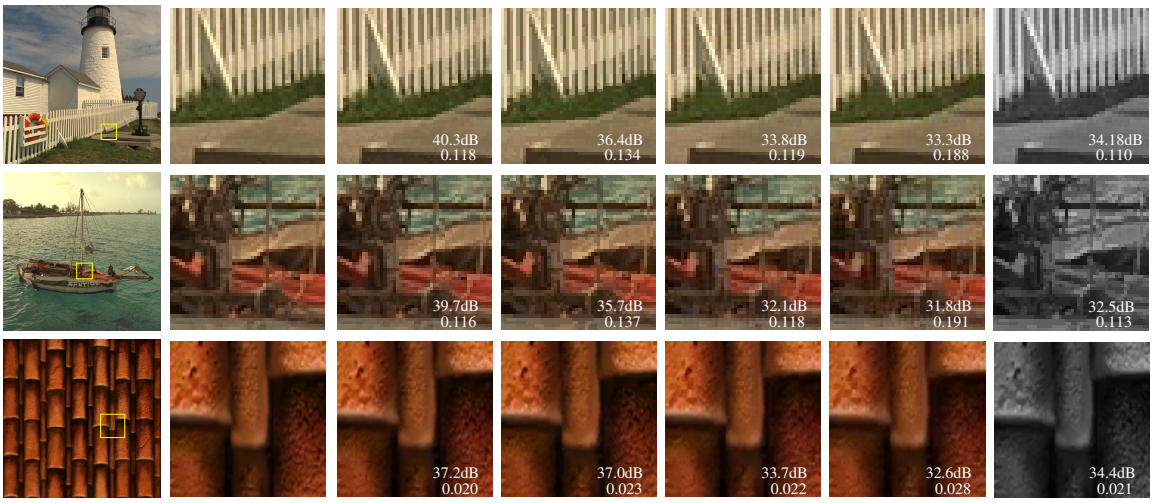


Figure 7: *Compression of kodim19 and kodim06 from the Kodak set and a typical game texture at various bitrates. The BC7 mode is used in our method to encode the transform coefficients.*

Cache Size: 0.5 Kbytes

			1024x1024		512x512	
			C	M	C	M
Sponza	DXT1	N	91.1%	5972k	66.4%	5637k
		T	96.2%	2550k	85.3%	2466k
	Ours	N	99.2%	2818k	97.0%	3019k
		T	99.6%	1207k	98.5%	1509k
Arena	DXT1	N	95.8%	2818k	84.2%	2650k
		T	98.1%	1275k	95.3%	788k
	Ours	N	99.6%	1610k	98.3%	1711k
		T	99.8%	805k	99.5%	503k
			59.3%		60.7%	

N:Nearest T:Trilinear C:Cache Hits M:Bandwidth(bits/frame)

Figure 8: *Bandwidth analysis of our compression scheme using a software rendering simulation. The last row indicates the average amount of bandwidth compared to DXT1 for each resolution.*

4.3. Bandwidth Analysis

The biggest advantage of hardware texture compression is perhaps not the reduced texture storage, but the reduction in the consumed memory bandwidth.To accurately study the impact of our method on the total memory bandwidth, we have integrated our compression scheme on a software rendering system. The rasterization order is from left to right in 32×32 tiles and inside each tile we follow the scanline order. A FIFO cache eviction strategy is used and the blocks in the compressed textures are arranged in linear order. Furthermore, a z-prepass was used during rendering, as in many modern game engines, in order to eliminate overdraw. This is not an optimal architecture to reduce cache misses, but it provides a common ground to compare the algorithms.

The results in Fig. 8 report the cache behavior and the consumed bandwidth when rendering two typical 3D scenes. A cache miss in our format requires at the worst case three 128bit blocks to be fetched from memory, while a cache miss in DXT1 costs only one 64bit block. On the other hand, the size of the textures in our method is reduced due to the wavelet packing and the chrominance subsampling, thus the cache behavior is better than DXT1. Please note that the wavelet coefficients are stored and accessed like regular textures, without any complex access patterns, thus ensuring efficient use of the existing texture cache.

A very interesting fact, as shown in the results, is that the efficiency of the texture cache is reduced when rendering at lower screen resolutions, because the average size of a triangle in pixels is also reduced. Even when rendering a small triangle, we have to load an entire block. Since our method increases the block size, it could have a negative impact on how much bandwidth is wasted. To a small extent, this is reflected in the results, where the bandwidth gain is slightly smaller in the lower resolution.

4.4. Performance Evaluation on Existing GPUs

Our method can be implemented in both software, using the programmable shaders, or in dedicated hardware. Both implementations are rather minimal, since the already available decompression hardware is used to perform the decoding of the transform coefficients and some additional ALU instructions are required to implement Equation 3.

The shader implementation has an additional overhead associated with texture filtering, which should be computed after the textures are decompressed by the shader, thus preventing the use of native texture filtering. Direct filtering of the compressed textures is incorrect and will produce

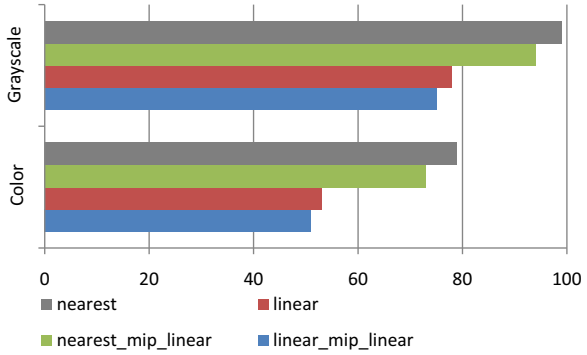


Figure 9: *The performance of the shader implementation of our method as a percentage of the native hardware texture fillrate. (Filter names use the OpenGL notation)*

excessive blurring and ghosting, because with our packing each texel encodes a 2×2 block. Therefore, bilinear filtering is computed in the shader, using four fetches per channel. With the same fetches we can also compute trilinear filtering, as noted in Section 3.3. A rather good performance/quality tradeoff is to implement "nearest_mipmap_linear" filtering (in OpenGL notation), which performs linear interpolation between the two texture mip-maps, but each mip-map is filtered with nearest filtering, avoiding the extra fetches. On the supplemental material of this paper we also discuss a very efficient method to perform anisotropic filtering with the help of the dedicated hardware. A more detailed discussion on texture filtering is beyond the scope of this paper. Figure 9 shows the performance of the shader implementation when rendering a tunnel scene as a percentage of the native hardware texturing on a Nvidia GTX460, using our proof-of-concept implementation in OpenGL and GLSL.

5. Discussion

A very important observation is that, depending on the nature of an application, some textures are never magnified significantly (or at all). For example this is true for the textures on a 2D or 3D side scrolling game, or the distant billboards of a racing game. Taking advantage of this knowledge, application developers can allocate less memory for these textures, by using one of our low bit-rate formats. Potentially, this can be also true for many grayscale textures that are never observed directly, like blending masks, alpha maps, gloss maps etc. The memory saved by encoding these textures at lower bitrates can be used to improve other aspects of an application, like increase the total number of textures and models. This fact highlights the importance of the flexibility in the texture compression formats.

6. Conclusions and Future Work

In this paper we have presented a new flexible compression scheme that combines transform coding concepts with standard block compression methods. A variation of the Haar transform is used to decompose the input texture into coefficients, which are then quantized and stored using the DXT5 or BC7 formats. To achieve better quality, coefficients with higher contribution to the final image are stored with more precision. Color images are compressed with chroma subsampling in the YCoCg-R color space. The resulting method improves the flexibility of the currently available DXTC formats, providing a variety of additional low bitrate encoding modes for the compression of grayscale and color textures.

The success of our method at combining DXTC/BC7 with transform coding methods potentially opens a new venue of research looking to further improve the results. This paper provides some first insights towards this direction and perhaps more improvements could be made in the future with further research, involving either a better approach to decompose the textures into coefficients and/or alternative ways to pack and quantize these coefficients.

Acknowledgements

We would like to thank all the anonymous reviewers and industry experts for their helpfull comments. Their insights helped us to improve our method. The first author would also like to thank Nikos Karampatziakis (Cornell University) for his insights on statistical analysis.

References

[BAC96] BEERS A. C., AGRAWALA M., CHADDHA N.: Rendering from compressed textures. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 373–378. 1, 2

[Bou08] BOULTON M.: Using wavelets with current and future hardware. In *ACM SIGGRAPH 2008 classes* (New York, NY, USA, 2008), SIGGRAPH '08, ACM. 2

[BPT09] BPTC: *ARB_texture_compression_bptc specification*. Available online: http://www.opengl.org/registry/specs/ARB/texture_compression_bptc.txt, 2009. 2, 5

[CDF*86] CAMPBELL G., DEFANTI T. A., FREDERIKSEN J., JOYCE S. A., LESKE L. A.: Two bit/pixel full color encoding. *SIGGRAPH Comput. Graph. 20* (August 1986), 215–223. 2

[CDSY97] CALDERBANK A., DAUBECHIES I., SWELDENS W., YEO B.-L.: Lossless image compression using integer to integer wavelet transforms. In *Proceedings of the 1997 International Conference on Image Processing (ICIP '97) 3-Volume Set-Volume 1 - Volume 1* (Washington, DC, USA, 1997), ICIP '97, IEEE Computer Society, pp. 596–. 8

[DCH05] DIVERDI S., CANDUSSI N., HLLERER T.: *Real-time rendering with wavelet-compressed multi-dimensional textures on the GPU*. Tech. rep., University of California, Santa Barbara, 2005. 2

[DM79] DELP E., MITCHELL O.: Image compression using block truncation coding. *IEEE Transactions on Communications 27* (1979), 1335–1342. 2

[Fen03] FENNEY S.: Texture compression using low-frequency signal modulation. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (Aire-la-Ville, Switzerland, 2003), HWWS '03, pp. 84–91. 2

[INH99] IOURCHA K., NAYAK K., HONG Z.: *System and Method for Fixed-Rate Block-Based Image Compression with Inferred Pixel Values*. United States Patent 5,956,431, 1999. 2

[Kap10] KAPLANYAN A.: Cryengine 3: reaching the speed of light. Advances in real-time rendering, ACM SIGGRAPH 2010 Course Notes, August 2010. 3

[MS03] MALVAR H., SULLIVAN G.: *YCoCg-R: A Color Space with RGB Reversibility and Low Dynamic Range*. Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, Document No. JVTI014r3, July 2003, 2003. 6

[NLO11] NYSTAD J., LASSEN A., OLSON T. J.: Flexible texture compression using bounded integer sequence encoding. In *SIGGRAPH Asia 2011 Sketches* (New York, NY, USA, 2011), SA '11, ACM, pp. 32:1–32:2. 2, 7

[OBGB11] OLANO M., BAKER D., GRIFFIN W., BARCZAK J.: Variable bit rate gpu texture decompression. *Computer Graphics Forum 30*, 4 (2011), 1299–1308. 2

[SAM05] STRÖM J., AKENINE-MÖLLER T.: iPACKMAN: high-quality, low-complexity texture compression for mobile phones. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (New York, NY, USA, 2005), HWWS '05, ACM. 2

[SDS95] STOLLNITZ E. J., DEROSE T. D., SALESIN D. H.: Wavelets for computer graphics: A primer, part 1. *IEEE Comput. Graph. Appl. 15* (May 1995), 76–84. 3

[SLDJ05] SENECAL J. G., LINDSTROM P., DUCHAINEAU M. A., JOY K. I.: Investigating lossy image coding using the plhaar transform. In *Proceedings of the Data Compression Conference* (Washington, DC, USA, 2005), DCC '05, IEEE Computer Society, pp. 479–479. 8

[SP07] STRÖM J., PETTERSSON M.: ETC2: texture compression using invalid combinations. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware* (2007), Eurographics Association, pp. 49–54. 2

[vW06] VAN WAVEREN J.: *Real-Time Texture Streaming and Decompression*. Tech. rep., Id Software Technical Report, available at http://software.intel.com/file/17248/., 2006. 2

[vWC07] VAN WAVEREN J., CASTANO I.: *Real-Time YCoCg-DXT Compression*. Tech. rep., NVIDIA Corporation, 2007. 3

[Wal91] WALLACE G. K.: The JPEG still picture compression standard. *Commun. ACM 34* (April 1991), 30–44. 3

[YPG01] YEE H., PATTANAIK S., GREENBERG D. P.: Spatiotemporal sensitivity and visual attention for efficient rendering of dynamic environments. *ACM Trans. Graph. 20* (January 2001), 39–65. 8