PYRAMID FILTERS BASED ON BILINEAR INTERPOLATION

Martin Kraus

Computer Graphics and Visualization Group, Technische Universität München, Germany

Magnus Strengert

Visualization and Interactive Systems Group, Universität Stuttgart, Germany

Keywords: Signal processing, image processing, multi resolution, pyramid algorithm, graphics hardware.

Abstract: The implementation of several pyramid methods on programmable graphics processing units (GPUs) in recent years led to additional research interest in pyramid algorithms for real-time computer graphics. Of particular interest are efficient analysis and synthesis filters based on hardware-supported bilinear texture interpolation because they can be used as building blocks for many GPU-based pyramid methods. In this work, several new and extremely efficient GPU-implementations of pyramid filters are presented for the first time. The discussion employs a new notation, which was developed for the consistent and precise specification of these filters and also allowed us to systematically explore appropriate filter designs. The presented filters cover analysis and synthesis filters, (quasi-)interpolation and approximation, as well as discontinuous, continuous, and smooth filters. Thus, a toolbox of filters and their efficient implementations for a great variety of GPU-based pyramid methods is presented.

1 INTRODUCTION

Many techniques in real-time image processing employ the pyramid algorithm by Burt (Burt, 1981), and GPU-based image processing is no exception to this rule (Williams, 1983; Krüger and Westermann, 2003; Lefebvre et al., 2005; Strengert et al., 2006). Pyramid methods are of particular interest because they usually feature a linear time complexity and require only a limited number of switches of the render target.

Although modern GPUs offer an enormous rasterization performance, the actual rasterization budget for each pixel often consists of the equivalent to only a few dozens of texture reads. This is already a serious limitation for many image processing techniques, which often employ large two-dimensional convolution filters. Thus, even GPU-based implementations of complex image processing techniques are often restricted to small image sizes and/or low frame rates. Therefore, several very efficient techniques have been developed to implement specific (often small) convolution filters on GPUs (Sigg and Hadwiger, 2005; Green, 2005). Unfortunately, many of these techniques are restricted to very specific filters and partic-

ular applications; thus, the efficient implementation of many other filters is still a challenging task in GPU-based image processing.

In this work, we generalize a recently published technique (Strengert et al., 2006) for a 2×2 box analysis filter and a 2×2 B-spline synthesis filter. By formalizing the underlying concept with the help of a new notation, we are able to systematically explore appropriate filter designs and present several filters that can be more efficiently implemented on GPUs using bilinear texture interpolation than previous publications suggested.

In order to make these filters and their implementations more easily accessible to readers who are looking for particular filters, we have also included some previously published filter implementations. Moreover, these filters should help to illustrate our new notation and the systematic filter construction suggested in this work. We classify filters as *basic filters* presented in Section 3; i.e., convolution filters without reduce or expand operation; *analysis filters* presented in Section 4; i.e., convolution filters combined with a reduce operation; and *synthesis filters* presented in Section 5, which are combined with an

top of pyramid: $g^{(2)}$

$g^{(1)} = h_{\text{analysis}} \otimes_\downarrow g^{(0)}$ $g^{(0)} = h_{\text{synthesis}} \otimes_\uparrow g^{(1)}$

image g

filter $h^{2\times 2}$

Σ

$h^{2\times 2} \otimes g$

(a)   (b)

$$
h^{2\times 2} \otimes g = \begin{bmatrix} h_{1,1} & h_{1,2} \\ h_{2,1} & h_{2,2} \end{bmatrix} \otimes \begin{bmatrix} g_{1,1} & g_{1,2} & 0 \\ g_{2,1} & g_{2,2} & 0 \\ 0 & 0 & 0 \end{bmatrix} \equiv \begin{bmatrix} 0 & 0 & 0 \\ 0 & h_{1,1} & h_{1,2} \\ 0 & h_{2,1} & h_{2,2} \end{bmatrix} \otimes \begin{bmatrix} g_{1,1} & g_{1,2} & 0 \\ g_{2,1} & g_{2,2} & 0 \\ 0 & 0 & 0 \end{bmatrix}
$$

$$
= \begin{bmatrix} h_{1,1}g_{1,1} & h_{1,2}g_{1,1}+h_{1,1}g_{1,2} & h_{1,2}g_{1,2} \\ h_{2,1}g_{1,1}+h_{1,1}g_{2,1} & h_{2,2}g_{1,1}+h_{2,1}g_{1,2}+h_{1,2}g_{2,1}+h_{1,1}g_{2,2} & h_{2,2}g_{1,2}+h_{1,2}g_{2,2} \\ h_{2,1}g_{2,1} & h_{2,2}g_{2,1}+h_{2,1}g_{2,2} & h_{2,2}g_{2,2} \end{bmatrix}
$$

(c)
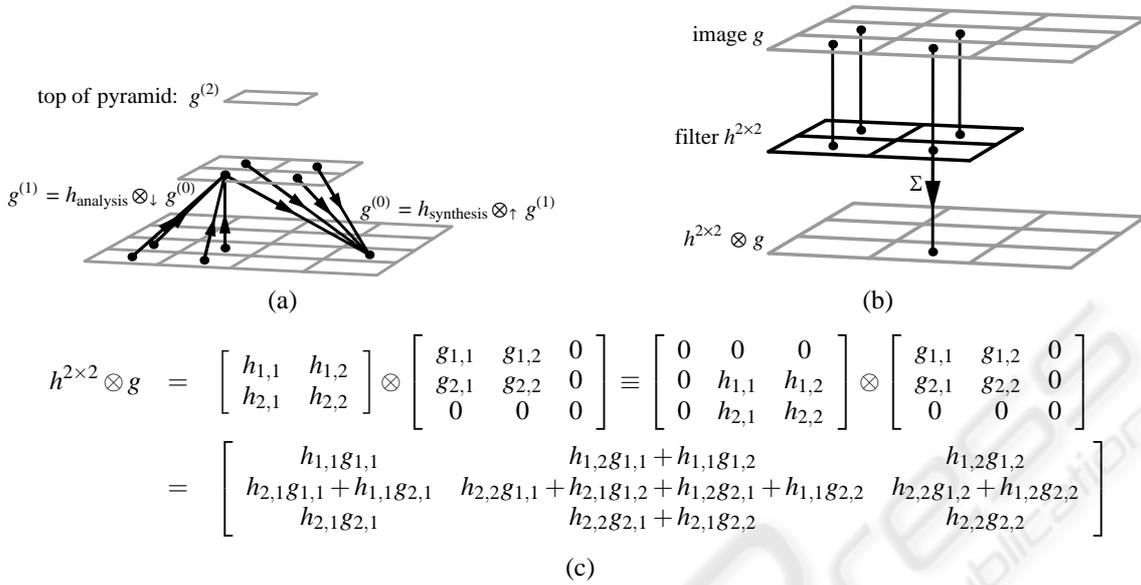
Figure 1: (a) An image pyramid consisting of 3 levels: $g^{(0)}$, $g^{(1)}$, and $g^{(2)}$. (b) Illustration of the convolution of a 3×3 image g with a 2×2 filter $h^{2\times 2}$. (c) The formal equation of the convolution illustrated in (b) including the equivalent 3×3 filter padded with zeros. Note the "mirrored" indices of $h^{2\times 2}$ and the "shift" of non-zero components in the resulting matrix.

expand operation. Analysis and synthesis filters are further classified according to their featured smoothness (in the limit of infinitely many reduce or expand operations). Section 6 concludes this work with our plans for future work.

2   PYRAMID METHODS ON GPUS

Pyramid methods usually convolve image data of one pyramid level with a small analysis filter and reduce the resulting image data to compute coarser pyramid levels. This process is called analysis while the opposite process, called synthesis, expands image data and convolves it with small synthesis filters to compute finer pyramid levels; see Figure 1a. The reduce and expand operation are also called downsampling and upsampling, respectively.

Pyramid methods—more specifically spoken, pyramid images—have been most successful in computer graphics in the form of mipmap textures as proposed by Williams (Williams, 1983). More recently, the possibility to read pixel data of a rasterized image by texture interpolation without crucial overhead led to new real-time image processing techniques on GPUs (Green, 2005). For pyramid methods in GPU-based image processing, we suggested to employ hardware-supported bilinear texture interpo-

lation for the reduce operation combined with a 2×2 convolution filter (Strengert et al., 2006). However, the only analysis filter presented in that work is a simple 2×2 box filter. For the synthesis, we proposed to employ bilinear texture interpolation for the combination of the expand operation and a 2×2 synthesis filter corresponding to the biquadratic B-spline subdivision scheme (Catmull and Clark, 1978), which is better known as the Doo-Sabin subdivision scheme for regular quadrilaterals.

3   BASIC FILTERS

The filters discussed in this section are simple convolution filters without reduce or expand operation. Thus, strictly speaking, they are not pyramid filters. However, they act as building blocks for the more complex filters discussed in Sections 4 and 5. A discrete convolution of a filter mask h and an image g resulting in an image f is denoted by

$$f = h \otimes g.$$

If the filter mask h is represented by an $n_i \times n_j$ matrix and the image g is of dimensions $n_r \times n_c$, the matrix component $f_{r,c}$ for the row index r and the column

index c is defined by:

$$f_{r,c} = \sum_{i=1}^{n_i} \sum_{j=1}^{n_j} h_{i,j} g_{r-\lfloor i-n_i/2 \rfloor, c-\lfloor j-n_j/2 \rfloor}.$$

Standard matrix notation is employed; i.e., row and column indices are given in this order and indices start with 1. Moreover, matrix products of column vectors times row vectors (i.e., outer products of vectors) are employed for separable filters.

Components of the image g with indices outside the ranges from 1 to n_r and 1 to n_c, respectively, are either set to 0 if the image g represents a filter mask, or determined by clamping the indices to the valid ranges. The dimensions of f depend on the particular application; in this work we usually determine the dimensions of f by g's dimensions; exceptions are mentioned explicitly.

For even filter dimensions n_i and n_j, it is often useful to think of the pixel positions of image f being shifted by half a pixel along the diagonal relatively to image g. In order to make this shift explicit, one can use zero padding of the convolution mask; e.g., for a 2×2 convolution mask $h^{2\times2}$ there is an equivalent 3×3 convolution mask with zeros in the first row and column:

$$h^{2\times2} \overset{\text{def}}{=} \begin{bmatrix} h_{1,1} & h_{1,2} \\ h_{2,1} & h_{2,2} \end{bmatrix} \equiv \begin{bmatrix} 0 & 0 & 0 \\ 0 & h_{1,1} & h_{1,2} \\ 0 & h_{2,1} & h_{2,2} \end{bmatrix}.$$

For an illustration of these definitions and the described shift of components of f, see Figures 1b and 1c.

As mentioned, g—and therefore f—may also represent filter masks. In this case, g and h denote two convolution masks (applied from right to left), which can be combined in one convolution mask f. In fact, the main motivation for our formalism is to decompose complex filter masks (e.g., f) into multiple smaller filters (e.g., h and g, but usually more than two), which are small enough (i.e., 2×2) to be implemented by bilinear texture interpolations. In this way, complex filters can be implemented by a sequence of bilinear texture interpolations.

3.1 2×2 Box Filter

The most important building block for the filters discussed in this work is the 2×2 box filter, also known as uniform, average, or mean filter:

$$h_{\text{box}}^{2\times2} \overset{\text{def}}{=} \frac{1}{4} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}.$$

Since this filter multiplies four neighboring pixels with equal weights, it is easily implementable with

one hardware-accelerated bilinear texture image interpolation. The sampling position for this texture interpolation (usually specified by texture coordinates) is determined by the position of the shared corner of the four pixels (in the "little squares" model of pixels) or the barycenter of the four pixels (if the pixels themselves represent sampling points in a uniform grid).

Due to our definition of the convolution, the sampling point for the resulting matrix component with indices r and c is located at the upper, left corner of the pixel specified by r and c in the original source matrix. (We assume a coordinate system that corresponds to traditional matrix notation with the (positive) r axis pointing downwards and the (positive) c axis pointing to the right.) This shift by half a diagonal of one pixel is more explicit in the equivalent zero-padded 3×3 filter mask, which will be called $h_{\text{box}}^{\searrow}$:

$$h_{\text{box}}^{2\times2} \equiv h_{\text{box}}^{\searrow} \overset{\text{def}}{=} \frac{1}{4} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

The arrow in the symbol $h_{\text{box}}^{\searrow}$ indicates the position of non-zero elements in the 3×3 matrix as well as the shift of non-zero elements in the resulting matrix illustrated in Figure 1c. Note, however, that the sampling position for the bilinear texture interpolation is shifted in the opposite direction relatively to the original pixel position.

Obviously, there are further (non-equivalent) zero-padded 3×3 filters, which shift pixel positions in other directions; e.g., the opposite direction for the filter $h_{\text{box}}^{\nwarrow}$:

$$h_{\text{box}}^{\nwarrow} \overset{\text{def}}{=} \frac{1}{4} \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

This filter $h_{\text{box}}^{\nwarrow}$ can also be implemented by a bilinear texture interpolation if the sampling point is set to the opposite (lower, right) pixel corner. These two box filters are the only building blocks for all filters presented in this section and Section 4; i.e., all these filters can be implemented by a decomposition into a sequence of convolutions with $h_{\text{box}}^{\searrow}$ and $h_{\text{box}}^{\nwarrow}$, and the application of the corresponding sequence of bilinear texture interpolations. The most basic example is the 3×3 Bartlett filter discussed next.

3.2 3×3 Bartlett Filter

Bartlett filters are also called triangular or (in particular in one dimension) triangle filters. They are separable filters; therefore, they may be decomposed into

matrix products of column times row vectors. In this work, the 3×3 Bartlett filter is of particular interest:

$$h_{\text{Bartlett}}^{3\times 3} \quad \overset{\text{def}}{\equiv} \quad \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$= \quad \frac{1}{4} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \cdot \frac{1}{4} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}.$$

The separation into two one-dimensional filters can also be expressed by a sequence of two convolutions with these filters (with appropriate filter dimensions and index mirroring). However, the representation as a convolution of box filters leads to a more efficient implementation:

$$h_{\text{Bartlett}}^{3\times 3} \quad = \quad \frac{1}{4} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \otimes \frac{1}{4} \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$= \quad h_{\text{box}}^{\searrow} \otimes h_{\text{box}}^{\nwarrow} = h_{\text{box}}^{\nwarrow} \otimes h_{\text{box}}^{\searrow}.$$

I.e., a sequence of two convolutions with $h_{\text{box}}^{\searrow}$ and $h_{\text{box}}^{\nwarrow}$ is equivalent to a single convolution with the 3×3 Bartlett filter $h_{\text{Bartlett}}^{3\times 3}$. Therefore, the 3×3 Bartlett filter may be implemented by a sequence of two bilinear texture interpolations corresponding to the two 2×2 box filters. Note that the second texture interpolation has to access the result of the first convolution; thus, a hardware-accelerated implementation will usually have to switch the render target in order to access the previously rasterized image. Note also that it is crucial to alternate between $h_{\text{box}}^{\searrow}$ and $h_{\text{box}}^{\nwarrow}$; otherwise, the discussed shifts would not cancel and the resulting image would be shifted by one full pixel position.

Since the 3×3 Bartlett filter $h_{\text{Bartlett}}^{3\times 3}$ is particularly useful, we will also use it for building up more complex filters. However, it is always understood that a convolution with $h_{\text{Bartlett}}^{3\times 3}$ is equivalent to a sequence of one convolution with $h_{\text{box}}^{\searrow}$ and one convolution with $h_{\text{box}}^{\nwarrow}$.

3.3 Gaussian Filters

Repeated convolutions of Bartlett filters are equivalent to approximations of Gaussian filters if all non-zero matrix components of the resulting filters are considered; for example:

$$h_{\text{Gauss}}^{5\times 5} \quad \overset{\text{def}}{\equiv} \quad \frac{1}{16^2} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

$$\equiv \quad h_{\text{Bartlett}}^{3\times 3} \otimes h_{\text{Bartlett}}^{3\times 3},$$

$$h_{\text{Gauss}}^{7\times 7} \quad \overset{\text{def}}{\equiv} \quad \frac{1}{16^3} \begin{bmatrix} 1 & 6 & 15 & 20 & 15 & 6 & 1 \\ 6 & 36 & 90 & 120 & 90 & 36 & 6 \\ 15 & 90 & 225 & 300 & 225 & 90 & 15 \\ 20 & 120 & 300 & 400 & 300 & 120 & 20 \\ 15 & 90 & 225 & 300 & 225 & 90 & 15 \\ 6 & 36 & 90 & 120 & 90 & 36 & 6 \\ 1 & 6 & 15 & 20 & 15 & 6 & 1 \end{bmatrix}$$

$$\equiv \quad h_{\text{Bartlett}}^{3\times 3} \otimes h_{\text{Bartlett}}^{3\times 3} \otimes h_{\text{Bartlett}}^{3\times 3}.$$

The basic reason for the approximation of Gaussian filters is the central limit theorem; in fact, any reasonable, positive filter will converge to a Gaussian filter in the limit of infinitely many convolutions. In the case of the box filter—and therefore also for the Bartlett filter—the continuous convolutions are actually higher-order B-splines.

Since each convolution with a Bartlett filter can be implemented by two bilinear texture interpolations, $n - 1$ texture interpolations are necessary to implement an approximation to the convolution with an $n \times n$ Gaussian filter. For comparison, a separable $n \times n$ filter with fixed filter weights for the two one-dimensional filters would require $2n$ nearest-neighbor texture reads.

4 ANALYSIS FILTERS

Analysis filters are convolution filters that are combined with a reduce operation, which reduces the number of pixels by a factor of 2 in each dimension. Therefore, this operation is also called downsampling. In our notation it is indicated by a downward pointing arrow:

$$f = h \otimes_{\downarrow} g,$$

which defines the components of f as:

$$f_{r,c} = \sum_{i=1}^{n_i} \sum_{j=1}^{n_j} h_{i,j} g_{2r-\lfloor i-n_i/2 \rfloor, 2c-\lfloor j-n_j/2 \rfloor}.$$

The issue of shifts by half a pixel diagonal is different from the problem discussed in Section 3. Since the number of components is divided by 2, it is preferable to work with even dimensions of images and filter masks. Moreover, it is often preferable to use sym-

metric filter mask (in the sense of $h_i = h_{n_i-i+1}$) in order to avoid asymmetric weighting of even and odd pixels.

For convenience we introduce a particular notation for the combination of an analysis filter with a reduction by a factor of 2^m in each dimension:

$$f = h \otimes_{\downarrow}^m g,$$

which results in these components of f:

$$f_{r,c} = \sum_{i=1}^{n_i} \sum_{j=1}^{n_j} h_{i,j} g_{2^m r - \lfloor i - n_i/2 \rfloor, 2^m c - \lfloor j - n_j/2 \rfloor}.$$

Our notation combines a reduce operation with a convolution in a single operation. Thus, if the convolution can be implemented with a bilinear texture interpolation, the combination with the reduce operation is also implementable with a bilinear texture interpolation. To this end, it is only necessary to use a sparser grid of texture sampling points.

4.1 Discontinuous Filter

In order to illustrate our notation for analysis filters, it is first applied to the 2×2 box filter, which is the standard analysis filter for mipmap generation:

$$h_{\text{box}}^{2 \times 2} \otimes_{\downarrow} g \equiv h_{\text{box}}^{\searrow} \otimes_{\downarrow} g.$$

In the case of analysis filters, the use of $h_{\text{box}}^{2 \times 2}$ might be preferable, although it is equivalent to $h_{\text{box}}^{\searrow}$. One bilinear texture interpolation is sufficient to implement this analysis filter in pyramid methods (Strengert et al., 2006).

The 2×2 box filter is classified as "discontinuous" analysis filter because the equivalent filter for a reduction of the dimensions by a factor of 2^m is always a discontinuous box filter—even in the limit of $m \to \infty$: Consider the squared filter, which is defined by a sequence of two reduce operations and convolutions:

$$\left(h_{\text{box}}^{2 \times 2} \right)^2 \otimes_{\downarrow}^2 g \stackrel{\text{def}}{=} h_{\text{box}}^{2 \times 2} \otimes_{\downarrow} \left(h_{\text{box}}^{2 \times 2} \otimes_{\downarrow} g \right).$$

Thus, the actual (separable) box filter is:

$$\left(h_{\text{box}}^{2 \times 2} \right)^2 = \frac{1}{4} \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}^{\top} \cdot$$
$$\cdot \frac{1}{4} \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

For a general factor 2^m, the equivalent filter

$$\left(h_{\text{box}}^{2 \times 2} \right)^m = \frac{1}{2^m} \big[\underbrace{0 \ldots 0}_{\times (2^m - 1)} \underbrace{1 \ldots 1}_{\times 2^m} \big]^{\top} \frac{1}{2^m} \big[\underbrace{0 \ldots 0}_{\times (2^m - 1)} \underbrace{1 \ldots 1}_{\times 2^m} \big]$$

is still a discontinuous box filter. The one-dimensional filter, which this separable filter is constructed from, is illustrated in Figure 2a for various values of m. The same technique is employed to discuss and classify the smoothness of all analysis filters presented in this section.
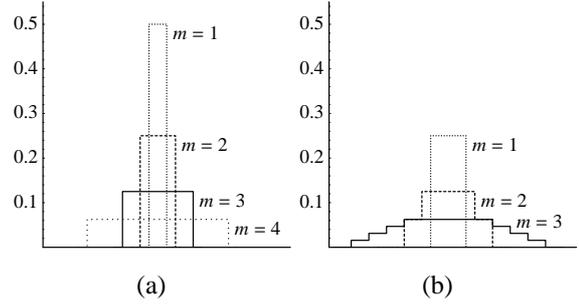
Figure 2: (a) Powers of the 1D box filter mask corresponding to $h_{\text{box}}^{2 \times 2}$. (b) Same as (a) for $h_{\text{box}}^{4 \times 4}$.

4.2 C^0-Continuous Filter

While the 2×2 box filter is a discontinuous analysis filter, the 4×4 box filter results in a C^0-continuous analysis filter in the limit of infinitely many analysis steps. It is defined as:

$$h_{\text{box}}^{4 \times 4} \stackrel{\text{def}}{=} \frac{1}{16} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix},$$

$$h_{\text{box}}^{4 \times 4} \otimes_{\downarrow} g \equiv h_{\text{box}}^{\searrow} \otimes \left(h_{\text{box}}^{\nwarrow} \otimes_{\downarrow} g \right).$$

As indicated by our notation, this filter can be constructed by a reduce operation that includes a convolution with $h_{\text{box}}^{\nwarrow}$ and a simple convolution with $h_{\text{box}}^{\searrow}$. Therefore, the implementation requires only two bilinear texture interpolations. However, there exists one additional implementation difficulty: the first reduce operation will in general compute non-zero components for the 0-th row and column. These intermediate results have to be stored and used in the second convolution, otherwise the components of the first row and column of the total result will be corrupted.

For the discussion of the smoothness of this analysis filter, we consider the equivalent (separable) squared filter first:

$$\left(h_{\text{box}}^{4 \times 4} \right)^2 = \frac{1}{16} \big[0\,0\,0\,0\,1\,1\,2\,2\,2\,2\,2\,2\,1\,1\,0 \big]^{\top} \cdot$$
$$\cdot \frac{1}{16} \big[0\,0\,0\,0\,1\,1\,2\,2\,2\,2\,2\,2\,1\,1\,0 \big].$$

The first three powers are illustrated in Figure 2b. In contrast to the 2×2 box filter, the 4×4 box filter has overlapping domains and therefore becomes C^0-continuous in the limit $m \to \infty$; in fact, the 1D filter becomes piecewise-linear with a linear ascending part, a constant part, and a linear descending part.

Our notation also suggests the construction of an

| (a) | (b) |
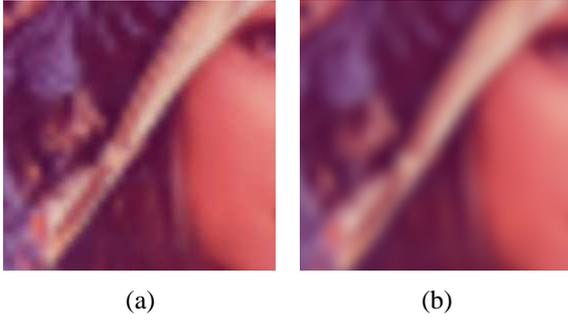
Figure 3: Pyramid image blurring using (a) the 2×2 box analysis filter and (b) the 4×4 box analysis filter. (The employed synthesis filter is discussed in Section 5.3.).

alternative combination of two box filters with one reduce operation:

$$h_{\text{Bartlett}}^{3\times3} \otimes_\downarrow g \equiv h_{\text{box}}^{\searrow} \otimes_\downarrow \left(h_{\text{box}}^{\nwarrow} \otimes g \right).$$

In the limit of $m \to \infty$ this filter is also C^0-continuous (in fact, it is a triangle function); however, it includes an additional undesirable shift and—what is worse—the analysis of even and odd pixels becomes asymmetric, which is likely to result in flickering artifacts in animations. Therefore, the $h_{\text{box}}^{4\times4}$ filter appears to be the preferable C^0-continuous analysis filter; e.g., for pyramidal image blurring (Strengert et al., 2006) as illustrated in Figure 3.

4.3 C^1-Continuous Filter

By combining one reduce operation and three convolutions with box filters a C^1-continuous filter can be constructed, which we call $h_{\text{quadratic}}^{4\times4}$ since it corresponds to a biquadratic B-spline in the limit of infinitely many analysis steps. The filter is defined as

$$h_{\text{quadratic}}^{4\times4} \stackrel{\text{def}}{\equiv} \frac{1}{64} \begin{bmatrix} 1 & 3 & 3 & 1 \\ 3 & 9 & 9 & 3 \\ 3 & 9 & 9 & 3 \\ 1 & 3 & 3 & 1 \end{bmatrix},$$

$$h_{\text{quadratic}}^{4\times4} \otimes_\downarrow g \equiv h_{\text{box}}^{\searrow} \otimes_\downarrow \left(h_{\text{Bartlett}}^{3\times3} \otimes g \right).$$

The decomposition shows that three bilinear texture interpolations are sufficient for an implementation of this analysis filter.

The squared filter already indicates a shape similar to the quadratic B-spline:

$$\left(h_{\text{quadratic}}^{4\times4}\right)^2 = \frac{1}{64}\left[0\,0\,0\,0\,1\,3\,6\,10\,12\,12\,10\,6\,3\,1\,0\right]^\top \cdot$$

$$\cdot \frac{1}{64}\left[0\,0\,0\,0\,1\,3\,6\,10\,12\,12\,10\,6\,3\,1\,0\right].$$

In the limit of infinitely high powers it actually converges to the C^1-continuous biquadratic B-spline function.

Another combination of three box filters and one reduce operation is:

$$h_{\text{Bartlett}}^{3\times3} \otimes \left(h_{\text{box}}^{\searrow} \otimes_\downarrow g \right);$$

however, this analysis filter is only C^0-continuous according to our classification and requires one more texture interpolation than $h_{\text{box}}^{4\times4}$. Yet another combination is:

$$h_{\text{box}}^{\nwarrow} \otimes \left(h_{\text{Bartlett}}^{3\times3} \otimes_\downarrow g \right),$$

which results in a C^1-continuous analysis filter but includes an undesirable shift and an asymmetric weighting of pixels. Thus, $h_{\text{quadratic}}^{4\times4}$ is usually the preferable C^1-continuous analysis filter.

The construction of higher-order B-spline analysis filters consists of additional convolutions with 2×2 box filters before the reduce operation is applied, e.g.:

$$h_{\text{cubic}}^{5\times5} \otimes_\downarrow g \quad \equiv \quad h_{\text{box}}^{\searrow} \otimes_\downarrow \left(h_{\text{Bartlett}}^{3\times3} \otimes \left(h_{\text{box}}^{\searrow} \otimes g \right) \right).$$

5 SYNTHESIS FILTERS

Analogously to analysis filters, synthesis filters are convolution filters that are combined with an expand operation, which increases the number of pixels by a factor of 2 in each dimension. This operation is also called upsampling and is indicated by a upwards pointing arrow in our notation:

$$f = h \otimes_\uparrow g,$$

where the components of f are:

$$f_{r,c} = \sum_{i=1}^{n_i} \sum_{j=1}^{n_j} h_{i,j}^{r \bmod 2, c \bmod 2}$$
$$\times g_{\lfloor (r+1)/2 \rfloor - \lfloor i - n_i/2 \rfloor, \lfloor (c+1)/2 \rfloor - \lfloor j - n_j/2 \rfloor}.$$

The synthesis filters presented in this section are strongly related to popular subdivision schemes, which are very well known in computer graphics; thus, we will considerably shorten the discussion by refering the reader to the corresponding concepts for subdivision schemes as discussed, for example, by Zorin et al. (Zorin et al., 2000).

Similarly to the case of analysis filters, the combination of an expand operation and convolution filters in our notation is closer to an efficient implementation using bilinear texture interpolation than traditional notations. One crucial difference to analysis filters is the choice of sampling positions in the corresponding bilinear texture interpolation. For most interpolating synthesis filters, the sampling points are

the centers of pixels, their corners, and the midpoints of their edges. This corresponds to face-split subdivision schemes, which are also known as primal schemes.

The most important alternative sampling positions are the positions of the Doo-Sabin subdivision scheme for regular quadrilaterals (Strengert et al., 2006). This alternative corresponds to vertex-split subdivision schemes (also known as dual schemes) and offers the advantage of symmetric computations for all sampling positions while face-split schemes distinguish between old and new positions.

In both cases, there are four different kinds of sampling positions, which correspond to four convolution filters $h^{1,1}$, $h^{1,0}$, $h^{0,1}$, and $h^{0,0}$ with the superscripts determined by the new row index modulo 2 and the new column index modulo 2. This notation is illustrated with the help of the well-known synthesis filters for nearest-neighbor interpolation, bilinear interpolation, and the biquadratic B-spline filter. On the other hand, the suggested implementation of the bicubic B-spline synthesis filter and the construction of higher-order synthesis filters is a new result.

5.1 Discontinuous Filter

A discontinuous synthesis filter is already provided by nearest-neighbor texture interpolation; thus, it is not of particular interest for our work. However, we have included it here for completeness and to demonstrate our notation for this very basic synthesis filter:

$$h^{1,1}_{\text{nearest}} \overset{\text{def}}{=} h^{1,0}_{\text{nearest}} \overset{\text{def}}{=} h^{0,1}_{\text{nearest}} \overset{\text{def}}{=} h^{0,0}_{\text{nearest}} \overset{\text{def}}{=} \left[\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{array} \right].$$

5.2 C^0-Continuous Filter

Since bilinear texture interpolation already provides a C^0-continuous filter even without the overhead of a pyramid method, the equivalent synthesis filter is not very useful in itself; however, it may be used as a building block for more complex synthesis filters. It is called h_{midpoint} because of the corresponding midpoint subdivision scheme; its definition is:

$$h^{1,1}_{\text{midpoint}} \overset{\text{def}}{=} \frac{1}{4} \left[\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{array} \right],$$

$$h^{1,0}_{\text{midpoint}} \overset{\text{def}}{=} \frac{1}{2} \left[\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{array} \right],$$

$$h^{0,1}_{\text{midpoint}} \overset{\text{def}}{=} \left(h^{1,0}_{\text{midpoint}} \right)^{\top},$$

$$h^{0,0}_{\text{midpoint}} \overset{\text{def}}{=} h^{0,0}_{\text{nearest}}.$$

Note that $h^{0,0}_{\text{midpoint}}$ corresponds to the convolution filter for sampling positions at pixel centers while $h^{1,1}_{\text{midpoint}}$ corresponds to the convolution filter for sampling positions at their corners, and the remaining two filters correspond to the centers of edges of pixels.

Also note that the first row and the first column of the new image data will be sampled at the upper, left corners of "old" pixels and the midpoints of edges between these positions. Therefore, the presented convolution filters will access "old" pixels of the 0-th row and 0-th column. In practice, these lookups could return the corresponding pixel data of the first row and first column, respectively. An alternative is to discard the first row and first column of the resulting image.

5.3 C^1-Continuous Filter

The synthesis filter discussed here corresponds to the biquadratic B-spline subdivision scheme (Catmull and Clark, 1978), also known as the Doo-Sabin subdivision scheme for regular quadrilaterals or the two-dimensional generalization of the Chaikin scheme. It is an approximating vertex-split subdivision scheme; thus, all pixels are processed in symmetric ways:

$$h^{1,1}_{\text{Doo-Sabin}} \overset{\text{def}}{=} \frac{1}{16} \left[\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 9 & 3 \\ 0 & 3 & 1 \end{array} \right],$$

$$h^{1,0}_{\text{Doo-Sabin}} \overset{\text{def}}{=} \frac{1}{16} \left[\begin{array}{ccc} 0 & 0 & 0 \\ 3 & 9 & 0 \\ 1 & 3 & 0 \end{array} \right],$$

$$h^{0,1}_{\text{Doo-Sabin}} \overset{\text{def}}{=} \left(h^{1,0}_{\text{Doo-Sabin}} \right)^{\top},$$

$$h^{0,0}_{\text{Doo-Sabin}} \overset{\text{def}}{=} \frac{1}{16} \left[\begin{array}{ccc} 1 & 3 & 0 \\ 3 & 9 & 0 \\ 0 & 0 & 0 \end{array} \right].$$

The implementation of this synthesis filter requires only one bilinear texture interpolation (Strengert et al., 2006). Note that the convolution for all boundary pixels of the resulting image accesses pixels outside of the original image.

Our notation also suggests an equivalent but less efficient implementation variant with two texture interpolations instead of one:

$$h_{\text{Doo-Sabin}} \otimes_{\uparrow} g \equiv h^{\nwarrow}_{\text{box}} \otimes \left(h_{\text{midpoint}} \otimes_{\uparrow} g \right).$$

Another synthesis filter may be constructed by re-ordering the convolution filters:

$$h_{\text{midpoint}} \otimes_{\uparrow} \left(h^{\nwarrow}_{\text{box}} \otimes g \right).$$

Since the resulting synthesis filter is only C^0-continuous and requires one additional texture interpolation, there is no apparent advantage compared to the C^1-continuous $h_{\text{Doo-Sabin}}$ filter.

5.4 C^2-Continuous Filter

The C^2-continuous synthesis filter corresponding to the bicubic B-spline subdivision scheme can be constructed easily in our notation with the help of one additional convolution with a 2×2 box filter:

$$h_{\text{cubic}} \otimes_\uparrow g \equiv h_{\text{box}}^{\searrow} \otimes \left(h_{\text{Doo-Sabin}} \otimes_\uparrow g \right).$$

Permutations of the components of this construction result in less symmetric and/or less smooth synthesis filters. On the other hand, the construction of higher-order B-spline filters by additional convolutions with 2×2 box filters should now be obvious. Note, however, that $h_{\text{box}}^{\nearrow}$ and $h_{\text{box}}^{\searrow}$ should appear in alternating order as discussed in Section 3.

5.5 (Quasi-)Interpolating Filters

Unfortunately, the construction of a C^1-continuous interpolation synthesis filter is considerably more difficult than the approximation synthesis filter corresponding to B-splines. If quasi-interpolation is sufficient, the image data of the coarsest level can be convolved with a filter before the synthesis is performed. The computation of appropriate convolution filters is discussed by Litke et al. (Litke et al., 2001). For example, a convolution filter for quasi-interpolation with bicubic B-splines can be constructed that is implementable with three texture interpolations:

$$\frac{1}{24} \begin{bmatrix} -1 & -2 & -1 \\ -2 & 36 & -2 \\ -1 & -2 & -1 \end{bmatrix} \equiv \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{10}{6} & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{16}{24} h_{\text{Bartlett}}^{3 \times 3}.$$

Our recommendation for an actually interpolating, C^1-continuous synthesis filter corresponds to the tensor-product generalization of the four-point subdivision scheme. Since it is separable by construction, it can be implemented by a synthesis operation for the rows followed by a synthesis operation for the columns. This appears to result in the most efficient implementation using 4.5 bilinear texture interpolations per pixel of the resulting image.

6 CONCLUSION

In this work, a set of discrete pyramid filters that are suitable for an efficient implementation based on bilinear texture interpolation has been presented. A new, precise and consistent notation for convolutions with these filters enabled us to construct appropriate filters in a systematic way and helped us to discuss many important implementation details. In particular, we have proposed an efficient implementation of biquadratic (and higher-order) B-spline analysis filters and of bicubic (and higher-order) B-spline synthesis filters.

Apart from applications of these filters, our plans for future work on pyramid filters include more efficient interpolating synthesis filters, three-dimensional filters, and efficient implementations of derivative filters and nonlinear filters.

REFERENCES

Burt, P. J. (1981). Fast Filter Transforms for Image Processing. *Computer Graphics and Image Processing*, 16:20–51.

Catmull, E. and Clark, J. (1978). Recursively Generated B-Spline Surfaces on Arbitrary Topological Meshes. *Computer Aided Design*, 10(6):350–355.

Green, S. (2005). Image Processing Tricks in OpenGL. Presentation at GDC 2005.

Krüger, J. and Westermann, R. (2003). Linear Algebra Operators for GPU Implementation of Numerical Algorithms. *ACM Transactions on Graphics*, 22(3):908–916.

Lefebvre, S., Hornus, S., and Neyret, F. (2005). Octree Textures on the GPU. In Pharr, M., editor, *GPU Gems 2*, pages 595–613. Addison Wesley.

Litke, N., Levin, A., and Schroeder, P. (2001). Fitting Subdivision Surfaces. In *Proceedings IEEE Visualization 2001*, pages 319–324.

Sigg, C. and Hadwiger, M. (2005). Fast Third-Order Texture Filtering. In Pharr, M., editor, *GPU Gems 2*, pages 313–329. Addison Wesley.

Strengert, M., Kraus, M., and Ertl, T. (2006). Pyramid Methods in GPU-Based Image Processing. In *Proceedings Vision, Modeling, and Visualization 2006*, pages 169–176.

Williams, L. (1983). Pyramidal Parametrics. In *Proceedings ACM SIGGRAPH '83*, pages 1–11.

Zorin, D., Schröder, P., DeRose, T., Kobbelt, L., Levin, A., and Sweldens, W. (2000). Subdivision for Modeling and Animation. SIGGRAPH 2000 Course Notes.