

# Hardware Based Wavelet Transformations

Matthias Hopf

Thomas Ertl

{hopf,ertl}@informatik.uni-erlangen.de

Computer Graphics Group, University of Erlangen-Nürnberg

## Abstract

Many filtering and feature extraction algorithms use wavelet or related multiscale representations of volume data for edge detection and processing. Due to the computational complexity of these approaches no interactive visualization of the extraction process is possible nowadays. Using the hardware of modern graphics workstations for wavelet decomposition and reconstruction is 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 [6, 14]. 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 [4] using the Fourier rendering approach [7, 13].

Wavelet decomposition and reconstruction is usually implemented by applying multiple convolution and down- / up-sampling steps to the volume data. The convolution steps will not scale with new computer hardware as well as pure computational problems, as they are already mainly memory-bound. When using typical tensor-product wavelets the complete volume data has to be accessed three times for each wavelet filtering step.

Additionally, when visualizing three-dimensional data, the volume has regularly to be downloaded in form of a regular grid to the graphics hardware after filtering to visualize

it interactively [8, 10, 15]. Here, the data set is stored for texture based volume rendering in special texture memory, which can be addressed by the graphics pipe very fast. The loading process itself is relatively slow, taking several seconds for big data sets even on the fastest available graphics workstations. As the data set has to be reloaded after a filter operation has been performed in software, interactive filtering will benefit a lot from algorithms that directly work on the graphics hardware.

On the other hand, modern graphics hardware of several vendors, as for instance Silicon Graphics [9], has support for two dimensional convolution. Three dimensional convolution with separable filter kernels can be implemented by using these hardware supported convolution filters along with volume textures [3]. Together with the ability to scale bitmaps by arbitrary factors all necessary steps needed for wavelet decomposition and reconstruction are available.

The memory interface of graphics hardware is usually designed in a much more parallel way in high end systems compared to the interface of the CPU. Additionally, the volume data do not have to be downloaded for visualization, when the wavelet analysis can be done completely in hardware, as it already resides in the volume texture memory of the graphics system. However, there are still several pitfalls to be circumvented, which we will address in more detail in the Sections 4 to 6.

## 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 [11, 16]. 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 section gives a short introduction into the basics of wavelet theory. Details on the theory can be found in [1, 2, 5].

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  $L^2(\mathbf{R})$

$$P_j f = P_{j-1} f + Q_{j-1} f, \quad (1)$$

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

$$\begin{aligned} P_j f &= \sum_{n \in \mathbf{Z}} \langle f, \phi_{j,n} \rangle \phi_{j,n} \\ Q_j f &= \sum_{n \in \mathbf{Z}} \langle f, \psi_{j,n} \rangle \psi_{j,n}. \end{aligned}$$

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

$$\phi = \sum_n h_n \phi_{1,n}, \quad (2)$$

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

$$\begin{aligned} f^j &= \sum_n c_n^j \phi_{j,n}, \\ \delta^j &= \sum_n d_n^j \psi_{j,n}, \end{aligned}$$

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

$$\begin{aligned} c_n^{j-1} &= \sum_k h_{k-2n} c_k^j, \\ d_n^{j-1} &= \sum_k g_{k-2n} c_k^j \end{aligned} \quad (3)$$

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  $f$ :

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

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

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

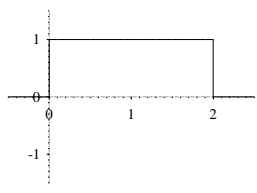


Figure 1: The Haar scaling function

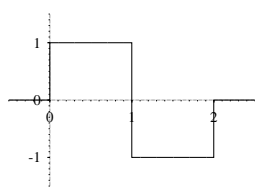


Figure 2: The Haar wavelet

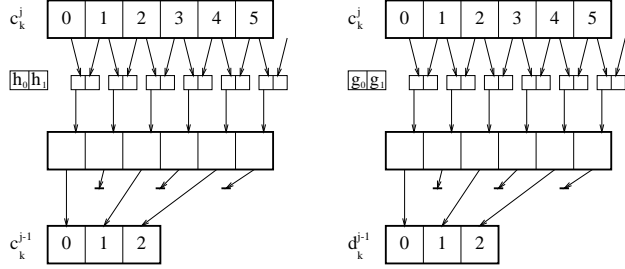


Figure 3: Decomposition using Haar wavelets

Now let us take a look at an example. The simplest possible wavelet is the *Haar* wavelet. Figures 1 and 2 depict the scaling function and the mother wavelet, respectively. The filter coefficients for the Haar wavelets are

$$h_n = \begin{cases} \frac{1}{\sqrt{2}} & n = 0, 1 \\ 0 & \text{otherwise} \end{cases},$$

$$g_n = \begin{cases} \frac{1}{\sqrt{2}} & n = 0 \\ -\frac{1}{\sqrt{2}} & n = 1 \\ 0 & \text{otherwise} \end{cases}.$$

We will now decompose a set of coefficients  $c_k^j$  into the  $c_k^{j-1}$  of the next coarser level. In Figure 3 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 4. 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

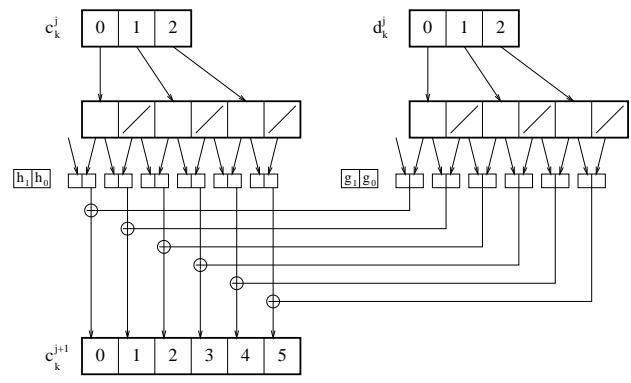


Figure 4: Reconstruction using Haar wavelets



Figure 5: Two-dimensional wavelet decomposition using tensor product wavelets

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.

Figure 5 reveals, how a two-dimensional wavelet decomposition is performed. First, the data are decomposed line by line in the direction of the x-axis. Both the low-pass and the high-pass filtered data are stored side by side to each other, so they can be filtered in a second step column by column in the direction of the y-axis. Afterwards, the lower left part of the final figure reveals the two-dimensional low-pass filtered data, the upper left and lower right parts contain the perpendicular to one of the axes high-pass filtered data, and the upper right area shows the completely high-pass filtered coefficients.

### 3 The Rendering Pipeline

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

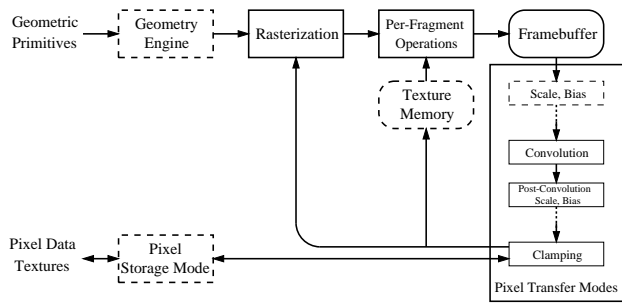


Figure 6: The OpenGL graphics pipeline

hand is performed by first up-sampling and filtering afterwards. Modern 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 6.

In order to simplify the description of the process we will show the process for a one dimensional wavelet transformation. As we have seen in the previous chapter, tensor product based multi-dimensional wavelet transformations are a straight-forward extension to this approach. In particular, [3] covers the details of how to employ 3D texture hardware in order to perform three dimensional convolution with separable filter kernels, which can be easily extended to cope with wavelet transformations.

Now let us consider how the graphics pipeline works on 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, the rasterizer and the per-fragment operation system 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. In the final per-fragment operations step, the resulting pixel values are blended with the pixel values of the destination area using several pre-defined blending functions. This step includes a final clamping

step as well.

In order to map the wavelet transformation onto the graphics hardware, we need a mathematical specification of the pixel operations of the graphics pipe. Let  $p^{n+1}$  be the pixel data that results from a graphical operation on  $p^n$ . Again, 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), two clamping steps (cl), a transposition (tr), the scaling step (sc), and a blending operation (bl):

$$p^{n+1} = \text{cl} \circ \text{bl} \circ \text{sc} \circ \text{tr} \circ \text{cl} \circ \text{co} (p^n) \quad (5)$$

$$\text{bl}(p_i) = \Gamma(p_i, p_i^n) \quad (6)$$

$$\text{sc}(p_i) = p_{[zi]} \quad (7)$$

$$\text{tr}(p_i) = p_{i-x_s+x_d} \quad (8)$$

$$\text{cl}(p_i) = \max(0, \min(1, p_i)) \quad (9)$$

$$\text{co}(p_i) = s \cdot \sum_{j=0}^m k_j p_{i+j} + b, \quad (10)$$

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, (tr) and (sc) describe the task of the rasterizer, and (bl) and (cl) illustrate the per-fragment operations.

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  $p_i^n$ . The blending function  $\Gamma$  can be chosen from a predefined set. For wavelet filter operations we use identity  $\Gamma_{\text{id}}(x, y) = x$ , addition  $\Gamma_{\text{add}}(x, y) = x + y$  and subtraction  $\Gamma_{\text{sub}}(x, y) = y - x$ , which are available as extensions to the OpenGL standard.

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 6, wavelet decomposition fits neatly into its scheme. Remembering that scaling is performed as 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

$$\check{c}_n^{j-1} = \sum_i h_i c_{n+i}^j, \quad (11)$$

$$\check{d}_n^{j-1} = \sum_i g_i c_{n+i}^j, \quad (12)$$

$$c_n^j = \check{c}_{2n}^j, \quad d_n^j = \check{d}_{2n}^j \quad (13)$$

and compare it to Equations (5) to (10), it is easy to see that each of the wavelet decomposition filter steps matches the calculations of the OpenGL graphics pipe perfectly, except for the clamping steps. We will address this problem in Section 6. (7) implements the down-scaling in (13), (11) and (12) can be expressed with the convolution filters (10) and an identity blending operation  $\Gamma_{\text{id}}$  in (6).

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.

Because the input data have to be convolved using two different filters, the resulting images have to be written to another part of the frame buffer so that the original data set is not overwritten. These two parts of the frame buffer can be used alternately when tensor product wavelet decompositions have to be computed.

Unfortunately, OpenGL is no pixel exact specification. In particular, zooming is only well defined according to (7) for up-sampling, that is

Create convolution filters:  $\check{h}_i = h_{i+\alpha_h}$ ,  $\check{g}_i = g_{i+\alpha_g}$ .

Set pixel zoom to 0.5.

Set blending function to  $\Gamma_{\text{id}}$ .

$$\begin{aligned} \check{h}_{\text{neg}} &= \sum_k \min(0, h_k), & \check{h}_{\text{pos}} &= \sum_k \max(0, h_k), \\ \check{g}_{\text{neg}} &= \sum_k \min(0, g_k), & \check{g}_{\text{pos}} &= \sum_k \max(0, g_k). \end{aligned}$$

$$s_h = \frac{1}{\check{h}_{\text{pos}} - \check{h}_{\text{neg}}}, \quad s_g = \frac{1}{\check{g}_{\text{pos}} - \check{g}_{\text{neg}}}.$$

Set post-convolution scaling to  $s_h$ .

Set post-convolution bias to  $b_h = -\check{h}_{\text{neg}} \cdot s_h$ .

Copy area  $[\delta + \alpha_h + o_i, \delta + \alpha_h + o_i + w + \Delta_h - 1]$  to  $[o_c, o_c + \frac{1}{2}w)$ , using convolution filter  $\check{h}$  (size  $\Delta_h$ ).

Set post-convolution scaling to  $s_g$ .

Set post-convolution bias to  $b_g = -\check{g}_{\text{neg}} \cdot s_g$ .

Copy area  $[\delta + \alpha_g + o_i, \delta + \alpha_g + o_i + w + \Delta_g - 1]$  to  $[o_d, o_d + \frac{1}{2}w)$ , using convolution filter  $\check{g}$  (size  $\Delta_g$ ).

---

$h_j, g_j$	Low- and high-pass filters, respectively
$\alpha_x$	Index of first non-zero element of filter $x$
$\Delta_x$	Size of filter $x$
$\delta$	Shift offset (see text)
$o_i, w$	Input image offset and size
$o_c, o_d$	Output image offsets

Figure 7: Implementation sequence for wavelet decomposition in hardware

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 7 shows the implementation sequence for wavelet decomposition using graphics hardware. It also computes the scal-

ing and bias values that are discussed in detail in Section 6.

## 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 6). 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 expensive than decomposition.

Moreover, we will discover in Section 6, that 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

$$\bar{c}_n^{j+1} = \sum_i (h_i^{\text{ev}} c_{i+n}^j + g_i^{\text{ev}} d_{i+n}^j), \quad (14)$$

$$c_k^{j+1} = c_{2n}^{j+1} = \bar{c}_n^{j+1}. \quad (15)$$

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

$$\hat{c}_n^{j+1} = \sum_i (h_i^{\text{od}} c_{i+n}^j + g_i^{\text{od}} d_{i+n}^j), \quad (16)$$

$$c_k^{j+1} = c_{2n+1}^{j+1} = \hat{c}_n^{j+1}. \quad (17)$$

Again, we will concentrate on the low pass filtered data first and simply neglect  $g$  in the terms above. We can see that (15) and (17) can be performed by setting according zoom factors in (7). (14) and (16) can be implemented in (10) by choosing  $h^{\text{ev}}$  and  $h^{\text{od}}$  as filter kernels, respectively. The blending function is set to  $\Gamma_{\text{id}}$  for this step, just as in the decomposition mechanism.

Of course, when rendering the odd coefficients, we have to make sure that we do not overwrite the previously rendered even coefficients. OpenGL provides the 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  $c_n^j$ . We now have to add the convoluted  $d_n^j$  to the values that already reside in the frame buffer. Therefore, we perform another rendering step in which we copy the high pass filtered coefficients with the convolution kernels  $g^{\text{ev}}$  and  $g^{\text{od}}$  just over the previously low-pass convolved coefficients. This time, we select  $\Gamma_{\text{add}}$  as the blending function, by which the rendered data are added to the values in the frame buffer rather than overwriting them.

Unfortunately, the second clamping step in (5) prohibits values  $< 0$  to be correctly subtracted from the frame buffer. Therefore, the same convolution has to be rendered twice, one time using the scale and bias values discussed in the next section and  $\Gamma_{\text{add}}$  as blending function, one time using the negated scale and bias values, using  $\Gamma_{\text{sub}}$  for blending.

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 8. The scaling and bias values that are computed here are discussed in detail in the next section. Again, 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 here, as the reconstruc-

Create convolution filters:

$$\begin{aligned}\tilde{h}_i^{\text{ev}} &= h_{2\lfloor \frac{\alpha_h + \Delta_h}{2} \rfloor - 2i}, & \tilde{h}_i^{\text{od}} &= h_{2\lceil \frac{\alpha_h + \Delta_h}{2} \rceil - 2i + 1}, \\ \tilde{g}_i^{\text{ev}} &= g_{2\lfloor \frac{\alpha_g + \Delta_g}{2} \rfloor - 2i}, & \tilde{g}_i^{\text{od}} &= g_{2\lceil \frac{\alpha_g + \Delta_g}{2} \rceil - 2i + 1}.\end{aligned}$$

Set pixel zoom to 2.0.

Initialize stencil buffer with  $\begin{cases} 0 & \text{even pixels} \\ 1 & \text{odd pixels} \end{cases}$ .

$$\begin{aligned}\tilde{h}_{\text{neg}} &= \sum_k \min(0, h_k), & \tilde{h}_{\text{pos}} &= \sum_k \max(0, h_k), \\ \tilde{h}_{\text{ev}} &= \sum_k h_{2k}, & \tilde{h}_{\text{od}} &= \sum_k h_{2k+1}, \\ \tilde{g}_{\text{neg}} &= \sum_k \min(0, g_k), & \tilde{g}_{\text{pos}} &= \sum_k \max(0, g_k), \\ \tilde{g}_{\text{ev}} &= \sum_k g_{2k}, & \tilde{g}_{\text{od}} &= \sum_k g_{2k+1}, \\ \delta_h^{\text{ev}} &= -\lfloor \frac{\alpha_h + \Delta_h - 1}{2} \rfloor, & \delta_h^{\text{od}} &= 1 - \lceil \frac{\alpha_h + \Delta_h - 1}{2} \rceil, \\ \delta_g^{\text{ev}} &= -\lfloor \frac{\alpha_g + \Delta_g - 1}{2} \rfloor, & \delta_g^{\text{od}} &= 1 - \lceil \frac{\alpha_g + \Delta_g - 1}{2} \rceil, \\ \Delta_h^{\text{ev}} &= -\delta_h^{\text{ev}} - \lfloor \frac{\alpha_h}{2} \rfloor + 1, & \Delta_h^{\text{od}} &= -\delta_h^{\text{od}} - \lfloor \frac{\alpha_h}{2} \rfloor + 1, \\ \Delta_g^{\text{ev}} &= -\delta_g^{\text{ev}} - \lfloor \frac{\alpha_g}{2} \rfloor + 1, & \Delta_g^{\text{od}} &= -\delta_g^{\text{od}} - \lfloor \frac{\alpha_g}{2} \rfloor + 1, \\ \bar{b}_h^{\text{ev}} &= \tilde{h}_{\text{ev}} \cdot \tilde{h}_{\text{neg}}, & \bar{b}_h^{\text{od}} &= \tilde{h}_{\text{od}} \cdot \tilde{h}_{\text{neg}}, \\ \bar{b}_g^{\text{ev}} &= \tilde{g}_{\text{ev}} \cdot \tilde{g}_{\text{neg}}, & \bar{b}_g^{\text{od}} &= \tilde{g}_{\text{od}} \cdot \tilde{g}_{\text{neg}}, \\ \bar{s}_h &= \tilde{h}_{\text{pos}} - \tilde{h}_{\text{neg}}, & \bar{s}_g &= \tilde{g}_{\text{pos}} - \tilde{g}_{\text{neg}}.\end{aligned}$$

Disable stencil test.

Set blending function to  $\Gamma_{\text{id}}$ .

Set post-convolution scaling and bias to  $\bar{s}_h$  and  $\bar{b}_h^{\text{ev}}$ .

Copy area  $[o_c + \delta_h^{\text{ev}}, o_c + \delta_h^{\text{ev}} + w + \Delta_h^{\text{ev}} - 1]$  to  $[o_o, o_o + \frac{1}{2}w)$ , using convolution filter  $\tilde{h}^{\text{ev}}$  (size  $\Delta_h^{\text{ev}}$ ).

Set blending function to  $\Gamma_{\text{add}}$ .

Set post-convolution scaling and bias to  $\bar{s}_g$  and  $\bar{b}_g^{\text{ev}}$ .

Copy area  $[o_d + \delta_g^{\text{ev}}, o_d + \delta_g^{\text{ev}} + w + \Delta_g^{\text{ev}} - 1]$  to  $[o_o, o_o + \frac{1}{2}w)$ , using convolution filter  $\tilde{g}^{\text{ev}}$  (size  $\Delta_g^{\text{ev}}$ ).

Set blending function to  $\Gamma_{\text{sub}}$ .

Set post-convolution scaling and bias to  $-\bar{s}_g$  and  $-\bar{b}_g^{\text{ev}}$ .

Copy area  $[o_d + \delta_g^{\text{ev}}, o_d + \delta_g^{\text{ev}} + w + \Delta_g^{\text{ev}} - 1]$  to  $[o_o, o_o + \frac{1}{2}w)$ , using convolution filter  $\tilde{g}^{\text{ev}}$  (size  $\Delta_g^{\text{ev}}$ ).

Enable stencil test, render only pixels with stencil value 1.

Set blending function to  $\Gamma_{\text{id}}$ .

Set post-convolution scaling and bias to  $\bar{s}_h$  and  $\bar{b}_h^{\text{od}}$ .

Copy area  $[o_c + \delta_h^{\text{od}}, o_c + \delta_h^{\text{od}} + w + \Delta_h^{\text{od}} - 1]$  to  $[o_o, o_o + \frac{1}{2}w)$ , using convolution filter  $\tilde{h}^{\text{od}}$  (size  $\Delta_h^{\text{od}}$ ).

Set blending function to  $\Gamma_{\text{add}}$ .

Set post-convolution scaling and bias to  $\bar{s}_g$  and  $\bar{b}_g^{\text{od}}$ .

Copy area  $[o_d + \delta_g^{\text{od}}, o_d + \delta_g^{\text{od}} + w + \Delta_g^{\text{od}} - 1]$  to  $[o_o, o_o + \frac{1}{2}w)$ , using convolution filter  $\tilde{g}^{\text{od}}$  (size  $\Delta_g^{\text{od}}$ ).

Set blending function to  $\Gamma_{\text{sub}}$ .

Set post-convolution scaling and bias to  $-\bar{s}_g$  and  $-\bar{b}_g^{\text{od}}$ .

Copy area  $[o_d + \delta_g^{\text{od}}, o_d + \delta_g^{\text{od}} + w + \Delta_g^{\text{od}} - 1]$  to  $[o_o, o_o + \frac{1}{2}w)$ , using convolution filter  $\tilde{g}^{\text{od}}$  (size  $\Delta_g^{\text{od}}$ ).

---

$h_j, g_j$	Low- and high-pass filters, respectively
$\alpha_x$	Index of first non-zero element of filter $x$
$\Delta_x$	Size of filter $x$
$o_c, o_d, w$	Input image offsets and size
$o_o$	Output image offset

Figure 8: Implementation sequence for wavelet reconstruction in hardware

tion filters have the size 1, which is a mere scaling.

## 6 Data Scaling

Up to now we dealt with the graphics pipe as if it could cope with floating point values. This is apparently not correct, as all frame buffer values are clamped to the interval  $[0, 1)$ . Luckily OpenGL provides the possibility to scale and bias pixel data after the convolution step, just before the clamping takes place.

In order to uphold consistency while performing the decomposition, we introduce scaling parameters  $s_h, s_g$ , and offset values  $b_h, b_g$ , that fit the resulting scaled wavelet coefficients  $\tilde{c}$  and  $\tilde{d}$ , represented by the pixel values  $p_j^n$ , to the interval  $[0, 1)$ . In the following we will only address the low-pass filtering sequence, because the high-pass filtering sequence is handled exactly the same way.

In particular we define the *scaled* decomposition equation

$$\tilde{c}_n^{j-1} = s_h \cdot \sum_k h_{k-2n} \tilde{c}_k^j + b_h \quad (18)$$

and initialize the decomposition with  $\tilde{c}_n^J = c_n^J$ . We can see that for positive  $\tilde{c}_n^j$  the sum  $\sum_k h_{k-2n} \tilde{c}_k^j$  gets minimal for

$$\tilde{c}_k^j = \begin{cases} 0 & h_{k-2n} \geq 0 \\ 1 & h_{k-2n} < 0 \end{cases}.$$

The maximum of the sum can be determined equivalently. By imposing these extrema to the restriction  $\tilde{c}_n^j \in [0, 1)$ , we get the scaling factor and the offset bias as

$$\begin{aligned}\tilde{h}_{\text{neg}} &= \sum_k \min(0, h_k), \\ \tilde{h}_{\text{pos}} &= \sum_k \max(0, h_k), \\ s_h &= \frac{1}{h_{\text{pos}} - h_{\text{neg}}}, & b_h &= -h_{\text{neg}} s_h.\end{aligned}$$

$n$		0	1	2	3
Haar	$h_n$	$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$		
	$g_n$	$\frac{1}{\sqrt{2}}$	$-\frac{1}{\sqrt{2}}$		
Daubechies (4)	$h_n$	$\frac{1+\sqrt{3}}{4\sqrt{2}}$	$\frac{3+\sqrt{3}}{4\sqrt{2}}$	$\frac{3-\sqrt{3}}{4\sqrt{2}}$	$\frac{1-\sqrt{3}}{4\sqrt{2}}$
	$g_n$	$\frac{1-\sqrt{3}}{4\sqrt{2}}$	$\frac{-3+\sqrt{3}}{4\sqrt{2}}$	$\frac{3+\sqrt{3}}{4\sqrt{2}}$	$\frac{-1-\sqrt{3}}{4\sqrt{2}}$

Table 1: Filter coefficients for different wavelet types.

During reconstruction, the scaling and bias has to be compensated. In order to accomplish this, we first insert (3) into (4):

$$\tilde{c}_k^j = \sum_n h_{k-2n} \cdot \sum_i h_{i-2n} \tilde{c}_i^j + \sum_n g_{k-2n} \cdot \sum_i g_{i-2n} \tilde{d}_i^j. \quad (19)$$

Now we insert Equation (18) into the equivalently scaled reconstruction equation and yield

$$\begin{aligned} \tilde{c}_k^j &= \\ \bar{s}_h \cdot \sum_n h_{k-2n} \cdot (s_h \cdot \sum_i h_{i-2n} \tilde{c}_i^j + b_h) &+ \bar{b}_h + \\ \bar{s}_g \cdot \sum_n g_{k-2n} \cdot (s_g \cdot \sum_i g_{i-2n} \tilde{d}_i^j + b_g) &+ \bar{b}_g \end{aligned} \quad (20)$$

The reconstruction process is performed in two steps, of which both are clamped to  $[0, 1)$ . The condition  $\tilde{c}_n^j \in [0, 1)$  and the lossless wavelet reconstruction ensures that the final value  $\tilde{c}_n^j$  does not exceed the clamping interval.

Now we decompose Equation (20) and compare the coefficients with (19). That way we yield:

$$\begin{aligned} \bar{s}_h &= \frac{1}{s_h}, \quad \bar{s}_g = \frac{1}{s_g} \\ \bar{b}_h &= h_{\text{neg}} \cdot \begin{cases} \sum_i h_{2i} & k \text{ even} \\ \sum_i h_{2i+1} & k \text{ odd} \end{cases} \\ \bar{b}_g &= g_{\text{neg}} \cdot \begin{cases} \sum_i g_{2i} & k \text{ even} \\ \sum_i g_{2i+1} & k \text{ odd} \end{cases}. \end{aligned}$$

Note that the bias parameters are different for odd and even input pixels. Therefore, only variant two of the reconstruction algorithms can be

successfully implemented using graphics hardware.

## 7 Results

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. Therefore, we take a closer look at differences of software and hardware filtered images.

First, Figures 9 and 12 show the original head and lena data sets, which were used for this analysis. The (enhanced) Figures 10, 11, 13, and 14 show the completely decomposed data sets for Haar and Daubechies wavelets of order 4. Table 1 lists the filter kernel coefficients for these two wavelet types.

The next images reveal the differences between software and hardware based Haar wavelet filtering in form of difference images of completely decomposed as well as decomposed and afterwards reconstructed data. The images had to be equalized in order to reveal *any* differences at all, as the hardware filtered variants differ only in the least significant bit. Figure 15 reveals all 1-bit differences between hardware and software decomposed data sets in the case of Haar wavelets for the head data set, a  $512^2$  sized slice of a computer tomography. Figures 16 and 17 show the differences on completely Haar wavelet decomposed and reconstructed data sets to their originals. Note that software decomposition using floats is accurately enough to restore image data during reconstruction without any noticeable differences.

As already mentioned in the previous sections, special care has to be taken at the borders. Without handling these special cases, artifacts will occur, as it can be seen in Figures 18 and 21. In order to verify, that only border data are affected, difference images have been calculated, which can be investigated in Figures 19 and 22. As apparently no differences except for the borders can be detected, equalized versions are presented in the Figures 20 and 23.

Wavelet decomposition and reconstruction have in principle the same order of complexity. However, hardware based reconstruction is about two to three times slower than decomposition due to the limitations of the graphics pipe. In general this is no major drawback, as wavelets are most often used for decomposition in order to accelerate volume rendering and feature detection. Currently, the expansion of compressed data, which would be a major application for wavelet reconstruction, does not map onto our algorithms very well, because all wavelet coefficients have to be stored in the images regardless of their values. Though hardware based wavelets are currently not much faster than software based filtering, a lot of time can be saved during a typical visualization cycle, as the data do not have to cross the graphics / host memory barrier. Still, there are several concepts to increase the rendering speed during wavelet filtering even more.

## 8 Conclusion

We have introduced a wavelet decomposition and reconstruction algorithm, that directly works on the graphics hardware of modern OpenGL capable workstations. By using the convolution and some blending extensions together with OpenGL's facilities to scale images during copy instructions, we are able to perform all necessary steps of one-dimensional wavelet filtering without copying data from or to the machine's main memory, thus avoiding typical bottlenecks in the visualization cycle. By using tensor product wavelets the algorithm can easily be extended to the two- and three-dimensional case, again

without touching main memory. Still, there are several approaches to speed up the filtering steps even more. Different strategies for reducing the graphical complexity as well as several 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 [12] due to the limited depth of the frame buffer. However, wavelet decomposition and reconstruction have proven to be relatively robust, as only single-bit differences between software and hardware decomposed data could be detected.

## 9 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. Technical Report 7/1999, Universität Erlangen-Nürnberg, Lehrstuhl für Graphische Datenverarbeitung (IMMD IX), Erlangen, April 1999.
- [4] 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.
- [5] A. K. Louis, P. Maass, and A. Rieder. *Wavelets*. B. G. Teubner Stuttgart, Germany, 1994.

- [6] 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.
- [7] T. Malzbender. Fourier-Volume-Rendering. *ACM Transactions on Graphics*, 12(3):233–250, July 1993.
- [8] SGI. Volume rendering using re2 technology. Technical report, SGI, 1994.
- [9] SGI. *OpenGL on Silicon Graphics Systems*. Silicon Graphics Inc., Mountain View, California, 1996.
- [10] O. Sommer, A. Dietz, R. Westermann, and T. Ertl. An Interactive Visualization and Navigation Tool for Medical Volume Data. In N. M. Thalmann and V. Skala, editors, *WSCG '98, The Sixth International Conference in Central Europe on Computer Graphics and Visualization '98*, volume II, pages 362–371, Plzen, Czech Republic, February 1998. University of West Bohemia Press.
- [11] G. Strang and T. Nguyen. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, Wellesley, Massachusetts, 1996.
- [12] 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.
- [13] T. Totsuka and M. Levoy. Frequency Domain Volume Rendering. *Computer Graphics*, 27(4):271–78, August 1993.
- [14] 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.
- [15] R. Westermann and T. Ertl. Efficiently Using Graphics Hardware in Volume Rendering Applications. In *Computer Graphics Proceedings, Annual Conference Series*, pages 169–177, Orlando, Florida, 1998. ACM SIGGRAPH.
- [16] M. V. Wickerhauser. *Adapted Wavelet Analysis from Theory to Software*. IEEE Press, New York, 1994.

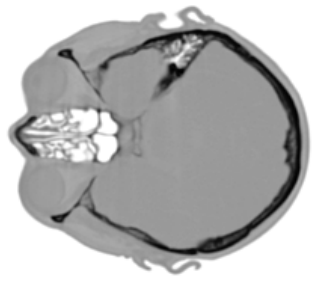


Figure 9: The head data set

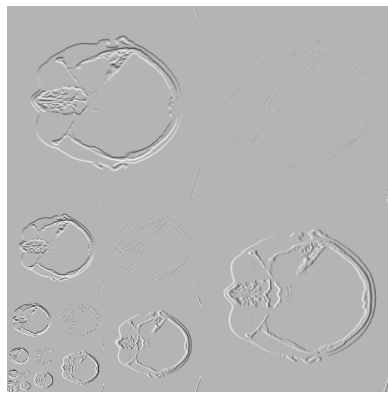


Figure 10: Haar wavelet decomposition

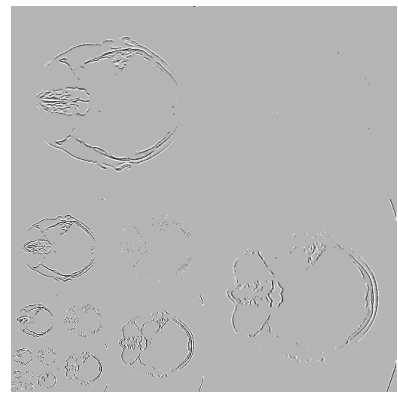


Figure 11: Daubechies wavelet decomposition



Figure 12: The lena data set



Figure 13: Haar wavelet decomposition



Figure 14: Daubechies wavelet decomposition

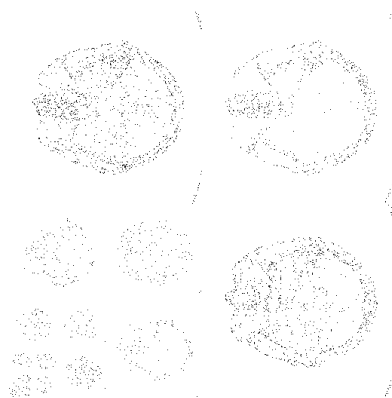


Figure 15: 1-bit differences between software and hardware Haar decomposition

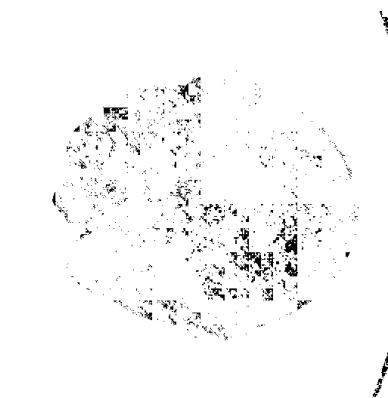


Figure 16: 1-bit differences after full Haar decomposition and reconstruction

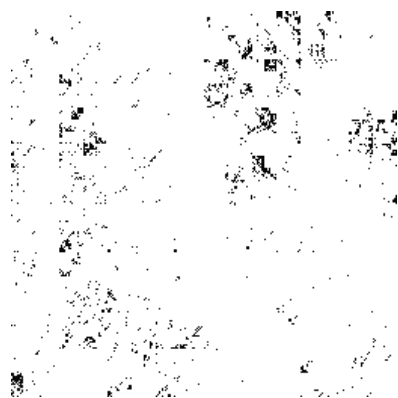


Figure 17: 1-bit differences on the lena data set

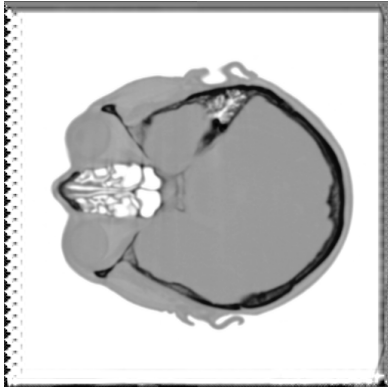


Figure 18: The head data set decomposed with Daubechies wavelets to a level depth of 4 and reconstructed afterwards

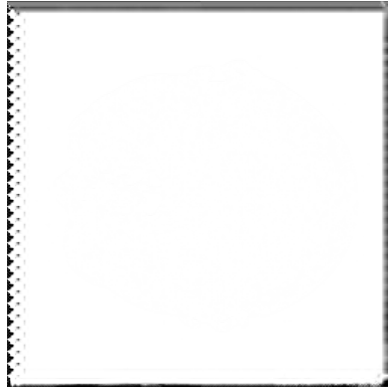


Figure 19: Differences between software and hardware Daubechies filtered data

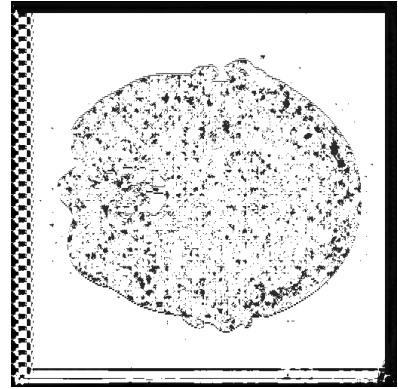


Figure 20: Enhanced version, making 1-bit differences visible



Figure 21: The lena data set decomposed with Daubechies wavelets to a level depth of 4 and reconstructed afterwards



Figure 22: Differences between software and hardware Daubechies filtered data

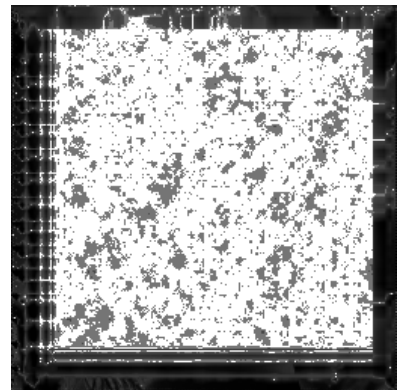


Figure 23: Enhanced version, making 1-bit differences visible