Convolution Pyramids

Zeev Farbman
The Hebrew University

Raanan Fattal
The Hebrew University

Dani Lischinski
The Hebrew University

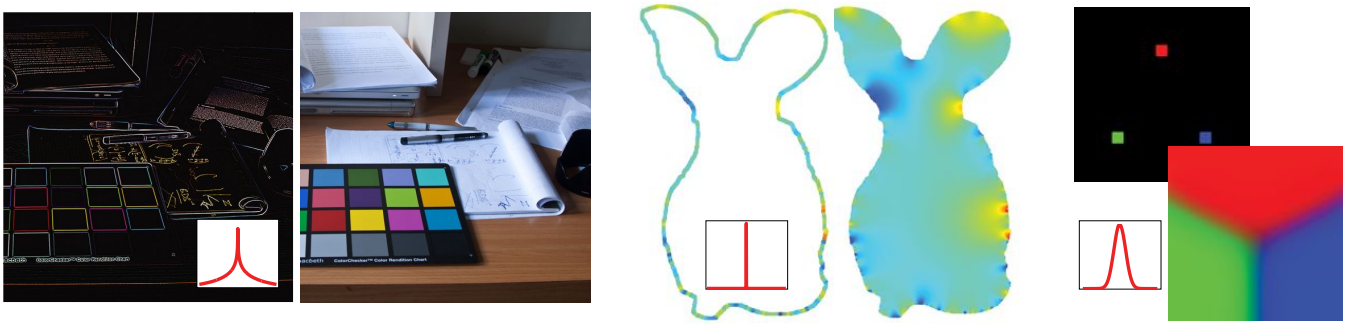


Figure 1: *Three examples of applications that benefit from our fast convolution approximation. Left: gradient field integration; Middle: membrane interpolation; Right: scattered data interpolation. The insets show the shapes of the corresponding kernels.*

Abstract

We present a novel approach for rapid numerical approximation of convolutions with filters of large support. Our approach consists of a multi-scale scheme, fashioned after the wavelet transform, which computes the approximation in linear time. Given a specific large target filter to approximate, we first use numerical optimization to design a set of small kernels, which are then used to perform the analysis and synthesis steps of our multiscale transform. Once the optimization has been done, the resulting transform can be applied to any signal in linear time. We demonstrate that our method is well suited for tasks such as gradient field integration, seamless image cloning, and scattered data interpolation, outperforming existing state-of-the-art methods.

Keywords: convolution, Green's functions, Poisson equation, seamless cloning, scattered data interpolation, Shepard's method

Links: DL PDF WEB

1 Introduction

Many tasks in computer graphics and image processing involve applying large linear translation-invariant (LTI) filters to images. Common examples include low- and high-pass filtering of images, and measuring the image response to various filter banks. Some less obvious tasks that can also be accomplished using large LTI filters are demonstrated in Figure 1: reconstructing images by integrating their gradient field [Fattal et al. 2002], fitting a smooth membrane to interpolate a set of boundary values [Pérez et al. 2003; Agarwala 2007], and scattered data interpolation [Lewis et al. 2010].

While convolution is the most straightforward way of applying an LTI filter to an image, it comes with a high computational cost: $O(n^2)$ operations are required to convolve an n-pixel image with a kernel of comparable size. The Fast Fourier Transform offers a more efficient, $O(n \log n)$ alternative for periodic domains [Brigham 1988]. Other fast approaches have been proposed for certain special cases. For example, Burt [1981] describes a multiscale approach, which can *approximate* a convolution with a large Gaussian kernel in $O(n)$ time at hierarchically subsampled locations. We review this and several other related approaches in the next section.

In this work, we generalize these ideas, and describe a novel multiscale framework that is not limited to approximating a specific kernel, but can be tuned to reproduce the effect of a number of useful large LTI filters, while operating in $O(n)$ time. Specifically, we demonstrate the applicability of our framework to convolutions with the Green's functions that span the solutions of the Poisson equation, inverse distance weighting kernels for membrane interpolation, and wide-support Gaussian kernels for scattered data interpolation. These applications are demonstrated in Figure 1.

Our method consists of a multiscale scheme, resembling the Laplacian Pyramid, as well as certain wavelet transforms. However, unlike these more general purpose transforms, our approach is to custom-tailor the transform to directly approximate the effect of a given LTI operator. In other words, while previous multiscale constructions are typically used to transform the problem into a space where it can be better solved, in our approach the transform itself directly yields the desired solution.

Specifically, we repeatedly perform convolutions with three small, fixed-width kernels, while downsampling and upsampling the image, so as to operate on all of its scales. The weights of each of these kernels are numerically optimized such that the *overall* action of the transform best approximates a convolution operation with some *target filter*. The optimization only needs to be done once for each target filter, and then the resulting multiscale transform may be applied to any input signal in $O(n)$ time. The motivation behind this design was to avoid dealing with the analytical challenges that arise from the non-idealness of small finite filters, on the one hand, while attempting to make the most out of the linear computational budget, on the other.

Our scheme's ability to closely approximate convolutions with the

free space Green's function [Evans 1998] is high enough to allow us to invert a certain variant of the Poisson equation. We show that for this equation our approach is faster than state-of-the-art solvers, such as those based on the multigrid method [Kazhdan and Hoppe 2008], offering improved accuracy for a comparable number of operations, and a higher ease of implementation.

While we demonstrate the usefulness of our approach on a number of important applications, and attempt to provide some insights on the limits of its applicability later in the paper, our theoretical analysis of these issues is certainly incomplete, and is left as an interesting avenue for future work.

2 Background

Besides the straightforward $O(n^2)$ implementation of convolution, there are certain scenarios where it can be computed more efficiently. Over periodic domains, every convolution operator can be expressed as a circulant Topelitz matrix, which is diagonalized by the Fourier basis. Thus, convolutions with large kernels over periodic domains may be carried out in $O(n \log n)$ time using the Fast Fourier Transform [Brigham 1988].

Convolution with separable 2D kernels, which may be expressed as the outer product of two 1D kernels, can be sped up by first performing a 1D horizontal convolution, followed by one in the vertical direction (or vice versa). Thus, the cost is $O(kn)$, where k is the length of the 1D kernels. Non-separable kernels may be approximated by separable ones using the SVD decomposition of the kernel [Perona 1995].

B-spline filters of various orders and scales may be evaluated at constant time per pixel using repeated integration [Heckbert 1986] or generalized integral images [Derpanis et al. 2007].

Many methods have been developed specifically to approximate convolutions with Gaussian kernels, because of their important role in image processing [Wells, III 1986]. Of particular relevance to this work is the *hierarchical discrete correlation* scheme proposed by Burt [1981]. This multiscale scheme approximates a Gaussian pyramid in $O(n)$ time. Since convolving with a Gaussian band-limits the signal, interpolating the coarser levels back to the finer ones provides an approximation to a convolution with a large Gaussian kernel in $O(n)$. However, Burt's approximation is accurate only for Gaussians of certain widths.

Burt's ideas culminated in the *Laplacian Pyramid* [Burt and Adelson 1983], and were later connected with wavelets [Do and Vetterli 2003]. More specifically, the Laplacian pyramid may be viewed as a *high-density wavelet transform* [Selesnick 2006]. These ideas are also echoed in [Fattal et al. 2007], where a multiscale bilateral filter pyramid is constructed in $O(n \log n)$ time.

The Laplacian pyramid, as well as various other wavelet transforms [Mallat 2008] decompose a signal into its low and high frequencies, which was shown to be useful for a variety of analysis and synthesis tasks. In particular, it is possible to approximate the effect of convolutions with large kernels. However, even though these schemes rely on repeated convolution (typically with small kernels), they are not translation-invariant, i.e., if the input is translated, the resulting analysis and synthesis will not differ only by a translation. This is due to the subsampling operations these schemes use for achieving their fast $O(n)$ performance. Our scheme also uses subsampling, but in a more restricted fashion, which was shown by Selesnick [2006] to increase translation invariance.

One scenario, which gained considerable importance in the past decade, is the recovery of a signal from its convolved form, e.g., an image from its gradient field [Fattal et al. 2002]. This gives rise to a translation-invariant system of linear equations, such as the Poisson Equation. The corresponding matrices are, again, Topelitz matrices, which can be inverted in $O(n \log n)$ time, using FFT over periodic domains. However, there are even faster $O(n)$ solvers for handling specific types of such equations over both periodic and non-periodic domains. The multigrid [Trottenberg et al. 2001] method and hierarchical basis preconditioning [Szeliski 1990] achieve linear performance by operating at multiple scales. A state-of-the-art multigrid solver for large gradient domain problems is described by Kazhdan and Hoppe [2008], and a GPU implementation for smaller problems is described by McCann and Pollard [2008].

Since the matrices in these linear systems are circulant (or nearly circulant, depending on the domain), their inverse is also a circulant convolution matrix. Hence, in principle, the solution can be obtained (or approximated) by convolving the right-hand side with a kernel, e.g., the Green's function in cases of the infinite Poisson equation. Our approach enables accurately approximating the convolution with such kernels, and hence provides a more efficient, and easy to implement alternative for solving this type of equations.

Gradient domain techniques are also extensively used for seamless cloning and stitching [Pérez et al. 2003], yielding a similar linear system, but with different (Dirichlet) boundary conditions, typically solved over irregularly shaped domains. In this scenario, the solution often boils down to computing a smooth membrane that interpolates a set of given boundary values [Farbman et al. 2009]. Agarwala [2007] describes a dedicated quad-tree based solver for such systems, while Farbman *et al.* [2009] avoid solving a linear system altogether by using mean-value interpolation. In Section 5 we show how to construct such membranes even more efficiently by casting the problem as a ratio of convolutions [Carr et al. 2003].

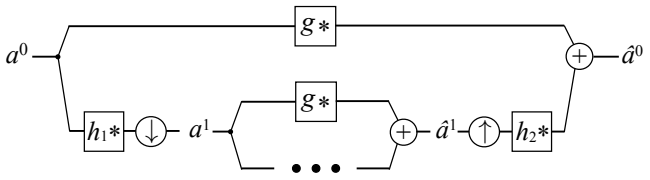
This approach can also be useful in more general scattered data interpolation scenarios [Lewis et al. 2010], when there are many data points to be interpolated or when a dense evaluation of the interpolated function is needed. Our work provides an efficient approximation for scattered data interpolation where it is important to use large convolution filters that propagate the information to the entire domain.

3 Method

In this section we describe our framework in general and motivate our design decisions. The adaptation of the framework to several specific problems in computer graphics is discussed in Section **??**.

Linear translation-invariant filtering is used extensively in computer graphics and image processing for scale separation. Indeed, many image analysis and manipulation algorithms cannot succeed without a suitably constructed multiscale (subband) decomposition. In most applications, the spectral accuracy needed when extracting a particular band is proportional to the band level: high-frequency components are typically extracted with small compact filters that achieve low localization in the frequency domain, while low frequencies are typically needed at a higher spectral accuracy and, hence, large low-pass filters are used.

Subband architectures such as wavelet transforms and Laplacian pyramids rely on a spectral "divide-and-conquer" strategy where, at every scale, the spectrum is partitioned via filtering and then the lower end of the spectrum, which may be subsampled without a major loss of data, is further split recursively. The subsampling step allows extracting progressively lower frequencies using filters of small fixed length, since the domain itself is shrinking and distant points become closer, due to the use of subsampling. This approach leads to highly efficient linear-time processing of signals and is capable of isolating low-frequency modes up to the largest



Algorithm 1 *Our multiscale transform*

Determine the number of levels L
{Forward transform (analysis)}
$a^0 = a$
for each level $l = 0 \dots L-1$ **do**
  $a_0^l = a^l$
  $a^{l+1} = \downarrow(h_1 * a^l)$
end for
{Backward transform (synthesis)}
$\hat{a}^L = g * a^L$
for each level $l = L-1 \dots 0$ **do**
  $\hat{a}^l = h_2 * (\uparrow \hat{a}^{l+1}) + g * a_0^l$
end for

Figure 2: *Our subband architecture flow chart and pseudocode.*

spatial scale (the DC component of the image).

While it may not be a major obstacle for some applications, these decompositions suffer from two main shortcomings. First, the resulting transformed coordinates, and therefore the operations performed using these coordinates, are not invariant to translation. Thus, unlike convolution, shifting the input image may change the outcome by more than just a spatial offset. Second, to achieve the $O(n)$ running time, it is necessary to use finite impulse response filters. These filters can achieve some spatial and spectral localization but do not provide an ideal partitioning of the spectrum. As we demonstrate here, these properties are critical for some applications (such as solving the Poisson equation) and for the design of particularly shaped kernels. In fact, these two shortcomings dictate the design of the new scheme we describe below, whose aim is to achieve an optimal approximation of certain translation-invariant operators under the computational cost budget of $O(n)$.

Figure 2 illustrates the multiscale filtering scheme that we use and is inspired by the architectures mentioned above. The forward transform consists of convolving the signal with an *analysis* filter h_1, and subsampling the result by a factor of two. This process is then repeated on the subsampled data. Additionally, an unfiltered and unsampled copy of the signal is kept at each level. Formally, at each level we compute:

$$a_0^l = a^l, \tag{1}$$

$$a^{l+1} = \downarrow(h_1 * a^l), \tag{2}$$

where the superscript l denotes the level in the hierarchy and a_0^l is the unfiltered data kept at each level, and $\downarrow$ denotes the subsampling operator. The transform is initiated by setting $a^0 = a$, where a is the input signal to be filtered.

The backward transformation (*synthesis*) consists of upsampling by inserting a zero between every two samples, followed by a convolution, with another filter, h_2. We combine this interpolated signal with the signal stored at that level after it is convolved with a third filter g, i.e.,

$$\hat{a}^l = h_2 * (\uparrow \hat{a}^{l+1}) + g * a_0^l, \tag{3}$$

where $\uparrow$ denotes the zero upsampling operator. Note that unlike in most subband architectures, our synthesis is *not* intended to invert the analysis and reproduce the input signal a, but rather the combined action of the forward and backward transforms $\hat{a}^0$ is meant to approximate the result of some specific linear translation-invariant filtering operation applied to the input a.

This scheme resembles the discrete fast wavelet transform, up to the difference that, as in the Laplacian pyramid, we do not subsample the decomposed signal (the high-frequency band in these transformations and the all-band a_0^l in our case). Similarly to the high density wavelet transformation [Selesnick 2006], this choice is made in order to minimize the subsampling effect and to increase the translation invariance.

Assuming ideal filtering was computationally feasible, h_1 and h_2 could be chosen so as to perfectly isolate and reconstruct progressively lower frequency bands of the original data, in which case the role of g would be to approximate the desired filtering operation. However, since we want to keep the number of operations $O(n)$, the filters h_1, h_2 and g must be finite and small. This means that the design of these filters must account for this lack of idealness and the resulting complex interplay between different frequency bands. Thus, rather than deriving the filters h_1, h_2 and g from explicit analytic filter design methodologies, we numerically optimize these filters such that their joint action will best achieve the desired filtering operation. In summary, our approach consists of identifying and allocating a certain amount of computations with reduced amount of subsampling, while remaining in the regime of $O(n)$ computations and then optimizing this allocated computations to best approximate convolution with large filters.

3.1 Optimization

In order to approximate convolutions with a given target kernel f, we seek a set of kernels $\mathscr{F} = \{h_1, h_2, g\}$ that minimizes the following functional

$$\arg\min_{\mathscr{F}} \|\hat{a}^0_{\mathscr{F}} - f * a\|_2, \tag{4}$$

where $\hat{a}^0_{\mathscr{F}}$ is the result of our multiscale transform with the kernels $\mathscr{F}$ on some input a. In order to carry out this optimization it remains to determine the types of the kernels and the number of their unknown parameters. The choice of the training data a depends on the application, and will be discussed in the next section. Note that once this optimization is complete and the kernels have been found, our scheme is ready to be used for approximating $f * a$ on *any* given signal a. All of the kernels used to produce our results are provided in the supplementary material, and hence no further optimization is required to use them in practice.

In order to minimize the number of total arithmetic operations, the kernels in $\mathscr{F}$ should be small and separable. The specific choices reported below correspond, in our experiments, to a good trade-off between operation count and approximation accuracy. Using larger and/or non-separable filters increases the accuracy, and hence the specific choice depends on the application requirements. Remarkably, we obtain rather accurate results using separable kernels in $\mathscr{F}$, even for non-separable target filters f. This can be explained by the fact that our transform sums the results of these kernels, and the sum of separable kernels is not necessarily separable itself.

Furthermore, the target filters f that we approximate have rotational and mirroring symmetries. Thus, we explicitly enforce symmetry on our kernels, which reduces the number of degrees of freedom and the number of local minima in the optimization. For example, a separable 3-by-3 kernel is defined by only two parameters ($[a,b,a] \cdot [a,b,a]^\top$) and a separable 5-by-5 kernel by three parameters. As for non-separable kernels, a 5-by-5 kernel with these symmetries is defined by six parameters. Depending on the application, the nature of the target filter f and the desired approximation accuracy,

we choose different combinations of kernels in $\mathscr{F}$. For example, we used between 6 and 9 parameters in order to produce each of the kernel sets used in Sections 4 and 5.

To perform the optimization we use the BFGS quasi-Newton method [Shanno 1970].[1] To provide an initial guess we set the kernels to Gaussians. Each level is zero-padded by the size of the largest kernel in $\mathscr{F}$ at each level before filtering. Typically, we start the optimization process on a low-resolution grid (32x32). This is fast, but the result may be influenced by the boundaries and the specific padding scheme used. Therefore, the result is then used as an initial guess for optimization over a high resolution grid, where the influence of the boundaries becomes negligible.

In the next sections we present three applications that use this approach to efficiently approximate several different types of filters.

4 Gradient integration

Many computer graphics applications manipulate the gradient field of an image [Fattal et al. 2002; McCann and Pollard 2008; Orzan et al. 2008]. These applications recover the image u that is closest in the L_2 norm to the modified gradient field $\mathbf{v}$ by solving the Poisson equation:

$$\Delta u = \operatorname{div} \mathbf{v}, \tag{5}$$

where Δ is a discrete Laplacian operator. Typically von Neumann boundary conditions, $\partial u/\partial \mathbf{n} = 0$ where $\mathbf{n}$ is the unit boundary normal vector, are enforced.

Green's functions $G(\mathbf{x},\mathbf{x}')$ define the fundamental solutions to the Poisson equation, and are defined by

$$\Delta G(\mathbf{x},\mathbf{x}') = \delta(\mathbf{x}-\mathbf{x}'), \tag{6}$$

where δ is a discrete delta function. When (5) is defined over an infinite domain with no boundary conditions, the Laplacian operator becomes spatially-invariant, and so does its inverse. In this case, the Green's function becomes translation-invariant depending only on the (scalar) distance between $\mathbf{x}$ and $\mathbf{x}'$. In two dimensions this *free-space* Green's function is given by

$$G(\mathbf{x},\mathbf{x}') = G(\|\mathbf{x}-\mathbf{x}'\|) = \frac{1}{2\pi} \log \frac{1}{\|\mathbf{x}-\mathbf{x}'\|}. \tag{7}$$

Thus, for a compactly supported right-hand side, the solution of (5) is given by the convolution

$$u = G * \operatorname{div} \mathbf{v}. \tag{8}$$

This solution is also known as the Newtonian potential of $\operatorname{div}\mathbf{v}$ [Evans 1998]. It can be approximated efficiently using our convolution pyramid, for example by zero-padding the right-hand side.

The difference between this formulation and imposing von Neumann boundary conditions is that the latter enforces zero gradients on the boundary, while in the free-space formulation zero gradients on the boundary can be encoded in the right-hand side serving only as a soft constraint. Forcing the boundary gradients to zero is a somewhat arbitrary decision, and one might argue that it may be preferable to allow them to be determined by the function inside the domain. In fact, a similar approach to image reconstruction from its gradients was used by Horn [1974] in his seminal work.

To perform the optimization in (4) we chose a natural greyscale image I, and set the training signal a to the divergence of its gradient field: $a = \operatorname{div}\nabla I$, while $f * a = I$. We chose a natural image I since the training data a should not be too localized in space (such as a delta function). Our scheme is not purely translation-invariant and a localized a might lead to an over-fitted optimum that would not preform as well in other regions where the training signal a is zero. Natural images are known to be stationary signals with no absolute spatial dependency and, moreover, this is the class of signals we would like to perform best on. We experimented with several different natural images as training data, but did not observe a significant difference in the accuracy of the resulting optimized kernel sets. We also attempted averaging the kernels obtained from different training signals, but that also did not have a significant effect on the accuracy. *[* **Dani:** *Zeev, is the last statement true?]*

[1] Specifically, we use `fminunc` from Matlab's Optimization Toolbox.

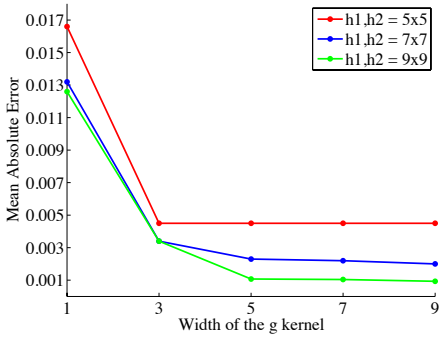


Figure 3: *Reconstruction accuracy for kernel sets of different sizes.*

We experimented with a number of different kernel size combinations for the kernel set $\mathscr{F}$. The results of these experiments are summarized in Figure 3. These results indicate that increasing the widths of the h_1, h_2 kernels reduces the error more effectively than increasing the width of g. We found the set $\mathscr{F}_5$ that uses a 5-by-5 kernel for h_1, h_2 and a 3-by-3 kernel for g to be particularly attractive, as it produces results that are visually very close to the ground truth while employing compact kernels. A more accurate solution (virtually indistinguishable from the ground truth) may be obtained by increasing the kernel sizes to 7-by-7/5-by-5 ($\mathscr{F}_7$) in exchange for a modest increase in running time, or even further to 9-by-9/7-by-7 ($\mathscr{F}_9$). It should be noted that we have no evidence that the best kernels we were able to obtain in our experiments correspond to the global optimum, so it is conceivable that even better accuracy may be achieved using another optimization scheme, or a different initial guess.

Table 1 summarizes the running times of our optimized CPU implementation for kernel sets $\mathscr{F}_5$ and $\mathscr{F}_7$ on various image sizes.

grid size (millions)	time (5x5/3x3) (sec, single core)	time (7x7/5x5) (sec, single core)
0.26	0.0019	0.00285
1.04	0.010	0.015
4.19	0.047	0.065
16.77	0.19	0.26
67.1	0.99	1.38

Table 1: *Performance statistics for convolution pyramids. Reported times exclude disk I/O and were measured on a 2.3GHz Intel Core i7 (2820qm) MacBook Pro.*

Figure 4(a) shows a reconstruction of the free-space Green's function obtained by feeding our method a centered delta function as input. A comparison to the ground truth (the solution of (6)) reveals that the mean absolute error (MAE) is quite small even for the $\mathscr{F}_5$ kernel set: *[* **Dani:** *Zeev, do you want to add a third column for F9 here?]*

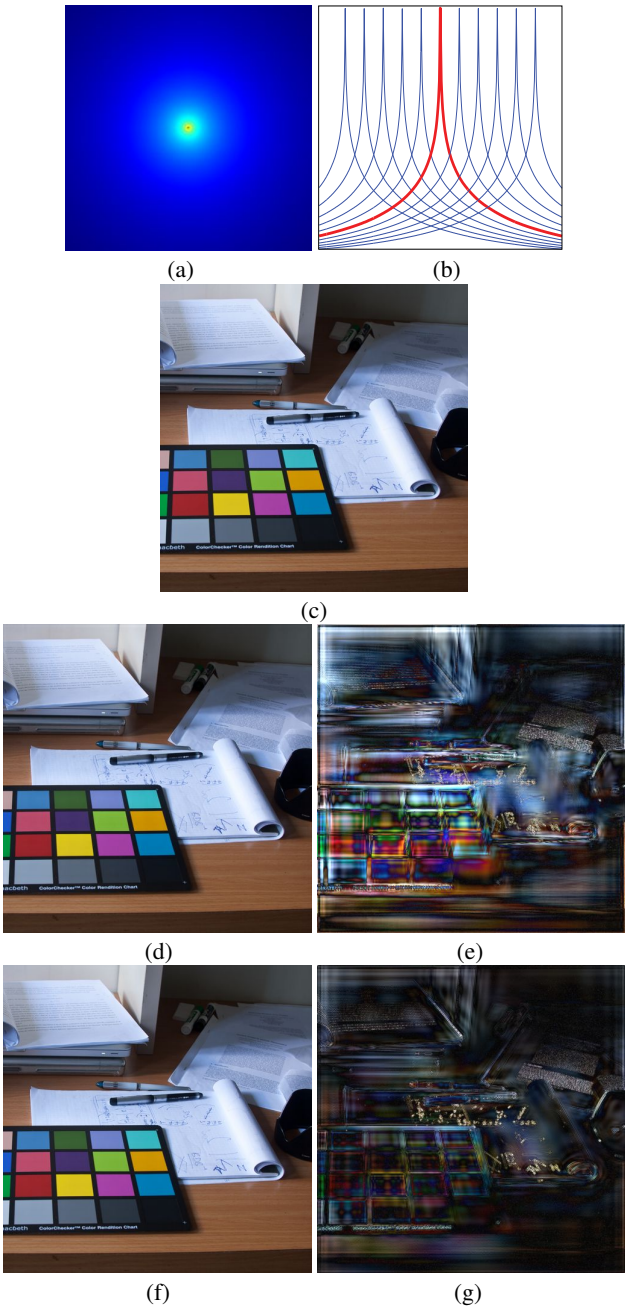


Figure 4: *Gradient field integration. (a) Reconstruction of the Green's function. (b) Our reconstructions exhibit spatial invariance. (c) A natural test image. (d) Reconstruction of (c) from its gradient field using $\mathscr{F}_5$. (e) Absolute errors (magnified by x50). (f)-(g) Reconstruction of (c) from its gradient field using $\mathscr{F}_7$, and the corresponding errors.*

grid size	$\mathscr{F}_5$ MAE	$\mathscr{F}_7$ MAE
1024^2	0.0031	0.0019
2048^2	0.0027	0.0015

Changing the position of the delta function's spike along the x-axis and plotting a horizontal slice through the center of each resulting reconstruction (Figure 4(b)), reveals that our reconstructions exhibit very little spatial variance.

Figure 4(c) shows a natural image (different from the one used as training data by the optimization process), whose gradient field divergence was given as input to our method. Two reconstructions, with $\mathscr{F}_5$ and $\mathscr{F}_7$, and their corresponding errors are shown in images (d)–(g). The mean absolute error of our reconstruction is 0.0052 with $\mathscr{F}_5$ and 0.0027 with $\mathscr{F}_7$. *[* **Dani:** *Zeev, please update these numbers]* Visually, it is difficult to tell the difference between either reconstruction and the original image.

Figure 5: *Gradient domain HDR compression. Left column: results using a Poisson solver with von Neumann boundary conditions. Right column: results using our approximate convolution.*

In Figure 5 we show HDR compression results produced using gradient domain image compression [Fattal et al. 2002]. The results on the left were obtained using a Poisson solver with von Neumann boundary conditions, while those on the right uses our approximation. In both cases, we use the same post-processing, which consists of stretching the result so 0.5% of the pixels are dark-clipped. While some differences between the results may be noticed, it is hard to prefer one over the other. Additional results are included in the supplementary material.

In this context of gradient field integration, our method presents a faster alternative to linear solvers, while also being easier to implement. Figure 6 plots the reconstruction error as a function of running time of our method to that of two other solvers: the in-core version of Kazhdan and Hoppe's solver [2008], and the implementation of a standard multigrid solver that was used by Fattal *et al.* [2002]. A comparison performed by Bhat *et al.* [2008] indicates that the in-core version of the Kazhdan-Hoppe solver is one of the fastest currently available CPU-based multigrid solvers for this kind of reconstruction. Indeed, the plots in Figure 6 confirm its superior convergence. However, for most practical graphics applications, a reconstruction error in the range of 0.001–0.0001 is sufficient, and our method is able to achieve this accuracy considerably faster than the Kazhdan-Hoppe solver, while being much simpler to implement and to port to the GPU.

McCann and Pollard [2008] describe a GPU-based implementation of a multigrid solver, which, at the time, enabled to integrate a one-megapixel image about 20 times per second, supporting interactive

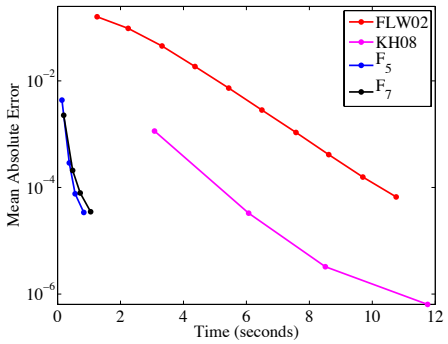


Figure 6: *Error vs. time for our method with $\mathscr{F}_5$ and $\mathscr{F}_7$ kernel sets, the in-core streaming multigrid solver (KH08), and the multigrid solver used by Fattal et al. [2002] (FLW02).*

gradient domain painting. Our current single core CPU-based implementation already enables to integrate such an image 33 times per second. We expect a GPU implementation to bring forth a significant additional speedup factor.

Since the exact running times depend greatly on the desired accuracy and on the implementation specifics, it is important to gain a better understanding of the speedup in terms of operation counts. A standard multigrid solver, such as the one used by Fattal *et al.* [2002], performs restriction and prolongation operations to change grid resolutions, with a few relaxation iterations and one residual computation at each resolution [Trottenberg et al. 2001]. Although the restriction/prolongation kernels may vary between different schemes, their sizes (and hence costs) are typically comparable to those of our h_1 and h_2 kernels. The cost of a single Gauss-Seidel relaxation iteration is $6n$ operations (multiplications and additions). Together with the residual computation this yields $18n$ operations on the fine resolution grid, when a single V-cycle is performed with only one relaxation iteration before and after each restriction (known as the V(1,1) scheme). In comparison, applying the 3-by-3 g kernel in our method costs $12n$ operations on the finest resolution grid.

Thus, our method's operation count is smaller than that of even the simplest multigrid V-cycle, while the accuracy we achieve is better, as demonstrated in Figure 6. In order to achieve higher accuracy, a multigrid solver might perform more V-cycles or use more relaxation iterations, with a corresponding increase in the number of operations. Similarly, we can also apply our transform iteratively. Specifically, at each iteration the transform is re-applied on the residual, and the result is added to the solution. This is how the curves corresponding to $\mathscr{F}_5$ and $\mathscr{F}_7$ in Figure 6 were produced.

5 Boundary interpolation

Applications such as seamless image cloning [Pérez et al. 2003] and image stitching [Szeliski 2006] can be formulated as boundary value problems and effectively solved by constructing a smooth membrane that interpolates the differences along a seam between two images across some region of interest [Farbman et al. 2009].

Such membranes have originally been built by solving the Laplace equation [Pérez et al. 2003]. However, Farbman *et al.* [2009] showed that other smooth interpolation schemes, such as mean-value interpolation may be used as well, offering computational advantages. Here we show how to construct a suitable membrane even faster by approximating Shepard's scattered data interpolation method [Shepard 1968] using a convolution pyramid.

Let Ω denote a region of interest (on a discrete regular grid), whose boundary values are given by $b(\mathbf{x})$. Our goal is to smoothly interpolate these values to all grid points inside Ω. Shepard's method defines the interpolant r at $\mathbf{x}$ as a weighted average of known boundary values:

$$r(\mathbf{x}) = \frac{\sum_k w_k(\mathbf{x}) b(\mathbf{x}_k)}{\sum_k w_k(\mathbf{x})}, \tag{9}$$

where $\mathbf{x}_k$ are the boundary points. In our experiments we found that a satisfactory membrane interpolant is obtained by using the following weight function:

$$w_k(\mathbf{x}) = w(\mathbf{x}_k, \mathbf{x}) = 1/d(\mathbf{x}_k, \mathbf{x})^3, \tag{10}$$

which has a strong spike at the boundary point x_k and decays rapidly away from it.

Naive evaluation of (9) is expensive: assuming that the number of boundary values is K and the number of points in Ω is n, the computational cost is $O(Kn)$. Our multiscale transform allows us to approximate the computation in $O(n)$ time. Following Carr *et al.* [2003], our first step is to re-write Shepard's method in terms of convolutions. We first define $\hat{r}$ as an extension of b to the entire domain:

$$\hat{r}(\mathbf{x}_i) = \begin{cases} b(\mathbf{x}_k), & \text{for } \mathbf{x}_i = \mathbf{x}_k \text{ on the boundary} \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

and rewrite (9) as a ratio of convolutions:

$$r(\mathbf{x}_i) = \frac{\sum_{j=0}^{n} w(\mathbf{x}_i, \mathbf{x}_j)\hat{r}(\mathbf{x}_j)}{\sum_{k=0}^{n} w(\mathbf{x}_i, \mathbf{x}_j)\chi_{\hat{r}}(\mathbf{x}_j)} = \frac{w * \hat{r}}{w * \chi_{\hat{r}}} \tag{12}$$

where $\chi_{\hat{r}}$ is the characteristic function corresponding to $\hat{r}$ ($\chi_{\hat{r}}$ is 1 where $\hat{r}$ is non-zero, 0 otherwise). Intuitively, including the characteristic function χ in the denominator ensures that the weights of the zeros in $\hat{r}$ are not accumulated.

Again, we use the optimization to find a set of kernels $\mathscr{F}$, with which our convolution pyramid can be applied to evaluate (12). To define the training data, we set b to be the boundary of a simple rectangular domain with random values assigned at the boundary grid points, and compute an exact membrane $r(x)$ using (12). Our optimization then attempts to match r, which is a ratio of two convolutions, directly, rather than matching each convolution separately. This is important to ensure that the decay of the filter w is accurately reproduced by our transform over the entire domain.

As in the previous application, we were able to produce satisfactory interpolating membranes using a kernel set $\mathscr{F}$ consisting of 5-by-5 separable filters for h_1, h_2 and a single 3-by-3 separable filter for g, regardless of the size and the shape of the domain. These kernels are provided in the supplementary material.

In Figure 7 we compare seamless image cloning using our method with that produced with a Laplacian membrane [Pérez et al. 2003]. Note that despite the slight differences between the two membranes the final seamless compositing results of both methods are difficult to distinguish visually.

The running time of the this method amounts to two evaluations of approximate convolution, one with $\hat{R}$ and one with $\chi_{\hat{R}}$, where the times for a single convolution are reported in Table 1. We have already established in the previous section that our approach outperforms state-of-the-art linear solvers. It remains, however, to compare our method to the fast Mean Value Cloning (MVC) method [Farbman et al. 2009], which computes mean value coordinates at the vertices of an adaptive triangulation of the domain. In contrast to that method, our approach does not require triangulating

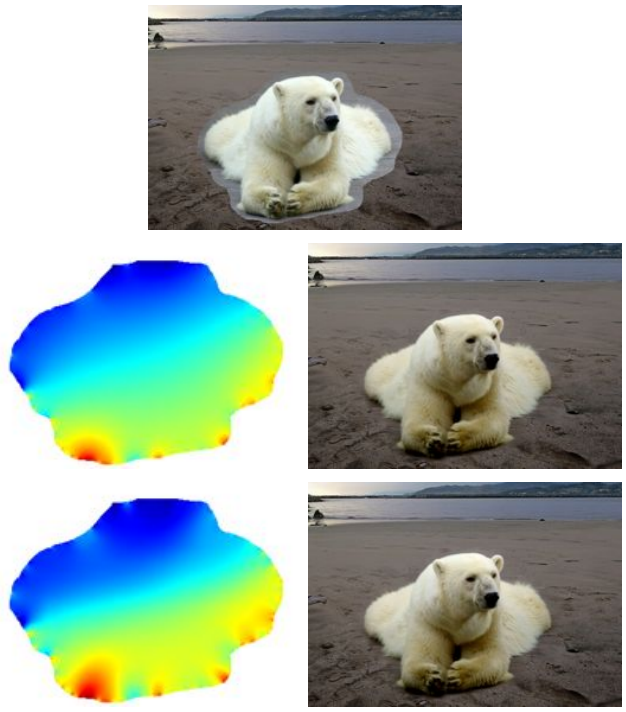


Figure 7: *Membrane construction for seamless cloning. Top row: image with a cloned patch superimposed; Middle row: Laplacian membrane [Pérez et al. 2003] and the resulting seamless cloning. Bottom row: Our membrane and the corresponding result.*

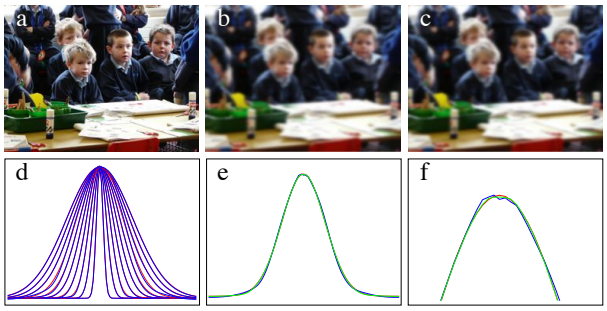


Figure 8: *Gaussian filters: (a) Original Image (b) Exact convolution with a Gaussian filter ($\sigma = 4$) (c) Convolution using our approximation for the same σ. (d) Exact kernels (in red) with approximate kernels (in blue). (e) exact Gaussian (in red), approximation using 5x5 kernels (in blue), approximation using 7x7 kernels. (f) shows a magnified part of (e).*

the domain and precomputing each pixel's barycentric coordinates to achieve interactive performance on the CPU. Furthermore, our method is completely independent of the size of the boundary, and avoids the somewhat sophisticated scheme that MVC employs for hierarchical sampling of the boundary. Farbman *et al.* [2009] report that after 3.6 seconds of preprocessing time, it takes 0.37 seconds to seamlessly clone a 4.2 megapixel region (on a 2.5 MHz Athlon CPU). In comparison, our method does not involve preprocessing, and clones the same number of pixels in 0.15 seconds.

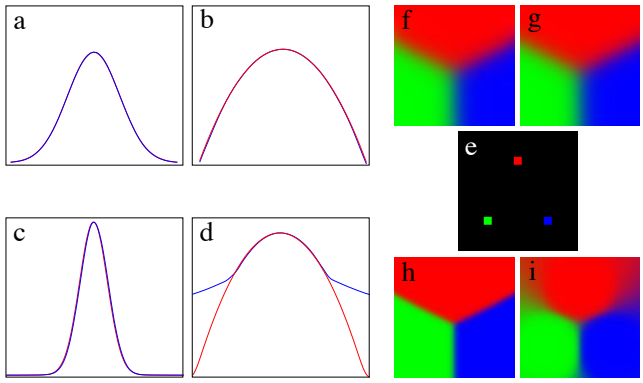


Figure 9: *Scattered data interpolation with two Gaussian kernels: (a,c) A horizontal slice through the exact Gaussian (in red) and our approximation (in blue) (b,d) A log plot of the same slice. (e) Scattered data interpolation input. (f,h) Exact results corresponding to the two Gaussians. (g,i) Our approximations.*

6 Gaussian kernels

In this section we demonstrate how to use convolution pyramids to approximate convolutions with Gaussian kernels $e^{-\|\mathbf{x}\|^2/2\sigma^2}$. Burt [1981] showed how this can be done using an $O(n \log n)$ multi-scale scheme and described an $O(n)$ variant that computes the result only on a coarse grid, whose spacing is inversely proportional to the Gaussian's width. While the resulting values can be interpolated to the original grid to yield an overall $O(n)$ method, the effective kernels in both variants are truncated and their support depends on the scale σ. Using our scheme we can approximate the solution at the original fine grid in $O(n)$ operations without truncating the filter support.

To carry out the optimization in (4) we set the training signal a to a sum of randomly translated and scaled delta functions, and $f * a$ is obtained by a full convolution with the target Gaussian filter. This is intended to force the optimal kernels to achieve an adequate approximation across the entire domain, resulting in a transform that is effectively translation-invariant. Our initial attempts to find a set $\mathscr{F} = \{h_1, h_2, g\}$ that will provide a good approximation failed. The reason is that Gaussians are rather efficient low-pass filters and, depending on their scale σ, they should not contain high-frequency components coming from the finer levels of the pyramid. These components are introduced by the convolutions with the g kernel in (3), and we were able to obtain considerably better results by modulating the contribution of g at each level l by a scalar weight w^l. These weights were added as additional degrees of freedom to optimize over, and indeed, the optimization resulted with significantly higher w^l at the levels that are closest to the target Gaussian's scale. Note that different sets of kernels $\mathscr{F}$ must be fit to obtain Gaussian filters of different scales.

Figure 8 shows the impulse responses of ten convolution pyramids optimized to approximate Gaussians of ten different values of σ, superimposed with the exact impulse responses of the target Gaussians. In these pyramids we use 5-by-5 separable kernels for h_1, h_2, and g. Notice that the approximation is less accurate for certain values of σ. To improve the accuracy, we increase the allocated computational budget and use 7-by-7 separable kernels instead. Figure 8(e-f) demonstrate the resulting improvement in accuracy.

The effective filters that Burt's method and integral images produce are truncated in space, i.e., they have a finite support that depends on the Gaussian scale σ. While this is not a major issue when the filtering is used for applications such as image blurring, this trun-

cation has a significant effect when the filters are used for scattered data interpolation, such as the Shepard's method outlined in the previous section. In this application, we need the data to propagate, in a monotonically decreasing fashion, across the entire domain. A truncated kernel may lead to divisions by zero in (12) or to introduce abrupt transitions in the interpolated values.

Figure 9 shows our approximation of two different Gaussian filters, differing in their width. These approximations were computed using the smaller 5-by-5 kernels. The approximation of the wider Gaussian in (a) provides a good fit across the entire domain. For the narrower Gaussian in (b), the approximation loses its relative accuracy as the Gaussian reaches very low values. This may be seen in the log plot in (d). Note however, that the approximation is still smooth and monotonically decaying. This slower decay leads to fuzzier transitions in the interpolated function compared to the exact scattered data interpolation, as shown in (h) and (i).

In summary, when using wide Gaussian weight functions or operating on a denser input dataset, our method provides an efficient and accurate approximation to scattered-data interpolation. Sparser datasets and narrower Gaussians reveal a limitation of our approximation.

7 Discussion and Future Work

We presented a multiscale scheme that achieves accurate approximations of the convolution operations with several widely-used filters. This is done using minimal computational efforts where we achieve large-scale non-separable filtering using small and separable kernels. We demonstrated the advantages of using our method over existing techniques, including state-of-the-art linear solvers. The two main ideas behind our approach are: (i) identifying the right computational scheme, which balances operation count with lack of translation-invariance, and (ii) optimizing filters, rather than tackling their lack of idealness analytically. The optimized kernels used in Sections 4 and 5 are provided in the supplementary material and can be used out-of-the-box for the applications described in these sections. As for the Gaussian filters, depending upon the application, in cases where the required values of σ are known in advance, such kernels can be computed up front.

While our paper provides new useful tools, our non-analytic approach has several fundamental limitations. While we succeeded in approximating certain filters which are commonly used in computer graphics, our work does not shed light on what other filters can be approximated using this approach. Specifically, it is not yet clear which filters can be approximated *efficiently* using *small* kernels. Another limitation arises from the use of black-box optimization in order to find the kernel set $\mathscr{F}$. In order to gain higher accuracy or approximate more challenging filters, larger kernels must be used. As the number of unknown parameters in the optimization increases, it will be harder to expect the optimization will indeed reach a global optimum (or even a satisfactory local one). As future work, we intend to conduct a thorough theoretical study in order to identify the scope of this approach in terms of filters it can approximate and gain insights that, if not replace the optimization, will at least aid its convergence.

Acknowledgements: The authors are indebted to Yoav HaCohen for crafting the optimized convolution pyramid implementation which produced the timings reported in this paper. This work was supported in part by the Israel Science Foundation founded by the Israel Academy of Sciences and Humanities.

References

AGARWALA, A. 2007. Efficient gradient-domain compositing using quadtrees. *ACM Trans. Graph. 26*, 3 (July), Article 94.

BHAT, P., CURLESS, B., COHEN, M., AND ZITNICK, C. L. 2008. Fourier analysis of the 2D screened Poisson equation for gradient domain problems. In *Proc. ECCV*, 114–128.

BRIGHAM, E. O. 1988. *The fast Fourier transform and its applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

BURT, P. J., AND ADELSON, E. H. 1983. The Laplacian pyramid as a compact image code. *IEEE Trans. Comm. 31*, 4, 532–540.

BURT, P. J. 1981. Fast filter transforms for image processing. *Computer Graphics and Image Processing 16*, 1 (May), 20–51.

CARR, J. C., BEATSON, R. K., MCCALLUM, B. C., FRIGHT, W. R., MCLENNAN, T. J., AND MITCHELL, T. J. 2003. Smooth surface reconstruction from noisy range data. In *Proc. GRAPHITE '03*, ACM, 119–ff.

DERPANIS, K., LEUNG, E., AND SIZINTSEV, M. 2007. Fast scale-space feature representations by generalized integral images. In *Proc. ICIP*, vol. 4, IEEE, 521–524.

DO, M., AND VETTERLI, M. 2003. Framing pyramids. *IEEE Transactions on Signal Processing 51*, 9 (Sept.), 2329–2342.

EVANS, L. C. 1998. *Partial Differential Equations*, vol. 19 of *Graduate Series in Mathematics*. American Mathematical Society.

FARBMAN, Z., HOFFER, G., LIPMAN, Y., COHEN-OR, D., AND LISCHINSKI, D. 2009. Coordinates for instant image cloning. *ACM Trans. Graph. 28*, 3, Article 67.

FATTAL, R., LISCHINSKI, D., AND WERMAN, M. 2002. Gradient domain high dynamic range compression. *ACM Trans. Graph. 21*, 3 (July), 249–256.

FATTAL, R., AGRAWALA, M., AND RUSINKIEWICZ, S. 2007. Multiscale shape and detail enhancement from multi-light image collections. *ACM Trans. Graph. 26*, 3 (July), Article 51.

HECKBERT, P. S. 1986. Filtering by repeated integration. In *Proc. ACM SIGGRAPH 86*, ACM, 315–321.

HORN, B. K. P. 1974. Determining lightness from an image. *Computer Graphics and Image Processing 3*, 1 (Dec.), 277–299.

KAZHDAN, M., AND HOPPE, H. 2008. Streaming multigrid for gradient-domain operations on large images. *ACM Trans. Graph. 27*, 3 (Aug.), 21:1–21:10.

LEWIS, J. P., PIGHIN, F., AND ANJYO, K. 2010. Scattered data interpolation and approximation for computer graphics. In *ACM SIGGRAPH ASIA 2010 Courses*, ACM, 2:1–2:73.

MALLAT, S. 2008. *A wavelet tour of signal processing*, 3rd ed. Academic Press.

MCCANN, J., AND POLLARD, N. S. 2008. Real-time gradient-domain painting. *ACM Trans. Graph. 27*, 3 (August), 93:1–93:7.

ORZAN, A., BOUSSEAU, A., WINNEMÖLLER, H., BARLA, P., THOLLOT, J., AND SALESIN, D. 2008. Diffusion curves: a vector representation for smooth-shaded images. *ACM Trans. Graph. 27*, 3 (August), 92:1–92:8.

PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. *ACM Trans. Graph. 22*, 3, 313–318.

PERONA, P. 1995. Deformable kernels for early vision. *IEEE Trans. Pattern Anal. Mach. Intell. 17*, 5, 488–499.

SELESNICK, I. 2006. A higher density discrete wavelet transform. *IEEE Trans. Signal Proc. 54*, 8 (Aug.), 3039–3048.

SHANNO, D. F. 1970. Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation 24*, 111 (July), 647.

SHEPARD, D. 1968. A two-dimensional interpolation function for irregularly-spaced data. In *Proc. 1968 23rd ACM National Conference*, ACM, 517–524.

SZELISKI, R. 1990. Fast surface interpolation using hierarchical basis functions. *IEEE Trans. Pattern Anal. Mach. Intell. 12*, 6, 513–528.

SZELISKI, R. 2006. Image alignment and stitching: a tutorial. *Found. Trends. Comput. Graph. Vis. 2* (January), 1–104.

TROTTENBERG, U., OOSTERLEE, C., AND SCHÜLLER, A. 2001. *Multigrid*. Academic Press.

WELLS, III, W. M. 1986. Efficient synthesis of Gaussian filters by cascaded uniform filters. *IEEE Trans. Pattern Anal. Mach. Intell. 8*, 2 (March), 234–239.