IEEE International Workshop on Multimedia Signal Processing (MMSP), Italy, 2013

Real-time texture sampling and reconstruction with wavelet filters

Bob Andries #*1, Adrian Munteanu #*2, Jan Lemeire #*3, Peter Schelkens #*4

# *Department of Electronics and Informatics, Vrije Universiteit Brussel, Pleinlaan 2, Brussels, Belgium*

**Department of Future Media and Imaging, iMinds V.Z.W., G. Crommenlaan 8, Ghent, Belgium*

1bob.andries@vub.ac.be, 2acmuntea@etro.vub.ac.be, 3jlemeire@etro.vub.ac.be, 4pschelke@etro.vub.ac.be

Abstract — **Currently, the use of the 2D wavelet transform in texture compression for real-time texture mapping on the GPU is still limited, mainly because of the lack of real-time texture filtering implementations. This work proposes a novel system to perform 2D wavelet reconstruction and bilinear texture filtering on a high performance GPU shader. The system is able to generate a GLSL shader for arbitrary wavelet filter configurations. This goes beyond earlier works in the literature proposing Haar wavelet and Discrete Cosine Transform (DCT) implementations on the GPU. We analyse the shader performance and run-time complexity for several wavelet filters. The experimental results show that filters longer than Haar are deployable on the GPU while maintaining accurate texture filtering and real-time performance.**

I. Introduction

Texture compression has been an essential tool for real time rendering for the past fifteen years. The DXT family of codecs [1] are the most popular texture compression systems, all relying on block truncation coding techniques. Despite of their popularity, the attainable compression performance obtained with such simple block-based quantization techniques is modest. In order to further improve compression performance, recent works focus on transform-based texture compression systems, mostly based on the DCT transform [2] and the Haar wavelet transform [3], [4], [5].

The transform decorrelates the input signal prior to quantization, which is known to improve the compression gain relative to quantization alone, as performed by the codecs in the DXT family. Moreover, transform-based codecs allow for more freedom than hard-coded DXT codecs. The decomposition of an image into subbands enables features such as scalable compression, per subband bitrate allocation and inherent mip-mapping support.

Most of the work in the area of transform-based texture compression cannot be deployed in practical applications, as texture filtering – an essential part of texture mapping – is usually not implemented or has some serious performance drawbacks. The systems that implement the wavelet transform in [3], [4], [5] are all limited to the Haar transform, which is known to be inferior to higher order filters required for efficiently compressing natural images. The approach in [6] focuses on practical implementations of the wavelet transform [7] on the GPU, but this technique still cannot be used in practical systems as it does not perform bilinear texture filtering in the wavelet domain. Currently there is only one transform-based scheme in the literature that performs transform-domain bilinear filtering in an efficient way, and this is the DCT-based system of [2].

Another drawback of transform-based compression is the lack of specialized hardware support, contrary to DXT. This results in additional run-time complexity, especially when texture filtering is required. However, the added complexity can be countered nowadays by the computational power of current GPUs, which has tremendously increased in the recent past. By making efficient use of this increased capacity, the fill rate – i.e. the rate at which pixels can be processed – of a transform-based reconstruction system can be sufficient for real-time texture mapping.

In this work, we propose a 2D wavelet reconstruction implementation on the GPU and investigate the transform-domain filtering problem for wavelet filters of arbitrary length. The resulting systems will be analysed in terms of computational complexity, floating-point error and bandwidth requirements. We analyse various biorthogonal wavelet filters, with varying number of vanishing moments, starting from the simple 1.1 (Haar) transform and ending with the popular 4.4 wavelet transform.

The remainder of the paper is organised as follows. Section II details the transform-domain texture filtering problem and the approach of Hollemeersch et al. [2]. Section III covers the construction of the 2D wavelet reconstruction matrix, suitable for a GPU implementation. In section IV, we explain how border extension can be used in conjunction with the matrix constructed in section III. Section V shows the results of the proposed implementation, followed by conclusions and future work given in section VI.

II. Transform domain filtering

2D texture mapping is the process of mapping the data contained in a 2D image onto a surface in 3D space, which is in turn projected onto the user's 2D screen. Bilinear filtering is an essential part of this process, merging 4 neighbouring texture elements (texels) into one texture sample.

Modern GPU's can handle bilinear texture filtering on the fly, i.e. there is no difference in performance between sampling just one texel or sampling an interpolation of four neighbouring texels. In particular, performance would suffer greatly if we would implement the reconstruction and filtering operations without making use of the built-in hardware interpolators.

A. Transform domain filtering for a block transform

Hollemeersch et. al. [2] use a 4x4 2D DCT transform, decomposing an input image into 16 frequency bands. In the lossless case, each of these 16 frequency bands is sent to the GPU as a separate texture. The advantage of this system is its ability to reconstruct and filter a pixel by using just one sample instruction per DCT frequency band.

The principle behind this system is the linearity of the filtering operators and the transform. In each dimension, two different filtering cases have to be covered: a sampling position can require samples from the same block (intra block filtering), or it can require samples from two neighbouring blocks (inter block filtering).

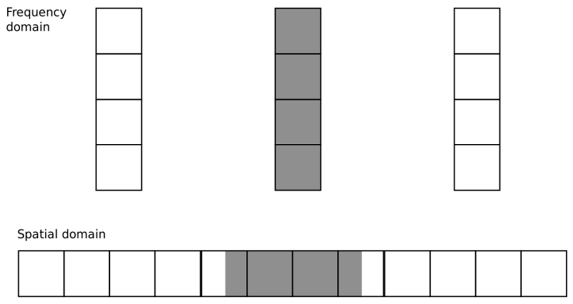


Figure 1. Intra block filtering. In this case, only DCT coefficients in the current block are required to reconstruct a filtered pixel; the 3 adjacent spatial domain blocks are delimited by thicker vertical lines.

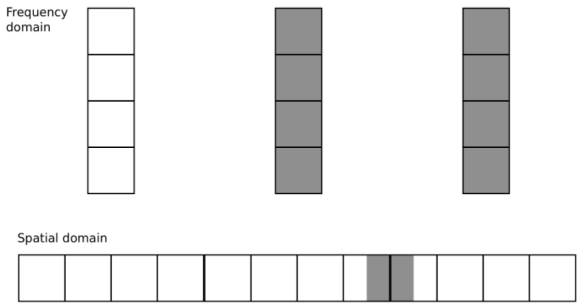


Figure 2. Inter block filtering. In this case, coefficients from neighbouring blocks are required to reconstruct a filtered pixel; the 3 adjacent spatial domain blocks are delimited by thicker vertical lines.

Intra block filtering is straightforward (Fig. 1), the linearity of the transform is exploited and equation (1) shows how multiplication of the filter weights $\overline{w}$ and the transform matrix T can be used to achieve the desired result:

$$x_{filtered} = \overline{w} \cdot \overline{x} = \overline{w} \cdot T \cdot \overline{c} = \overline{t} \cdot \overline{c}\ , \qquad (1)$$

where $\overline{x}$ is a column vector denoting the spatial-domain pixel values in the current block, $\overline{c}$ is a column vector denoting the corresponding DCT coefficients, and $\overline{t}$ is the resulting row vector performing joint reconstruction and interpolation.

Note that the filter weights $\overline{w}$ depend on the sampling position within the current block. The GPU samples the coefficients exactly at their centre, the hardware interpolation being actually not performed in this case.

In the case of inter block filtering (Fig. 2), Hollemeersch et al. [2] propose a solution employing the hardware texture filtering units. Instead of sampling at the center of each texel in each frequency band, the sampling position is shifted to a position between neighbouring texels, by an amount proportional to the actual sampling position between two neighbouring blocks. A block mirroring scheme makes sure the neighbouring texels at the edges of the blocks can be reconstructed by using exactly the same weights; for more details, the interested reader is referred to [2]. It should be noted that this mirroring scheme is applied before executing the transform, lowering spatial coherence and thus potentially lowering compression performance.

B. Transform domain filtering for a wavelet transform

Although the wavelet transform is not block-based, the approach above can be applied to handle hardware filtered wavelet reconstruction due to the localization properties of the wavelet transform. In order to use this approach for an arbitrary 2D wavelet transform, we have to define a block of samples in the spatial domain and a corresponding matrix which can reconstruct the elements of this block based on a corresponding set of wavelet coefficients. This requirement makes sure we can perform intra block filtering.

To enable inter block filtering, using the block-based approach of [2], just as many equally sized sub-bands are required in the wavelet transform representation as there are pixels in the reconstruction block. The classical wavelet transform [3] does not fulfil this requirement, that is, the size of wavelet subbands decreases with a factor of four with each decomposition level. In general, the wavelet packet decomposition is an example of a transform that can produce equally sized of sub-bands. For ease of implementation, we start from a conventional 2D discrete wavelet transform and apply polyphase decompositions of the high-frequency bands, which are appropriately performed at each resolution level until equally sized subbands are obtained across all levels. The next section will elaborate on the construction of a 2D wavelet reconstruction matrix suitable for transform domain filtering.

III. 2D DWT RECONSTRUCTION

In classical multiresolution representations, the wavelet transform is a global transform applied on the entire input image, which is performed row- and column-wise, using either conventional low- and band-pass filtering operations [7] or the lifting scheme [8].

Each texture sample has to be calculated independently of its neighbours on a GPU. Performing a global inverse wavelet transform on the entire image to get a single texture sample would be completely inefficient; that is, employing the conventional filter-bank or lifting-based implementation of the inverse wavelet transform followed by bilinear filtering in the spatial domain does not make sense in practice. Furthermore, we cannot benefit either of the well-known halving of the computational burden [8] brought by lifting when compared to traditional filtering. Even when GPGPU is used to reconstruct the entire image, it has been shown [6] that the lifting scheme [8] does not bring any performance improvements on a GPU when compared to a conventional filtering-based approach [7].

Therefore, we will base our approach on the traditional filter bank implementation [7], which can be carried out as a matrix-vector multiplication. This matrix enables us to easily analyse the data dependencies for each texel in the spatial domain and identify the corresponding samples in the wavelet domain that are necessary for its accurate reconstruction. Additionally, writing the transform as a matrix-vector multiplication allows for grouping the filtering and inverse transform operations, similar to (1). We will first present the trivial cases, corresponding to the single-level 1D and 2D matrices. Using these notations and algorithms, we will finally construct a multi-level 2D wavelet reconstruction matrix.

C. 1D case

Equations (2) and (3) below can be used to express the single level 1D wavelet transform as a matrix-vector multiplication:

$$\begin{bmatrix} l \\ h \end{bmatrix} = T_{1d} \cdot \begin{bmatrix} a_1 \\ b_1 \\ \vdots \\ a_n \\ b_n \end{bmatrix} \tag{2}$$

$$\overline{x}_k = \begin{bmatrix} a_k \\ b_k \end{bmatrix} = \tilde{T}_{1d} \cdot \overline{c}_k \tag{3}$$

where a_i, b_i are even and odd spatial-domain samples needed to compute the low- and high-pass coefficients l, h respectively, $\overline{x}_k$ is the vector containing the reconstructed spatial-domain pair of samples a_k, b_k. T_{1d}, $\tilde{T}_{1d}$ are the 1D decomposition and reconstruction matrices respectively, and $\overline{c}_k$ is the vector of wavelet coefficients needed to reconstruct $\overline{x}_k$. To further fully define the 1D reconstruction matrix we introduce an input mapping composed of $band_{1d}(i)$ and $inputoffset_{1d}(i)$, which define the band and the offset at which the coefficient can be found for index i of vector $\overline{c}_k$:

$$\overline{c}_k[i] = b\left[k + inputOffset_{1d}(i)\right]; \quad b = band_{1d}(i)$$

In other words, given the index i, one identifies the band b corresponding to i, and one extracts the element $k + inputOffset_{1d}(i)$ from this band. The matrices T_{1d} and $\tilde{T}_{1d}$ can be easily constructed making use of the filter coefficients for each filter bank instantiation imposing that perfect reconstruction is attained for any $\overline{x}_k$.

D. 2D case

In 2D, the reconstruction footprint has four elements, as can be seen in equation (4) where the 2D equivalent of equation (2) is given:

$$\overline{x}_{k,p} = \begin{bmatrix} aa_{k,p} \\ ab_{k,p} \\ ba_{k,p} \\ bb_{k,p} \end{bmatrix} = \tilde{T}_{2D} \cdot \begin{bmatrix} ll_{\mathcal{D}_1} \\ lh_{\mathcal{D}_2} \\ hl_{\mathcal{D}_3} \\ hh_{\mathcal{D}_4} \end{bmatrix} = \tilde{T}_{2D} \cdot \overline{c}_{k,p} \tag{4}$$

where $\mathcal{D}_i$ are sets of indices in the corresponding wavelet subbands (LL, LH, HL, HH) indicating the wavelet coefficients needed to reconstruct $\overline{x}_{k,p}$. Examples of such sets are given in [9].

Generating the single level matrix for the 2D wavelet reconstruction is done by convolving two 1D transforms. A first step is to generate the new elements of the transformation matrix, along with their input and output mapping, as seen in Table 1. The second step is to assemble a complete mapping of all unique inputs, defining the input vector and thus the number of columns of the new 2D reconstruction matrix $\tilde{T}_{2d}$. Similarly, the number of output offsets determines the number of rows in the new matrix. Each coefficient can then be put in the matrix according to its input and output mapping.

Table 1. Pseudo-code describing the generation of the reconstruction matrix $\tilde{T}_{2d}$ for a one level inverse wavelet transform.

Inputs:

- matrix $\tilde{T}_{1d}$, the 1D wavelet reconstruction matrix
- the mappings $band_{1d}(i)$ and $inputoffset_{1d}(i)$, mapping each row index i in $\overline{c}_k$ to a corresponding input band and an input offset
- the mapping $outputoffset_{2d}(i)$, mapping each row index i in $\overline{x}_{k,p}$ to a corresponding 2D output offset in the 2 by 2 block of reconstructed elements

Outputs:

- matrix $\tilde{T}_{2d}$, the 2D wavelet reconstruction matrix
- the mappings $band_{2d}(i)$ and $inputoffset_{2d}(i)$, mapping each row index i in $\overline{c}_{k,p}$ to a corresponding 2D input band and 2D offset

```
for each index $i$ in $\overline{x}_{k,p}$
  $rowX =$ get row $outputoffset_{2d}(i).x$ in $\tilde{T}_{1d}$
  $rowY =$ get row $outputoffset_{2d}(i).y$ in $\tilde{T}_{1d}$
  for each index $x$ in $rowX$
    for each index $y$ in $rowY$
      $value = rowX[x] \cdot rowY[y]$
      $inputBand = band2d(band_{1d}(x), band_{1d}(y))$
      $inputOffset = 2dOffset(inputOffset_{1d}(x), inputOffset_{1d}(y))$
      $index = findOrCreateIndex(inputBand, inputOffset)$
      $band_{2d}(index) = inputBand$
      $inputoffset_{2d}(index) = inputOffset$
      $\tilde{T}_{2d}[i, index] = value$
    end
  end
end
```

E. Multiple levels

When dealing with multiple levels, the size of vector $\overline{x}_{k,p}$ increases; in the case of 2 levels, the 4 element output vector in equation (4) becomes a 16 element output vector. To create the corresponding extended matrix, we first apply the algorithm in Table 1 on an extended output vector. The corresponding mapping $band_{2d}(i)$ will still depend on a level 1 low-pass sub-band, which is not available when N > 1 decompositions have been performed. Conceptually, each of the level 1 low-pass sub-band coefficients can be in turn expressed using (4) as a function of the coefficients in the level 2 subbands. Similar to (4), this will finally yield a 2D transformation matrix for 2 decomposition levels, which multiplies the coefficients of the level 1 and level 2 subbands.

A last issue is the requirement of using equally sized bands. The generated reconstruction matrices can be easily modified to support polyphase reconstruction in which case the input mappings have to be appropriately adapted.

IV. Border extension

Border extension has to be performed on the 2D signal for any wavelet transform except for the simple Haar wavelet transform. Commonly used extension methods are the half-point and the point symmetric extensions. One of these two extension techniques has to be applied both when decomposing and reconstructing the signal.

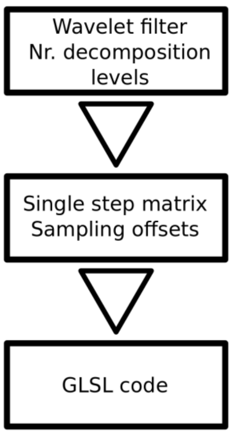
When reconstructing a sample, several texels from the wavelet sub-bands are required for reconstruction. Which texels are exactly required is determined by the position of the desired reconstructed sample and the subband and offsets stored in the input mappings $band_{2d}(i)$ and $inputoffset_{2d}(i)$. Calculating the sample positions for a single level decomposition is straightforward: for each element in $\overline{c}_{k,p}$, add the corresponding offset to the sample position at (k,p), followed by the application of border extension. Calculating the positions for a decomposition with more than one level is more complicated. These positions should be derived from the sample positions from the level above them, which complicates the construction of the one pass matrix, as we cannot simply use offsets anymore to define the input mapping. This can be alleviated by replacing the offsets introduced in section II.C by referenced positions. A referenced position consists of an offset and a parent relationship. At runtime, this results in a tree like database of positions, which can only be resolved by first calculating the root value, after which its children can be calculated and so on until all the sampling positions have been resolved.

V. Complexity and performance

F. GLSL shader generation

As a rather unbiased platform and because of the support for multiple platforms, GLSL was chosen as implementation language. To facilitate benchmarking and fast exploration of different algorithms, an automatic GLSL shader generation system was developed. The core of this system is the matrix generation described in section III and the border extension given in section IV. The system can generate a shader with just two parameters: the 1D wavelet reconstruction filter and the number of decomposition levels. It allows us to send fully unrolled GLSL code to the OpenGL drivers, which is then further optimized and compiled into a binary shader program.

Figure 1. Sequence of operations for the generation of the GLSL code.



A second advantage of this system is that it allows us to easily calculate the theoretical complexity of each configuration. This might differ from the actual run time complexity as the GLSL compiler is free to optimize and merge the unrolled calculations in the GLSL code.

G. Results

The run time performance of the proposed 2D wavelet reconstruction and bilinear texture filtering depends mainly on two factors: the number of texture sample instructions and the number of floating point operations. These two groups of instructions compose the majority of the generated GLSL code. As modern GPUs still have a hard time performing well on non-locally grouped branching instructions, these were entirely avoided.

The results obtained for wavelet filters having various number of vanishing moments for 1 and 2 decomposition levels are given in Tables 2 and 3. In the naming convention bior*xy*, *x* and *y* indicate the number of vanishing moments for the decomposition and reconstruction band-pass filters respectively.

Table 2. Computational complexity for various filter kernels and decomposition levels.

configuration	Texture samples / pixel	Matrix size (elements)
1 level Haar	4	16
1 level bior13	16	64
1 level bior31	36	144
1 level bior22	25	100
1 level bior44	81	324
2 level Haar	16	256
2 level bior13	48	768
2 level bior31	84	1344
2 level bior22	65	1040
2 level bior44	265	4240

The fill rate was derived by rendering a texture to a 1024 by 1024 render target. This number can be used as a guideline when the average number of visible textures and the screen resolution of a real time application is known. For example, our render target of about 1 Mpixel was used to render the same texture 32 times during each render pass, which requires a fill rate of at least 1 Gpixel/s to achieve a smooth frame rate of 30 fps. As for the complexity, we notice that there is very little relation between the size of the reconstruction matrix and the real time performance on the GPU; the performance of the analysed GPU shaders is probably constrained by the texture sampling units, which have to process a high amount of sample instructions per pixel.

Table 3. Fill rates for different filter kernels and decomposition levels for two different GPUs. Several OpenGL drivers had troubles compiling the GLSL source of the 2 level bior44 reconstruction shader.

	Fill rate (Gpixel/s)	
configuration	**AMD 7970**	**NVIDIA GTX 660 Ti**
1 level Haar	20.8	12.5
1 level bior13	6.31	6.74
1 level bior31	2.52	3.34
1 level bior22	5.02	4.47
1 level bior44	1.29	1.36
2 level Haar	4.29	5.25
2 level bior13	1.45	1.93
2 level bior31	0.88	1.10
2 level bior22	1.26	1.44
2 level bior44	0.072	N/A

VI. Conclusions and Future Work

This paper presents a system which enables the use of wavelet reconstruction in filtered texture mapping scenarios. The performance results show that real-time implementations are achievable for 2 level wavelet transforms with filters having up to 4 vanishing moments in total for the decomposition and reconstruction band-pass filters.

The next step is to avoid mirroring the blocks (see section II.A), as this has a detrimental effect on compression performance. Recent developments in GPU shading language, especially the newer sampling instructions, might enable us to employ some more fundamental methods to avoid this problem.

VIII. References

[1] P. Brown, S3 texture compression specification, http://www.opengl.org/registry/specs/EXT/texture_compression_s3tc.txt

[2] 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.

[3] 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.

[4] A. V. Pereberin et al. “Hierarchical approach for texture compression,” *Proceedings of GraphiCon ’99*, pp. 195–199, 1999.

[5] C. Sun, Y. Tsao, and S. 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.

[6] 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.

[7] S. 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, July 1989.

[8] W. Sweldens, “The Lifting Scheme: a Custom Design Construction of Biorthogonal Wavelets,” *Journal of Appl. and Comput. Harmonic Analysis*, vol. 3, no. 2, pp. 186-200, 1996.

[9] A. Alecu, A. Munteanu, P. Schelkens, J. Cornelis, and S. Dewitte, “Wavelet-based Fixed and Embedded Linf-constrained Image Coding,” *SPIE Journal of Electronic Imaging*, vol. 12, no. 3, pp. 522-538, July 2003.