Presented at: Proc. Computer Graphics Intl. '93, Lausanne, Switzerland

A FAST ALGORITHM FOR DIGITAL IMAGE SCALING

George Wolberg
Department of Computer Science
City College of New York
New York, NY 10031
wolberg@cs-mail.engr.ccny.cuny.edu

Henry Massalin
Microunity Corporation
Sunnyvale, CA 94089
qua@microunity.com

ABSTRACT

This paper describes a fast algorithm for scaling digital images. Large performance gains are realized by reducing the number of convolution operations, and optimizing the evaluation of those that remain. We achieve this by decomposing the overall scale transformation into a cascade of smaller scale operations. As an image is progressively scaled towards the desired resolution, a multi-stage filter with kernels of varying size is applied. We show that this results in a significant reduction in the number of convolution operations. Furthermore, by constraining the manner in which the transformation is decomposed, we are able to derive optimal kernels and implement efficient convolvers. The convolvers are optimized in the sense that they require no multiplication; only lookup table and addition operations are necessary. This accelerates convolution and greatly extends the range of filters that may be feasibly applied for image scaling. The algorithm readily lends itself to efficient software and hardware implementation.

1. INTRODUCTION

Image scaling is a geometric transformation that is used to resize digital images. This is a classic uniform resampling problem with widespread use in computer graphics and image processing. It plays a key role in many applications including pyramid construction [1, 3] supersampling, multi-grid solutions [6], and geometric normalization [7].

Despite a flurry of activity in this area, image scaling remains a computationally expensive operation. Its cost is dominated by convolution, which is necessary to bandlimit the discrete input and thereby mitigate undesirable reconstruction and aliasing artifacts. Convolution may prove to be prohibitively expensive, especially when large (high-quality) filter kernels or large scale factors are applied. This poses a quality-time tradeoff in which high-quality output must often be compromised in the interest of time. Not surprisingly, commonly used resampling filters utilize small kernels to achieve satisfactory processing rates.

This paper addresses the computational bottleneck imposed by convolution. The goal of this work is two-fold: to reduce the number of convolution operations, and to optimize the evaluation of those that remain. Although the quality-time tradeoff will always exist, accelerated image scaling promises to extend the range of high-quality resampling filters that may be feasibly applied. This stands to benefit many applications in such fields as computer graphics, computer vision, remote sensing, and medical imaging.

We present a fast algorithm for image scaling that progressively scales the image towards the desired resolution. Since each stage in the progression imposes different constraints on the resampling filter, we make use of a multi-stage filter with suitably tailored kernels that are each applied to its respective intermediate image. We demonstrate that by decomposing the overall scale transformation into a series of smaller scale operations, a significant reduction in the number of convolution operations is possible. Furthermore, by constraining the manner in which we decompose the transformation, we are able to derive optimal kernels and implement efficient convolvers. The convolvers are optimized in the sense that they require no multiplication; only lookup table accesses and addition operations are necessary. This makes the algorithm particularly attractive for both software and hardware implementation.

2. MAGNIFICATION

Image magnification is achieved by upsampling the discrete input image. The central component to this process is image reconstruction, an interpolation stage that fits a continuous function through the data samples. This serves to recover the continuous input image from its discrete representation so that it may be subjected to a higher sampling rate. In practice, magnification is usually implemented by interpolating only those points which are needed to produce the output image. For equally spaced data, interpolation can be expressed as

$$g(x) \;=\; \sum_{k=0}^{K-1} g_s(x_k)h(x-x_k)$$

where g_s is the discrete input signal defined over integer values of x_k, g is the output signal defined over real values of x, and h is the interpolation kernel weighted by K data samples. The equation given above formulates interpolation as a convolution operation. Nearly always, h is a symmetric kernel, i.e., $h(-x) = h(x)$. In addition, h must satisfy the following two conditions in order to interpolate the data: $h(0) = 1$, and $h(k) = 0$ for all integer values of $k \neq 0$. This lets $g(x_k) = g_s(x_k)$, thereby allowing $g(x)$ to pass through the data.

The computation of one interpolated point is illustrated in Fig. 1. The interpolation kernel is centered at x, the location of the point to be interpolated. We refer to the kernel as a 4-point kernel because it extends over four points. In general, any kernel which extends over N data samples will be referred to as an N-point kernel in this paper. Note that this applies when the kernel is *not* centered at a data sample. Otherwise, the kernel will actually extend over $N+1$ samples but this case is not of interest to us since $g(x)$ is known to pass through the data.

The value of the point at x is equal to the sum of the values of the discrete input scaled by the corresponding values of the interpolation kernel. This follows directly from the definition of convolution. Unfortunately, this is computationally expensive for the wide kernels that are necessary to perform accurate interpolation.

A wide range of filters are now in common use. The shortest ones include the 1-point box and 2-point triangle filters used for nearest neighbor and linear interpolation, respectively. Superior

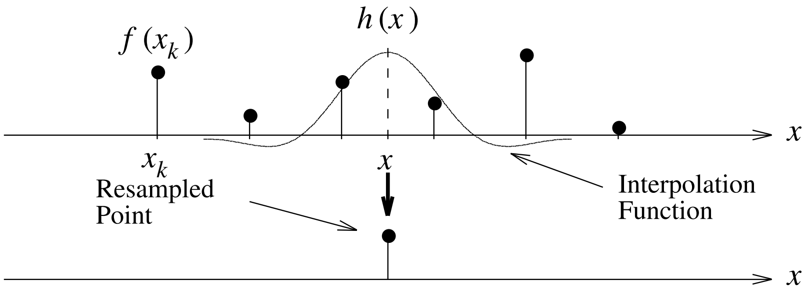


Figure 1: Interpolation of a single point.

results are possible with cubic convolution [2, 5]. More recently, a two-parameter family of cubic filters was introduced to offer a broader range of behaviors [4]. Applications requiring wider kernels typically use windowed sinc functions. Popular windows include the Hann, Hamming, Blackman, Kaiser, Gaussian, and Lanczos windows [7].

2.1 A Strategy For Reducing the Cost of Magnification

Consider the problem of magnifying a 1-D signal $g_s(x)$ by a factor of a, where $a \geq 1$. Let $g_s(x)$ consist of M samples, and let $h(x)$ be an N-point interpolation kernel. In the general case, the interpolation cost is $O(aMN)$ because $h(x)$ must be centered at aM equispaced positions in the input. For integer values of a, M of these positions is guaranteed to lie on the data samples themselves. Since the output at those positions is known, the cost of interpolation for integer values of a is $O((a-1)MN)$.

One apparent way of reducing this cost is to select a narrower kernel and compromise on reconstruction quality. This may be warranted if the spectrum $G(f)$ has a narrow bandwidth, with no high frequency components that can contribute significantly to postaliasing. Although we cannot predict the frequency content of $G(f)$, we can be sure that magnification will reduce its bandwidth relative to the new sampling rate. This suggests that a series of filters, with increasingly relaxed constraints, can be applied to a set of progressively magnified signals for reconstruction.

Fig. 2 demonstates this reconstruction. It shows three stages in the magnification of a sampled signal with spectrum $G_s(f)$. The first stage applies a high-quality reconstruction filter to effectively bandlimit the signal to f_{max}. Notice that the reconstruction filter, drawn with dashed lines, must have a sharp cutoff to retain the baseband spectrum and discard the broadband replicas. That reconstructed signal is then enlarged and sampled to produce a signal with a narrower baseband bandwidth. Since there is now more space between replicas, the next reconstruction filter applied need not have a sharp cutoff. This is depicted in the second row of Fig. 2 by a filter with a wider transition band that slowly tapers off. The third row repeats this process, making use of an even poorer filter to bandlimit the magnified signal. In the spatial domain, these results imply that ever-smaller kernels can be applied in each stage for adequate interpolation. This cycle of reconstruction, enlargement, and sampling ends once the desired signal resolution is reached.

Let's assume for now that $a = 2$ for each stage in the progression. In the example given above, three stages were necessary to achieve an overall eight-fold magnification. The total cost of interpolation is $O(MN_0 + 2MN_1 + 4MN_2)$, where N_i refers to the N_i-point kernel used in stage i. Collectively, these kernels comprise the $multi\text{-}stage\ filter$ used to bandlimit the input and

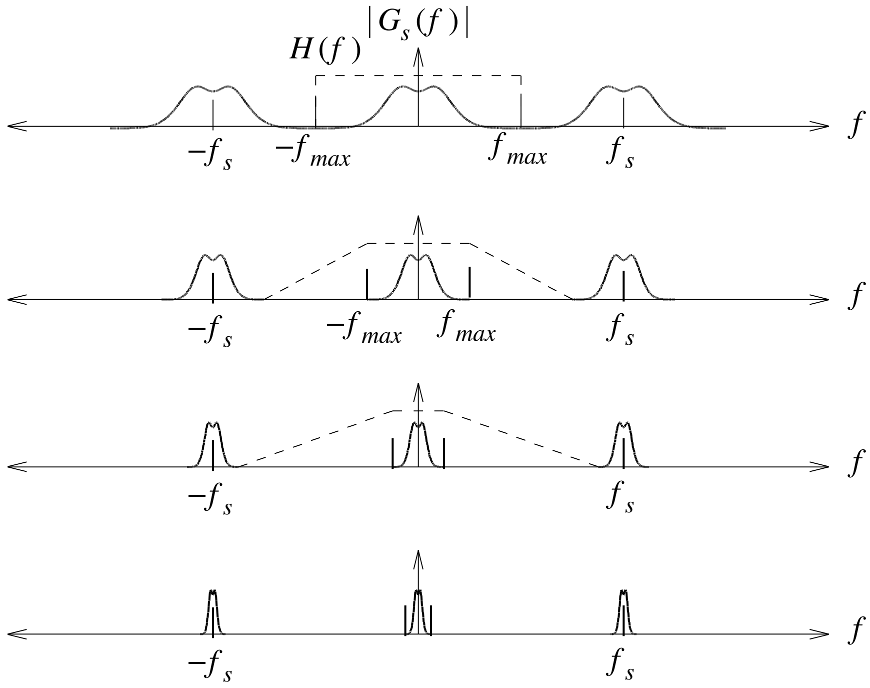


Figure 2: Spectra of a progressively magnified signal.

intermediate signals. The computational savings of using a multi-stage filter in this progressive approach is due to the fact that $N_2 < N_1 < N_0$, i.e., ever-smaller kernels are applied in successive stages. This proves to be cheaper than direct convolution with a single wide kernel ($N_0 = N_1 = N_2$) which was predicted earlier to have a cost of $O(7MN)$.

For K magnification stages, the cost of using a multi-stage filter is $O(\sum_{i=0}^{K-1} 2^i MN_i)$. This fares better than direct convolution, with a cost of $O((2^K - 1)MN_0)$. Multi-stage filters are particularly attractive for large magnification factors, where more stages induce more savings. In addition, their low cost bears significant consequences on image quality. We are not deterred, for instance, from using superior reconstruction filters with wide kernels since they are applied early in the process when the signal has the fewest samples. As the size of the signal grows, we avoid prohibitively expensive convolution by turning to smaller kernels. This balance makes it feasible to apply higher quality filters than would otherwise be possible with direct convolution.

Multi-stage filters exploit the frequency characteristics of increasingly magnified signals to perform adequate nonideal reconstruction. Although the enlargement at each stage can be arbitrary, we choose to limit it to two-fold magnification. As we have seen earlier, it is computationally cheaper to deal with integer magnification factors, with two being the smallest. In addition, a two-fold magnifier readily lends itself to efficient software and hardware implementation. The next section discusses how to derive N-point kernels for several values of N.

2.2 Deriving Optimal Kernels

Interpolation kernels are generally continuous functions that may be centered anywhere in the input. The weights applied to the neighbors must be evaluated by sampling the centered kernel at positions coinciding with the input samples. In the case of two-fold magnification, the allowable set of positions at which we must sample the interpolation kernel is greatly reduced. Since the kernel can either be centered directly on the data samples or halfway between them, the kernel must be evaluated at half-unit intervals. An example of a 6-point kernel sampled for use in two-fold magnification is shown in Fig. 3

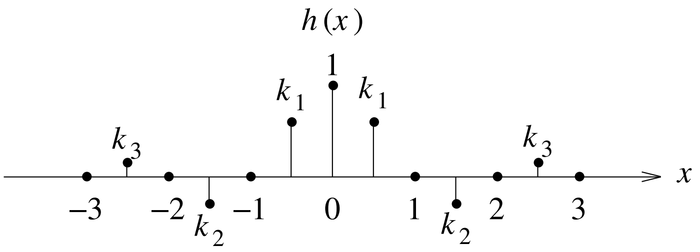


Figure 3: Kernel samples needed for two-fold magnification.

We note that those weights defined at integer values of x are applied when the kernel is centered on a data sample. In order to conform with the standard requirement of interpolation kernels, $h(0) = 1$ and $h(x) = 0$ for all nonzero integer values of x. This is necessary to guarantee that the output passes through the input. In practice, we avoid this computation at integer input coordinates by directly copying the input value to the output.

The remainder of the weights are applied when the kernel is centered halfway between input samples. This results in a symmetric set of weights that offers several important benefits. First, the weights never have to be re-evaluated during run-time because the kernel always maintains the same alignment with the data samples. Second, the number of multiplications necessary to compute one output value is cut in half to reflect the number of distinct weights. Third, and most importantly, symmetric weights allow us to express the frequency response of the kernel as a particularly simple closed-form expression. That expression shall be used to derive optimal kernel values.

We begin by noting that the kernel samples in Fig. 3 constitute a summation of symmetric impulse pairs:

$$h(x) \quad = \quad \delta(0) + \sum_{i=1}^{3} k_i \delta(x + i - .5) + k_i \delta(x - i + .5)$$

In order to compute $H(f)$, we make use of the well-known Fourier transform pair that relates symmetric impulse pairs in one domain to cosines in the other to give us:

$$H(f) \quad = \quad 1 + 2k_1 \cos(\pi f) + 2k_2 \cos(3\pi f) + 2k_3 \cos(5\pi f)$$

This result can be generalized to $2N$-point filters as follows:

$$H(f) \quad = \quad 1 + 2 \sum_{i=1}^{N} k_i \cos((2i - 1)\pi f)$$

Since our kernel samples are known to be symmetric, we now use N to refer to the number of distinct weights and $2N$ to refer to the filter support.

The above equation is a simple closed-form expression for $H(f)$ in terms of k_i. This result is significant because it now becomes possible to derive optimal kernel values by selecting those k_i's that minimize the difference between $H(f)$ and the ideal filter response. Due to point symmetry about $f = .5$, it is sufficient to directly minimize $H^2(f)$ over the stopband, i.e., $.5 \leq f \leq 1$. The square term ensures that the integrand $H(f)$ is always positive. We now use this least-squared error constraint to derive optimal kernels for several $2N$-point filters.

2.2.1 2-Point Filters

The expression for $H(f)$ for a 2-point filter is $H(f) = 1 + 2k_1 \cos(\pi f)$. Minimizing $H^2(f)$ over the interval $.5 \leq f \leq 1$ gives us $k_1 = .63662$. Note that although we only minimized $H^2(f)$ over the stopband, the passband shares its same characteristics due to point symmetry about $f = .5$. This guarantees that any improvements made in the stopband will also reflect back into the passband.

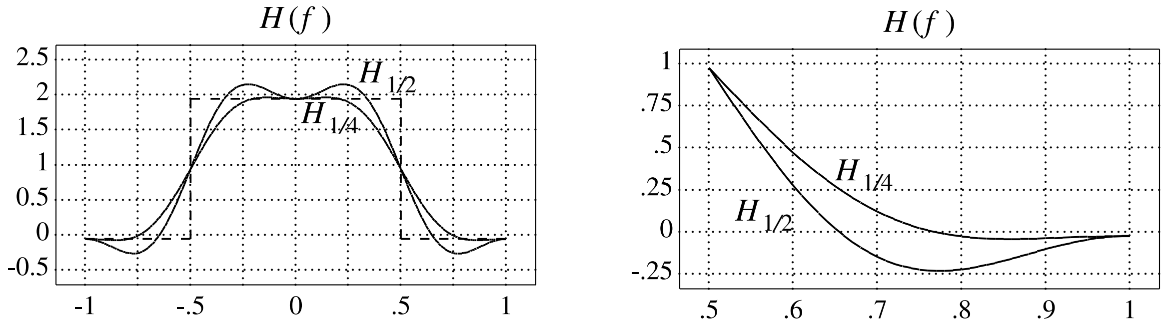
Close inspection of this result reveals a problem: $H(1) \neq 0$, or equivalently $H(0) \neq 1$. This leads to an artifact known as *sample-frequency ripple* which produces a noticable grid pattern on the image. Although the ideal filter (dashed) requires $H(f) = 0$ for $f > .5$, the failure to satisfy this condition at the sampling frequency $f = 1$ is most disturbing because the strongest component of the signal (at $f = 0$) is now aliasing as a high frequency (at $f = f_s = 1$). This problem is remedied by constraining $H(1) = 0$. That gives us $0 = 1 + 2\sum_{i=1}^{N} k_i \cos((2i-1)\pi)$ which simply reduces to $1 = 2\sum_{i=1}^{N} k_i$. The 2-point filter is now fully constrained, forcing $k_1 = .5$ i.e., the usual triangle filter. Due to the point symmetry about $f = .5$, we also satisfy $H(0) = 1$ as well.

The equation given above demonstrates that sample-frequency ripple can be designed out of the filter by requiring the sum of all of the $2N$ kernel samples to equal 1. Therefore, for a $2N$-point kernel having N points on each side, there are $N - 1$ free parameters. The last parameter must be defined as $k_N = .5 - \sum_{i=1}^{N-1} k_i$. This constraint enforces a *flat-field response*, meaning that if the digital input has constant sample values, then the reconstructed signal will also have constant value. It should be noted that many others have pointed out the benefits of this constraint [4].

2.2.2 4-Point Filters

The expression for $H(f)$ for a 4-point filter is $H(f) = 1 + 2k_1 \cos(\pi f) + (1 - 2k_1)\cos(3\pi f)$. Minimizing $H^2(f)$ over the interval $.5 \leq f \leq 1$ now yields $k_1 = .674413$. Plugging k_1 back into the above expression for k_N gives us the remaining kernel sample: $k_2 = -.174413$. The frequency response of this filter is labeled $H_{1/2}$ in Fig. 4a. Its stopband response is depicted more prominently in Fig. 4b. Although $H_{1/2}(1) = 0$ ensures that there is no sample-frequency ripple, its stopband response illustrates that there still remains plenty of frequency leakage, particularly near $f = 1$.

We are better served by introducing a transition region in which the frequency response $H(f)$ may smoothly drop from the passband to the stopband. This significantly reduces ringing because $H(f)$ can now be more closely approximated by the summation of cosines. Filter $H_{1/4}$ in Fig. 4 demonstrates the superior response gained by selecting kernel values that minimize $H^2(f)$ over the interval $.75 \leq f \leq 1$. The kernel values are now $k_1 = .587051$ and $k_2 = -.087051$. Although $H_{1/4}$ does not drop to 0 in the stopband as sharply as $H_{1/2}$, it does remain there more closely when it reaches $f = .75$. This proves to be a very important property for reconstruction filters used in magnification. As already demonstrated in Fig. 2, a flat passband/stopband response is more essential than a sharp cutoff, especially for increasingly enlarged images that can tolerate wide transition bands anyway.



(a) (b)

Figure 4: 4-point filters. (a) Spectrum; (b) Stopband response.

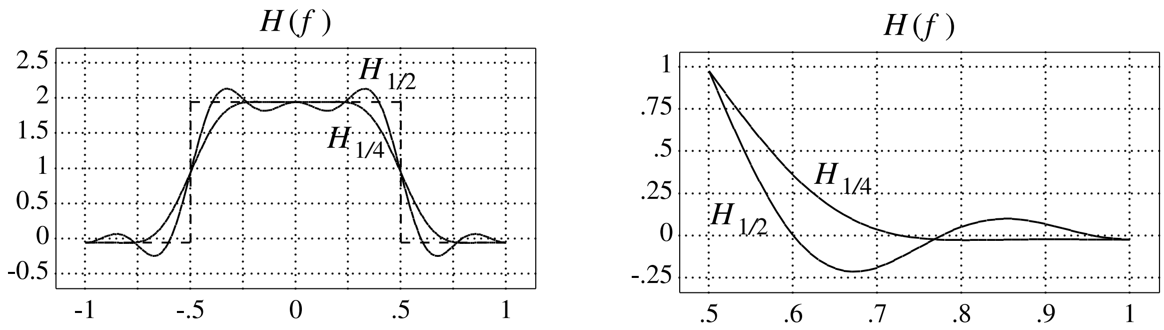
2.2.3 6-Point Filters

The expression for $H(f)$ for a 6-point filter is

$$H(f) = 1 + 2k_1 \cos(\pi f) + 2k_2 \cos(3\pi f) + (1 - 2k_1 - 2k_2) \cos(5\pi f)$$

The kernel values for $H_{1/2}$, and $H_{1/4}$ are given below. Their frequency responses are shown in Fig. 5.

	k_1	k_2	k_3
$H_{1/2}$	.619374	-.229452	.110078
$H_{1/4}$	.600816	-.123529	.022713



(a) (b)

Figure 5: 6-point filters. (a) Spectrum; (b) Stopband response.

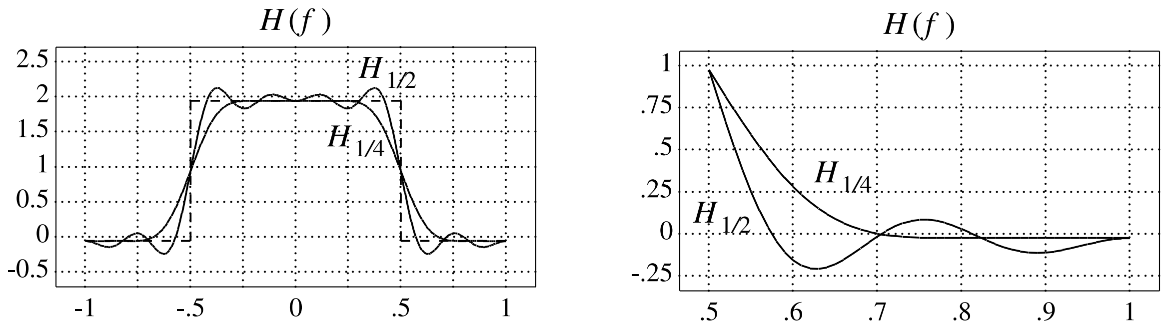
2.2.4 8-Point Filters

The expression for $H(f)$ for an 8-point filter is

$$H(f) = 1 + 2k_1\,(\pi f) + 2k_2 \cos(3\pi f) + 2k_3 \cos(5\pi f) + (1 - 2k_1 - 2k_2 - 2k_3) \cos(7\pi f)$$

The kernel values for $H_{1/2}$ and $H_{1/4}$ are given below. Their frequency responses are shown in Fig. 6.

	k_1	k_2	k_3	k_4
$H_{1/2}$	.646422	-.202404	.137126	-.081144
$H_{1/4}$	.60964	-.142133	.0390404	-.0065474



(a) (b)

Figure 6: 8-point filters. (a) Spectrum; (b) Stopband response.

2.3 Magnification Algorithm

Multi-stage filters accelerate magnification by significantly reducing the number of convolution operations with each successive stage. Further savings are possible when each stage is limited to two-fold magnification. In this section, we examine several means for accelerating an individual stage, and propose a novel method for optimizing convolution. 1-D scanlines are considered here for simplicity. This will later be extended to 2-D using a separable implementation.

Two-fold magnification is essentially a cycle of fetch-convolve operations. Fig. 7 illustrates this process on an input scanline consisting of five samples. Those samples are directly applied to even-addressed positions in a working buffer, trivially generating half of the magnified signal. The odd-addressed positions are reserved for the remaining output values that must be derived by interpolation. Since a 4-point kernel will be used in this example, padding is added to each end of the buffer so that the kernel has enough data to compute the interpolated results near the borders. The padding shown in the shaded areas of Fig. 7 is generated by reflecting the input values about the borders. Due to the nature of the interpolation, the right side needs one additional padding element more than the left side.

Convolution is applied directly on the input elements stored in the buffer, with the output interleaved among them. There is therefore a distance of two between successive input samples and successive interpolated output samples in the buffer. The two-fold magnification is implicit in this layout of the data. Since the kernel must now be applied to input data that has twice the inter-sample spacing, it too must be scaled to coincide with the samples. Therefore, the kernel samples must now lie at $x = 0, \pm 1, \pm 3, \pm 5,$ ... as opposed to $x = 0, \pm .5, \pm 1.5, \pm 2.5,$ ... as

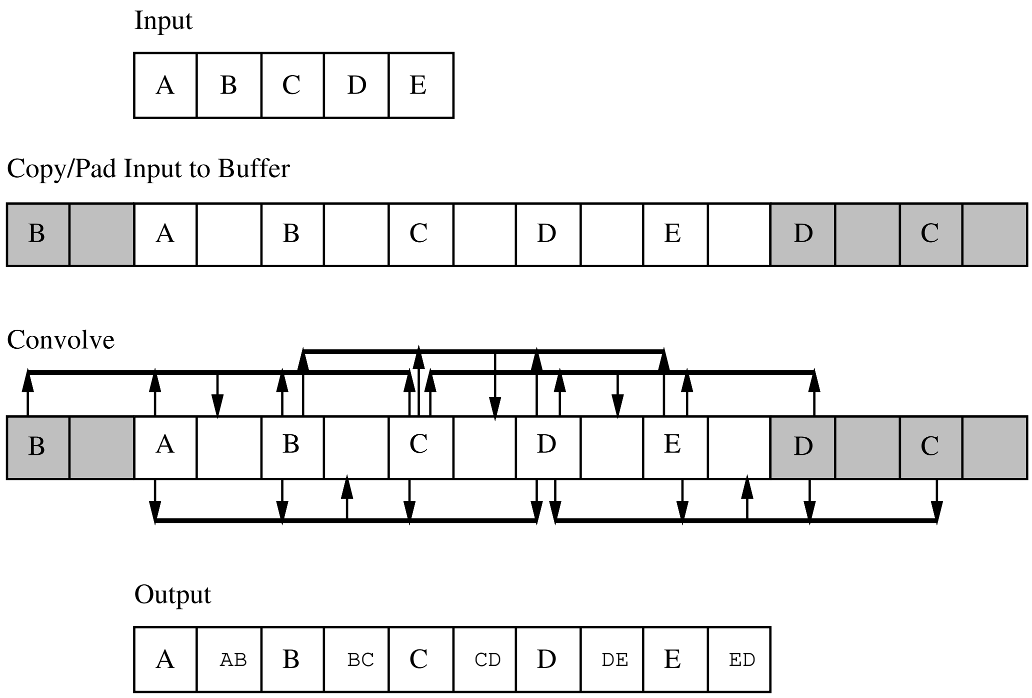


Figure 7: Two-fold magnification.

in Fig. 3. This spacing permits the kernel, which is centered at odd addresses, to be applied to consecutive input samples lying at even buffer addresses. The convolution process is depicted in the third row of Fig. 7. Note that the padded elements prove to be necessary in computing output values AB, DE, and ED.

The 1-D algorithm magnification algorithm can be readily extended to scale 2-D images. This is achieved in a separable manner. First, each horizontal scanline (row) is scaled and stored in an intermediate image I. Then each vertical scanline (column) of I is scaled to produce the final output image.

3. MINIFICATION

Image minification is achieved by downsampling the discrete input image, in a manner akin to magnification. We have already established that magnification narrows each baseband replica in the spectrum, as demonstrated in Fig. 2. Due to the reciprocal relationship between the spatial and frequency domains, minification serves to broaden each replica. This introduces complications because there now exists the possibility that the replicas in the spectrum of the minified signal may overlap. This is a symptom of undersampling, and contributes to aliasing. Unlike the spurious high frequencies retained by nonideal reconstruction filters, the spectral components passed due to undersampling are more serious since they actually corrupt the components in the original signal.

The filtering necessary to combat aliasing is known as *antialiasing*. This is most typically done by low-pass filtering the input *before* sampling at the lower rate. This method, known as *prefiltering*, bandlimits (truncates) the signal spectrum to eliminate the offending high frequencies.

Minification is usually implemented by convolving the input with a low-pass filter centered at only those sparse positions which are needed to produce the output image. This procedure is identical to the interpolation performed for magnification. This should not be surprising since interpolation and prefiltering play similar roles: they convert the discrete input into a continuous signal that is suitable for resampling.

The only difference between prefiltering and interpolation is the shape of the filter kernel. Whereas the cutoff frequency f_c is held constant at .5 cycle/pixel for magnification, f_c must now vary with the scale factor: $f_c = .5a$ cycle/pixel for minification, where $0 < a < 1$ is the scale factor. This implies that minification requires a broader filter kernel whose width N is inversely proportional to the scale factor. As with magnification, we are faced with the same challenge to reduce the cost of high-quality low-pass filtering.

3.1 A Strategy For Reducing the Cost of Minification

Consider the problem of minifying a 1-D signal $g_s(x)$ by a factor of a, where $0 < a < 1$. Let $g_s(x)$ consist of M samples, and let filter $h(x)$ be an N-point kernel. The cost of prefiltering is $O(aMN)$ because $h(x)$ must be centered at aM equispaced positions in the input. For integer values of $1/a$, these positions coincide with the data samples themselves. In that case, N must be odd to accommodate symmetric kernels.

One apparent way of reducing this cost is to select a narrower kernel and accept more aliasing artifacts. This may be warranted if $G(f)$ has a narrow bandwidth, with no high frequency components that can contribute significantly to aliasing. As already noted, though, the value of N is inversely proportional to a, and so we must consider the scale factor as well. In general, the cost of prefiltering rises with decreasing a and higher bandwidth signals.

We can reduce this cost by minifying the signal in stages in a manner akin to magnification. This approach derives its benefit by exploiting relaxed filter constraints that can be applied to a set of progressively minified signals. A key observation to be made here is that the narrow slice of the spectrum that falls below $f_c = .5a$ will ultimately span the entire baseband bandwidth in the output. It is important that this frequency band remain uncorrupted as it grows in size with each successive stage. As we shall see, this may be achieved by applying a series of filters with increasingly tighter cutoff constraints in each stage of minification.

The progressive minification that we consider downsamples the input by a factor of two in each stage. For notational convenience, we refer to the frequency band between $-f_c$ and f_c as B_0. With each successive minification stage i, B_0 doubles in size to become B_i with a range of $0 \le |f| < 2^i f_c$. In order to prevent any degradation in B_i, ideal passband and stopband responses must be defined over $0 \le |f| < 2^i f_c$ and $.5 - 2^i f_c < |f| \le .5$, respectively. The ideal passband prevents the attenuation of B_i while the stopband guards against aliasing by preventing frequency foldover in B_{i+1} after the signal in stage i undergoes 2:1 minification.

Fig. 8 demonstrates this process. It shows a sampled signal with spectrum $G_s(f)$ undergoing 8:1 minification as it passes through a succession of three 2:1 stages. The first stage applies a low-quality low-pass filter to the input. The filter, drawn with dashed lines, has a narrow passband that cuts off at $f_c = .0625$ cycles/pixel. This serves to retain B_0, the frequency band that will ultimately comprise the output spectrum. Unfortunately, the cutoff is not sharp, allowing some high frequencies to remain. That signal is then minified and resampled to produce the signal shown in the second row of Fig. 8. The high frequencies retained earlier now give rise to aliasing, as

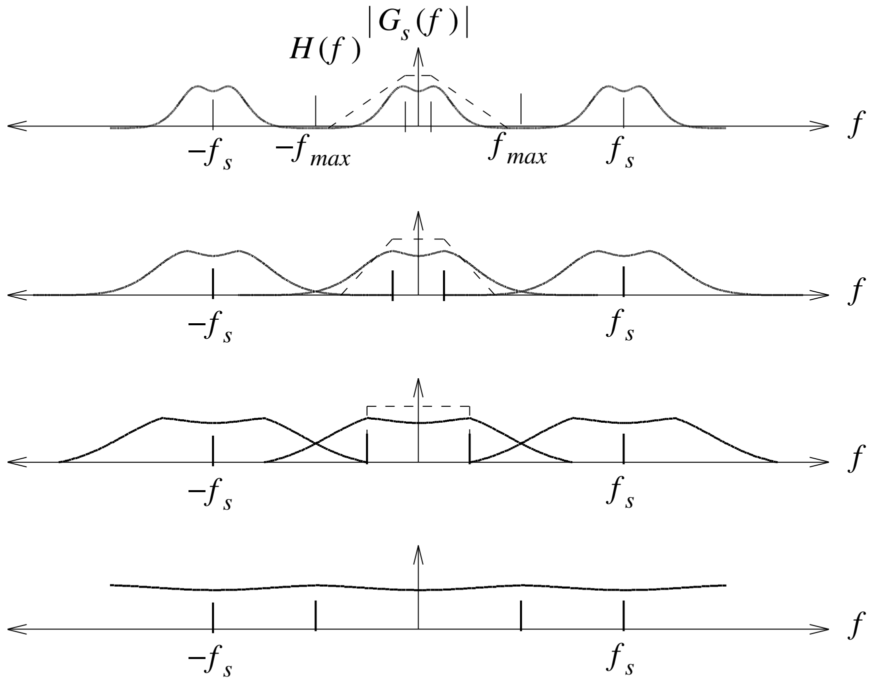


Figure 8: Spectra of a progressively minified signal.

depicted by the overlapping spectral components.

Although the signal suffers from aliasing, B_1, which spans over $|f| < .125$, remains uncorrupted. This is due to our careful choice for the stopband range. By attenuating all frequency components in the range $.5 - .0625 < f \leq .5$ in stage 0, the frequency components in the range $1 - .125 < f \leq 1$ are fully suppressed in stage 1. Therefore, frequency foldover onto the baseband from the adjacent replicas centered at $f = \pm 1$ have no effect on B_1. That signal is then prefiltered in stage 1 with a filter having ideal passband and stopband responses over $0 \leq f < .125$ and $.5 - .125 < f \leq .5$, respectively. After scale reduction, aliasing is again present in the signal, as depicted in the third row of the figure. However, frequency foldover is now limited to the range $|f| > .25$, leaving B_2 untampered. Finally, a filter having a sharp cutoff at $f = .25$ is applied to retain B_2 before it doubles in size to become B_3, the final output spectrum.

In the example given above, three stages were necessary to achieve an overall eight-fold minification. The total cost of prefiltering is $O(MN_0 + .5MN_1 + .25MN_2)$, where N_i refers to the N_i-point kernel used in stage i. Collectively, these kernels comprise the multi-stage filter used to bandlimit the input and intermediate signals. The computational savings of using a multi-stage filter in this progressive approach is due to the fact that $N_0 < N_1 < N_2$, i.e., the smallest kernels are applied in the earliest stages. This proves to be cheaper than direct convolution with a single wide kernel having extent N_2/a (with $f_c = .5a$). For K minification stages, the cost of using a multi-stage filter is $O(\sum_{i=0}^{K-1} .5^i MN_i)$. This is particularly attractive for large-scale minification, where more stages induce more savings.

Several observations can be made about this process. First, the most relaxed constraints on the low-pass filter occur in the first stage of processing. Any low-quality filter may be applied as long as it has good response over the narrow passband and stopband ranges. Note that the filter is considered to be low-quality only in the sense that it has a wide transition band. As the passband and stopband regions become wider with each successive stage, the transition band becomes narrower until finally, the filter must have a sharp cutoff.

A second observation is that this process is the dual of that for magnification. Whereas the signal grows and the kernel size decreases with each successive stage of magnification, the opposite is true for minification. This maintains a desirable balance between the sizes of the kernel and the signal to which it is applied. For instance, minification (magnification) applies a wide high-quality kernel in the last (first) stage when the signal has the fewest samples. At first glance, this is a rather surprising result for minification. It suggests that we may apply a poor filter and thereby tolerate aliasing early in the process without consequences to the final minified signal. As we have already demonstrated, this is possible because we are careful to confine aliasing to frequencies outside of B_i. Those frequency components degraded by aliasing will eventually be discarded by a high-quality filter having a sharp cutoff.

3.2 Minification Algorithm

Minification is implemented in essentially the same way as magnification. Fig. 9 illustrates the minification algorithm applied to an input scanline consisting of ten samples. Those samples are copied to a working buffer, where padding (shown shaded) is added on each side to accommodate filtering near the borders. The minification kernel $h_{min}(x)$ is centered directly on every other input sample. Due to the reciprocal relationship between the spatial and frequency domains, h_{min} is related to the magnification kernel h_{mag} as follows: $h_{min}(x) = .5h_{mag}(x/2)$. Since h_{min} is now twice as wide as h_{mag}, its kernel samples lie at $x = 0, \pm1, \pm3, \pm5, ...$ as opposed to $x = 0, \pm.5, \pm1.5, \pm2.5, ...$ as in Fig. 3. This explains why h_{min} is not applied to consecutive input samples in Fig. 9. For instance, samples B, D, E, F, and H, are filtered to replace the value of E with E'.

The minification algorithm shown above bears a strong resemblance to the magnification algorithm depicted in Fig. 7. This becomes apparent if we consider the evaluation of a single output sample. Consider, for instance, the computation of E' above. The straightforward use of h_{min} on the data gives us:

$$E' \quad = \quad \frac{E + k_1(D+F) + k_2(B+H)}{2} = \frac{E + DF}{2}$$

where DF is the output of the magnification algorithm applied to samples B, D, F, and H. This result shows that h_{min} differs only slightly from h_{mag}. Since h_{mag} is centered at odd addresses in between input samples, it does not apply $k_0 = 1$. The application of h_{mag} therefore only accounts for the weighting of samples B, D, F, and H. The central data sample E must be added explicitly. That sum is divided by two because the weights applied by h_{mag} are not appropriate for minification, i.e., recall that $h_{min} = .5h_{mag}$.

These observations allow us to recast minification as a variation of the magnification algorithm. Since we simply want to average the even-addressed input elements with the interpolated results of the odd-addressed elements, only a slight modification to the algorithm shown in Fig. 7 need be done. This implies that minification can be implemented with the same software/hardware that is used to realize magnification. The resulting minification algorithm is illustrated in Fig. 10.

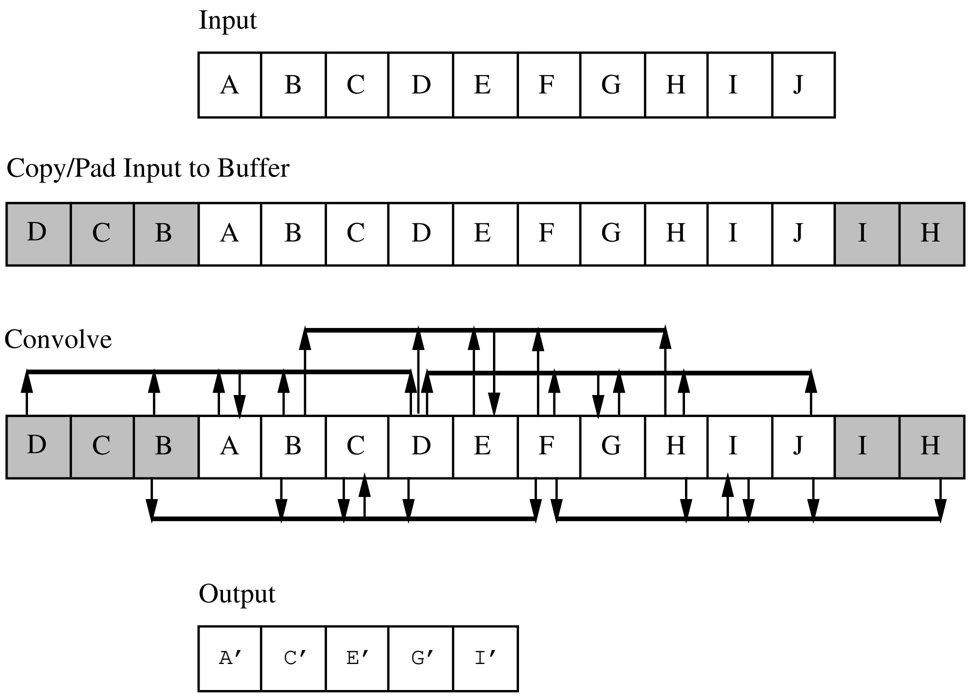


Figure 9: Two-fold minification.

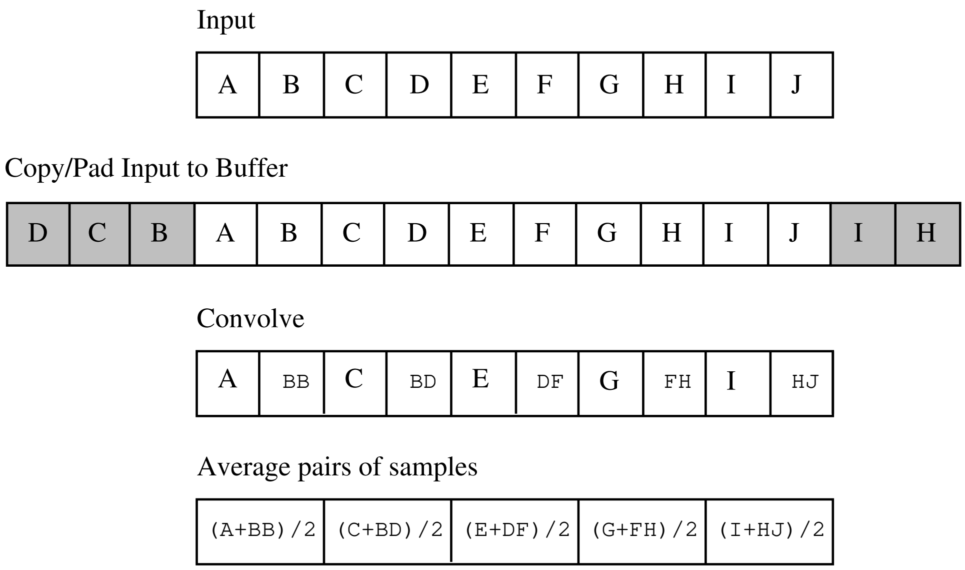


Figure 10: Two-fold minification using magnification.

3.3 Fast Convolution

The most time-consuming operation in image scaling is convolution. Although the use of multi-stage filters helps reduce this cost, the core multiply-add operation remains costly. In this section, we describe an efficient means for implementing convolution using lookup table operations. For convenience, our discussion will assume that we are convolving 8-bit data with a 6-point kernel.

A 6-point kernel has only three kernel values: k_1, k_2, and k_3. Since each kernel value k_i can be applied to an integer in the range [0, 255], we may precompute their products and store them in three lookup tables tab_i, for $1 \leq i \leq 3$. The product of data sample s with weight k_i now reduces to a simple lookup table access, e.g., $tab_i[s]$. This makes it possible to implement a 6-point convolver without multiplication; only lookup table and addition operations are necessary. In order to retain numerical accuracy during partial evaluations, we designate each 256-entry lookup table to be 10-bits wide. This accommodates 8-bit unsigned integers with 2-bit fractions.

The use of lookup tables to eliminate multiplication becomes unfeasible when a large number of distinct kernel values are required in the convolution summation. This is particularly true of general convolution. Fortunately, two-fold magnification and minification require only a few distinct kernel values. The memory demands to support the corresponding lookup tables are very modest, i.e., $256^(N/2)$ 10-bit entries, for an N-point kernel.

Further computational savings are possible by exploiting some nice properties of the convolution summation for our special two-fold rescaling problem. These properties are best understood by considering the output expressions after a 6-point kernel is applied to input samples A through H. The expanded expressions for convolution output CD, DE, and EF are given below.

$$
\begin{aligned}
CD &= k_3A + k_2B + k_1C + k_1D + k_2E + k_3F \\
DE &= k_3B + k_2C + k_1D + k_1E + k_2F + k_3G \\
EF &= k_3C + k_2D + k_1E + k_1F + k_2G + k_3H
\end{aligned}
$$

These results demonstrate a pattern: each input sample s is weighted by all k_i values during the course of advancing the kernel across the data. This is apparent for samples C and F in all three expressions. Rather than individually accessing each of the three tables with sample s, all three tables may be packed side-by-side into one wide table having 30 bits in width. This permits one index to access three packed products at once: k_3s, k_2s, and k_1s. The largest number of tables that may be packed together is limited only by the precision with which we store the products and the width of the longest integer, e.g., 32 bits on most computers.

Consider table entries for input samples A through H. Three 10-bit fields are used to pack three fixed point products in each entry. The organization of the data in this manner not only reduces the number of table accesses, but it also lends itself to a fast convolution algorithm requiring only shift and add operations. Let fwd be an in integer that is initialized with the table entry for A. After shifting it by one field (10-bits) to the right, we add the entry for B. We then right-shift that result and add the C entry. This process continues until all the input is processed. At the same time, we perform left-shift operations on the integer rev, beginning with entry D. The first few shift-add operations produce fwd and rev with the following fields. Notice that the low-order 10-bit fields of fwd contain half of the convolution summation necessary to compute the output. The other half is contained in the high-order 10-bit fields of rev. By simply adding both fields together, the output values are generated.

fwd			rev		
k_3B	$k_3A + k_2B$	$k_2A + k_1B$	$k_3E + k_2D$	$k_2E + k_1D$	k_1E
k_3C	$k_3B + k_2C$	$k_3A + k_2B + k_1C$	$k_3F + k_2E + k_1D$	$k_2F + k_1E$	k_1F
k_3D	$k_3C + k_2D$	$k_3B + k_2C + k_1D$	$k_3G + k_2F + k_1E$	$k_2G + k_1F$	k_1G
k_3E	$k_3D + k_2E$	$k_3C + k_2D + k_1E$	$k_3H + k_2G + k_1F$	$k_2H + k_1G$	k_1H

This scheme is hampered by one complication: addition may cause one field to spill into the next, thereby corrupting its value. This will happen if a field value exceeds the range [0, $2^8 - 1$]. We modify the fast convolver by adding a .25 bias to the k_2 field. The bias is removed from the computation when we add the low-order 10-bit field of fwd to the high-order 10-bit field of rev.

Operating with symmetric kernels has already been shown to reduce the number of arithmetic operations: $N/2$ multiplies and $N - 1$ for an N-point kernel. This algorithm, however, does far better. It requires no multiplication (other than that needed to initialize lut), and a total of four adds per output sample, for a 6-point kernel. Furthermore, no distinction is made between 2-, 4-, and 6-point kernels because they are all packed into the same integer. That is, a 4-point kernel is actually implemented as a 6-point kernel with $k_3 = 0$. Since there is no additional penalty for using a 6-point kernel, we are induced to use a superior (6-point) filter at low cost. Larger kernels can be assembled by cascading integers together.

4. SUMMARY AND CONCLUSIONS

We have presented an algorithm to accelerate the scaling of digital images. Since scaling is an exercise in convolution, our goal has been to relieve the computational bottleneck by reducing the number of convolution operations and optimizing the evaluation of those that remain. In the process, we have also derived optimal kernels.

The algorithm is motivated by the observation that filtering constraints change as an image is progressively scaled towards the desired resolution. At each stage of processing, a different kernel is used to reflect these changing filtering requirements. The use of these multi-stage filter kernels is less costly than direct convolution. We have demonstrated that as the size of the input grows, we avoid prohibitively expensive convolution by turning to smaller kernels. The dual is true as well: as the input becomes increasingly decimated, the kernel size grows. This maintains a desirable balance between the kernel size and the signal size, i.e., large kernels are limited to small signals.

We have shown that the multi-stage filter has many desirable properties when the scale change across each stage is limited to a factor of two. Since each kernel is now guaranteed to be symmetric, its frequency response can be defined by a simple closed-form expression $H(f)$. This permits us to derive optimal kernels by solving for the unknown kernel samples that minimize the difference between $H(f)$ and the ideal filter. Optimal N-point kernels for even values of $N \leq 8$ have been given.

A very interesting relationship is shown to exist between two-fold magnification and minification. We have shown that two-fold minification is achieved by averaging the even-addressed input elements with the interpolated results of the odd-addressed elements. This result implies that minification can be implemented with the same software/hardware used to realize magnification.

Scale factors that are not powers of two are realized by scaling to the closest power of two and then applying direct convolution.

The final result of this paper has focused on optimizing the evaluation of the convolution summation. We achieve large performance gains by packing all weighted instances of an input sample into one 32-bit integer and then using these integers in a series of shift-add operations to compute the output. In this manner, we essentially mimic a pipelined vector processor on a general 32-bit computer. This approach will likely find increased use with the forthcoming generation of 64-bit computers. The additional bits will permit us to handle wider kernels at finer precision.

REFERENCES

[1] Chin, F., A. Choi, and Y. Luo. Optimal Generating Kernels for Image Pyramids by Piecewise Fitting. *IEEE Trans. Pattern Analysis and Machine Intelligence 14*, 12 (1992), 1190–1198.

[2] Keys, Robert G. Cubic Convolution Interpolation for Digital Image Processing. *IEEE Trans. Acoust., Speech, Signal Process. ASSP-29* (1981), 1153–1160.

[3] Meer, P., E.S. Baugher, and A. Rosenfeld. Frequency Domain Analysis and Synthesis of Image Pyramid Generating Kernels. *IEEE Trans. Pattern Analysis and Machine Intelligence 9*, 4 (1987), 512–522.

[4] Mitchell, Don P., and Arun N. Netravali. Reconstruction Filters in Computer Graphics. *Computer Graphics (Proc. SIGGRAPH '88) 22*, 4 (1988), 221–228.

[5] Park, Stephen K., and Robert A. Schowengerdt. Image Reconstruction by Parametric Cubic Convolution. *Computer Vision, Graphics, and Image Processing 23* (1983), 258–272.

[6] Vaidyanathan, P.P. *Multirate Systems and Filter Banks*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[7] Wolberg, George. *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, CA, 1990.