Noname manuscript No.
(will be inserted by the editor)

Scalable Texture Compression using the Wavelet Transform

Bob Andries · Jan Lemeire · Adrian Munteanu

Draft version, published in the Visual Computer, pp. 1-19, 2016.

the date of receipt and acceptance should be inserted later

Abstract 2D texture data represents one of the main data sources in 3D graphics, requiring large amounts of memory and bandwidth. Texture compression is of critical importance in this context to cope with these bottlenecks. To improve upon the available supported texture compression systems, several transform-based solutions have been proposed. These solutions, however are not suitable for real-time texture sampling or provide insufficient image quality at medium to low rates. We propose a new scalable texture codec based on the 2D wavelet transform suitable for real-time rendering and filtering, using a new subband coding technique. The codec offers superior compression performance compared to the state-of-the-art, resolution scalability coupled with a wide variety of quality versus rate trade-offs as well as complexity scalability supported by the use of different wavelet filters.

Keywords Texture compression · Texture mapping · Wavelet transform · Quantization

B. Andries
Vrije Universiteit Brussel - ETRO
Pleinlaan 2
B-1050 Brussels
E-mail: bob.andries@etro.vub.ac.be

J. Lemeire
Vrije Universiteit Brussel - INDI
Pleinlaan 2
B-1050 Brussels
E-mail: jan.lemeire@etro.vub.ac.be

A. Munteanu
Vrije Universiteit Brussel - ETRO
Pleinlaan 2
B-1050 Brussels
E-mail: adrian.munteanu@etro.vub.ac.be

1 Introduction

The majority of source data in real-time 3D engines consists of 2D images called textures. We refer to the samples stored in these textures as *texels*, which have to be mapped on the geometry of the 3D environment. Executing this process at interactive frame-rates and high spatial resolutions requires high amounts of memory and bandwidth. These requirements can be lowered by compressing the texture data using specialized fixed-rate codecs. By using a fixed compression ratio, accessing a single texel in memory can be done in constant time and with little data dependencies.

The inclusion of S3TC [1] (also known as DXTC) in DirectX 6.0 was the beginning of widespread adoption of hardware-accelerated texture compression, resulting in a significantly lower memory and bandwidth footprint for texture mapping operations. DXTC, nowadays extended and better known as BC [2], is a collection of texture compression formats based on block truncation coding [3], still representing one of the most popular texture compression technologies today.

More recently, mobile hardware support for a potential successor of the DXTC texture formats became available: ASTC [4]. This format surpasses the existing hardware-supported formats in terms of compression performance and is widely applicable thanks to its wide range of available bitrates and variable number of encoded color channels.

The work presented in this paper will build upon existing texture coding techniques and leverage them to develop a state of the art wavelet-based texture codec. Due to the utilization of the wavelet decomposition as transform of choice, the codec features excellent resolution and complexity scalability.

2 Related work

2.1 Block truncation-based texture coding

A vector quantization based texture compression system was proposed by Beers et. al. [5] in 1996 featuring straightforward decompression but complicating the implementation of an efficient cache through its large global lookup table. This issue is not present in techniques using block truncation coding.

The general idea of block truncation-based texture compression is to divide the texture in relatively small (e.g. 4x4) blocks and to characterize each of these blocks by using derived data, such as the mean and the variance of the coefficients within each block. The subsequent coding step then uses these figures for each block to perform the actual quantization.

LATC [6](also known as DXT5/A) performs this process on single-channel data, using 4x4 blocks. This codec characterizes each block using a minimum and a maximum, which are both stored as 8 bit normalized values. Each block is then quantized using a 3 bit uniform quantizer, of which the endpoints are defined by these two 8 bit values. Each 4x4 block is thus defined by sixteen 3 bit indices and two 8 bit values, resulting in an average rate of 4 bits per pixel (bpp).

DXT1, a codec targeting RGB data, performs a similar procedure: it is also using 4x4 blocks, but the quantization is done by using a 2 bit uniform quantizer of the 3D RGB space. The quantizer endpoints are stored as normalized RGB565 values, requiring 32 bits of memory per block for endpoint storage. Finding the optimal quantizer endpoints is more complicated in this case, as we should make sure the 16 RGB values of each block lie as close as possible to the 4 uniformly interpolated color values. Hence, this procedure only performs well on correlated RGB data.

More advanced texture codecs such as BC7 [2] are still based on the same principles, but feature additional per-block modes, altering the way the data is quantized and stored.

2.2 Transform-based texture coding

Several publications propose transform-based texture compression formats, either supported by newly designed hardware or by a programmable decoding process performed by the pixel shader on the GPU. One of the earliest approaches for transform-based texture compression was made by Perebrin et. al. in 1999, proposing a format [7] based on the 2-level Haar wavelet transform and a hardware decoding system. Fenney proposed a texture compression format [8] targeting low power applications, which has received hardware support in the PowerVR chips, primarily used in Apple smartphones and tablets. It has not been implemented in any major AMD or NVIDIA GPU though. In 2005, DiVerdi proposed a system [9] featuring a pixel shader which could decompress multi-dimensional textures using the Haar transform. An additional codec requiring dedicated hardware was proposed by Sun et. al. [10], which was also based on the Haar transform. Longer wavelet filters were combined with a tree based coding approach by Grund et. al. [11]. However, tree based coding introduces significant and irregular indirection, limiting real-time performance and complicating resolution scalability by streaming selected subbands to the GPU. Aiming to further increase the fillrate in order to provide both filtered texture samples and a real-time frame rate, Hollemeersch et al. proposed a shader-based system [12] which uses the DCT transform and LATC texture compression. This system outperformed DXT1 and other previous works in terms of compression performance. At the same time, Mavridis et. al. presented a Haar-based texture compression approach [13], also outperforming previously proposed texture compression systems. These last two compression techniques are considered to be the state-of-the-art in transform-based texture coding and are used as benchmark techniques in our work.

Other works [14], [15], [16], [17] have been focusing on utilizing the GPU as a device to accelerate the wavelet transform for general purposes usage. These approaches cannot be used in real-time rendering as they are unable to efficiently perform per pixel reconstruction, which is a critical component of texture sampling. Similarly, the system proposed by Treib et. al. [18] is unable to perform in-situ texture sampling, requiring additional GPU memory to store the decoded data before it can be sampled.

The work presented in this paper explores and enables efficient application of wavelet transforms for texture encoding and real-time texture decoding, sampling and filtering. We design, implement and test a shader-based wavelet reconstruction system, targeting consumer GPUs. This lies in contrast to some of the previous designs, which require specifically-designed reconstruction hardware [7], [8], [10].

The proposed system is able to utilize wavelet filters of lengths beyond Haar, surpassing the work done by DiVerdi et. al. [9] and Mavridis et. al. [13]. The proposed system reconstructs pixels independently, making it suitable for real-time texture sampling. Compression performance is greatly elevated by using a new texture quantization scheme optimized for subband data. By combining the existing and our newly proposed subband coding techniques, we can offer a wide range of compression ratios and achieve superior compression performance compared to the state of the art.

The remainder of the paper is structured as follows: in the next sections we will elaborate on the proposed encoding architecture. Section 6 details the encoding of subband data, including our new technique. In section 7 we overview our rate allocation system. Per pixel wavelet reconstruction is explained is section 8. Sections 9 and 10 detail the design and implementation aspects of the shader-based reconstruction. In sections 11, 12 and 13 we present the experimental results, reporting on the compression performance, run-time complexity and comparisons against the state-of-the-art respectively. The conclusions of our work are drawn in section 14.

3 Codec architecture

Similar to some of the previous works [12], [13], the architecture of the proposed fixed-rate texture coder consists of four stages. First, when dealing with color textures, the three color channels are decorrelated using the YCoCg-R transform [19], yielding a luminance channel and two chrominance channels. Second, these channels are decomposed into subbands using the wavelet transform. The resulting subband coefficients are then shifted and rescaled in the third step after which the subbands are finally quantized and packed as texture data according to the specified codec configuration, ready to be sent to the GPU.

The next sections introduce the three most important coding steps of the proposed system: the wavelet transform, subband scaling and subband coding. Additionally, we present the rate allocation system which is used to determine the optimal coding configurations for a given image.

4 Wavelet transform

For one-dimensional (1D) signals, the wavelet transform [20] decomposes an input signal A_0 into two sets of coefficients, i.e. a low-pass subband A_{-1} and a high-pass subband D_{-1}, as illustrated in figure 1. This decomposition is performed by convolving the input signal with respectively the low-pass filter $\tilde{H}$ and the high-pass filter $\tilde{G}$, followed by downsampling the convolved signals with a factor of two. A multiresolution representation can be achieved by repetitively applying the decomposition to the resulting approximation signal

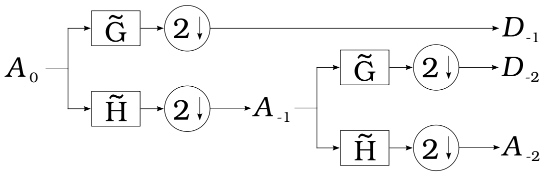

Fig. 1 A two-level wavelet decomposition of a discrete signal A_0 performed using a filter bank.

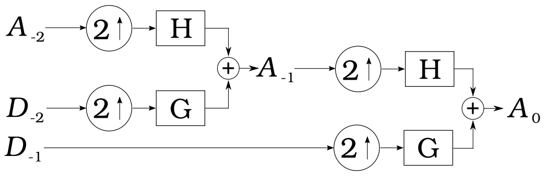

Fig. 2 The reconstruction of A_0, starting from the decomposed signals A_{-2}, D_{-2} and D_{-1}.

A_j, resulting in a single final approximation signal A_{-J} and a set of detail signals D_j, where $0 > j \geq -J$ and J denotes the number of decomposition levels. An example with two decomposition levels is shown in figure 1.

The approximation signal and the detail signal are used to reconstruct the higher-resolution approximation signal using the filters H and G, as shown in figure 2. Specifically, the wavelet reconstruction step computes A_{j+1} based on the lower-resolution approximation and detail signals A_j and D_j respectively as follows:

$$A_{j+1}(i) = \sum_m H(i-2m)A_j(m) + \sum_m G(i-2m)D_j(m) \tag{1}$$

The wavelet transform can be applied to 2D image data by performing the transform on each dimension separately. E.g. each row first is transformed using the 1D transform, followed by column-wise filtering of the resulting signals - for details the reader is referred to [20]. The 2D subbands will be referred to using the LL, HL, LH and HH notations, where HL subband contains the coefficients resulting from the horizontal high-pass filter and the vertical low-pass filter.

This work focuses on the Haar wavelet and the biorthogonal (2,2) wavelet. More information about the biorthogonal wavelet can be found in the work by Cohen [21] and the book by Mallat [22] (section 7.4.3). However, the proposed system is not limited to these two and can be used with other wavelets.

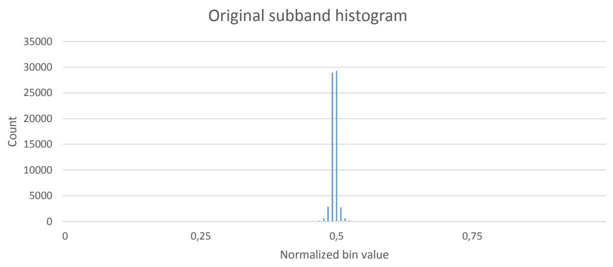

Fig. 3 Histogram of the biorthogonal (2,2) LH subband of the Lena image. For this visualization, the floating point data was quantized using 128 bins.

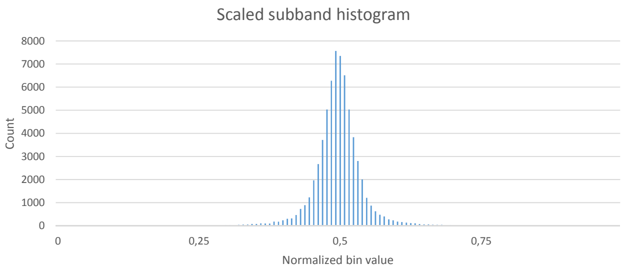

Fig. 4 Histogram of the scaled and clipped biorthogonal (2,2) LH subband of the Lena image. The applied scaling factor was 7.525. For this visualization, the rescaled and clipped floating point data was quantized using 128 bins.

5 Subband scaling

The coefficients of each wavelet subband are rescaled such that the dense center area of the histograms is stretched out while sparsely populated sides are clipped. This can result in a lower quantization error on the quantized subband coefficients compared to the case when no stretching is applied. To make sure the scaling process also performs well for low-pass data, the subband data is centered around zero before scaling and clipping is applied, using the average coefficient value of this particular subband.

This process is illustrated with an example subband in figures 3 and 4. Although the clipping process alters the high-magnitude wavelet coefficients, the overall quantization distortion will in general be reduced by scaling and clipping. This is shown next.

It is well know that the Laplace distribution is a simple model for the distribution of the wavelet coefficients in a wavelet subband - see e.g. [20]. The variance of the quantization error of a uniformly quantized error on a uniformly quantized and clipped Laplace distribution can be modeled as follows:

$$\begin{aligned} &E(K,\varDelta,\lambda) = \\ &2\cdot\int_0^{\frac{\varDelta}{2}} x^2\cdot\frac{\lambda}{2}e^{-\lambda\cdot x}dx \\ &+2\cdot\sum_{k=1}^{K-1}\int_{\frac{(2k-1)\cdot\varDelta}{2}}^{\frac{(2k+1)\cdot\varDelta}{2}}(x-k\cdot\varDelta)^2\cdot\frac{\lambda}{2}e^{-\lambda\cdot x}dx \\ &+2\cdot\int_{\frac{(2K-1)\cdot\varDelta}{2}}^{\infty}\left(x-\frac{(2K-1)\cdot\varDelta}{2}\right)^2\cdot\frac{\lambda}{2}e^{-\lambda\cdot x}dx \end{aligned} \tag{2}$$

where $K+1$ is the number of reconstruction points, the variance of the Laplace distribution is equal to $\frac{2}{\lambda^2}$ and $\varDelta$ is the distance between each of the reconstruction points. The first term integrates the error over the deadzone, the second term integrates the error on the bins to the left and the right of the deadzone and the last term integrates the error of the part of the curve that is beyond the outer reconstruction points (i.e. the overload distortion).

Performing the calculations in (2) leads to a closed form for the variance of the quantization error. This is given by:

$$\begin{aligned} &E(K,\varDelta,\lambda) = \\ &\frac{2}{\lambda^2}-\frac{\varDelta}{\lambda}\Big(1+\coth(\frac{\lambda\varDelta}{2})\Big)e^{\frac{-\lambda\varDelta}{2}} \\ &-\Big(\frac{\varDelta^2}{4}-\frac{\varDelta}{\lambda}\coth(\frac{\lambda\varDelta}{2})\Big)e^{-\lambda K\varDelta+\frac{\lambda\varDelta}{2}} \end{aligned} \tag{3}$$

One of the important steps in our coding approach is to to scale the subband data prior to quantization and clipping. It can be shown that, given K and $\varDelta$, i.e. for a given number of quantization cells and quantizer cell-size, one can find an optimal subband scaling factor C for which the variance of the quantization error is minimal.

Scaling the subband data with a factor C is equivalent to replacing λ with $\frac{\lambda}{C}$ in 3. By quantizing the scaled data in the same way as in 2 and then applying the inverse scaling operation, we get the following expression for the variance of the quantization error:

$$\tilde{E}(K,\varDelta,\lambda,C) = \frac{E(K,\varDelta,\frac{\lambda}{C})}{C^2} \tag{4}$$

By minimizing equation 4 with respect to C for given parameters $\lambda, K, \varDelta$, the optimal scaling factor C can be found. An example plot of equation 4 is given in figure 5.

6 Subband coding

Before introducing our new subband quantization modes, we will first investigate the performance of both

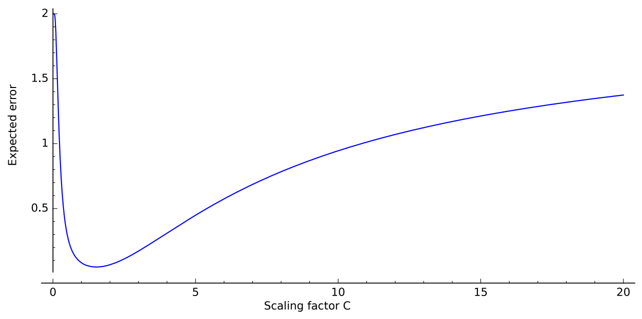

Fig. 5 A plot of equation 4, where $\lambda = \sqrt{2}$, $K = 8$ and $\Delta = 1$

single- and multi-channel block truncation based modes when employed to encode wavelet subbands.

6.1 Single channel versus multi-channel coding

Subbands of a single decomposition level have relatively little inter-band correlation, as pointed out by Mavridis et. al. [13]. Hence, the different subbands of a single color channel are difficult to be compressed using multi-channel texture formats, as such codecs expect correlation between the channels.

We demonstrate this with an experiment of which the results are shown in table 1. In this experiment we compare two advanced multi-channel codecs, BC7 and ASTC [4], with two simple single-channel reference encoding techniques. For each encoding technique, only the low-pass subband and the LH and HL subbands were encoded, while the HH subband was discarded, similar to the way Mavridis et. al. encode the subbands. The subbands were generated using a single-level Haar transform of the luminance channel of each image. The first reference technique uses essentially a single-channel version of DXT1. This was created by using only the green (second) channel of the DXT1 encoder. In other words, the DXT1 encoder was forced to ignore the data of the two other channels, effectively turning it into a low bitrate single-channel codec. The encoded data consists of two times 6 bpp for the quantizer endpoints and another 32 bpp for the 16 quantization indices, resulting in a bitrate of 2.75 bpp per channel (corresponding to 8.25 bpp reported in Table 1).

The second single-channel reference technique has a more finetuned distribution of bitrates over the three channels: the low-pass subband is encoded using LATC while the HL and LH subbands are encoded with the 2 bpp single-channel version of ASTC.

Table 1 shows that on decorrelated channels, even a simple, non optimized single-channel version of DXT1 can perform equally well or even better than an advanced multi-channel codec. This inspired us to develop relatively simple but still specialized single quantization modes, designed to perform well on wavelet subband data.

Table 1 Single channel vs multi channel performance, PSNR of the reconstructed luminance channel compared to the original channel.

Image	Rate (bpp)	Encoding	PSNR (dB)
kodim01	8	BC7 (multi)	30.4
	8	ASTC (multi)	30.8
	8.25	DXT1 (single)	31.6
	8	LATC ASTC (single)	31.8
kodim02	8	BC7 (multi)	36.7
	8	ASTC (multi)	37.0
	8.25	DXT1 (single)	37.5
	8	LATC ASTC (single)	38.0
kodim03	8	BC7 (multi)	36.7
	8	ASTC (multi)	37.0
	8.25	DXT1 (single)	37.9
	8	LATC ASTC (single)	38.5

We notice that by independently coding the subbands with single-channel quantization modes, resolution scalability can be easily implemented. Coupled with existing texture coding techniques, this creates a solid basis to perform subband quantization and represents the core of our wavelet-based texture codec.

6.2 Fast local subband coding

For reasons explained in the previous section, we chose to focus on a single-channel quantization system for our subband data. Inspired by LATC and our previous work [23], we propose a set of local uniform quantization modes optimized for subband coding. Similar to LATC [6], for each 4x4 texel block, a local uniform quantizer is defined by storing the quantizer endpoints using a lower and upper endpoint value. Sticking to uniform quantization is justified, as it allows for low complexity reconstruction and on average, the normalized 4x4 blocks feature approximately uniformly distributed data. This can be seen in the histogram of the per-block normalized coefficients of a high-pass subband in figure 6.

LATC spends 16 bits per 4x4 block defining the local quantizer endpoints in a uniform way; this is necessary for spatial-domain data, as in that case the global histogram approximates a uniform distribution. For our subband oriented technique, we are spending $2 \cdot M$ bits per block: our system stores the local quantizer

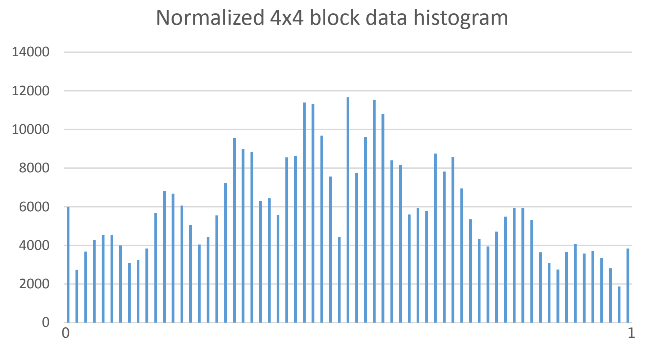

Fig. 6 Each local quantizer scales and clips the data by choosing both a discretized minimum and maximum value. By using this local minimum and maximum value in each block, the data of each block is rescaled to a range between 0 and 1. The resulting normalized floating point data of all blocks is visualized in this figure using a histogram featuring 128 bins.

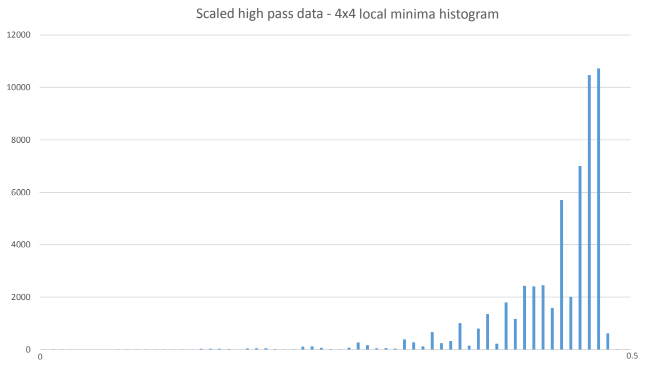

Fig. 7 The minima of the blocks are not uniformly distributed. This figure shows a 128-bin histogram of the minima of the 4x4 blocks of a high-pass subband containing coefficients normalized to a range between 0 and 1.

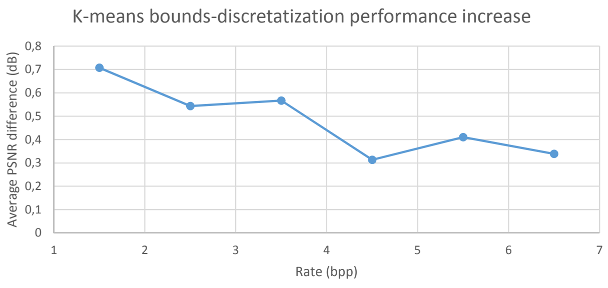

Fig. 8 A visualization of the averaged impact of the k-means discretization method for the per block quantizer endpoints of a high-pass subband. The level 2 luminance LH subbands of the first 10 Kodim images were used to evaluate the performance of both the uniform and the k-means discretization methods. The resulting PSNR numbers were averaged, after which for each of the available bitrates the average performance difference was calculated.

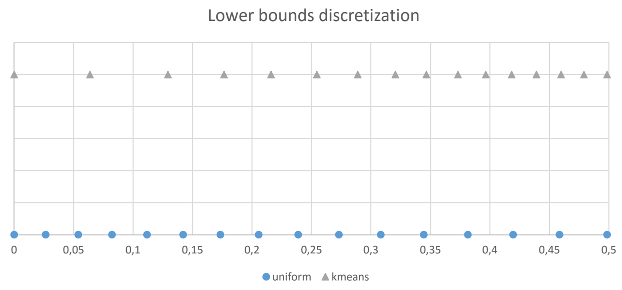

Fig. 9 A visualization of the different discretizations of the lower quantizer endpoints. The k-means discretization puts more emphasis on more prevalent lower bounds close to the center of the subband coefficients distribution.

endpoints using two M bit indices which point to values in a global lookup table. Each of the modes features an N bit uniform quantizer. For example, choosing $M = 4$ and $N \in [1..6]$ results in $2 \cdot M/16 = 0.5$ bpp for quantizer endpoint storage and N bpp for index storage.

The ideal local quantizer endpoints in each 4x4 block are dependent on each blocks' content. Hence, this would result in a very large set of optimal endpoints. However, the global lookup table can only contain a limited set of quantizer endpoints: 2^M for the lower endpoints and 2^M for the upper endpoints. Hence, the block bounds have to be discretized globally in an optimal way. Figure 7 shows the distribution of the lower bounds of 4x4 blocks of an arbitrary high-pass subband. The performance impact of different discretizations is shown in figure 8, where a sample subband was encoded at six different rates, using two different lookup tables for the quantizer endpoints. The actual values generated by the discretization are shown in figure 9. The final discretization used in the proposed system was achieved by performing k-means clustering on the observed block bounds.

We note that we have also investigated Lloyd-Max quantization of the wavelet subbands in the past [23]. However, it turns out that, due to its local adaptive, data-dependent nature, the proposed block-based quantization scheme yields better results than global Lloyd-Max quantization of the wavelet subbands. For this reason, the experimental section focuses only on the proposed adaptive quantization scheme.

7 Rate allocation

Essentially, for each subband, a number of coding modes is evaluated in terms of rate and distortion. The mode that yields the steepest distortion-rate slope is selected. In this way we build a global rate distortion function for the system. Note that no explicit downsampling of the chrominance channels is performed. Instead, the rate allocation system is free to perform implicit downsampling by allocating zero bits to certain high pass subbands, effectively erasing the high pass chrominance data.

In terms of coding modes, we evaluate the proposed adaptive quantization approach of section 6.2 but also assess established texture codecs (e.g. LATC) adapted to subband data. Details on the later are given in section 11.1.

This rate allocation process can be speeded up, if necessary, by utilizing heuristics: instead of building a global rate-distortion curve, one targets fixed texture coding rates, which speeds up rate allocation. E.g. it is of no use encoding the low-pass luminance subband with very low rate quantization methods while targeting a relatively high bitrate. Vice versa, when targeting low bitrates, one should not evaluate the quantization error of high rate quantizers on the sparsely populated high pass subbands. This was implemented using a constraint on the subband rates using a hierarchy. E.g. for a single level decomposition, the rate of the LL subband should be higher or equal to the rate of the LH or the HL subband, of which the rate should be higher than the rate of the HH subband.

8 Per pixel 2D wavelet reconstruction

As stated in the introduction, a performant GPU shader program is characterized by many independent lightweight threads, each outputting one element: a reconstructed and filtered spatial-domain pixel in our case.

We can generalize our problem as follows: each shader instance (thread) should independently reconstruct a set of spatial domain pixels, which are then filtered and output as a single value. This section focuses on one aspect of the proposed shader code: the wavelet reconstruction algorithm. The subsequent sections will focus on the algorithmic developments required to perform filtering and multi-level wavelet reconstruction.

We will give first the mathematical expressions required to reconstruct a single pixel located in a spatial-domain position $(2m + a, 2n + b)$, with m, n positive integers, and $a, b \in \{0, 1\}$. The values a and b indicate the parity of the spatial-domain position where the reconstruction is performed.

First, we introduce a couple of notations to ease the identification of the reconstruction dependencies of a single spatial-domain element.

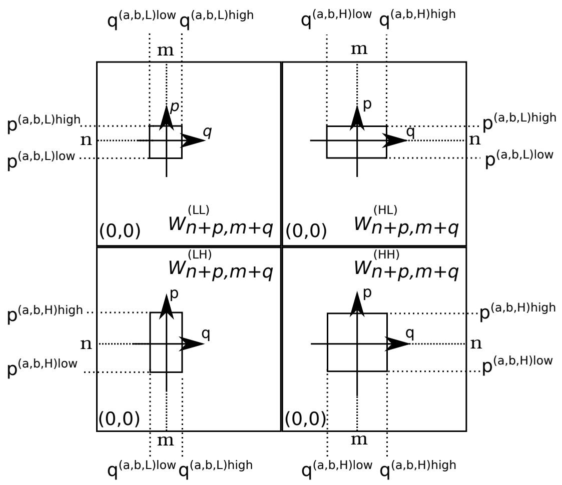

Fig. 10 2D wavelet reconstruction notations. The dependency domain in each subband is defined a set of offsets (p, q), depending on the parity (a, b) of the reconstruction element. Each of the four illustrations depicts on of the four available parity combinations, which imply different reconstruction weights.

The set of offsets relative to (m, n) in a single subband s associated with the reconstruction at spatial-domain positions $(2m + a, 2n + b)$ with parity (a, b) is given by:

$$O(a,b,s) = \Big\{(p,q) \;\Big|\; p \in \{p^{(a,b,s)\text{low}}, \dots, p^{(a,b,s)\text{high}}\}, \\ q \in \{q^{(a,b,s)\text{low}}, \dots, q^{(a,b,s)\text{high}}\}\Big\} \quad (5)$$

By adding these offsets to (m, n) we get the positions of the samples in subband s required to perform the reconstruction of the texture values at spatial-domain locations $(2m + a, 2n + b)$. We refer to figure 10, which clarifies the notations in equation 5. As shown in the figure, reconstructing a spatial-domain pixel at position $2m + a, 2n + b$ requires all the wavelet coefficients depicted by the four rectangles in the four subbands. For more details on the identification of these dependencies, we refer to our past work - see [24] (section 2.1.1), [25] (section II-A).

The set of total dependencies for reconstruction at a position with parity (a, b) is thus expressed by:

$$O(a,b) = \Big\{(s, O(a,b,s)) \;\Big|\; s \in S\Big\} \quad (6)$$

where S denotes the set of the four subbands s.

The set of subband samples required to reconstruct a spatial-domain element at position $(2m + a, 2n + b)$ is defined by:

$$dep(2m+a, 2n+b) = \{W^{(s)}_{m+p,n+q} | (s,p,q) \in O(a,b)\} \quad (7)$$

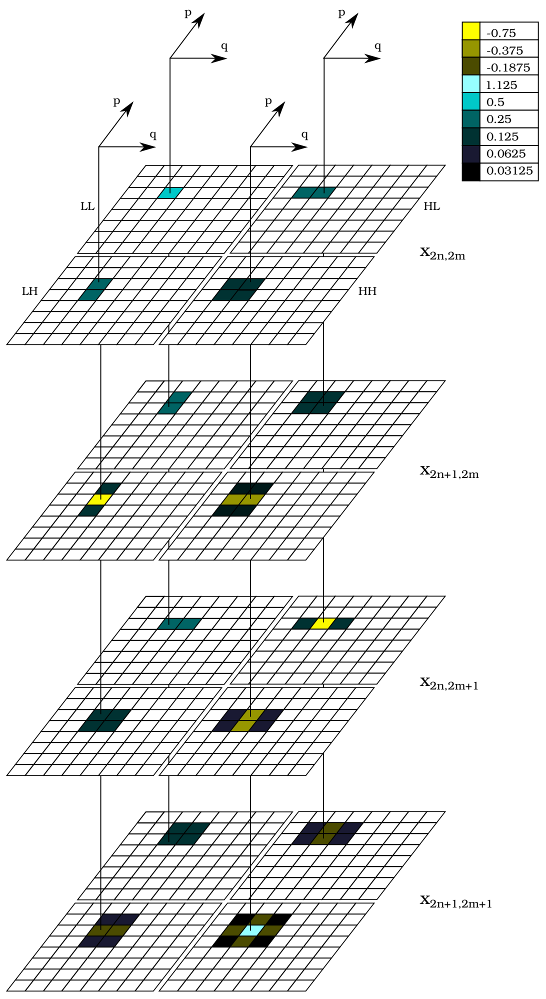

Fig. 11 2D wavelet reconstruction weights for the biorthogonal (2,2) [21] filter.

where $W^{(s)}_{m+p,n+q}$ represents a sample at position $(m + p, n + q)$ in subband s.

Using these notations, the 2D wavelet reconstruction of a single element can be written as [24], [25]:

$$x_{2m+a,2n+b} = \sum_{(s,p,q)\in O(a,b)} k^{(s)}_{p,q,a,b} W^{(s)}_{m+p,n+q} \tag{8}$$

where $k^{(s)}_{p,q,a,b}$ represents a 2D reconstruction coefficient derived from the 1D reconstruction filter coefficients. As an example, the 2D reconstruction coefficients $k^{(s)}_{p,q,a,b}$ for the biorthogonal (2,2) wavelet transform are depicted in figure 11. Equation 8 can be applied to other wavelet transforms as well.

Often, more than one spatial domain pixel will be required; e.g. whenever a filtered pixel is desired. For each of these pixels, equation 8 has to be evaluated. By examining the combined set of subband dependencies, redundant memory accesses can be avoided by buffering the necessary subband samples before performing the reconstruction calculations. The next section shows how a set of dependencies can be derived to enable the reconstruction of a single filtered spatial-domain pixel.

9 Shader design

The equations presented in the previous chapter allow us to reconstruct spatial-domain texture values depending on corresponding wavelet-domain coefficients. In practice, texture mapping is an operation which usually combines (1) fetching spatial-domain values and (2) filtering them to a single value which can be displayed as a pixel on the screen. This minimizes aliasing artifacts and provides a more smooth appearance for close-up views of objects.

This section will first detail generic texture filtering, after which we will investigate the relation between texture filtering and wavelet reconstruction. A solution using transform-domain filtering is proposed. Finally, details about border extension for GPU-based wavelet reconstruction are presented.

9.1 Filtering

Using the spatial-domain samples (texels) $\mathbf{x}$ and the filter weights $\mathbf{f}$, the generic filtering equation is given by:

$$x_{filtered} = \mathbf{f}^{\mathbf{T}} \cdot \mathbf{x} \tag{9}$$

When traditional texture sampling and filtering is performed on a spatial-domain texture, the filtering operation is performed efficiently by the GPU. The filter weights $\mathbf{f}$ are derived from a given floating point texture coordinate $\mathbf{t}$, where $0 < t_x < 1$ and $0 < t_y < 1$ and $\mathbf{t}$ corresponds to a normalized position on the texture. The GPU reads the required texels $\mathbf{x}$ from texture memory and blends them according to equation 9.

The set of spatial-domain positions covering the texels $\mathbf{x}$ required for a filtering operation can be written as:

$$P(\mathbf{t}, X_f) = \bigcup_{\mathbf{o}\in X_f} (p(\mathbf{t}) + \mathbf{o}) \tag{10}$$

where X_f is a set of filter-specific spatial-domain offsets and $p(\mathbf{t})$ is an integer position derived from texture coordinate $\mathbf{t}$:

$$p(\mathbf{t}) = \lfloor (N_x t_x - 0.5, N_y t_y - 0.5) \rfloor = (2m + a, 2n + b) \tag{11}$$

with $\mathbf{N} = (N_x, N_y)$ the size of the texture.

Equation 10 shows that spatial-domain filtering depends on as many spatial-domain pixels as there are offsets in X_f. In our case, the textures contain decomposed data and the spatial-domain pixels covered by X_f have to be reconstructed using equation 8. Hence, we cannot simply perform the spatial-domain filtering process using hardware-supported filtering, as equation 8 requires separate unfiltered subband samples. The filtering hardware would provide blended texels instead of separate texels, which are of no use to the reconstruction process.

Given the filter weights $w_{i,j}$, the interpolated pixel value is calculated as follows:

$$x_{filtered} = \sum_{(i,j)\in X_f} w_{i,j} \cdot x(p(\mathbf{t}) + (i,j)) \tag{12}$$

The combined reconstruction and filtering equation can always be written as:

$$x_{filtered} = \mathbf{f^T} \cdot \mathbf{M} \cdot \mathbf{c} \tag{13}$$

where $\mathbf{M} \cdot \mathbf{c}$ reconstructs the spatial-domain pixels specified by equation 10 and $\mathbf{f^T}$ is the column vector containing the filter weights as specified in equation 12. The elements of the matrix $\mathbf{M}$ are the 2D reconstruction coefficients $k^{(s)}_{p,q,a,b}$ while the vector $\mathbf{c}$ contains subband samples at positions $(m + p, n + q)$ as seen in equation 8.

9.2 Dealing with parity at run-time

As illustrated in Figure 11, the weights in equation 8 depend on the parity of (a, b). That is, equation 13 depends on the parity of (a, b). This is problematic, as this implies very heavy non localized branching at run time. This is caused by the way the shader is executed on the GPU: for each screen space pixel, a 'thread' is launched which ideally should execute the same instructions as its neighboring pixel-threads but with different data.

Avoiding this issue can be done by altering the outputs of the matrix $\mathbf{M}$ and the corresponding sampling positions. Instead of using outputs based on X_f and $p(\mathbf{t})$ - a set of positions of which the parity is unknown until run time - we use an altered version of equation 11:

$$p_{even}(\mathbf{t}) = 2\lfloor(\mathbf{t} \cdot \mathbf{N} - 0.5)/2\rfloor = (2m, 2n) \tag{14}$$

As the parity of $(2m, 2n)$ always equals $(0, 0)$, the offsets in X_f thereby define the parity of each of the reconstructed elements. Hence, each elements reconstruction equation is known at compile-time, avoiding costly indirection operations in the pixel shader. A direct consequence is that by applying the offsets of X_f relative to $(2m, 2n)$ instead of the unaltered reconstruction position $(2m + a, 2n + b)$ with parity (a, b), in 3 out of the 4 different parity cases we do not reconstruct the intended spatial-domain samples. Because of this, we have to use an enlarged window of output offsets X_{f+p}, as shown in the example in figure 12. This enlarged window is one pixel larger in each dimension to cover each of the parity cases. Selection and filtering of the correct spatial-domain samples of the enlarged window is then ensured by calculating and applying an extended set of filter weights $\mathbf{f}$, which can be constructed based on the equation 12.

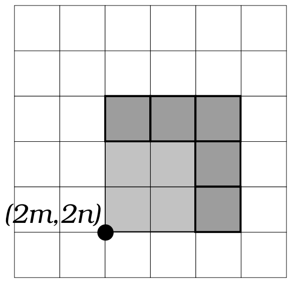

Fig. 12 Enlargement of the set of offsets X_f to X_{f+p} to cover all four parity cases of $2m + a, 2n + b$. In this case, the original X_f covers a 2 by 2 window to support bilinear filtering.

In our work we employ bilinear filtering, which is the most commonly chosen method in GPU texture mapping. Bilinear filtering takes 4 neighboring input samples and applies 2D linear interpolation to produce one filtered pixel value. Hence, X_f contains the 4 offsets $(0, 0)$, $(0, 1)$, $(1, 0)$ and $(1, 1)$, whereas in X_{f+p} this set is extended with 5 additional positions.

9.3 Transform-domain filtering

The preceding sections covered a generic way to reconstruct spatial-domain pixels, after which they are explicitly filtered by executing shader commands. However, the GPU hardware can perform texture sampling and bilinear filtering at the same time; more specifically, we can get a bilinear interpolation of four neighboring texture samples at the cost of one texture sampling operation.

A bilinear filtering operation is traditionally expressed as:

$$x_{filtered} = \frac{1}{n} \cdot \begin{bmatrix} w_{(0,0)} & w_{(0,1)} & w_{(1,0)} & w_{(1,1)} \end{bmatrix} \cdot \begin{bmatrix} x_{(0,0)} \\ x_{(0,1)} \\ x_{(1,0)} \\ x_{(1,1)} \end{bmatrix} \tag{15}$$

where the normalization factor is $n = w_{(0,0)} + w_{(0,1)} + w_{(1,0)} + w_{(1,1)}$ and the weights w are subject to the following constraints:

$$\frac{w_{(0,0)}}{w_{(0,1)}} = \frac{w_{(1,0)}}{w_{(1,1)}} \text{ and } \frac{w_{(0,0)}}{w_{(1,0)}} = \frac{w_{(0,1)}}{w_{(1,1)}} \tag{16}$$

We can use this hardware-supported filtering and sampling process to perform parts of the reconstruction shown in equation 13.

We introduce the following notation representing a subset of (13):

$$x^*_{filtered} = \mathbf{f^T} \cdot M^* \cdot \mathbf{c}^* \tag{17}$$

where M^* contains four specific columns of M and $\mathbf{c}^*$ contains the elements of $\mathbf{c}$ corresponding to these columns. This subset of equations can be selected from (13) under the following circumstances:

– $\mathbf{c}^*$ should form a group of 4 neighboring samples of the same subband.
– The values of the 4-element vector $\mathbf{f^T} \cdot M^*$ should adhere to the same constraints as four bilinear weights, apart from a normalization value. This can be verified at compile-time.

When these conditions are fulfilled, a temporary texture coordinate $\mathbf{t_{temp}}$ and a normalization factor n are calculated. We can then perform hardware-accelerated bilinear sampling on the subband s:

$$x^*_{filtered} = n \cdot sample(\mathbf{t_{temp}}, s) \tag{18}$$

In practice, these conditions are fulfilled when sampling and reconstructing from the LL subband using the Haar or the bior(2,2) filter.

Note that the way [12] uses the GPU to perform transform-domain filtering requires manipulation of the original image before decomposing, resulting in degraded compression performance. Our approach avoids this problem.

9.4 Border extension

Wavelet filters with a wider footprint than Haar require border extension to avoid additional reconstruction errors at the borders. Whenever $(m + p, n + q)$ (eq. 8) refers to pixels outside the actual subband texture, the correct extension mechanism must be used.

Most literature covers border extension as it is performed on interleaved decomposed data. In our case - where reconstruction is performed using non interleaved data - it is beneficial to perform the extension during the sampling operation itself, as half-point extension can be handled by the GPU hardware without any performance penalty. Consequently, the GPU performs half-point extension by default, without any modification to the reconstruction algorithms[1]. To perform extension correctly in all cases, we must use the correct extension mechanism depending on the boundary and whether or not a band is high-pass.

Table 2 Extension mechanisms for odd wavelet filters during texture sampling. Odd wavelet filters use point extension on traditional interleaved subband data.

	lower extension	upper extension
low-pass	point	half-point
high-pass	half-point	point

Table 2 shows how the classic point extension translates to extension mechanisms that can be used during texture sampling on non interleaved data.

A one-dimensional texture coordinate $t \in [0, 1]$ can be modified such that the GPU sampler, which can perform half-point extension by default, performs point symmetric extension on a one-dimensional texture of length N as

$$t = t - (1.0/N) * (1 - \lceil (t - 0.5/N) \rceil) \tag{19}$$

This equation moves the location of t one texel to the left whenever $t < 0.5/N$. Note that this particular equation is used when we are performing a lower-bound extension, i.e. the texture coordinate t approaches 0 and we have to deal with samples with a negative index.

In the same way a texture coordinate t is modified for upper bound point symmetric extension as:

$$t = t + (1.0/N) * (\lceil (t + 0.5/N) \rceil - 1) \tag{20}$$

10 Multi-level wavelet reconstruction shaders

The preceding sections focused on the algorithms needed to develop a single-level wavelet reconstruction shader. This section covers the changes required to develop a multi-level wavelet reconstruction shader.

10.1 Immediate multi-level wavelet reconstruction

The immediate reconstruction algorithm does not use any intermediate buffers nor does it require any indirection instructions; the contribution of any subband sample to the spatial domain result can be immediately calculated. Hence, the parity of the

[1] We can do this by setting the OpenGL texture coordinate wrapping mode to GL_MIRRORED_REPEAT.

spatial-domain elements has to be known at compile-time. Once again, this implies the enlargement of the filter-specific spatial-domain window X_f to cover all the parity cases possible. These additional spatial-domain parities are a result of the parities of the intermediate low-pass dependencies. E.g., a two-level decomposition features a 4 by 4 sized window with 16 unique parity combinations, whereas a single level decomposition features only 4 unique parity combinations.

Starting from two-levels, the output window of the reconstruction matrix M has to be significantly enlarged to cover the parity cases of all levels. The filter weights $\mathbf{f}$ ensure the extraction of the filtered value of this enlarged output window. Hence, the final equation can still be written as equation 13.

The construction of the matrix M starts with the extension of the output vector, corresponding to the enlarged set of offsets X_{f+p}. For instance, a two-level decomposition implies a 4 by 4 output footprint to cover all the parity cases, enlarged to 5 by 5 to be able to perform bilinear filtering. The coefficients of M are based on the coefficients used in equation 8. For each additional decomposition level (> 1), M will feature low-pass coefficient dependencies which cannot be sampled. These dependencies are converted to dependencies on the next decomposition level, according to the appropriate reconstruction equations, leading to multiplications of filter coefficients.

Although this approach might seem wasteful due to the increased size of M at first sight, it is in practice very efficient when reconstructing just the low pass contributions, especially when combined with transform domain filtering.

10.2 Recursive reconstruction

This algorithm mirrors classical wavelet reconstruction: for a given Mallat decomposition, the low-pass subband and its corresponding equally sized high-pass subbands are assembled to form a new low-pass band. This process is repeated until the desired resolution level has been reconstructed.

Given a final window of spatial-domain pixels to be reconstructed, as defined by X_{f+p}, we identify the low-pass dependencies using equation 5, resulting in a set of low-pass offsets X_{-1} (see figure 13), which in turn will have to be reconstructed if more than one decomposition was performed on the original image data.

While the parity of each of the spatial-domain elements is known at compile-time, the parity of the elements of the intermediate low-pass levels is not known. E.g. when reconstructing elements of A_{-1} (notation as in figure 2), the offsets of X_{-1} will be applied to $(2m, 2n)/2$. In contrast to equation 11, any compile-time information about the parity of this position is lost. Consequently, the calculations to reconstruct the elements of A_{-1} would require conditional equations. Therefore, this particular algorithm opts to reconstruct intermediate elements of A_{-1} relative to position $2\lfloor(m, n)/2\rfloor$. Once again, in 3 out of 4 cases, we will lack values which are required according to the window of offsets relative to $(2m, 2n)/2$. To alleviate this, we enlarge the window of low-pass dependencies with one element in each dimension, in the same way as shown in figure 12. The intermediate samples are calculated, stored in a buffer and finally, based on the actual (run time) parity of $(2m, 2n)/2$, the values are rearranged in memory such that they are in the correct position for the next stage of reconstruction. Note that rearranging the values does require conditional instructions, but the related divergent code path is very short compared to actual conditional reconstruction equations.

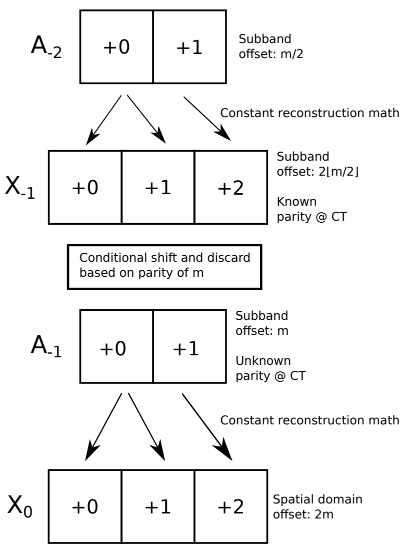

Fig. 13 A visualization of the recursive reconstruction process for the GPU, using a two-level Haar transform. Note the use of a conditional shift for the intermediate low-pass samples of A_{-1}. This step makes sure the correct samples are used for the last reconstruction step.

This procedure is repeated for each additional decomposition level, e.g. in case of three decomposition levels an additional set of intermediate samples is constructed as defined by X_{-2} and $2\lfloor(m, n)/4\rfloor$.

An overview of this reconstruction process using a two-level one-dimensional Haar transform is given in figure 13.

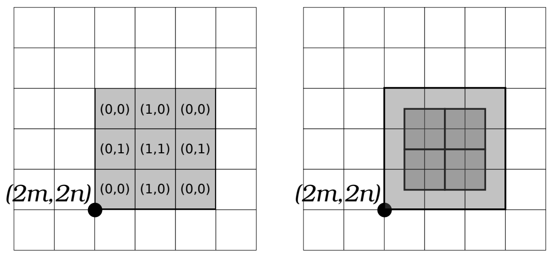

Fig. 14 On the left, we show how X_f features 9 offsets and 4 different parities. On the right, we show the 4 different areas in which bilinear filtering is performed, each using just 4 spatial-domain elements with parities $(0,0)$, $(0,1)$, $(1,0)$ and $(1,1)$. Note how each of the areas depends on the middle $(1,1)$ element, while the position of the $(0,0)$, $(0,1)$ and $(1,0)$ dependencies is variable.

10.3 Recursive reconstruction with indirection

A modification of the previous algorithm allows us to reduce the number of calculations in the final reconstruction step (reconstructing the elements defined by X_f) at the cost of some indirection. In the previous section, we reconstructed 9 spatial-domain elements, while we actually only need 4 elements to perform bilinear filtering at a given position $\mathbf{t}$. We exploit this by reconstructing exactly one element of each of the four parity combinations, which covers the source elements of each of the 4 filtering areas shown in figure 14.

For simplicity, we will explain this process for 1D reconstruction, where just two spatial-domain parities exist, even and odd, as defined by a in $2m + a$. We calculate two values: x_{even} and x_{odd}. Depending on a, we alter the reconstruction of x_{even} by shifting its input dependencies by one position. This shift in the wavelet domain allows us to shift between the reconstruction of $2m + 0$ and $2m + 2$, both even elements. At position $2m + 1$, x_{odd} is reconstructed as usual. A downside of this approach is the introduction of additional memory indirection, as the positions of the samples needed for the reconstruction of the even element now depend on a and m, while the samples needed for the odd element only depend on m. To minimize the impact of the memory indirection, a sample buffer is filled with dependencies for all 3 potential reconstructed elements at positions $2m + 0$, $2m + 1$ and $2m + 2$. Values in this buffer are then shifted whenever necessary during construction, which turns out to be faster than accessing the buffer using the extra indirection variable a.

Linear filtering can in this case be efficiently computed as:

$$x = \begin{cases} (1-\alpha)x_{even} + \alpha x_{odd} & \text{if } \lfloor tN - 0.5 - p \rfloor = 0 \\ (1-\alpha)x_{odd} + \alpha x_{even} & \text{else} \end{cases} \tag{21}$$

11 Subband coding performance

This section compares the performance of the quantization modes proposed in section 6.2 to some of the most efficient existing single-channel texture compression systems, justifying our choice to incorporate these quantization modes in the proposed texture compression system.

We will briefly overview these codecs, explain how they are used to compress subband data and assess their performance.

11.1 Evaluated subband coding methods

11.1.1 Subband encoding using NVTT LATC

LATC [6] works on 4x4 blocks of texels. For each block, a lower and upper bound value is encoded using 8 bits each. These values are the bounds of a 3 bit uniform quantizer, hence sixteen 3 bit indices are stored for each block. On the GPU, dequantization is performed using dedicated hardware at no additional cost. This particular encoder implementation is found in the open source project NVIDIA Texture Tools [26]. Subband data is converted to an 8 bit integer representation before being compressed.

11.1.2 Subband encoding using floating point customized LATC

This custom-made floating point encoder uses floating point values as input data instead of 8 bit integers. This can make a significant difference, as it allows each of the 4x4 block quantizations to optimize for floating point reconstruction points. Subband data is scaled to a [0..1] floating point range before being compressed.

11.1.3 Subband encoding using ASTC

Adaptive Scalable Texture Compression [4] is a new texture codec developed by ARM and AMD and is currently supported on certain mobile platforms. Although mainly developed for multi-channel content, it also supports single-channel data. Subband data is converted to an 8 bit integer representation before being

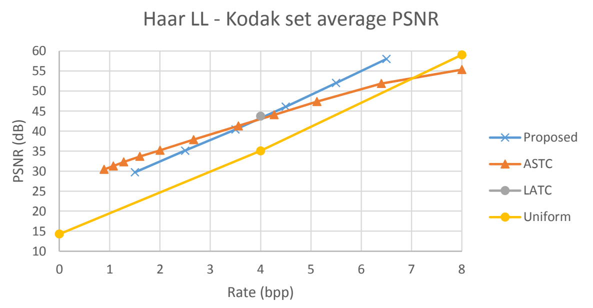

Fig. 15 2 level Haar LL subband compression performance averaged over the entire Kodak image set.

compressed. Compression software is freely available at [27]. Note that current desktop GPUs do not support this format; support is only available on certain mobile devices. Still, we include this format as it is a good experimental benchmark.

11.1.4 Subband encoding using global uniform quantization

For reference, 8 bit and 4 bit uniform quantization is also performed. In the figures in this section this mode is grouped together with the only encoding mode that does not store any data at all, effectively zeroing the values of the subband in question.

11.1.5 Subband encoding using the proposed subband coding technique

Our proposed subband coding system, explained in section 6.2, was developed focusing on single-channel, zero centered subband data. All four modes work on 4x4 blocks and spend 8 bits on quantizer endpoint storage per block. The different versions use respectively 1, 2, 3, 4, 5 or 6 bits per texel for the quantization indices, resulting in respectively 1.5, 2.5, 3.5, 4.5, 5.5 and 6.5 bpp.

11.2 Subband compression performance

Evaluation of the previously mentioned subband codecs was done on subbands of the images of the Kodak set [28]. These subbands were the result of single wavelet decomposition of a luminance channel, either using the Haar or the bior(2,2) 2D wavelet transform. Before quantization, the values of each subband were optimally rescaled. Low pass data was shifted using the average coefficient value whenever this resulted in improved compression performance.

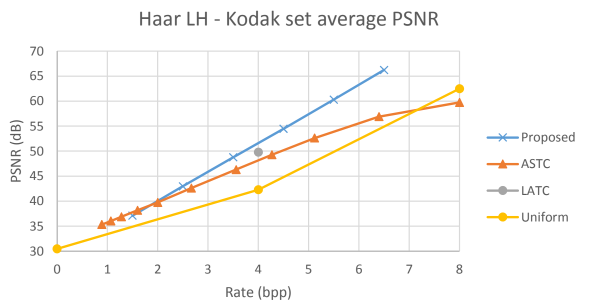

Fig. 16 2 level Haar LH subband compression performance averaged over the entire Kodak image set.

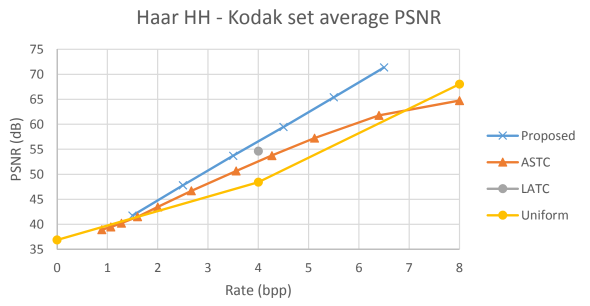

Fig. 17 2 level Haar HH subband compression performance averaged over the entire Kodak image set.

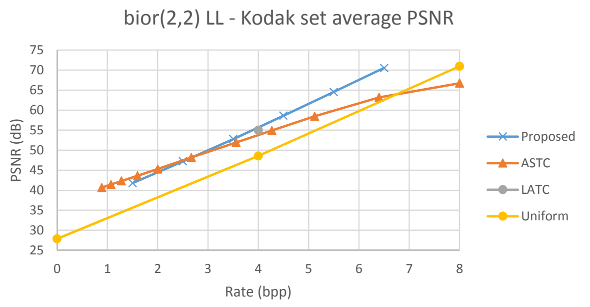

Fig. 18 2 level bior(2,2) LL subband compression performance averaged over the entire Kodak image set.

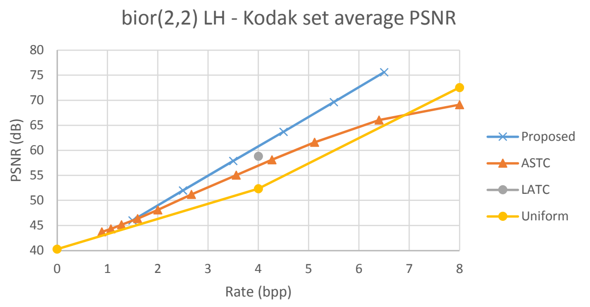

Fig. 19 2 level bior(2,2) LH subband compression performance averaged over the entire Kodak image set.

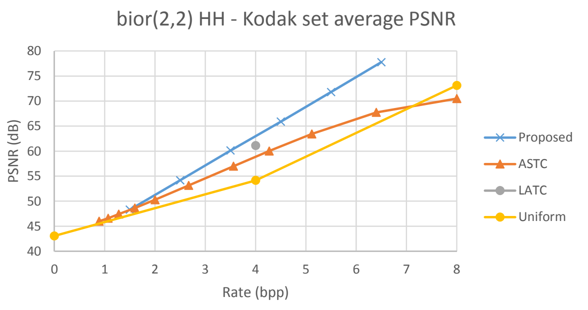

Fig. 20 2 level bior(2,2) HH subband compression performance averaged over the entire Kodak image set.

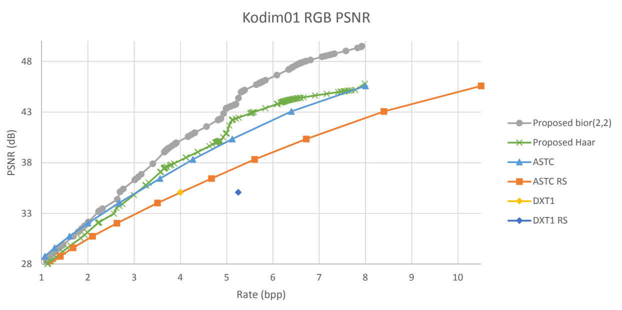

Fig. 21 Kodim01 - Proposed compression performance compared to spatial-domain codecs.

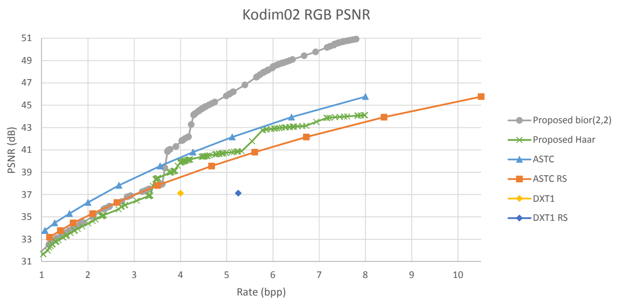

Fig. 22 Kodim02 - Proposed compression performance compared to spatial-domain codecs.

The charts in figures 15 - 20 depict the Peak Signal to Noise Ratio - PSNR (dB) as a function of bitrate when encoding the subbands of the images of the Kodak set. The charts show that the proposed system has a clear advantage when coding high-pass subband data, except for the lowest bitrate. As can also be seen in the charts, the added complexity of ASTC does not pay off in single-channel scenario's, as LATC - a much simpler codec - on average outperforms ASTC. Moreover, although the method used to discretize the local block's bounds used in our proposed method was optimized for high-pass subbands, the results show that our proposed method still outperforms existing implementations at most rates when applied to low-pass subband data.

12 Compression results

This section presents the spatial domain compression performance of the proposed texture compression system.

As each of the two state-of-the-art transform-based reference works ([13], [12]) has provided results for a different data set, separate comparisons will be made for each of these two techniques. First, we compare the proposed system with state-of-the-art spatial-domain techniques.

12.1 Comparison with conventional techniques

Although ASTC is not yet available on desktop GPUs, we include it as a baseline, as it offers a wide variety of bitrates. The test subjects are the images from the Kodak image set [28].

The results featured in this comparison were produced using the following set of subband quantizers: our local quantizers presented in section 6.2, LATC and ASTC. These schemes were picked whenever appropriate by the rate allocation algorithm.

Note that our proposed codec uses a two-level wavelet transform in these results. Hence, it features resolution scalability and inherently includes data that should be included in a mip map pyramid. One notes that ASTC would require $1 + 1/4 + 1/16 = 1.3125$ as much rate as the proposed solution if a mip map pyramid was required. Therefore, for ASTC we report two sets of results, one corresponding to single-resolution decoding, and a second corresponding to a resolution scalable (RS) version producing a two-level pyramid; this later version is indicated with the acronym ASTC RS in the graphs.

The results of these experiments are shown in figures 21 and 22, where we show the rate distortion curve of these two images. A summary of the experiments is shown in figure 23, where the average PSNR gain or loss is shown of both the Haar and bior(2,2) method versus ASTC RS.

The summary in figure 23 reveals that while the bior(2,2) version of our codec performs very well, the Haar version of our codec struggles with some of the images of the Kodak set. These images feature both smooth areas and areas with lots of high frequency detail. Hence, the resulting subband data also features areas with small magnitude coefficients and other areas with large coefficients. However, each subband is scaled using just one global scaling value, which fails to optimally scale the values of both regions. Note that

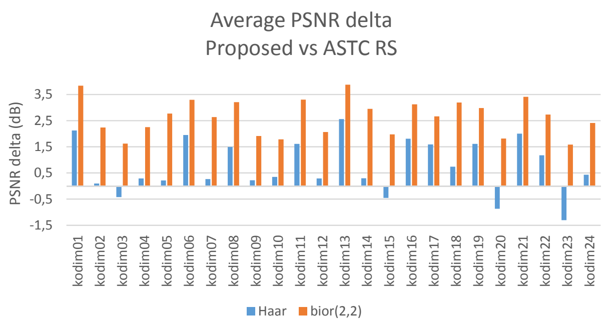

Fig. 23 Average quality difference between the proposed method and ASTC RS. The average PSNR delta was calculated using the method by Bjontegaard [29].

the bior(2,2) transform manages to capture first order transitions (smoothness) in the low pass subband, which is which is de facto encoded at higher rates and less vulnerable to scaling issues.

12.2 Comparison with Mavridis et al.

The work by Mavridis et. al. [13] features 4 encoding modes. The luminance 2 bpp mode performs a single-level partially inverted Haar transform. The resulting subbands are encoded using DXT5 or BC7. The 2.25 bpp and 3 bpp color modes encode the luminance data using the 2 bpp mode, and perform downsampling on the color channels, after which they are also encoded using the 2 bpp mode. The 5 bpp mode does not utilize the wavelet transform for the luminance channel, which is instead encoded as spatial domain data using LATC.

In this comparison, our codec performs similarly a single-level Haar or biorthogonal transform on the luminance channel. The color channels are decomposed using the corresponding 3 level wavelet transform. We do not downsample the color channels and simply rely on the rate allocation algorithm to optimally encode the image, given the quality metric - which is RGB PSNR in this case. The subband encoding methods used for this comparison are our proposed local quantization methods and LATC. Note that the three decompositions on the chrominance data result in a theoretical increase in computational complexity. However, in practice the rate allocation algorithm always discards the higher frequency bands of the chrominance data, which enables negating this effect.

The results are shown in figures 24 - 26 for the first two images of the Kodak image set and the rooftiles images used by Mavridis et.al.

Figure 27 shows the average PSNR gain or loss of both the Haar and bior(2,2) method versus the methods by Mavridis et. al.

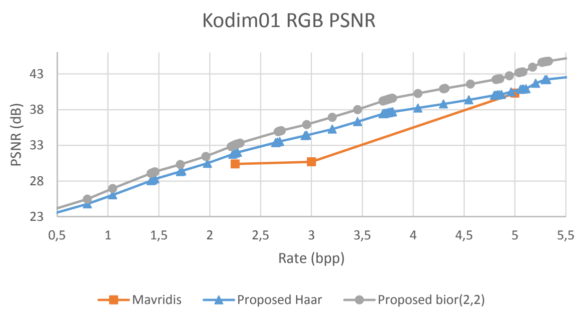

Fig. 24 Compression results for kodim01.

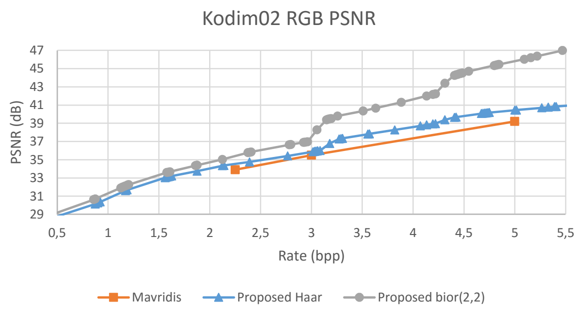

Fig. 25 Compression results for kodim02.

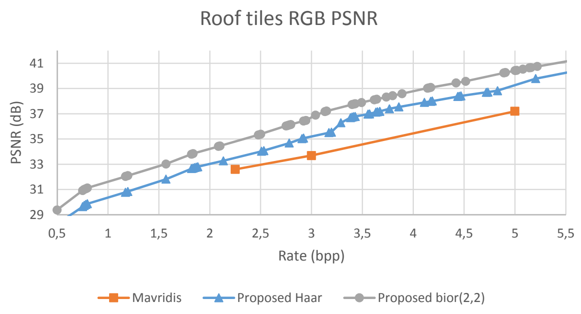

Fig. 26 RGB compression results for the rooftiles image used by Mavridis et. al.

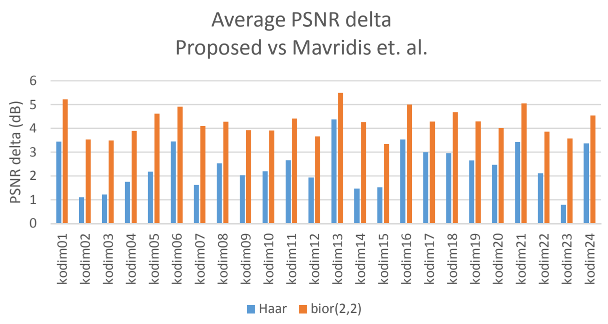

Fig. 27 Average quality difference between the proposed method and the method by Mavridis et.al. The average PSNR delta was calculated using the method by Bjontegaard [29].

Figures 28, 29 and 30 show 64x64 areas of three different sample images, decompressed at various rates using the method by Mavridis et. al. and our method. The denoted rate and PSNR values are measured on the entire images. The first two images feature lots of high frequency details and could be used as textures in games. The third image is an image taken from the Kodak image set.

Figure 28 shows some banding artifacts in our proposed lowest rate Haar image, whereas in the leftmost image (Mavridis et. al.) there is a is a bit more loss of high frequency details. The top three images (highest quality) show that the Haar based images show more pronounced high frequency detail at virtually equal RGB PSNR values.

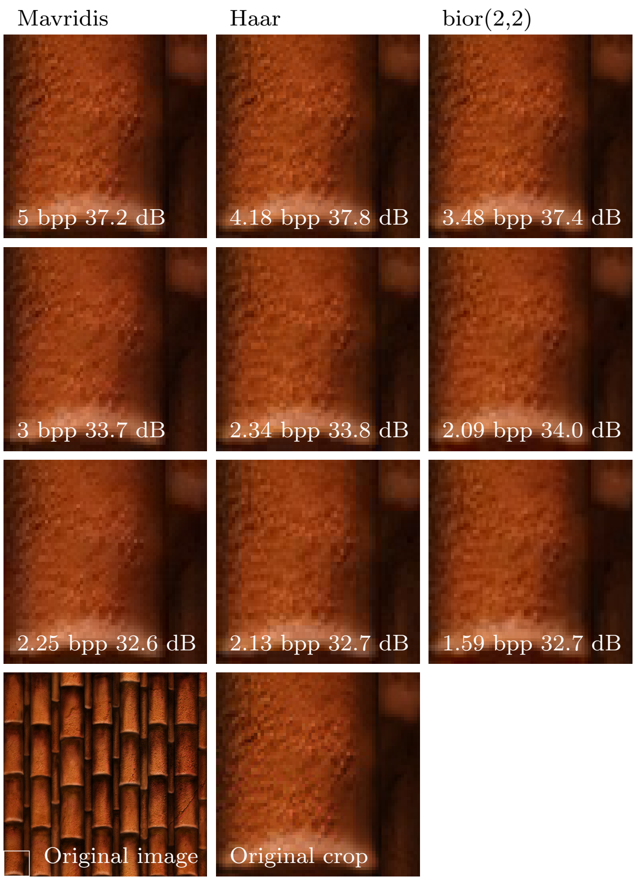

Fig. 28 A visual comparison of the rooftiles image provided by Mavridis et. al. The leftmost column features the images encoded with the three modes designed by Mavridis et. al., the two other columns feature images encoded using our solution at comparable qualities.

The image used in figure 29 is less dependent on color accuracy. This clearly shows in the lowest quality comparison, where our proposed images which feature much more luminance detail.

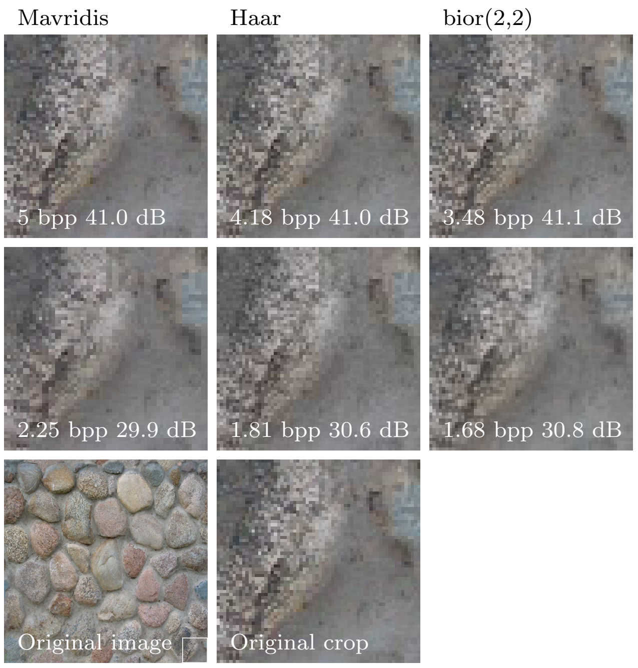

Fig. 29 A visual comparison of the stone image provided by Mavridis et. al. The leftmost column features the images encoded with the three modes designed by Mavridis et. al., the two other columns feature images encoded using our solution at comparable qualities. The 3 bpp result by Mavridis et. al. was omitted as it improved the PSNR by less than 0.2 dB compared to the 2.25 bpp version.

Figure 30 shows a crop of the kodim21 image, featuring sharp color transitions. This results in a visible color blocking artifact in our 1.81 bpp Haar image.

For future reference, we included compression results achieved with the previously explained configuration on the entire Kodak set in table 3.

12.3 Comparison with Hollemeersch et al.

The core of the codec presented by Hollemeersch et al. [12] is a 4x4 DCT transform, after which the coefficients are packed into 16 frequency bands. Each of these frequency bands is then either discarded or encoded using LATC. The work presented by Hollemeersch et al. reports PSNR values measured on the luminance channel only. To make a fair comparison possible, the bitrates of these results were transformed such that only the rate related to the luminance channel is taken into account. The subband encoding methods used for this comparison are our proposed local quantization methods and LATC.

Our proposed codec was configured to perform two decompositions on the luminance channel in order to feature at least the same resolution scalability as the

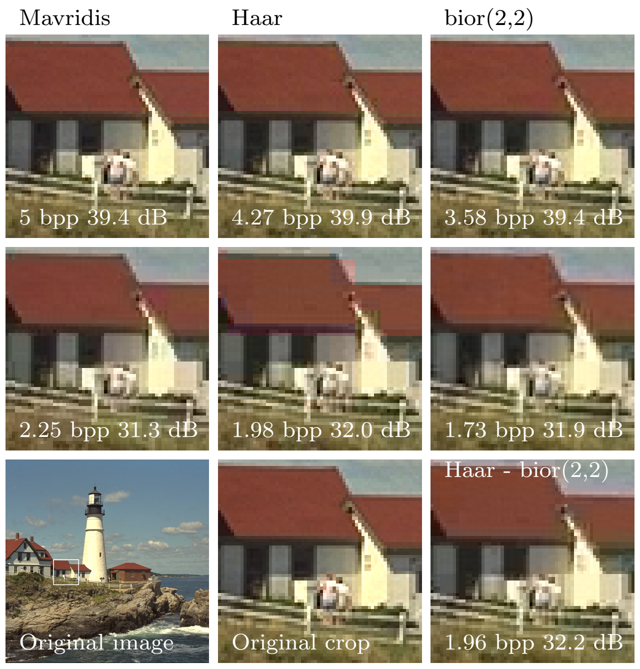

Fig. 30 A visual comparison of the kodim21 image. The 3 bpp result by Mavridis et. al. was omitted as it improved the PSNR by less than 0.3 dB compared to the 2.25 bpp version. The lower right image was produced using Haar for the luminance channel and bior(2,2) for the chrominance channels.

	Haar			bior(2,2)		
image	2	4	6	2	4	6
01	30.5	37.7	43.1	31.5	39.7	46.0
02	33.4	38.4	41.4	34.0	41.4	48.2
03	34.9	40.6	43.9	36.2	43.0	47.6
04	33.9	39.0	42.9	35.0	41.5	46.2
05	28.0	34.9	39.6	29.4	37.7	43.8
06	32.2	39.2	43.7	33.5	40.7	46.3
07	33.6	39.1	43.1	35.3	42.3	47.8
08	28.6	35.7	40.7	29.7	37.8	43.6
09	34.3	40.1	44.6	36.0	42.0	47.1
10	33.9	40.4	44.5	35.1	42.3	46.6
11	31.5	38.5	43.2	32.7	40.4	46.9
12	35.6	41.1	45.3	36.3	43.2	48.5
13	27.5	35.7	40.8	28.4	36.9	42.7
14	29.8	35.5	39.6	31.4	38.3	44.6
15	33.5	39.0	42.3	34.2	41.1	46.1
16	34.8	41.9	46.8	36.9	44.0	49.3
17	34.0	40.4	44.8	35.1	42.3	46.2
18	29.5	36.8	41.0	30.8	38.8	43.2
19	32.1	39.8	44.1	33.9	41.4	46.3
20	33.5	39.8	43.1	34.5	41.3	45.6
21	32.0	39.1	43.5	33.4	40.7	45.7
22	32.4	37.9	41.7	33.6	39.9	44.6
23	33.9	39.1	43.0	36.4	42.0	46.9
24	29.6	36.2	39.7	30.8	37.4	41.2

Table 3 Compression results for the Kodak image set using the compression configuration described in section 12.2. The row on top shows the rate in bits per pixel, the results below are RGB PSNR measured in dB.

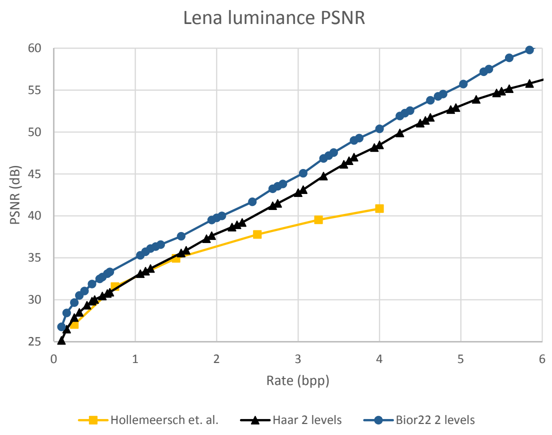

Fig. 31 Lena luminance PSNR. Compression was performed using the proposed system employing the Haar and bior(2,2) filter, and compared with the DCT system by Hollemeersch et al. [12]

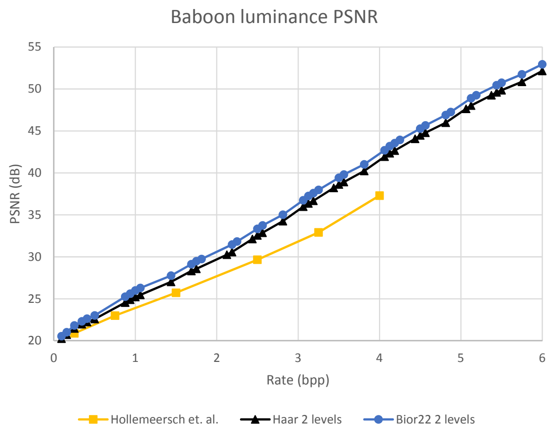

Fig. 32 Baboon luminance PSNR. Compression was performed using the proposed system employing the Haar and bior(2,2) filter, and compared with the DCT system by Hollemeersch et al. [12]

codec by Hollemeersch et al. Figures 31 and 32 show the obtained PSNR results on the luminance channel, compared with the results of Hollemeersch et al. Our Haar-based system performs at least as well, while our bior(2,2)-based system clearly outperforms the DCT-based results of [12].

13 Run-time performance

This section covers the complexity of the proposed system. Three aspects are independently evaluated: the complexity of the wavelet reconstruction, the performance of the different wavelet reconstruction algorithms, and a shader implementation of the proposed quantization scheme.

Filter	Subbands	Instr.	Fillr. (GP/s)
Haar	LL	1	25.3
bior(2,2)	LL	1	23.5
Haar	LL, LH, HL	3	20.7
bior(2,2)	LL, LH, HL	5	12.5
bior(4,4)	LL, LH, HL	16	4.34
Haar	2 levels, 5 bands	6	11.4
bior(2,2)	2 levels, 5 bands	13	6.0
bior(4,4)	2 levels, 5 bands	60	1.9

Table 4 Single channel wavelet reconstruction complexity, given by the the amount of texture sampling instructions required when using the recursive reconstruction scheme. Note that a single texture sampling instruction can yield 4 samples at the same time, e.g. when using *textureGather*. The fill rate was calculated for the reconstruction of the highest detail level using bilinear filtering on an NVIDIA GTX 980.

13.1 Reconstruction complexity analysis

As the amount of configurations of our system are virtually unlimited, we will limit the complexity analysis to a few cases. The configurations range from Haar low-pass data with all high-pass data discarded (essentially no difference with a downsampled texture) to long wavelet filters such as bior(4,4) featuring many decomposition levels. The complexity and the run-time performance of a selection of configurations is shown in table 4.

The big jump in complexity in case of the bior(2,2) filter when moving from the three-band single level configuration to the five-band two level configuration is caused by the inclusion of an HH subband. This introduces in the case of the bior(2,2) filter a 3x3 texel dependency, which requires 4 separate texture sampling instructions, as the *textureGather* instruction can only sample a 2x2 area.

13.2 Wavelet reconstruction performance

Similar to the benchmarking setup in the work by Mavridis et. al. [13], a tunnel scene was created and the performance of each compression configuration was compared to a native DXT1 texture. Our compression configuration used 5 compressed textures: 3 low-pass textures for each of the three channels and 2 additional high-pass luminance textures. For the trilinear filtering configuration (linear_mip_linear) an additional texture was used featuring the required mip levels. Two filtering configurations were tested: linear, featuring bilinear magnification filtering and thus suffering from filtering artifacts further away from the rendered objects, and linear_mip_linear, also known as trilinear filtering. This mode in turn was tested in two additional configurations. The naive shader first performs wavelet reconstruction and filters the values afterwards. The second one uses indirection; whenever appropriate, wavelet reconstruction is skipped and low resolution data is sampled instead. The relative performance figures are shown in figure 33.

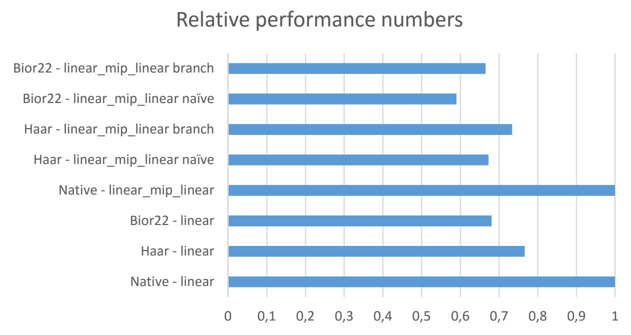

Fig. 33 Relative performance of the proposed reconstruction system.

We note that the performance loss compared to a bare shader is to be expected, and an actual implementation should combine the variants of the proposed algorithms into a heavily optimized shader. At sampling rates up to 25 GP/s, increased streaming capacity and decreased memory consumption is an interesting option in exchange for a lower sampling rate.

An additional fill rate test on Haar low-pass reconstruction showed a 6 % performance improvement for transform-domain filtering over a straightforward filtering implementation.

13.3 Fast local subband decoding performance

The proposed texture compression technique can be likely efficiently implemented in hardware, as it is very similar to already implemented texture coding techniques. Still, we implemented a proof of concept reconstruction shader. The implemented shader decodes a 1.5 bpp texture and a 2.5 bpp texture at the same time. This configuration is a relevant one, as it was often selected by the exhaustive search system of our proposed codec to encode both the LH and HL band.

The encoded texture data was packed in blocks of 64 bit by using a 32 bit unsigned integer texture and packing the data of a 4x4 block as two integers. Hence, consecutive integer 'texels' would actually encode 32 texels, 16 belonging to the 1.5 bpp encoded subband and 16 belonging to the 2.5 bpp subband. The actual packing of each 4x4 block was done as follows: 8 bits were used for the two indices of the quantizer bounds of the 1.5 bpp local quantizer, followed by 8 bits for the bounds of the quantizer of the 2.5 bpp block. Then, 16 bits were used to store the quantization indices of the 16 texels of the 1.5 bpp block. The remaining 32 bits were used for the 16 quantization indices of the 2.5 bpp values.

A straightforward shader-based implementation of a decoder for this packed texture data resulted in 47 % of the native fillrate of a hardware decoded DXT1 texture with an equal memory footprint. This strengthens our claim that an efficient hardware implementation will not be a significant challenge. The only main difference is the usage of a small lookup table for the quantizer bounds, compared to an integer to float conversion for the bounds of LATC and DXT1 style quantizers.

14 Conclusions

In this paper we presented a wavelet-based texture codec which outperforms the state-of-the-art in compression performance. A wide range of bitrates is available, ensuring that the user-defined quality requirements for each texture can be met. The codec performs combined decoding and texture sampling in real-time on commonly available consumer GPUs. The proposed system is able to utilize wavelet filters of lengths beyond Haar, which was the only wavelet transform used in the coding literature so far for texture compression. It was found that a good trade-off between compression performance and complexity is given by the bior(2,2) wavelet transform, resulting in real-time decompression and high reconstruction quality. Dynamic shader code generation and multiple reconstruction algorithms allow for per GPU optimized code and make it possible to adapt the system to future changes and improvements to consumer GPUs. The system in its entirety is highly flexible thanks to its inherent resolution scalability and configuration possibilities. It can be deployed in a wide range of applications as a drop-in GPU shader, significantly lowering the required memory and bandwidth.

Acknowledgements The Kodim test images used in this paper are courtesy of KODAK [28].

References

1. K. I. Iourcha, K. S. Nayak, and Z. Hong, "Fixed-rate block-based image compression with inferred pixel values," Dec. 2 2003. US Patent 6,658,146.
2. Microsoft, "Texture Block Compression in Direct3D 11." `https://msdn.microsoft.com/en-us/library/windows/desktop/hh308955`. Accessed 20 Jan 2016.
3. E. Delp and O. Mitchell, "Image compression using block truncation coding," *IEEE Transactions on Communications*, vol. 27, no. 9, pp. 1335–1342, 1979.
4. J. Nystad, A. Lassen, A. Pomianowski, S. Ellis, and T. Olson, "Adaptive Scalable Texture Compression.," in *High Performance Graphics* (C. Dachsbacher, J. Munkberg, and J. Pantaleoni, eds.), pp. 105–114, Eurographics Association, 2012.
5. A. C. Beers, M. Agrawala, and N. Chaddha, "Rendering from Compressed Textures," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, (New York, NY, USA), pp. 373–378, ACM, 1996.
6. M. J. Kilgard, P. Brown, Y. Zhang, and A. Barsi, "LATC OpenGL extension." `https://www.opengl.org/registry/specs/EXT/texture/compression/latc.txt`, 2009. Accessed 20 Jan 2016.
7. A. V. Pereberin *et al.*, "Hierarchical approach for texture compression," in *Proceedings of GraphiCon ¿99*, pp. 195–199, 1999.
8. S. Fenney, "Texture Compression Using Low-frequency Signal Modulation," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS '03, (Aire-la-Ville, Switzerland, Switzerland), pp. 84–91, Eurographics Association, 2003.
9. S. Diverdi, N. Candussi, and T. Höllerer, "Real-time rendering with wavelet-compressed multi-dimensional textures on the GPU," in *University of California, Santa Barbara*, Citeseer, 2005.
10. C.-H. Sun, Y.-M. Tsao, and S.-Y. Chien, "High-quality mipmapping texture compression with alpha maps for graphics processing units," *IEEE Transactions on Multimedia*, vol. 11, no. 4, pp. 589–599, 2009.
11. N. Grund, N. Menzel, and R. Klein, "High-Quality Wavelet Compressed Textures for Real-time Rendering," in *WSCG Short Papers*, no. 18, pp. 207–212, 2010.
12. C.-F. Hollemeersch, B. Pieters, P. Lambert, and R. Van de Walle, "A new approach to combine texture compression and filtering," *The Visual Computer*, vol. 28, no. 4, pp. 371–385, 2012.
13. P. Mavridis and G. Papaioannou, "Texture compression using wavelet decomposition," in *Computer Graphics Forum*, vol. 31, pp. 2107–2116, Wiley Online Library, 2012.
14. C. Tenllado, R. Lario, M. Prieto, and F. Tirado, "The 2d discrete wavelet transform on programmable graphics hardware," in *IASTED Visualization, Imaging and Image Processing Conference*, 2004.
15. T.-T. Wong, C.-S. Leung, P.-A. Heng, and J. Wang, "Discrete wavelet transform on consumer-level graphics hardware," *IEEE Transactions on Multimedia*, vol. 9, no. 3, pp. 668–673, 2007.
16. C. Tenllado, J. Setoain, M. Prieto, L. Piñuel, and F. Tirado, "Parallel implementation of the 2d discrete wavelet transform on graphics processing units: Filter bank versus lifting," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 3, pp. 299–310, 2008.
17. W. J. van der Laan, A. C. Jalba, and J. Roerdink, "Accelerating wavelet lifting on graphics hardware using CUDA," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 1, pp. 132–146, 2011.
18. M. Treib, F. Reichl, S. Auer, and R. Westermann, "Interactive editing of gigasample terrain fields," in *Computer Graphics Forum*, vol. 31, pp. 383–392, Wiley Online Library, 2012.
19. H. Malvar and G. Sullivan, "YCoCg-R: A color space with RGB reversibility and low dynamic range," *ISO/IEC JTC1/SC29/WG11 and ITU*, 2003.
20. S. G. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Transactions*

on Pattern Analysis and Machine Intelligence, vol. 11, no. 7, pp. 674–693, 1989.
21. A. Cohen, I. Daubechies, and J.-C. Feauveau, “Biorthogonal bases of compactly supported wavelets,” *Communications on pure and applied mathematics*, vol. 45, no. 5, pp. 485–560, 1992.
22. S. Mallat, “A Wavelet Tour of Signal Processing Academic,” *New York*, vol. 16, 1998.
23. B. Andries, J. Lemeire, and A. Munteanu, “Optimized quantization of wavelet subbands for high quality real-time texture compression,” in *IEEE International Conference on Image Processing 2014*, (Paris, France), 2014.
24. A. Alecu, A. Munteanu, P. Schelkens, J. Cornelis, and S. Dewitte, “Wavelet-based fixed and embedded L-infinite-constrained image coding.,” *J. Electronic Imaging*, vol. 12, no. 3, pp. 522–538, 2003.
25. A. Alecu, A. Munteanu, J. Cornelis, and P. Schelkens, “Wavelet-based scalable L-infinity-oriented compression.,” *IEEE Transactions on Image Processing*, vol. 15, no. 9, pp. 2499–2512, 2006.
26. “Nvidia Texture Tools.” `https://github.com/castano/nvidia-texture-tools`, 2010. Accessed 20 Jan 2016.
27. “ASTC encoder.” `https://github.com/ARM-software/astc-encoder`, 2015. Accessed 20 Jan 2016.
28. “Kodak image set.” `http://r0k.us/graphics/kodak/`. Accessed: 2015-08-10.
29. G. Bjontegaard, “Calcuation of average PSNR differences between RD-curves,” *Doc. VCEG-M33 ITU-T Q6/16, Austin, TX, USA, 2-4 April 2001*, 2001.