# Hardware Accelerated Wavelet Transformations

Matthias Hopf Thomas Ertl

{hopf,ertl}@informatik.uni-stuttgart.de
http://wwwvis.informatik.uni-stuttgart.de/

Visualization and Interactive Systems Group, University of Stuttgart

**Abstract.** Wavelets and related multiscale representations are important means for edge detection and processing as well as for segmentation and registration. Due to the computational complexity of these approaches no interactive visualization of the extraction process is possible nowadays. By using the hardware of modern graphics workstations for accelerating wavelet decomposition and reconstruction we realize a first important step for removing lags in the visualization cycle.

## 1 Introduction

Feature extraction has been proven to be a useful utility for segmentation and registration in volume visualization [7, 13]. Many edge detection algorithms used in this step employ wavelets or related basis functions for the internal representation of the volume. Additionally, wavelets can be used for fast volume visualization [5] using the Fourier rendering approach [8, 12].

Wavelet analysis is a mainly memory bound problem. Graphics hardware on the other hand regularly has memory systems that can be addressed extremely fast. As modern graphics hardware of several vendors, for instance Silicon Graphics [9], has support for two dimensional convolution and the ability to scale bitmaps by arbitrary factors, all necessary steps needed for wavelet decomposition and reconstruction are available.

Additionally, three dimensional convolution with separable filter kernels can be implemented by using these hardware supported convolution filters along with volume textures [3], paving the way to 3D wavelet analysis, which will benefit from the high memory bandwidth of the graphics hardware even more.

However, there are still several pitfalls to be circumvented, which are addressed in our previous paper about the first steps to hardware based wavelet analysis [4]. In this paper, we will emphasize new algorithmic aspects of the acceleration process by utilizing special OpenGL features.

## 2 Wavelets

In the past two decades, wavelet analysis has grown from a mathematical curiosity into a major source of new basis decomposition and signal processing algorithms [10, 14]. The importance of orthonormal basis of wavelets and multi-resolution analysis resides

in their hierarchical nature, which offers a mathematical framework for describing functions at different levels of resolution. Using basis functions with good approximation properties, i.e. with many vanishing moments, one can represent functions by keeping only the important coefficients (regularly called *features*) and discarding all others. This sections gives a short introduction into the basics of wavelet theory. Details on the theory can be found in [1, 2, 6].

A multi-resolution analysis can be thought of as a ladder of approximating closed subspaces $(V_j)_{j\in\mathbf{Z}}$ of $L^2(\mathbf{R})$. The functions in these subspaces have well defined scaling and translation properties. Furthermore, there exists a function $\phi \in V_0$ such that $\{\phi_{0,n};\, j,n \in \mathbf{Z}\}$ with $\phi_{j,n} = 2^{j/2}\phi(2^j x - n)$ is an orthonormal basis of $V_0$. Under these conditions one can construct an orthonormal wavelet basis $\{\psi_{j,n};\, j,n \in \mathbf{Z}\}$ with $\psi_{j,n} = 2^{j/2}\psi(2^j x - n)$, such that for any function $f$ in $L^2(\mathbf{R})$

$$P_j f = P_{j-1} f + Q_{j-1} f \;, \tag{1}$$

where $P_j$ and $Q_j$ are the orthogonal projections onto $V_j$ and $W_j$, respectively:

$$P_j f = \sum_{n\in\mathbf{Z}} < f, \phi_{j,n} > \phi_{j,n} \;\;, \quad Q_j f = \sum_{n\in\mathbf{Z}} < f, \psi_{j,n} > \psi_{j,n} \;.$$

The function $\psi$ is sometimes called the *mother* wavelet. The projection $P_j f$ onto the subspaces $V_j$ corresponds to the different resolution levels in which the function $f$ can be decomposed. These projections contain the *smooth* information of $f$ at a given level of resolution. The projections $Q_j f$ onto the subspaces $W_j$ spanned by the $\psi_{j,n}$ represent the *detail* information of $f$ required to move from one resolution approximation subspace to the next finer one. Equation (1) is the wavelet decomposition of the function $f$. The *scaling* function $\phi$ satisfies the *two-scale* relation

$$\phi = \sum_n h_n \phi_{1,n} \,, \tag{2}$$

which is a discrete *low-pass filter* operation with the filter $\{h_n\}_{n\in\mathbf{Z}}$.

Now we start with a scale approximation $f^{j+1} = P_{j+1} f$ of a function $f$ in $V_{j+1}$ and decompose it into a coarser approximation in $V_j$. Due to the fact that $V_{j+1} = V_j \oplus W_j$, we have $f^{j+1} = f^j + \delta^j$, where $\delta^j = Q_j f$. In terms of the orthonormal bases $\{\phi_{j,n}\}_{n\in\mathbf{Z}}$ and $\{\psi_{j,n}\}_{n\in\mathbf{Z}}$, we have

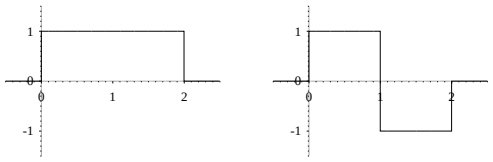
$$f^j = \sum_n c_n^j \phi_{j,n} \;\;, \quad \delta^j = \sum_n d_n^j \psi_{j,n} \;\;,$$

where the relation between the coefficients of the two levels of resolution is given by

$$c_n^{j-1} = \sum_k h_{k-2n}\, c_k^j \;\;, \quad d_n^{j-1} = \sum_k g_{k-2n}\, c_k^j \tag{3}$$

and $g_n = (-1)^n h_{1-n}$. $h$ and $g$ are the low-pass and high-pass filters, respectively. The decimation by a factor 2 corresponds to a down-sampling when going from one level to the next coarser one. This decomposition can be continued using the relation $V_{j+1} = V_j \oplus W_j$ and so on until a given level $J < j$, obtaining the following approximation for $f$:

$$f^{j+1} = \delta^j + \cdots + \delta^{J+1} + \delta^J + f^J$$




| $n$ | | 0 | 1 | 2 | 3 |
|---|---|---|---|---|---|
| Haar | $h_n$ | $\frac{1}{\sqrt{2}}$ | $\frac{1}{\sqrt{2}}$ | | |
| | $g_n$ | $\frac{1}{\sqrt{2}}$ | $-\frac{1}{\sqrt{2}}$ | | |
| Daub. | $h_n$ | $\frac{1+\sqrt{3}}{4\sqrt{2}}$ | $\frac{3+\sqrt{3}}{4\sqrt{2}}$ | $\frac{3-\sqrt{3}}{4\sqrt{2}}$ | $\frac{1-\sqrt{3}}{4\sqrt{2}}$ |
| | $g_n$ | $\frac{1-\sqrt{3}}{4\sqrt{2}}$ | $\frac{-3+\sqrt{3}}{4\sqrt{2}}$ | $\frac{3+\sqrt{3}}{4\sqrt{2}}$ | $\frac{-1-\sqrt{3}}{4\sqrt{2}}$ |

**Fig. 1.** The Haar scaling function, wavelet, and filter coefficients for Haar and Daubechies (4)

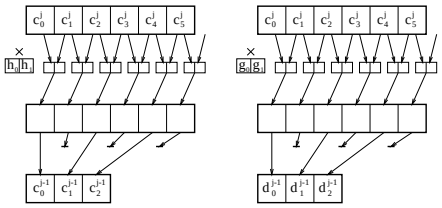


**Fig. 2.** Decomposition using Haar wavelets

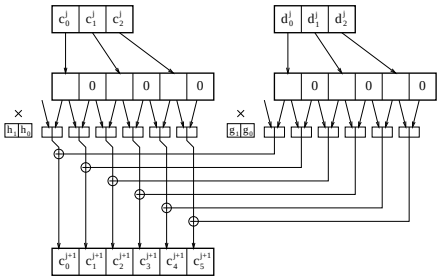


**Fig. 3.** Reconstruction using Haar wavelets

The inverse operation, the reconstruction of $f^{j+1}$ from $f^j$ and $\delta^j$, is simply given by:

$$c_k^{j+1} = \sum_n (h_{k-2n}\, c_n^j + g_{k-2n}\, d_n^j) \tag{4}$$

Now let us take a look at an example. The simplest possible wavelet is the *Haar* wavelet. Figure 1 depicts its scaling function and the mother wavelet together with the filter coefficients.

We will now decompose a set of coefficients $c_k^j$ into the $c_k^{j-1}$ of the next coarser level. In Figure 2 the decomposition process is explained. The input data are convolved with the filter kernels $h_n$ and $g_n$ and down-sampled by a factor of 2. This process can be continued with the low-pass filtered coefficients $c_k^{j-1}$, until only one coefficient is left.

In order to reconstruct the original signal, the low- and high-pass filtered coefficients are processed as shown in Figure 3. The coefficients are up-sampled and then convolved with the reverted filter kernels according to (4).

So far we have only dealt with one-dimensional data. For higher dimensions bases which are tensor products of the one-dimensional case are used. There exist other approaches for selecting orthogonal basis functions, but tensor product wavelets are easier to understand and faster to compute.

## 3 The Rendering Pipeline

As it can be directly derived from Equations (3) and (4), wavelet decomposition is practically done by an input signal filtering and a down-sampling step. Reconstruction on the other hand is performed by first up-sampling and filtering afterwards. Modern

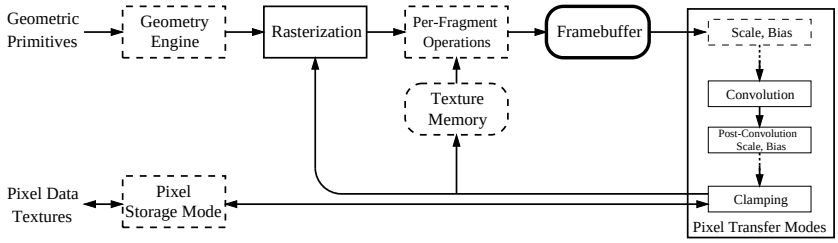



**Fig. 4.** The OpenGL graphics pipeline

graphics hardware supports filtering and scaling (resampling) for image transfer operations, which we will utilize for hardware based wavelet decomposition and reconstruction. The relevant part of the the OpenGL graphics pipeline is depicted in Figure 4.

In order to map the wavelet transformation onto the graphics hardware, we will use a mathematical specification of the graphics pipe. A more elaborated model has been derived in [4]. Let us consider the relevant parts of the graphics pipeline for image data. When a rectangular part of the frame buffer is to be copied from a source area, its color values are piped through the pixel transfer system and the rasterizer, before they are written to the destination area. Pixel transfer includes scaling and biasing of the color values, convolution with a prior defined filter kernel and clamping to the usual color value range $[0,1)$. The rasterizer transposes the input image to the designated destination area while zooming it with arbitrary zoom factors, in other words, it performs up- and down-sampling.

Now let $p^{n+1}$ be the pixel data that results from a graphical operation on $p^n$. For simplification we will assume that $p^n$ is one-dimensional. A first approximation of the relevant part of the graphics pipeline can be written as a composition of a convolution (co), a clamping step (cl), a transposition (tr), and the scaling step (sc):

$$p^{n+1} = \mathrm{sc} \circ \mathrm{tr} \circ \mathrm{cl} \circ \mathrm{co}\ (p^n) \tag{5}$$

$$\mathrm{sc}(p_i) = p_{\lfloor zi \rfloor} \tag{6}$$

$$\mathrm{tr}(p_i) = p_{i - x_s + x_d} \tag{7}$$

$$\mathrm{cl}(p_i) = \max(0, \min(1, p_i)) \tag{8}$$

$$\mathrm{co}(p_i) = s \cdot \sum_{j=0}^{m} k_j\, p_{i+j}\ + b\ , \tag{9}$$

with zoom $z$, source $x_s$ and destination $x_d$ position, scaling $s$, and bias $b$ parameters, and with a convolution kernel $k$ of size $m$. As explained above, (co) and (cl) are performed in the pixel transfer system, while (tr) and (sc) describe the task of the rasterizer.

These equations are applied to pixels $p_i^{n+1}$ of the destination area $i \in [x_d, (x_d + w + 1 - m) \cdot z)$, with $w$ being the image size. The remaining pixels stick to their old values, that is, they are equal to $p_i^n$.

As we now have a mathematical model of the rendering pipeline, we can address the problem of mapping wavelet transformations onto the hardware as the next logical step.

## 4 Hardware Based Decomposition

Compared to the order of operations in the graphics pipeline, of which the relevant part is depicted in Figure 4, wavelet decomposition fits neatly into its scheme. Remembering that scaling is a part of the rasterization process, convolution is performed in the graphics pipe just before image scaling.

When we write the wavelet decomposition (3) as

$$\check{c}_n^{j-1} = \sum_i h_i c_{n+i}^j \,, \quad \check{d}_n^{j-1} = \sum_i g_i c_{n+i}^j \,, \tag{10}$$

$$c_n^j = \check{c}_{2n}^j \,, \quad d_n^j \;=\; \check{d}_{2n}^j \tag{11}$$

and compare it to Equations (5) to (9), we see that each of the wavelet decomposition filter steps matches the calculations of the OpenGL graphics pipe perfectly, except for the clamping steps. Clamping introduces several problems to these algorithms, that have to be addressed by using arbitrary scale and bias parameters. This aspect is discussed in detail in [4]. (6) implements the down-scaling in (11) and (10) can be expressed with the convolution filters (9).

One thing to note is that the image data $p_j^n$ as well as the filter kernel $k_j$ are only defined for $j \geq 0$. The filter kernel size is further limited by hardware specific constants, which are rather small. Thus it is necessary to displace the filter kernel and the input and output image specifications before invocation. Of course, the displacement has to be compensated in the final convolution step.

The input data have to be convolved using two different filters, so either the resulting images have to be written to another part of the frame buffer, just like in our earlier approach, or they have to be done together in one step. Now remember that we are actually dealing with 2D images. When we combine both tensor product steps with the two different filters, we get a total of four filters that have to be applied to the data.

As the graphics pipeline works on RGBA images nevertheless, it seems to be straightforward to use RGBA convolution filters instead of luminance only filters to combine these four steps into one as depicted in Figure 5. This will speed up the decomposition significantly, as the raster manager needs to address only one fourth of the number of pixels of the previous mentioned approach, and the convolution pipeline is implemented for color filters anyway. Additionally, we do not have to copy the source image in order to save it for the second filter, which makes for another factor of two.

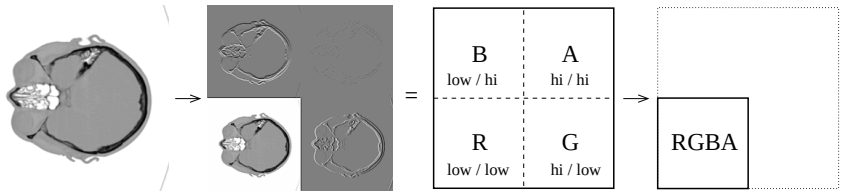



**Fig. 5.** Using one RGBA convolution instead of four different luminance only convolutions

Create convolution filter: $\tilde{h}_j = h_{j+\alpha}$ , $\tilde{g}_j = g_{j+\alpha}$ ,
$\mathbf{f}^R_{j,k} = \tilde{h}_j \cdot \tilde{h}_k$ , $\mathbf{f}^G_{j,k} = \tilde{g}_j \cdot \tilde{h}_k$ , $\mathbf{f}^B_{j,k} = \tilde{h}_j \cdot \tilde{g}_k$ , $\mathbf{f}^A_{j,k} = \tilde{g}_j \cdot \tilde{g}_k$ $\forall j,k$ .
Calculate scaling **s** and bias **b** . Set post-convolution scaling to **s** .
Set post-convolution bias to **b** .
Set pixel zoom to $1.0 \times 1.0$ . Set color matrix to $\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$ .
Copy area $[\delta_x + \alpha + i_x\ ,\ \delta_x + \alpha + i_x + w_x + \Delta - 1) \times [\delta_y + \alpha + i_y\ ,\ \delta_y + \alpha + i_y + w_y + \Delta - 1)$ to $[o_x\ ,\ o_x + w_x + \Delta - 1) \times [o_y\ ,\ o_y + w_y + \Delta - 1)$.
Set pixel zoom to $0.5 \times 0.5$ . Disable color matrix.
Copy area $[o_x\ ,\ o_x + w_x + \Delta - 1) \times [o_y\ ,\ o_y + w_y + \Delta - 1)$ to $[o_x\ ,\ o_x + \frac{1}{2}w_x) \times [o_y\ ,\ o_y + \frac{1}{2}w_y)$, using convolution filter **f** (size $\Delta^2$).

| | |
|---|---|
| $h_j, g_j$ | Low- and high-pass filters, respectively |
| $\alpha$ | Index of first non-zero element of both filters |
| $\Delta$ | Size of filters |
| $\delta$ | Shift offset (see text) |
| $i, w$ | Input image offset and size |
| $o$ | Output image offsets |

**Fig. 6.** Implementation sequence for wavelet decomposition in hardware

However, it turns out that we still have to copy the source image, because OpenGL does not provide a pre-convolution color matrix, which would be necessary to provide the same information to the four different filters. As we want to address only the low-pass filtered data of the previous step, which is stored in the red component of the calculated image, we have to spread this information to all four color channels using SGI's color matrix OpenGL extension before invoking the convolution filter. Still, we have the advantage of better utilization of the graphics pipe.

Unfortunately, OpenGL is no pixel exact specification. In particular, zooming is only well defined according to (6) for up-sampling, that is for zoom factors greater than one. When images are scaled down, it is up to the implementation which pixels to transfer. We have found that even the implementations of one vendor — Silicon Graphics in our case — vary from architecture to architecture. In order to address this problem, a so-called *shift offset* $\delta$ is determined. When added to the specification of the source image's left edge, it corrects the internal pixel offset. Currently the only way to determine the shift offset is to draw a scaled-down version of a well-known image for several different shift values and to read it back afterwards for comparison with the desired result.

Additionally, care has to be taken at the borders of the input image. Several strategies have already been discussed, with blanking being the easiest and input mirroring being one of the best methods in order to suppress high frequencies that are not part of the image, but introduced by aliasing effects.

Finally, Figure 6 shows the implementation sequence for wavelet decomposition using graphics hardware. The calculation of the scaling and bias values, which is left out here for clarity, is discussed in detail for the one dimensional case in [4].

## 5 Hardware Based Reconstruction

In contrast to the decomposition algorithm, wavelet reconstruction is much more complicated, because according to Equation (4) scaling and convolution is to be performed in inverse order compared to the rendering pipeline (Figure 4). Either scaling and convolution have to be performed in separate rendering steps, or the filters have to be split and special care has to be taken in order to render even and odd pixel positions separately. Either way, reconstruction is more complicated than decomposition.

Moreover, due to different scaling and bias values for odd and even pixels, using separate rendering steps is not a feasible option. Therefore, we will concentrate on the second possibility of splitting the filters.

Now we examine the wavelet reconstruction (4). In order to simplify the expression, we have to distinguish between $k$ being even and odd. For even $k$ we substitute $h_{k-2n}$ using $h_n^{\text{ev}} = h_{-2n}$ ($g$ accordingly) and get

$$\bar{c}_n^{j+1} = \sum_i (h_i^{\text{ev}} c_{i+n}^j + g_i^{\text{ev}} d_{i+n}^j) \,, \tag{12}$$

$$c_k^{j+1} = c_{2n}^{j+1} \;=\; \bar{c}_n^{j+1} \,. \tag{13}$$

For odd $k$ we use $h^{\text{od}} = h_{1-2n}$, which results in

$$\hat{c}_n^{j+1} = \sum_i (h_i^{\text{od}} c_{i+n}^j + g_i^{\text{od}} d_{i+n}^j) \,, \tag{14}$$

$$c_k^{j+1} = c_{2n+1}^{j+1} \;=\; \hat{c}_n^{j+1} \,. \tag{15}$$

Again, we will concentrate on the low pass filtered data first and simply neglect $g$ in the terms above. We can see that (13) and (15) can be performed by setting according zoom factors in (6). (12) and (14) can be implemented in (9) by choosing $h^{\text{ev}}$ and $h^{\text{od}}$ as filter kernels, respectively.

Of course, when rendering the odd coefficients, we have to make sure that we do not overwrite the previously rendered even coefficients. OpenGL knows about a so-called *stencil* buffer, which provides masking tests in the per-fragment operation part of the graphics pipeline. The stencil buffer has to be initialized with a striped pattern only once, after that the stencil test can be set to render even or odd pixels only. We activate the test for rendering odd pixels only due to speed reasons, as each activated test can slow down the rendering process.

Up to now we have only dealt with the low-pass filtered coefficients $c_n^j$. As we have the necessary hi-pass filtered coefficients $d_n^j$ stored as another component of the same pixels, we can use SGI's color matrix extension to combine them. Again, we use all four red, green, blue, and alpha components in order to work on 2D tensor product wavelets in one step. This is different to our previous approach, where we treated the different coefficients in separate steps. The new approach is not only faster, but even more accurate, because color matrix operations are performed with higher precision than blending operations in the frame buffer, and we do not have to deal with clamping artifacts in this case either. We disable rendering to the green, blue, and alpha channels in order to not overwrite the hi-pass filtered coefficients there, which will be needed in the next reconstruction step.

Create convolution filters:

$\tilde{h}_j^{\mathrm{ev}} = h_{2\lfloor \frac{\alpha_h+\Delta_h}{2} \rfloor - 2j}$ , $\tilde{h}_j^{\mathrm{od}} = h_{2\lceil \frac{\alpha_h+\Delta_h}{2} \rceil - 2j+1}$ , $\tilde{g}_j^{\mathrm{ev}} = g_{2\lfloor \frac{\alpha_g+\Delta_g}{2} \rfloor - 2j}$ , $\tilde{g}_j^{\mathrm{od}} = g_{2\lceil \frac{\alpha_g+\Delta_g}{2} \rceil - 2j+1}$ .

$\mathbf{f}_{j,k}^{x,y,R} = \tilde{h}_j^x \cdot \tilde{h}_k^y$ , $\mathbf{f}_{j,k}^{x,y,G} = \tilde{g}_j^x \cdot \tilde{h}_k^y$ , $\mathbf{f}_{j,k}^{x,y,B} = \tilde{h}_j^x \cdot \tilde{g}_k^y$ , $\mathbf{f}_{j,k}^{x,y,A} = \tilde{g}_j^x \cdot \tilde{g}_k^y \quad \forall j,k \,, \forall x,y \in \{\mathrm{ev}, \mathrm{od}\}$ .

$\delta^{\mathrm{ev}} = -\lfloor \frac{\alpha+\Delta-1}{2} \rfloor$ , $\delta^{\mathrm{od}} = 1 - \lceil \frac{\alpha+\Delta-1}{2} \rceil$ , $\Delta^{\mathrm{ev}} = -\delta^{\mathrm{ev}} - \lceil \frac{\alpha}{2} \rceil + 1$ , $\Delta^{\mathrm{od}} = -\delta^{\mathrm{od}} - \lfloor \frac{\alpha}{2} \rfloor + 1$ .

Calculate scaling $\mathbf{s}$ and bias $\mathbf{b}^{x,y}$, $x,y \in \{\mathrm{ev}, \mathrm{od}\}$ .

Set pixel zoom to $2.0 \times 2.0$ . Enable rendering to R only, disable rendering to G, B, and A.

Set color matrix to $\begin{pmatrix} 1&0&0&0 \\ 1&0&0&0 \\ 1&0&0&0 \\ 1&0&0&0 \end{pmatrix}$ . Initialize stencil buffer with $\begin{cases} 0 & x \text{ even}, y \text{ even} \\ 1 & x \text{ odd}, y \text{ even} \\ 2 & x \text{ even}, y \text{ odd} \\ 3 & x \text{ odd}, y \text{ odd} \end{cases}$ .

Disable stencil test. Set post-convolution scaling and bias to $\bar{\mathbf{s}}$ and $\bar{\mathbf{b}}^{\mathrm{ev,ev}}$ .

Copy area $[i_x + \delta^{\mathrm{ev}} \,,\, i_x + \delta^{\mathrm{ev}} + w_x + \Delta^{\mathrm{ev}} - 1) \times [i_y + \delta^{\mathrm{ev}} \,,\, i_y + \delta^{\mathrm{ev}} + w_y + \Delta^{\mathrm{ev}} - 1)$ to $[o_x \,,\, o_x + \frac{1}{2} w_x) \times [o_y \,,\, o_y + \frac{1}{2} w_y)$, using convolution filter $\mathbf{f}^{\mathrm{ev,ev}}$ (size $\Delta^{\mathrm{ev}} \times \Delta^{\mathrm{ev}}$) .

Do $\forall x,y \in \{\mathrm{ev}, \mathrm{od}\}$ :

Enable stencil test, render only pixels with stencil value $\begin{cases} 1 & x = \mathrm{od}, y = \mathrm{ev} \\ 2 & x = \mathrm{ev}, y = \mathrm{od} \\ 3 & x = \mathrm{od}, y = \mathrm{od} \end{cases}$ .

Set post-convolution bias to $\bar{\mathbf{b}}^{x,y}$ .

Copy area $[i_x + \delta^x \,,\, i_x + \delta^x + w_x + \Delta^x - 1) \times [i_y + \delta^y \,,\, i_y + \delta^y + w_y + \Delta^y - 1)$

to $[o_x \,,\, o_x + \frac{1}{2} w_x) \times [o_y \,,\, o_y + \frac{1}{2} w_y)$, using convolution filter $\mathbf{f}^{x,y}$ (size $\Delta^x \times \Delta^y$) .

| | |
|---|---|
| $h_j$, $g_j$ | Low- and high-pass filters, respectively |
| $\alpha$ | Index of first non-zero element of both filters |
| $\Delta$ | Size of both filters |
| $o_c$, $o_d$, $w$ | Input image offsets and size |
| $o_o$ | Output image offset |

**Fig. 7.** Implementation sequence for wavelet reconstruction in hardware

As we are up-sampling during reconstruction, we do not have to care about any shift offsets during zooming, as the OpenGL specification is pixel exact in this case. However, we have to care about the fact that hardware filter kernels $h_k$ are only to be specified for non-negative $k$. Together with the problem of odd sized filter kernels this leads to quite horrible filter kernel specifications, which can be noted in the implementation sequence in Figure 7. Again, the scaling and bias values that have to be computed here have been discussed in detail in our previous paper. Care has to be taken about image borders as well. The policy here depends heavily on the policy taken during the decomposition step. Note that Haar wavelets are quite uncomplicated, as the reconstruction filters have the size 1, which is a mere scaling.

## 6 Results

Table 1 reveals that Hardware based wavelet filtering is much faster than a well tuned software implementation. Only for very small images the software system outperforms the OpenGL hardware. Scaling and bias computation as well as filter kernel download adds an almost constant overhead which unsurprisingly leads to bad times for small images. On the other hand, performance analysis shows that the filter operations of current

| | Haar wavelet | | | | | Daubechies (4) wavelet | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Size | $32^2$ | $64^2$ | $128^2$ | $256^2$ | $512^2$ | $32^2$ | $64^2$ | $128^2$ | $256^2$ | $512^2$ |
| Software decomp. | 0.50 | 2.0 | 7.8 | 31 | 150 | 0.70 | 2.8 | 11 | 45 | 209 |
| Hardware decomp. | 0.65 | 1.4 | 4.5 | 16 | 62 | 0.70 | 1.8 | 5.5 | 19 | 74 |
| Factor | 0.77 | 1.4 | 1.7 | 1.9 | 2.4 | 1.0 | 1.6 | 2.0 | 2.4 | 2.8 |
| Software recons. | 0.80 | 3.6 | 14 | 55 | 240 | 1.2 | 5.0 | 19 | 78 | 340 |
| Hardware recons. | 1.4 | 2.0 | 5.0 | 18 | 66 | 1.4 | 2.0 | 5.1 | 18 | 66 |
| Factor | 0.57 | 1.8 | 2.8 | 3.1 | 3.6 | 0.86 | 2.5 | 3.7 | 4.3 | 5.2 |

**Table 1.** Filter times in ms per 2D wavelet step

graphics hardware are still not optimized and in the future much higher throughput can be expected.

All times have been measured on a Silicon Graphics Octane with R10000 195MHz processor and a MXE graphics pipe. We will add performance figures for the Intergraph Wildcat as well as soon as possible.

As hardware based wavelet filtering uses the frame buffer for its computations, which has only a limited depth, the accuracy of the computations cannot be as good as with software based techniques, which in contrast only have to tolerate the typically small floating point errors. On the other hand, when using a frame buffer with a depth of 12 bits per base color, only single bit errors can be found in images of size $512^2$ after complete wavelet decomposition and reconstruction, as it can be seen on the color plate in Figures 8 to 11. Note that the difference images have been enhanced so that one bit differences are visible.

On the other hand, frame buffers with only eight bits per base color yield less pleasing results. Figures 12 to 13 reveal the differences after complete decomposition and reconstruction. Again, the last image has been enhanced in order to reveal the differences. The maximum absolute difference between the original image and the wavelet decomposed image is 13, that is about 5% of the total 8 bit color range.

## 7 Conclusion

We have introduced a wavelet decomposition and reconstruction algorithm, that directly works on the graphics hardware of modern OpenGL capable workstations and accelerates the time consuming filtering steps a lot. By using the convolution and color matrix extensions together with OpenGL's facilities to scale images during copy instructions, we are able to perform all necessary steps of 2D tensor product wavelet filtering without copying data from or to the machine's main memory, thus avoiding typical bottlenecks in the visualization cycle. Different possibilities to use hardware based wavelets for enhanced feature detection are currently subject of further investigations.

Using the frame buffer for mathematical operations is usually problematic in terms of accuracy [11] due to the limited depth of the frame buffer. However, wavelet decom-

position and reconstruction have proven to be relatively robust. Only single-bit differences between software and hardware decomposed data can be detected when rendering intermediate images to 12 bit accurate frame buffers.

## 8 Acknowledgments

We would like to thank our colleague Rüdiger Westermann for his helpful discussion regarding wavelet basis and hardware implementation issues. Additionally, we would like to thank our former colleague Christoph Lürig for giving us some ideas about how to accelerate hardware based wavelet transformations even more.

## References

1. C. K. Chui. *An Introduction to Wavelets*. Academic Press, Inc., San Diego, 1992.
2. I. Daubechies. *Ten Lectures on Wavelets*. Number 61 in CBMS-NSF Series in Applied Mathematics. SIAM, Philadelphia, 1992.
3. M. Hopf and T. Ertl. Accelerating 3D Convolution using Graphics Hardware. In D. Ebert, M. Gross, and B. Hamann, editors, *Visualization '99*, pages 471–474, San Francisco, CA, 1999. IEEE Computer Society, IEEE Computer Society Press.
4. M. Hopf and T. Ertl. Hardware Based Wavelet Transformations. In B. Girod, H. Niemann, and H.-P. Seidel, editors, *Vision, Modeling, and Visualization '99*, pages 317–328, Erlangen, Germany, November 1999. SFB 603, Graduate Research Center, IEEE, and GI, Infix Press.
5. L. Lippert, M. H. Gross, and C. Kurmann. Compression Domain Volume Rendering for Distributed Environments. In D. Fellner and L. Szirmay-Kalos, editors, *EUROGRAPHICS '97*, volume 14, pages C95–C107. Eurographics Association, Blackwell Publishers, 1997.
6. A. K. Louis, P. Maass, and A. Rieder. *Wavelets*. B. G. Teubner Stuttgart, Germany, 1994.
7. C. Lürig, R. Grosso, and T. Ertl. Combining Wavelet Transform and Graph Theory for Feature Extraction and Visualization. In *Proc. 8th Eurographics Workshop on Visualization in Scientific Computing*, pages 137–144. Eurographics Association, 1997.
8. T. Malzbender. Fourier-Volume-Rendering. *ACM Transactions on Graphics*, 12(3):233–250, July 1993.
9. SGI. *OpenGL on Silicon Graphics Systems*. Silicon Graphics Inc., Mountain View, California, 1996.
10. G. Strang and T. Nguyen. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, Wellesley, Massachusetts, 1996.
11. C. Teitzel, M. Hopf, R. Grosso, and T. Ertl. Volume Visualization on Sparse Grids. Technical Report 8/1998, Universität Erlangen-Nürnberg, Lehrstuhl für Graphische Datenverarbeitung (IMMD IX), Erlangen, July 1998. Accepted for publication in *Computing and Visualization in Science*, Springer-Verlag, Heidelberg.
12. T. Totsuka and M. Levoy. Frequency Domain Volume Rendering. *Computer Graphics*, 27(4):271–78, August 1993.
13. R. Westermann and T. Ertl. A Multiscale Approach to Integrated Volume Segmentation and Rendering. In *Computer Graphics Forum 16(3) (Proc. EUROGRAPHICS '97)*, pages 117–129. Blackwell, 1997.
14. M. V. Wickerhauser. *Adapted Wavelet Analysis from Theory to Software*. IEEE Press, New York, 1994.

**Fig. 8.** The head data set

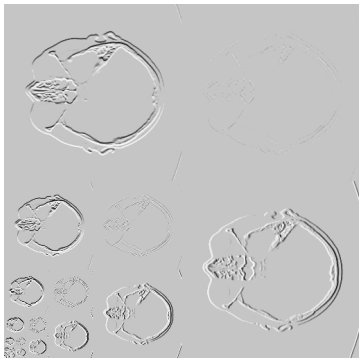


**Fig. 9.** Haar wavelet decomposition

**Fig. 10.** 1-bit differences after full Haar decomposition and reconstruction using a frame buffer with 12 bits per color

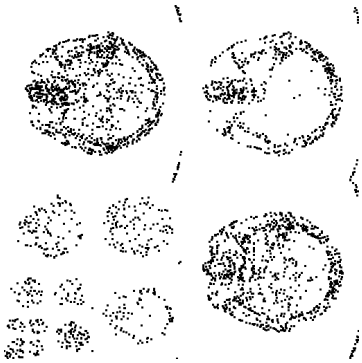


**Fig. 11.** 1-bit differences between software and hardware Haar decomposition using a frame buffer with 12 bits per color

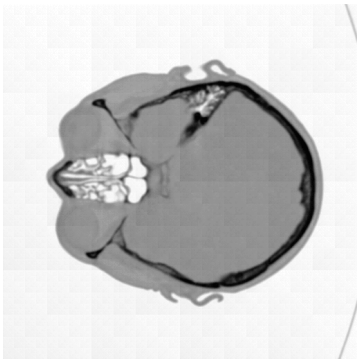


**Fig. 12.** Reconstructed image using a frame buffer with 8 bits per color

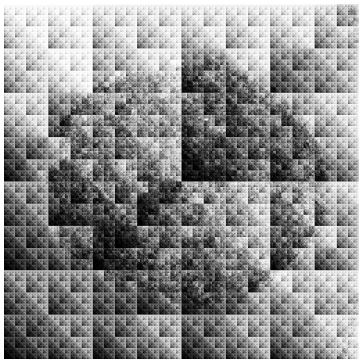


**Fig. 13.** Enhanced differences after full Haar decomposition and reconstruction using a frame buffer with 8 bits per color