Foundations and Trends® in
Computer Graphics and Vision
Vol. 8, No. 1 (2012) 1–84

DOI: 10.1561/0600000053

now
the essence of knowledge

A Fresh Look at Generalized Sampling

Diego Nehab
Instituto Nacional de Matemática Pura e Aplicada (IMPA)

Hugues Hoppe
Microsoft Research

Contents

Abstract

Discretization and reconstruction are fundamental operations in computer graphics, enabling the conversion between sampled and continuous representations. Major advances in signal-processing research have shown that these operations can often be performed more efficiently by decomposing a filter into two parts: a compactly supported continuous-domain function and a digital filter. This strategy of "generalized sampling" has appeared in a few graphics papers, but is largely unexplored in our community. This survey broadly summarizes the key aspects of the framework, and delves into specific applications in graphics. Using new notation, we concisely present and extend several key techniques. In addition, we demonstrate benefits for prefiltering in image downscaling and supersample-based rendering, and analyze the effect that generalized sampling has on the noise due to Monte Carlo estimation. We conclude with a qualitative and quantitative comparison of traditional and generalized filters.

D. Nehab and H. Hoppe. *A Fresh Look at Generalized Sampling*. Foundations and Trends® in Computer Graphics and Vision, vol. 8, no. 1, pp. 1–84, 2012.

DOI: 10.1561/0600000053.

1

Introduction

Many topics in computer graphics involve digital processing of continuous-domain data, so it is unsurprising that discretization and reconstruction are essential operations. Figure 1.1 shows the traditional sampling and reconstruction pipeline. During discretization (e.g., rasterization of a scene, or capture of a digital photograph), a continuous input signal f is passed through an

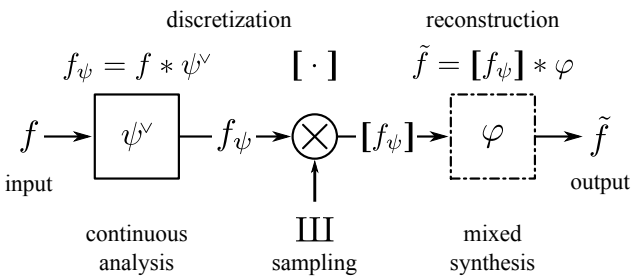


Figure 1.1: The traditional signal-processing pipeline is divided into two major stages: discretization and reconstruction. During discretization, a continuous input signal f is convolved with the reflection $\psi^\vee$ of a given analysis filter ψ. The resulting prefiltered signal $f_\psi = f * \psi^\vee$ is then uniformly sampled into a discrete sequence $[f_\psi]$. To obtain the output approximation $\tilde{f}$, the reconstruction stage computes the mixed convolution between $[f_\psi]$ and a given reconstruction kernel φ, i.e., a sum of shifted copies of φ, where each shifted copy scaled by the corresponding entry in $[f_\psi]$. (Our notation is explained in greater depth in section 3.)

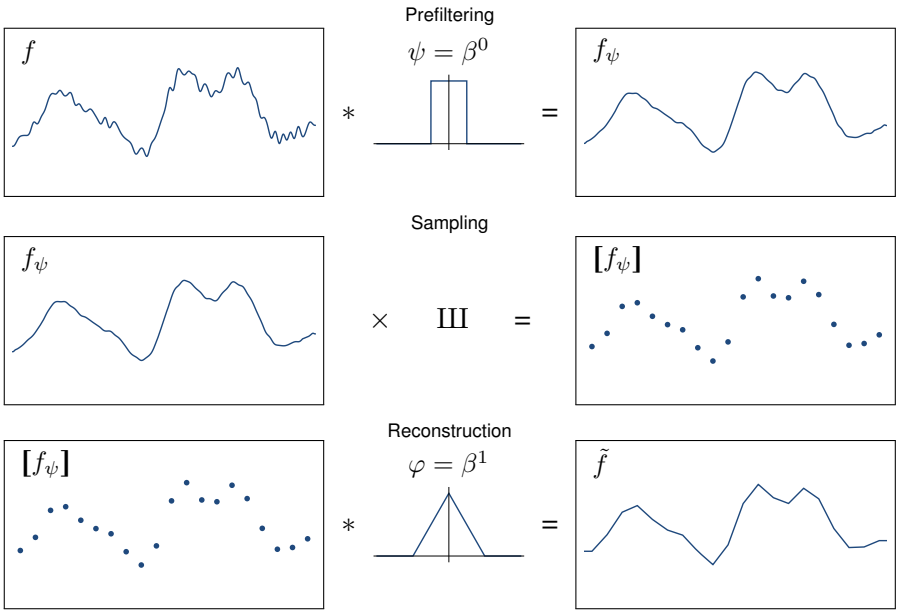


Figure 1.2: A continuous function f is prefiltered with analysis kernel ψ (here the box function β^0, not to scale). The resulting signal f_ψ is sampled into a discrete sequence $[f_\psi]$. The final output $\tilde{f}$ is obtained by mixed convolution between the discrete sequence $[f_\psi]$ and the reconstruction kernel φ (here the hat function β^1, not to scale).

analysis filter ψ (a.k.a. *sampling kernel*, *prefilter*, or *antialiasing filter*) before being sampled. The result is a discrete sequence $[f_\psi]$ (e.g., an image). During reconstruction (e.g., interpolation of a texture, or display of an image on a screen), the continuous approximation $\tilde{f}$ of the original signal is obtained by mixed convolution with a *reconstruction kernel* φ (a.k.a. *generating function*, *basis function*, or *postfilter*). Figure 1.2 illustrates each step of the process with a concrete example in 1D.

The roles of the analysis filter ψ and reconstruction kernel φ are traditionally guided by the sampling theorem [Shannon, 1949]. Given a sampling rate $1/T$, the analysis filter $\psi = \mathrm{sinc}(\cdot/T)$ eliminates from the input signal f those frequencies higher than or equal to $1/2T$ so that the bandlimited f_ψ can be sampled without aliasing. And in that case, the reconstruction kernel $\varphi = \mathrm{sinc}(\cdot/T)$ recreates $\tilde{f} = f_\psi$ exactly from the samples.

Sampling may also be interpreted as the problem of finding the function $\tilde{f}$ that minimizes the norm of the residual $\|f - \tilde{f}\|_{\mathcal{L}_2}$. If we restrict our attention

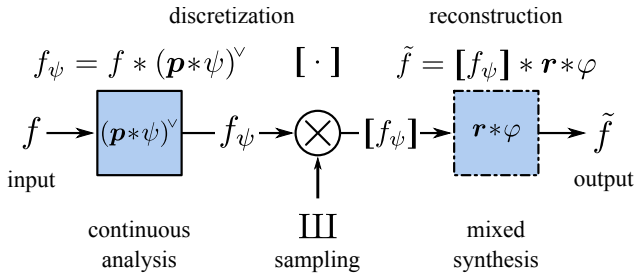


Figure 1.3: The main idea in generalized sampling is to broaden the analysis and reconstruction kernels by expressing these as mixed convolutions ($\boldsymbol{p} * \psi$ and $\boldsymbol{r} * \varphi$) with a pair of digital filters ($\boldsymbol{p}$ and $\boldsymbol{r}$) while retaining compact support for the functions ψ and φ.

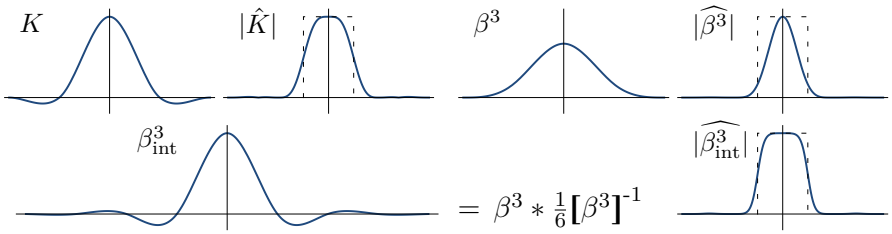


Figure 1.4: The traditional Keys cubic (Catmull-Rom spline) K has support 4 and a reasonably sharp frequency response $|\hat{K}|$. The cardinal cubic B-Spline β^3_{int} is a generalized kernel formed from the basic cubic B-spline β^3 and a digital filter. The digital filter acts to widen support to infinity (though with exponential decay) and to significantly sharpen the frequency response.

to the space of bandlimited functions, the ideal prefilter is still $\psi = \text{sinc}(\cdot/T)$. However, functions are often not bandlimited in practice (e.g., due to object silhouettes, shadow boundaries, vector outlines, detailed textures), and for efficiency we desire ψ and φ to be compactly supported.

In addressing these concerns, the signal-processing community has adopted a generalization of the sampling and reconstruction pipeline [Unser, 2000]. The idea is to represent the prefilter and reconstruction kernels as mixed convolutions of *compactly supported* kernels and *digital filters*. As shown in figure 1.3, digital filters $\boldsymbol{p}$ and $\boldsymbol{r}$ respectively modify the prefilter ψ and the reconstruction kernel φ. The additional degrees of freedom and effectively larger filter support enabled by $\boldsymbol{p}$ and $\boldsymbol{r}$ allow the design of generalized kernels with better approximation properties or sharper frequency response. Figure 1.4 compares a traditional piecewise cubic kernel (the Catmull-Rom spline, or Keys cubic) with a generalized cubic kernel (the cardinal cubic B-spline).

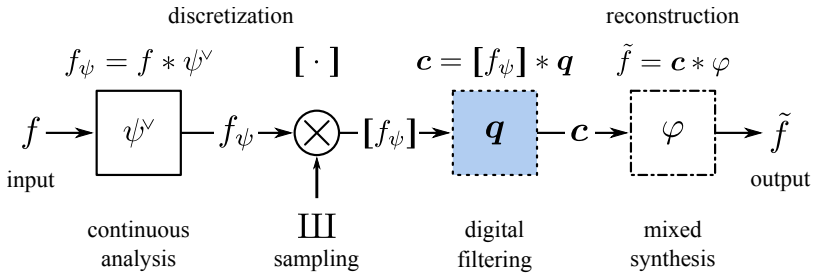


Figure 1.5: Generalized sampling adds a digital filtering stage to the pipeline. The output $\lfloor f_\psi \rceil$ of the sampling stage is convolved with a digital transformation filter $\boldsymbol{q} = \boldsymbol{p}^\vee * \boldsymbol{r}$. It is the result $\boldsymbol{c}$ of this stage (and not $\lfloor f_\psi \rceil$) that is convolved with the reconstruction kernel φ to produce the output signal.

Equivalently, the digital filters $\boldsymbol{p}$ and $\boldsymbol{r}$ can be combined as $\boldsymbol{q} = \boldsymbol{p}^\vee * \boldsymbol{r}$ into a separate filter stage as shown in figure 1.5. The result $\lfloor f_\psi \rceil$ of the sampling stage is transformed by the digital filter $\boldsymbol{q}$ (a.k.a. *correction* or *basis change*) to form a new discrete sequence $\boldsymbol{c}$, which is then convolved with φ as usual to reconstruct $\tilde{f}$. The key to the efficiency of this generalized sampling framework is that the digital filters $\boldsymbol{p}$ and $\boldsymbol{r}$ that arise in practice are typically compact filters or their inverses [Unser et al., 1991], both of which are parallelizable on multicore CPU and GPU architectures [Ruijters et al., 2008, Nehab et al., 2011]. Thus, the correction stage adds negligible cost.

An important motivation for generalized sampling is improved interpolation [Blu et al., 1999]. As demonstrated in figure 1.6, an image $\lfloor f_\psi \rceil$ is processed by a digital filter $\boldsymbol{q}$ resulting in a coefficient array $\boldsymbol{c}$ which can then be efficiently reconstructed with a simple cubic B-spline filter β^3. The resulting interpolation is sharper and more isotropic (i.e., has higher quality) than that produced by the popular Mitchell-Netravali filter [1988], even though both filters have the same degree and support. The implementation of the digital filtering stage is described in detail in section 4.2. The theory of image upscaling is described in section 5.2, with implementation notes in section 8.2. Source-code in provided in appendix A.

In graphics, careful prefiltering is often necessary to prevent aliasing. McCool [1995] describes an early application of generalized sampling, in which rendered triangles are antialiased analytically by evaluating a prism spline prefilter. The resulting image is then convolved with a digital filter. In this work, we apply generalized sampling to image downscaling and in general

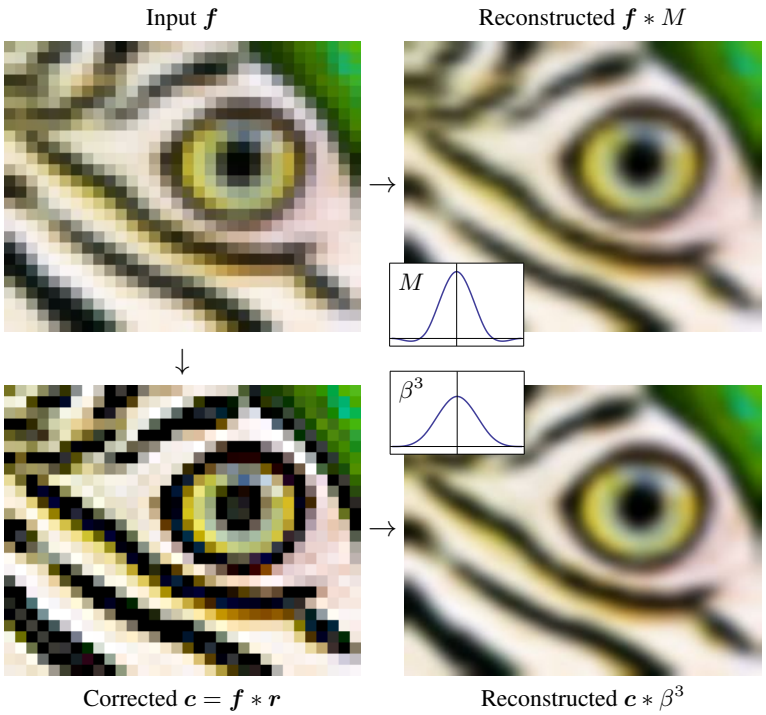


Figure 1.6: Reconstruction example. The top row shows the result of the traditional cubic Mitchell-Netravali filter M. The bottom row uses the generalized sampling approach, first applying a digital filter $\boldsymbol{r} = \left[\frac{1}{6}, \frac{4}{6}, \frac{1}{6}\right]^{-1}$ as a preprocess, and then reconstructing with the cubic B-spline β^3 — which is less expensive to evaluate on a GPU than filter M.

to rendering with supersampling. Figure 1.7 shows an example. The input f is prefiltered using the cubic B-spline basis β^3. The resulting over-blurred image is then transformed with a digital filter $\boldsymbol{p}^\vee$ that reshapes the antialiasing kernel *a posteriori*. The final low-resolution image is sharper and exhibits less aliasing than with a Catmull-Rom filter, for a similar computational cost. The theory of image downscaling is described in section 5.2, with implementation notes in section 8.2 and source-code in appendix A. Generalized supersampling is described in section 7.

Our aim is to present a concise overview of the major developments in generalized sampling and to extend these techniques to prefiltering in graphics. To facilitate exposition and exploration, we develop a new concise notation for sampling. With this parameter-free notation, key techniques are derived using simple algebraic manipulation. We conclude by comparing a variety of

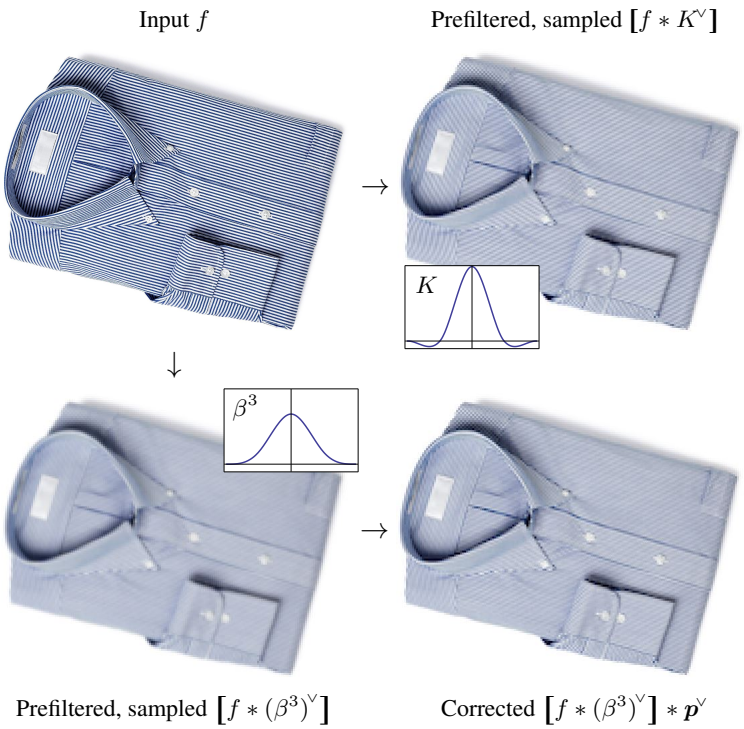


Figure 1.7: Prefiltering example. The top row shows the result of rendering with the Keys (Catmull-Rom) prefilter K. The bottom row shows rendering using a B-spline β^3, followed by convolution with a digital filter $\boldsymbol{p}^{\vee} = \left[\frac{1}{6}, \frac{4}{6}, \frac{1}{6}\right]^{-1}$. The generalized prefilter $\boldsymbol{p} * \beta^3$ equals the cubic cardinal B-spline β^3_{int}. Kernels K and β^3 have the same support, but the improved frequency response of β^3_{int} reduces aliasing while maintaining sharpness. (Our notation is explained in section 3.)

traditional and generalized filters, using frequency-domain visualizations and empirical experiments using both $\mathcal{L}_2$ and SSIM metrics, to identify the best strategies available.

2

Background

2.1 Reconstruction kernels

The search for finite-support alternatives to the ideal reconstruction filter has led to many non-polynomial windowed-sinc variants [Meijering et al., 1999a]. The most popular in graphics is the Lanczos window [Duchon, 1979]. (According to our experiments in section 10, the Hamming window [Hamming, 1977] performs better.)

Polynomial functions have been shown to have an efficiency advantage, and to match (and surpass) windowed-sinc approximations in quality [Meijering et al., 2001]. We therefore focus on piecewise polynomial kernels. Like Thévenaz et al. [2000], we use a set of properties that characterize reconstruction kernels to guide us through the bibliography. For an alternative, chronological survey, please refer to Meijering [2002].

The *degree* N of a kernel φ is the maximum degree of its polynomial pieces. The *support* W of φ is the width of the smallest interval outside of which φ vanishes (assuming a sample spacing of one). Increasing either the degree or support of a kernel φ introduces additional degrees of freedom for the design of good kernels, but unfortunately also adds to the runtime computational cost.

Most kernels are *symmetric* about the origin. The *regularity* R measures the smoothness of φ. That is, a kernel φ is said to be in $\mathcal{C}^R$ if it is differentiable R times. The space of functions spanned by a generator φ is denoted by V_φ. The *order of approximation* L of V_φ measures the rate at which the residual between f and its optimal approximation $\tilde{f}_T \in V_\varphi$ vanishes as the sample spacing T is reduced:

$$\|f - \tilde{f}_T\|_{\mathcal{L}_2} = C_f\, T^L \quad \text{as} \quad T \to 0.$$

Equivalently, a space with approximation order L can reproduce polynomial signals of up to degree $L-1$ [Strang and Fix, 1973]. Enforcement of regularity, symmetry, and approximation order in φ consume degrees of freedom from the design process. In fact, the best approximation order a kernel of degree N can attain is $L=N+1$. This optimal order is achievable even with a compact support $W=N+1$ [Blu et al., 2001]. Various strategies for setting the remaining degrees of freedom have led to the development of a multitude of reconstruction kernels.

Mitchell and Netravali [1988] design a family of cubic kernels by starting with $W = 4$, $R = 1$, and $L = 1$. They set the two remaining degrees of freedom by subjectively evaluating the amount of ringing, blur, and anisotropy in upsampled images. Alternatively, setting $L = 2$ leaves only one degree of freedom, and they similarly set its value based on subjective quality.

Interpolating kernels A reconstruction kernel φ is *interpolating* if it satisfies $\varphi(0) = 1$ and $\varphi(k) = 0, k \in \mathbb{Z} \setminus \{0\}$. Naturally, enforcing this property further eliminates degrees of freedom. Popular interpolating kernels include the ubiquitous nearest-neighbor (or *box*, $L = 1, W = 1$) and linear (or *hat*, $L = 2, W = 2$) interpolation kernels. These are members of the family of local Lagrangian interpolators, which have optimal order and minimum support $L = W = N + 1$ [Schafer and Rabiner, 1973, Schaum, 1993, Blu et al., 2001] but offer no regularity. Several authors have reached the Catmull-Rom cubic spline [Catmull and Rom, 1974] by different means [Keys, 1981, Park and Schowengerdt, 1983, Meijering et al., 1999b, Blu et al., 2001]. It is the only $\mathcal{C}^1$-continuous cubic interpolating kernel with optimal order and minimum support.

Other noteworthy kernels include an additional cubic ($W = 6, L = 4$) by Keys [1981], the $\mathcal{C}^0$-continuous (interpolating) and the $\mathcal{C}^1$-continuous

(not interpolating) quadratics ($N = 2, W = 3, L = 2$) of Dodgson [1997], a quartic ($L = 5, W = 7, R = 1$) by German [1997], and the quintic and septic kernels ($W = 6$ and 8, but $L = 3$) by Meijering et al. [1999b]. The support of these kernels is larger than necessary for their approximation order.

Generalized kernels A breakthrough came from the idea that the responsibility for interpolating the original samples need not be imposed on the continuous kernel φ itself, but can instead be achieved by using a digital correction filter $\boldsymbol{q}$. This was first demonstrated in the context of B-spline interpolation [Hou and Andrews, 1978, Unser et al., 1991].

B-splines are the most regular members of a class of functions called MOMS (for Maximal Order, Minimum Support) [Blu et al., 2001]. The best performing kernels, the O-MOMS (Optimal MOMS), trade off regularity to minimize the leading coefficient C_L in the optimal mean approximation error

$$\|f - \tilde{f}_T\|_{\mathcal{L}_2} = C_L \, T^L \, \|f^{(L)}\|_{L_2} \quad \text{as} \quad T \to 0.$$

If continuous derivatives are desired, the SO-MOMS (Sub-Optimal MOMS) minimize coefficient C_L subject to $\mathcal{C}^1$-continuity.

It is possible to further reduce reconstruction error by mimicking the low-frequency behavior of orthogonal projection (see below). The *shifted* linear interpolation scheme of Blu et al. [2004] gives up on symmetry and uses the additional freedom to minimize the approximation error. *Quasi-interpolation* schemes give up on interpolation of $\mathbf{[}f_\psi\mathbf{]}$, so that $\boldsymbol{q}$ is freed of this restriction. In that case, the interpolation property holds only when f is a polynomial of degree less than L (the quasi-interpolation order). Blu and Unser [1999a] describe an IIR design for $\boldsymbol{q}$, Condat et al. [2005] an all-pole design, and Dalai et al. [2006] a FIR design. The improvements due to this relaxation are often substantial, particularly for low-degree schemes ($N \leq 3$).

2.2 Analysis filters

As explained earlier, an important goal is *orthogonal projection*: minimizing the residual $\|f - \tilde{f}\|_{\mathcal{L}_2}$ between the reconstruction $\tilde{f}$ and the input signal f. Given a reconstruction kernel φ, orthogonal projection is achieved using a prefilter known as the *dual* of φ and denoted by the symbol $\mathring{\varphi}$. This prefilter may be written in the form $\boldsymbol{p} * \varphi$ (see section 4.3). Exploiting the fact that in

computer graphics we often have access to the input signal *prior to sampling*, Kajiya and Ullner [1981] use this orthogonal projection for antialiased text rasterization. In their work, φ is a Gaussian that models the CRT electron beam, and f corresponds to the text being rendered. They further restrict the optimization to non-negative coefficients $\boldsymbol{c}$ and explore a perceptual alternative to the $\mathcal{L}_2$ norm.

This idea was extended by McCool [1995] to include other prefilters of the form $\boldsymbol{p} * \psi$, where ψ is a compactly supported B-spline basis function. A similar factorization is used: first the input is prefiltered with ψ, then the result is transformed to what would be obtained if directly prefiltering with the cardinal spline.

Orthogonal projection is commonly used in processing of previously discretized input signals. Most image processing operations result in signals outside of the approximation space. These can then be projected back. In the case of scaling and translation, there are efficient algorithms to achieve this [Unser et al., 1995a,b, Muñoz et al., 2001].

The approach of orthogonal projection has not been widely adopted in computer graphics. In part, there is a lack of familiarity with the framework. But also, there is the issue that the perceptual quality of an approximation involves a subjective balance between aliasing, blurring, ringing, and positivity, to which the $\mathcal{L}_2$ norm is oblivious (as noted by Kajiya and Ullner [1981]). Some favor the sharpness offered by filters with negative lobes (the "negative lobists" [Blinn, 1989]), while others fear the accompanying ringing in dark regions (a problem accentuated by gamma correction). Rendering systems therefore offer several alternatives, including box, hat, Mitchell-Netravali, Catmull-Rom (Keys), Gaussian, and windowed-sinc filters [Pixar, 2005], but rarely (if ever) offer orthogonal or oblique projections.

3

Basic notation, definitions, and properties

Many of the derivations in generalized sampling involve tedious manipulations of integrals and summations, including changes of index variables. In this section, we introduce a new index-free notation to help eliminate this problem. Using this notation and a small set of properties, we are able to simplify the derivations of many algorithms and concepts to trivial algebraic manipulations.

For simplicity, we formulate the mathematics in one dimension. However, all results are easily extended to 2D images and 3D volumes in a *separable* way by making use of tensor-product basis functions. Furthermore, we assume all signals are real even though results are easily generalized to complex signals.

We denote the implicit argument to a univariate function with a dot "$\cdot$":

$$f(\cdot + k) \quad \stackrel{\text{def}}{=} \quad x \mapsto f(x + k). \tag{3.1}$$

The prefiltering stage in figures 1.1 and 1.5 simply perform a continuous convolution between the signal f and the prefilter kernel ψ:

$$f * \psi^{\vee} \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(t)\ \psi^{\vee}(\cdot - t)\,\mathrm{d}t. \tag{3.2}$$

We denote discrete sequences by bold letters, or write the values explicitly, enclosed in bold square-brackets **[ ]**:

$$\boldsymbol{c} \stackrel{\text{def}}{=} \mathbf{[}\ldots, c_{-2}, c_{-1}, c_0, c_1, c_2, \ldots\mathbf{]}, \quad c_k \in \mathbb{R},\ k \in \mathbb{Z}. \tag{3.3}$$

An element of a sequence can be selected by a subscript so that $\boldsymbol{c}_k \stackrel{\text{def}}{=} c_k$ above.

A key part of our new notation is the use of bold square-brackets, when enclosing an expression with a single free variable, to denote the discrete sequence that results from uniformly sampling the expression at all multiples of a sampling spacing T:

$$\boldsymbol{[}f\boldsymbol{]}_T \stackrel{\text{def}}{=} \boldsymbol{[}\ldots, f(-2T), f(-T),\ f(0),\ f(T), f(2T), \ldots\boldsymbol{]} \tag{3.4}$$

$$\stackrel{\text{def}}{=} \boldsymbol{[}f\boldsymbol{]},\ \text{in short when } T = 1. \tag{3.5}$$

This is the operation performed by the sampling stage of figures 1.1–1.5.

The digital filtering introduced by the generalized sampling pipeline in figure 1.5 corresponds to discrete convolution with the discrete sequence $\boldsymbol{q}$:

$$\boldsymbol{c} = \boldsymbol{b} * \boldsymbol{q}, \quad \text{where} \quad c_k = \sum_{i\in\mathbb{Z}} b_i\, q_{k-i},\ k \in \mathbb{Z}. \tag{3.6}$$

The reconstruction stage in figures 1.1–1.5 shares many properties with convolutions. We therefore introduce notation for a *mixed convolution with spacing* T between a discrete sequence $\boldsymbol{c}$ and a reconstruction kernel φ:

$$\boldsymbol{c} *_T \varphi \stackrel{\text{def}}{=} \sum_{k\in\mathbb{Z}} c_k \varphi(\cdot - kT) \quad \text{and in particular} \quad \boldsymbol{c} * \varphi \stackrel{\text{def}}{=} \boldsymbol{c} *_1 \varphi. \tag{3.7}$$

In most cases, commutativity and associativity apply to all combinations of convolutions. Parentheses are unnecessary as there is no danger of ambiguity. We can manipulate convolutions like products:

$$(\boldsymbol{b} * \boldsymbol{c}) * f = \boldsymbol{b} * (\boldsymbol{c} * f) \quad \text{and} \quad (f * g) * \boldsymbol{b} = f * (g * \boldsymbol{b}). \tag{3.8}$$

The only exception is in expressions involving multiple mixed convolutions with different spacings. Even then, it is still true that

$$\boldsymbol{b} *_T (\boldsymbol{c} *_S f) = \boldsymbol{c} *_S (\boldsymbol{b} *_T f), \tag{3.9}$$

but the spacings must be the same for us to factor out the discrete convolution:

$$\boldsymbol{b} *_T (\boldsymbol{c} *_T f) = \boldsymbol{c} *_T (\boldsymbol{b} *_T f) = (\boldsymbol{b} * \boldsymbol{c}) *_T f. \tag{3.10}$$

For repeated convolution, we use a notation reminiscent of exponentiation:

$$f^{*n} = \underbrace{f * \cdots * f}_{n} \quad \text{and} \quad \boldsymbol{b}^{*n} = \underbrace{\boldsymbol{b} * \cdots * \boldsymbol{b}}_{n} \tag{3.11}$$

The reflection of sequences and functions is denoted by

$$(\boldsymbol{c}^\vee)_k \stackrel{\text{def}}{=} \boldsymbol{c}_{-k},\ k \in \mathbb{Z} \quad \text{and} \quad f^\vee(x) \stackrel{\text{def}}{=} f(-x),\ x \in \mathbb{R} \tag{3.12}$$

and distributes over all flavors of convolution:

$$(\boldsymbol{b} * \boldsymbol{c})^\vee = \boldsymbol{b}^\vee * \boldsymbol{c}^\vee, \quad (f * g)^\vee = f^\vee * g^\vee, \quad \text{and} \quad (\boldsymbol{b} * \varphi)^\vee = \boldsymbol{b}^\vee * \varphi^\vee. \tag{3.13}$$

The simple connection between inner-products and convolutions,

$$\langle f,\, \psi \rangle \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(t)\, \psi(t)\, \mathrm{d}t = (f * \psi^\vee)(0), \tag{3.14}$$

is the source for the pervasive reflection operation. Its generalization

$$\big\langle f,\ \psi(\cdot - k) \big\rangle \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(t)\, \psi(t-k)\, \mathrm{d}t = (f * \psi^\vee)(k) \tag{3.15}$$

lets us express in compact form the fundamental operation of sampling a function f with an analysis filter ψ:

$$\big[\ldots, \big\langle f,\ \psi(\cdot - 1) \big\rangle,\ \langle f,\, \psi \rangle,\ \big\langle f,\ \psi(\cdot + 1) \big\rangle, \ldots\big] = \big[f * \psi^\vee\big]. \tag{3.16}$$

Using our notation, we can succinctly express the *key property* that sampling a mixed convolution with matching spacing T is equivalent to performing a discrete convolution between the sequence and the sampled function:

$$\big[\boldsymbol{b} *_T f\big]_T = \boldsymbol{b} * \big[f\big]_T. \tag{3.17}$$

In other words, these operations commute.

The continuous unit impulse δ (the Dirac delta) and the discrete unit impulse $\boldsymbol{\delta}$ (the Kronecker delta) are the identity elements for the convolution operations:

$$\delta * f \stackrel{\text{def}}{=} f,\ \forall f \quad \text{and} \quad \boldsymbol{\delta} * \boldsymbol{c} \stackrel{\text{def}}{=} \boldsymbol{c},\ \forall \boldsymbol{c}. \tag{3.18}$$

The discrete impulse can also be defined simply as

$$\boldsymbol{\delta} = \big[\ldots, 0, 0, \underset{\cdot}{1}, 0, 0, \ldots\big] \quad \text{where} \quad \boldsymbol{\delta}_0 = 1. \tag{3.19}$$

The discrete convolution operation $\boldsymbol{c} = \boldsymbol{b} * \boldsymbol{q}$ can often be efficiently inverted (see section 4.2). The inverse operation is again a discrete convolution:

$$\boldsymbol{b} = \boldsymbol{c} * \boldsymbol{q}^{-1} \quad \text{with} \quad \boldsymbol{q} * \boldsymbol{q}^{-1} = \boldsymbol{\delta}. \tag{3.20}$$

Reflection and convolution-inverse commute, so we can define:

$$\boldsymbol{b}^{-\vee} \stackrel{\text{def}}{=} (\boldsymbol{b}^{-1})^{\vee} = (\boldsymbol{b}^{\vee})^{-1}. \tag{3.21}$$

Interestingly, the derivative operation does *not* distribute over convolution:

$$(f * g)' = f' * g = f * g'. \tag{3.22}$$

This rule lets us express the derivative operation as a convolution, since

$$f = f * \delta \quad \Rightarrow \quad f' = (f * \delta)' = f * \delta'. \tag{3.23}$$

For higher-order derivatives, we use the short notation

$$f^{(n)} = f * (\delta')^{*n}. \tag{3.24}$$

The discrete analog of the derivative is the finite difference. We can express the (backward) differencing operation as convolution with the sequence

$$\boldsymbol{\Delta} = [\ldots, 0, 0, \underset{\cdot}{1}, -1, 0, \ldots] \quad \text{where} \quad \boldsymbol{\Delta}_0 = 1. \tag{3.25}$$

Just as with the derivative operation,

$$\boldsymbol{\Delta} * (\boldsymbol{b} * \boldsymbol{c}) = (\boldsymbol{\Delta} * \boldsymbol{b}) * \boldsymbol{c} = \boldsymbol{b} * (\boldsymbol{\Delta} * \boldsymbol{c}). \tag{3.26}$$

The continuous unit (or Heaviside) step function u is the integral of the unit impulse. Accordingly, the discrete unit step function $\boldsymbol{u}$ is the summation of the discrete unit impulse:

$$u(x) = \int_{-\infty}^{x} \delta(t)\,\mathrm{d}t \quad \text{and} \quad \boldsymbol{u}_i = \sum_{k=-\infty}^{i} \boldsymbol{\delta}_k. \tag{3.27}$$

From these definitions, it is clear that

$$(f * u)(x) = \int_{-\infty}^{x} f(t)\,\mathrm{d}t \quad \text{and} \quad (\boldsymbol{c} * \boldsymbol{u})_i = \sum_{k=-\infty}^{i} \boldsymbol{c}_k. \tag{3.28}$$

Furthermore,

$$u * \delta' = \delta \quad \text{and} \quad \boldsymbol{u} * \boldsymbol{\Delta} = \boldsymbol{\delta}, \tag{3.29}$$

so that the pairs u, δ' and $\boldsymbol{u}, \boldsymbol{\Delta}$ are convolution inverses.

We use τ_h to denote the translation of the impulse δ by an offset h:

$$\tau_h = \delta(\cdot - h). \tag{3.30}$$

This lets us express the translation of a function f as a convolution:

$$f(\cdot - h) = \tau_h * f. \tag{3.31}$$

The centered B-spline basis functions β^n are an important family of generating functions. The box filter β^0 can be defined as:

$$\beta^0 = \boldsymbol{\Delta} * u * \tau_{-1/2}. \tag{3.32}$$

The hat filter β^1 and the remaining B-splines are recursively defined as:

$$\beta^n = \beta^{n-1} * \beta^0, \tag{3.33}$$

or equivalently using the notation for repeated convolution:

$$\beta^n = (\beta^0)^{*(n+1)} = \boldsymbol{\Delta}^{*(n+1)} * u^{*(n+1)} * \tau_{-(n+1)/2}. \tag{3.34}$$

Note that the one-sided power function can be written equivalently as

$$u^{*(n+1)}(x) = \frac{x^n}{n!}\, u(x). \tag{3.35}$$

A useful shorthand for the cross-correlation between two functions is:

$$a_{\varphi,\psi} \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} \varphi(t)\, \psi(t - \cdot)\, \mathrm{d}t = \varphi * \psi^{\vee}. \tag{3.36}$$

In particular, we denote the auto-correlation as $a_\varphi \stackrel{\text{def}}{=} a_{\varphi,\varphi}$.

Two functions φ and ψ are biorthogonal if they satisfy

$$\big\langle \varphi(\cdot - i),\ \psi(\cdot - j) \big\rangle = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise,} \end{cases} \tag{3.37}$$

where $i, j \in \mathbb{Z}$, or equivalently in our notation,

$$\big[\!\big[a_{f,g} \big]\!\big] = \boldsymbol{\delta}. \tag{3.38}$$

The Discrete Time Fourier Transform (DTFT) of a sequence and the Fourier Transform of a function are defined respectively by

$$\hat{\boldsymbol{c}}(\xi) \stackrel{\text{def}}{=} \sum_{k \in \mathbb{Z}} \boldsymbol{c}_k e^{-2\pi i \xi k} \quad \text{and} \quad \hat{f}(\xi) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(x) e^{-2\pi i \xi x}\, \mathrm{d}x. \tag{3.39}$$

Convolution in the time domain becomes a product in the frequency domain. This is also true of mixed convolutions:

$$\widehat{(\boldsymbol{b} * \boldsymbol{c})} = \hat{\boldsymbol{c}}\,\hat{\boldsymbol{b}}, \quad \widehat{(f * g)} = \hat{f}\hat{g}, \quad \text{and} \quad \widehat{(f * \boldsymbol{c})} = \hat{f}\,\hat{\boldsymbol{c}}. \tag{3.40}$$

In our notation, the impulse train (Dirac comb) is a mixed convolution:

$$\text{Ш} \stackrel{\text{def}}{=} \sum_{k \in \mathbb{Z}} \delta(\cdot - k) = \big[1\big] * \delta = \mathbf{1} * \delta \tag{3.41}$$

It has the important property of being its own Fourier transform:

$$\widehat{\text{Ш}} = \text{Ш}. \tag{3.42}$$

See table 3.1 for a list of properties and constructions expressed using the notation we just presented.

Table 3.1: Some properties and constructions using the notation presented in this section.

Concept	Expressed in our notation
Symmetry of φ	$\varphi = \varphi^\vee$
Interpolation property	$\big[\varphi\big] = \boldsymbol{\delta}$
Unit integral	$\varphi * 1 = 1$
Partition of unity	$\varphi * \big[1\big] = 1$
Cross-correlation of φ and ψ	$a_{\varphi,\psi} = \varphi * \psi^\vee$
Biorthogonality of φ and ψ	$\big[a_{\varphi,\psi}\big] = \boldsymbol{\delta}$
Auto-correlation of φ	$a_\varphi = \varphi * \varphi^\vee$
Orthogonality of φ	$\big[a_\varphi\big] = \boldsymbol{\delta}$
Cardinal kernel φ_{int}	$\big[\varphi\big]^{-1} * \varphi$
Dual kernel $\mathring{\varphi}$	$\big[a_\varphi\big]^{-1} * \varphi$
Orthogonal kernel ϕ	$\big[a_\varphi\big]^{-\frac{1}{2}} * \varphi$

4

Fundamental algorithms

Using the new notation introduced in section 3, we now describe some fundamental algorithms in generalized sampling. While none of these algorithms are novel, the new notation allows more concise (and arguably more intuitive) derivation.

The algorithms in the next sections address different scenarios based on the selection of prefilter and reconstruction kernels:

- In **interpolation**, the input is a sampled representation assumed to be obtained without prefiltering, or equivalently by using a prefilter ψ equal to the unit impulse δ. Therefore the goal is to find the unique reconstruction $\tilde{f}$ that interpolates the samples — among functions in the space V_φ of the reconstruction kernel φ.

- The technique of **orthogonal projection** addresses the case where the prefilter ψ spans the same function space as the reconstruction kernel φ, e.g. the space of piecewise cubic B-splines. The approach is to minimize the $\mathcal{L}_2$ difference between the input signal and the reconstruction.

- Finally, **oblique projection** considers the case where the prefilter ψ and reconstruction kernel φ span different function spaces. It sets the reconstruction error to be orthogonal to the prefiltering space.

Table 4.1: Fundamental algorithms and their associated digital filters.

Approach	Prefilter	Reconstruction	Digital filter
Interpolation	$\psi = \delta$	φ	$\boldsymbol{q} = \left[\varphi\right]^{-1}$
Orthogonal projection	$\psi = \varphi$	φ	$\boldsymbol{q} = \left[a_\varphi\right]^{-1}$
Oblique projection	ψ	φ	$\boldsymbol{q} = \left[\varphi * \psi^\vee\right]^{-1}$

Table 4.1 summarizes the digital filter associated with each of the three approaches, as derived in the next sections. Interestingly, both interpolation and orthogonal projection can be seen as special cases of oblique projection.

4.1 Interpolation

When the prefiltering process is unknown (or absent if $\psi = \delta$), it is often desired that the reconstructed function $\tilde{f}$ interpolate the sample values $\left[f_\psi\right]$. With traditional sampling (figure 1.1), this interpolation property requires:

$$\begin{aligned} \left[f_\psi\right] &= \left[\tilde{f}\,\right] && (4.1)\\ &= \left[\left[f_\psi\right] * \varphi\right] && (4.2)\\ &= \left[f_\psi\right] * \left[\varphi\right]. && (4.3) \end{aligned}$$

As $\left[f_\psi\right]$ is arbitrary, an interpolating reconstruction kernel φ must therefore satisfy

$$\left[\varphi\right] = \boldsymbol{\delta}. \tag{4.4}$$

As noted earlier, these interpolation constraints may severely hinder the design of a reconstruction kernel with good approximation properties. Therefore, instead of requiring the kernel itself to satisfy the interpolation property, generalized sampling (figure 1.5) introduces a digital filtering stage for that purpose. We find an appropriate digital filter for interpolation with an arbitrary kernel φ as follows:

$$\begin{aligned} \left[f_\psi\right] &= \left[\tilde{f}\,\right] && (4.5)\\ &= \left[\boldsymbol{c} * \varphi\right] && (4.6)\\ &= \boldsymbol{c} * \left[\varphi\right] \;\Rightarrow && (4.7)\\ \boldsymbol{c} &= \left[f_\psi\right] * \left[\varphi\right]^{-1}. && (4.8) \end{aligned}$$

In other words,

$$\boldsymbol{c} = \left[f_\psi\right] * \boldsymbol{q} \quad \text{with} \quad \boldsymbol{q} = \left[\varphi\right]^{-1}. \tag{4.9}$$

The convolution of the digital filter $\boldsymbol{q} = \left[\varphi\right]^{-1}$ with the kernel φ is called the *cardinal* φ_{int} (see figure 4.1b):

$$\tilde{f} = \boldsymbol{c} * \varphi = \left[f_\psi\right] * \left[\varphi\right]^{-1} * \varphi \tag{4.10}$$

so that

$$\tilde{f} = \left[f_\psi\right] * \varphi_{\text{int}} \quad \text{with} \quad \varphi_{\text{int}} = \left[\varphi\right]^{-1} * \varphi. \tag{4.11}$$

We can use (4.4) to verify that φ_{int} is indeed an interpolating kernel:

$$\left[\varphi_{\text{int}}\right] = \left[\left[\varphi\right]^{-1} * \varphi\right] = \left[\varphi\right]^{-1} * \left[\varphi\right] = \boldsymbol{\delta}. \tag{4.12}$$

The key point is that the generalized sampling approach has the advantage of using the compactly supported φ during reconstruction, when performance is paramount. Since digital filtering with $\boldsymbol{q}$ is performed during preprocessing, it does not add any computational cost to the reconstruction stage. In the words of Blu et al. [1999], one obtains "higher-quality at no additional cost". In contrast, the cardinal kernel φ_{int} has infinite support. Finite approximations have a significantly wider support than φ, increasing reconstruction costs.

4.2 Inverse discrete convolution

Discrete convolution by a sequence $\left[\varphi\right]$ as in (4.7) is a linear operator and therefore expressible in matrix form:

$$\left[f_\psi\right] = \boldsymbol{c} * \left[\varphi\right] = \Phi\, \boldsymbol{c}, \text{ for some matrix } \Phi. \tag{4.13}$$

Inverting the process as in (4.9) is more challenging, as it amounts to solving a linear system:

$$\boldsymbol{c} = \left[f_\psi\right] * \left[\varphi\right]^{-1} = \Phi^{-1}\left[f_\psi\right]. \tag{4.14}$$

Fortunately, typical matrices Φ have regular structure: for even-periodic extensions, they are *almost* Toeplitz. We describe an algorithm adapted from Malcolm and Palmer [1974] for solving the common case where $\left[\varphi\right] = \left[p, q, p\right]$ is symmetric with support width of 3 elements. A similar algorithm appears in Hummel [1983] in the context of orthogonal projection. The tridiagonal matrix Φ and its LU-decomposition are:

$$\Phi = \begin{bmatrix} p+q & p & & & \\ p & q & p & & \\ & \ddots & \ddots & \ddots & \\ & & p & q & p \\ & & & p & q+p \end{bmatrix} = p \begin{bmatrix} 1 & & & \\ \ell_0 & 1 & & \\ & \ddots & \ddots & \\ & & \ell_{n-3} & 1 \\ & & & \ell_{n-2} & 1 \end{bmatrix} \begin{bmatrix} \ell_0^{-1} & & & & \\ & \ell_1^{-1} & 1 & & \\ & & \ddots & \ddots & \\ & & & \ell_{n-2}^{-1} & 1 \\ & & & & v \end{bmatrix}.$$

The factorization is performed and stored only once, then reused to solve multiple linear systems (one for each column and row in the image). Moreover, when Φ is diagonally dominant, the factorization is highly compressible, as the sequence $[\ell_0, \ell_1, \ldots]$ quickly converges to a limit value ℓ_∞. The last diagonal element $v = 1 + 1/\ell_\infty$ of matrix U needs to be handled separately.

Computing $\boldsymbol{c} = [f_\psi] * [\varphi]^{-1}$ when $[\varphi]$ has support 3 requires only $3n$ products and $2n$ additions. Coincidentally, this is the same cost as computing $[f_\psi] = \boldsymbol{c} * [\varphi]$. If one can make do with a scaled version $\boldsymbol{d} = p\,\boldsymbol{c}$ of the solution, for example by pre-scaling the kernel so that $p = 1$, the computation requires n fewer products.

The forward- and back-substitutions are very simple, and we include full source-code in appendix A (function `linear_solve`, lines 12–32). For convenience, we also provide the LU-decomposition arising from the cubic B-spline and O-MOMS interpolation problems. The cubic B-spline has $[\varphi] = \frac{1}{6}[1, \underset{\cdot}{4}, 1]$ and needs only 8 coefficients before the sequence converges to single-precision floating-point. (See lines 160–162.) For the cubic O-MOMS, $[\varphi] = \frac{1}{21}[4, \underset{\cdot}{13}, 4]$ and only 9 coefficients are needed (See lines 189–192.) Our vectorized, multicore implementation of the same algorithm in 2D runs at 800MiPix/s on an Intel Core i7 980X CPU.

An equivalent approach favored by the signal processing community is to formulate this inverse convolution as a sequence of recursive filters [Hou and Andrews, 1978, Unser et al., 1991]. The difficulty is dealing with the poles in $\boldsymbol{q} = [\varphi]^{-1}$ and the trick is to factor the filter into cascaded causal and anticausal parts, each realized as a stable recursive filter. Interestingly, the operations are almost exactly the same as those described in appendix A. The only difference is in the treatment of elements close to the boundaries. Whereas the recursive-filtering approach conceptually extends the input so as to compute the initial feedback, the LU-factorization simply uses different coefficients for the elements close to the boundary.

Even if $\big[\varphi\big]$ has larger support (i.e., $W > 4$), the inverse convolution can still be performed efficiently using LU-factorization in a similar way. The matrices L and U have bandwidth $\lfloor \frac{W+1}{2} \rfloor$, so the forward/back-substitution steps use several feedback elements, much like higher-order causal/anti-causal recursive filters.

In signal processing, higher-order recursive filters are sometimes further factored into chains of first- and second-order recursive filters. This is equivalent to factoring the L and U matrices into products of lower and upper bidiagonal or tridiagonal matrices. However, computations using such factorizations traverse the data additional times and do not save arithmetic operations. Thus, the additional factorization is not beneficial in modern computer architectures where memory access is expensive.

A modern parallelization of recursive filters for GPUs recently achieved over 6GiPix/s on an NVIDIA GTX 480 GPU for the bicubic B-spline interpolation problem [Nehab et al., 2011]. The key message is that, whatever the approach followed or architecture used, the digital processing stage in generalized sampling is extremely fast.

4.3 Orthogonal projection

We now derive an algorithm to obtain the orthogonal projection $P_\varphi f$ of the input signal f into the reconstruction space V_φ. The algorithm computes the coefficient array $\boldsymbol{c}$ expressing this projection as $P_\varphi f = \boldsymbol{c} * \varphi$. To compute $\boldsymbol{c}$, we need to determine the appropriate prefilter ψ and digital filter $\boldsymbol{q}$. The orthogonality condition is:

$$(f - \tilde{f}\,) \perp V_\varphi \quad \Leftrightarrow \quad \big\langle f - \tilde{f},\ \varphi(\cdot - k) \big\rangle = 0 \quad \text{for} \quad k \in \mathbb{Z}, \tag{4.15}$$

$$\Leftrightarrow \quad \big[(f - \tilde{f}\,) * \varphi^\vee\big] = \mathbf{0}. \tag{4.16}$$

From there,

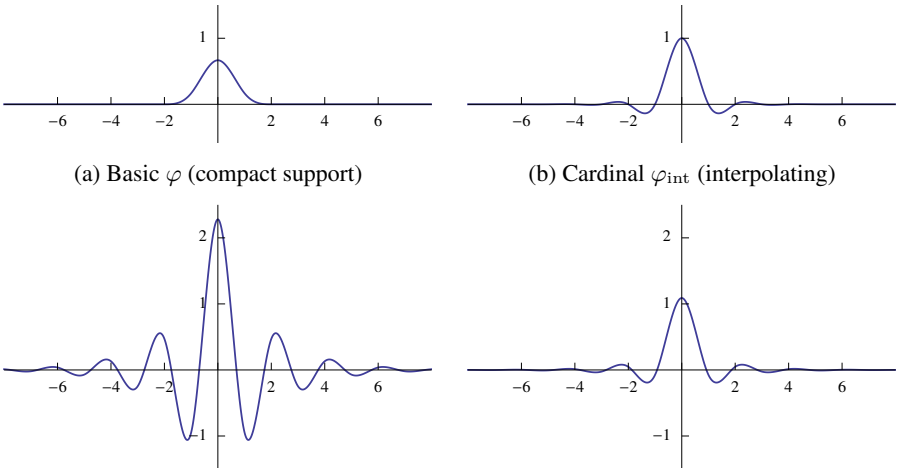
$$\big[(f - \tilde{f}\,) * \varphi^\vee\big] = \mathbf{0} \ \Rightarrow \tag{4.17}$$

$$\big[f * \varphi^\vee\big] = \big[\tilde{f} * \varphi^\vee\big] \tag{4.18}$$

$$= \big[\boldsymbol{c} * \varphi * \varphi^\vee\big] \tag{4.19}$$

$$= \boldsymbol{c} * \big[\varphi * \varphi^\vee\big] \ \Rightarrow \tag{4.20}$$

$$\boldsymbol{c} = \big[f * \varphi^\vee\big] * \big[\varphi * \varphi^\vee\big]^{-1}. \tag{4.21}$$



(a) Basic φ (compact support)

(b) Cardinal φ_{int} (interpolating)

(c) Dual $\mathring{\varphi}$ (biorthogonal to φ)

(d) Orthogonal ϕ (own dual)

Figure 4.1: Equivalent basis functions for cubic B-splines.

In other words,

$$\boldsymbol{c} = \left[f * \psi^{\vee}\right] * \boldsymbol{q} \quad \text{with} \quad \psi = \varphi \quad \text{and} \quad \boldsymbol{q} = \left[a_{\varphi}\right]^{-1}. \tag{4.22}$$

Here, $a_{\varphi} = \varphi * \varphi^{\vee}$ is the auto-correlation of φ, so that the matrix associated with the linear system is none other than the Gramiam matrix reached by Kajiya and Ullner [1981] and Hummel [1983]. It is important to point out that, unless the output device uses φ for reconstruction, it makes little sense to directly use the coefficient array $\boldsymbol{c}$ for display. It may be necessary to prefilter $P_{\varphi} f$ for display, or *at least* sample its reconstruction (see section 10 and figure 10.4):

$$\left[P_{\varphi} f\right] = \left[\boldsymbol{c} * \varphi\right] = \boldsymbol{c} * \left[\varphi\right]. \tag{4.23}$$

Recall that the orthogonal projection P_{φ} is equivalent to convolution with the dual $\mathring{\varphi}$ (figure 4.1c). To find the expression for $\mathring{\varphi}$ and verify that it belongs to V_{φ}, note that[1]:

$$\begin{aligned} \boldsymbol{c} &= \left[f * \varphi^{\vee}\right] * \left[a_{\varphi}\right]^{-1} = \left[f * \varphi^{\vee}\right] * \left[a_{\varphi}\right]^{-\vee} \\ &= \left[f * \varphi^{\vee} * \left[a_{\varphi}\right]^{-\vee}\right] = \left[f * \left(\varphi * \left[a_{\varphi}\right]^{-1}\right)^{\vee}\right] \end{aligned}$$

[1] Remember that $a_{\varphi} = \varphi * \varphi^{\vee}$ is always symmetric.

so that

$$\boldsymbol{c} = \left[f * \mathring{\tilde{\varphi}}^{\vee}\right] \quad \text{with} \quad \mathring{\varphi} = \varphi * \left[a_{\varphi}\right]^{-1}. \tag{4.24}$$

Thus, the traditional sampling approach would be to set the prefilter $\psi = \mathring{\varphi}$. The generalized sampling approach lets us avoid working directly with the dual $\mathring{\varphi}$, which typically has infinite support. We instead prefilter f with the more convenient, compactly supported kernel φ, and convolve with the inverse of the sampled auto-correlation.

As a side note, it is easy to verify the *biorthogonality* of φ and $\mathring{\varphi}$, i.e.

$$\left\langle \mathring{\varphi}(\cdot - i),\ \varphi(\cdot - j) \right\rangle = \delta_{ij}, \tag{4.25}$$

using our notation:

$$\begin{aligned}
\left[\varphi * \mathring{\tilde{\varphi}}^{\vee}\right] &= \left[\varphi * \left(\varphi * \left[a_{\varphi}\right]^{-1}\right)^{\vee}\right] && (4.26)\\
&= \left[\varphi * \varphi^{\vee} * \left[a_{\varphi}\right]^{-\vee}\right] && (4.27)\\
&= \left[\varphi * \varphi^{\vee}\right] * \left[a_{\varphi}\right]^{-\vee} && (4.28)\\
&= \left[a_{\varphi}\right] * \left[a_{\varphi}\right]^{-1} && (4.29)\\
&= \boldsymbol{\delta}. && (4.30)
\end{aligned}$$

McCool [1995] explored orthogonal projection in the context of rendering, but may have mistakenly used the correction filter $\boldsymbol{q} = \left[\varphi\right]^{-1} * \left[\varphi\right]^{-1}$ instead of $\boldsymbol{q} = \left[a_{\varphi}\right]^{-1}$ for generalized prefiltering.

Orthogonal kernels It is easy to show (via the frequency domain) that the ideal low-pass filter sinc equals its own auto-correlation. Because sinc satisfies the interpolation condition, it happens that sinc is also its own dual, i.e., it is an *orthogonal* kernel. This means that the ideal sampling procedure amounts to the orthogonal projection of function f into the space of bandlimited functions. Another example of orthogonal kernel is the box filter (its auto-correlation is the hat filter, which also interpolates). In general, other typical kernels are *not* orthogonal, so that φ and $\mathring{\varphi}$ are quite different.

Notable equivalent generating functions We have already seen three generating functions for the same approximation space V_{φ}: The basic φ itself, its cardinal φ_{int}, and its dual $\mathring{\varphi}$. To complete the picture, we now find an

orthogonal generating function ϕ for V_φ. This is mostly of theoretical interest. To do so, we numerically compute the "convolution square root" of $\left[a_\varphi\right]^{-1}$, via the Fourier series expansion of $\mathrm{DTFT}^{-1/2}(\left[a_\varphi\right])$. Then,

$$\phi = \varphi * \left[a_\varphi\right]^{-\frac{1}{2}} \quad \text{where} \quad \left[a_\varphi\right]^{-\frac{1}{2}} * \left[a_\varphi\right]^{-\frac{1}{2}} = \left[a_\varphi\right]^{-1}, \tag{4.31}$$

and we can verify that ϕ is indeed orthogonal:

$$\begin{aligned}
\left[\phi * \phi^\vee\right] &= \left[\varphi * \left[a_\varphi\right]^{-\frac{1}{2}} * \left(\varphi * \left[a_\varphi\right]^{-\frac{1}{2}}\right)^\vee\right] && (4.32)\\
&= \left[\varphi * \varphi^\vee\right] * \left[a_\varphi\right]^{-\frac{1}{2}} * \left[a_\varphi\right]^{-\frac{1}{2}} && (4.33)\\
&= \left[a_\varphi\right] * \left[a_\varphi\right]^{-1} && (4.34)\\
&= \boldsymbol{\delta}. && (4.35)
\end{aligned}$$

Figure 4.1 shows the four bases associated with the cubic B-splines.

4.4 Oblique projection

Unlike in rendering applications, many signal-processing applications typically have little control over the analysis filter $\psi^\vee$ (e.g., it is part of an acquisition device). Moreover, there may be little control over the reconstruction filter φ (e.g., it is part of a display device). Naturally, this prevents the use of the orthogonal projection. Instead, given both $\psi^\vee$ and φ, *consistent sampling* [Unser and Aldroubi, 1994] is a strategy for obtaining the *oblique projection* $P_{\varphi\perp\psi} f$ of a signal f into space V_φ, where the residual is orthogonal to V_ψ (rather than being orthogonal to V_φ):

$$\begin{aligned}
\left[(f - \tilde{f}) * \psi^\vee\right] &= \mathbf{0} \;\Rightarrow && (4.36)\\
\left[f * \psi^\vee\right] &= \left[\tilde{f} * \psi^\vee\right] && (4.37)\\
&= \left[\boldsymbol{c} * \varphi * \psi^\vee\right] && (4.38)\\
&= \boldsymbol{c} * \left[\varphi * \psi^\vee\right] \;\Rightarrow && (4.39)\\
\boldsymbol{c} &= \left[f * \psi^\vee\right] * \left[\varphi * \psi^\vee\right]^{-1}. && (4.40)
\end{aligned}$$

In other words,

$$\boldsymbol{c} = \left[f_\psi\right] * \boldsymbol{q} \quad \text{with} \quad \boldsymbol{q} = \left[\varphi * \psi^\vee\right]^{-1} = \left[a_{\varphi,\psi}\right]^{-1}, \tag{4.41}$$

where $a_{\varphi,\psi}$ is the cross-correlation of φ and ψ.

An equivalent characterization follows from the projection property. If a signal $f = \boldsymbol{c} * \varphi$ already belongs to V_φ, then its oblique projection $P_{\varphi\perp\psi} f$ must be f itself (hence, "consistent"):

$$\boldsymbol{c} * \varphi = P_{\varphi\perp\psi} (\boldsymbol{c} * \varphi) \;\Rightarrow \tag{4.42}$$

$$\boldsymbol{c} = \left[\boldsymbol{c} * \varphi * \psi^\vee\right] * \boldsymbol{q} = \boldsymbol{c} * \left[\varphi * \psi^\vee\right] * \boldsymbol{q} \;\Rightarrow \tag{4.43}$$

$$\boldsymbol{q} = \left[\varphi * \psi^\vee\right]^{-1} \tag{4.44}$$

Yet another characterization is that oblique projection selects $\boldsymbol{q}$ to make the effective analysis filter $\boldsymbol{q}^\vee * \psi$ biorthogonal to the reconstruction kernel φ:

$$\left[\varphi * (\boldsymbol{q}^\vee * \psi)^\vee\right] = \boldsymbol{\delta} \;\Rightarrow \tag{4.45}$$

$$\left[\varphi * \boldsymbol{q} * \psi^\vee\right] = \boldsymbol{\delta} \;\Rightarrow \tag{4.46}$$

$$\left[\varphi * \psi^\vee\right] * \boldsymbol{q} = \boldsymbol{\delta} \;\Rightarrow \tag{4.47}$$

$$\boldsymbol{q} = \left[\varphi * \psi^\vee\right]^{-1}. \tag{4.48}$$

The approximation error of the oblique projection is bounded by

$$\|f - P_\varphi f\| \;\le\; \|f - P_{\varphi\perp\psi} f\| \;\le\; (\cos\theta_{\psi,\varphi})^{-1} \|f - P_\varphi f\|, \tag{4.49}$$

where $\theta_{\psi,\varphi}$ is a measure of the "maximum angle" between the two spaces, as computed from their *spectral coherence* [Unser and Aldroubi, 1994].

5

Translation and scaling

We next examine how the generalized sampling algorithms can be applied to the problems of signal translation and scaling. These operations are obviously crucial in processing images.

5.1 Translation of discretized signals

The common practice for translating a discrete signal by an offset h is to sample the translated reconstruction $\tilde{f}(\cdot - h) = \tau_h * \tilde{f}$ with no prefiltering (i.e., with $\psi = \delta$):

$$\boldsymbol{c}_h = \left[\tau_h * \tilde{f}\,\right] = \left[\tau_h * \boldsymbol{c} * \varphi\right] = \boldsymbol{c} * \left[\tau_h * \varphi\right]. \tag{5.1}$$

The result is the discrete convolution between the coefficient array and the sampled, translated basis φ.

Translation using generalized sampling When the reconstruction kernel includes a digital filter $\boldsymbol{r}$, equation (5.1) does not directly produce the desired coefficient array. Instead, we must apply the filter:

$$\boldsymbol{c}_h = \boldsymbol{c} * \left[\tau_h * \varphi\right] * \boldsymbol{r}. \tag{5.2}$$

The new coefficient array $\boldsymbol{c}_h$ is then ready for reconstruction with φ (and further processing).

To complete the generalization, we must add a prefiltering stage to equation 5.2. Thanks to our notation, this can be done with simple algebraic manipulations:

$$\begin{aligned}
\boldsymbol{c}_h &= \left[(\tau_h * \tilde{f}) * (\boldsymbol{p} * \psi)^\vee\right] * \boldsymbol{r} && (5.3)\\
&= \left[\tau_h * \boldsymbol{c} * \varphi * \psi^\vee\right] * \boldsymbol{p}^\vee * \boldsymbol{r} && (5.4)\\
&= \boldsymbol{c} * \left[\tau_h * a_{\varphi,\psi}\right] * \boldsymbol{p}^\vee * \boldsymbol{r} && (5.5)\\
&= \boldsymbol{c} * \boldsymbol{q} \quad \text{with} \quad \boldsymbol{q} = \left[\tau_h * a_{\varphi,\psi}\right] * \boldsymbol{p}^\vee * \boldsymbol{r}. && (5.6)
\end{aligned}$$

To represent naïve translation within this generalized sampling framework, simply eliminate prefiltering by setting $\psi = \delta$ and $\boldsymbol{p} = \boldsymbol{r} = \boldsymbol{\delta}$ in (5.6). The result is (5.1). The generalized sampling method, however, can also express more sophisticated strategies.

Least-squares translation Unlike the space of band-limited functions V_{sinc}, the translation $\tau_h * \tilde{f}$ of a function $\tilde{f} \in V_\varphi$ does not in general belong to V_φ. Unser et al. [1995b] therefore explore forming its orthogonal projection $P_\varphi(\tau_h * \tilde{f})$ into V_φ and then sampling it. This is accomplished by setting $\boldsymbol{p} * \psi = \mathring{\varphi} = \varphi * \left[a_\varphi\right]^{-1}$ and $\boldsymbol{r} = \boldsymbol{\delta}$ in (5.6):

$$\boldsymbol{c}_h = \boldsymbol{c} * \left[\tau_h * a_\varphi\right] * \left[a_\varphi\right]^{-1} \quad \text{so that} \quad \left[P_\varphi(\tau_h * \tilde{f})\right] = \boldsymbol{c}_h * \left[\varphi\right]. \tag{5.7}$$

They further observe that (5.7) may be rewritten to resemble (5.1):

$$\boldsymbol{c}_h = \boldsymbol{c} * \left[\tau_h * (a_\varphi)_{\text{int}}\right]. \tag{5.8}$$

This means that least-squares translation in space V_φ is similar to naïve translation in the space V_{a_φ}. For example, if $\varphi = \beta^n$ is the B-spline of degree n (and order $L = n + 1$), then $a_\varphi = \beta^{2n+1}$ is the B-spline of degree $2n + 1$ (and order $L = 2n + 2$). Therefore, in this case, least-squares translation is equivalent to performing the naïve translation in a space that has twice the approximation order.

Note that in all of the translation algorithms, the reconstruction and prefilter kernels are sampled into the digital filter, so computing the translated sequences only involves a discrete convolution.

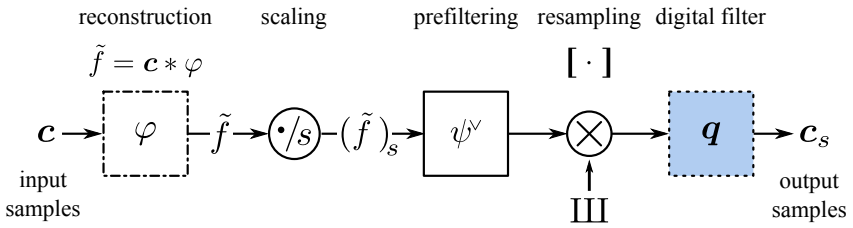


Figure 5.1: Scaling a sequence using generalized sampling. Given the input sequence $\boldsymbol{c}$, the reconstruction $\tilde{f} = \boldsymbol{c} * \varphi$ is scaled by a factor s. The scaled reconstruction $\tilde{f}_s$ is prefiltered and sampled, and the resulting sequence is convolved with a digital filter $\boldsymbol{q}$ to produce the new sequence $\boldsymbol{c}_s$.

5.2 Scaling of discretized signals

Scaling a discrete sequence is a more difficult operation than translation because in the general case it cannot be computed as a single convolution. The overall process is illustrated in figure 5.1.

We use the shorthand notation $f_s \stackrel{\text{def}}{=} f(\cdot/s)$ to denote f after a uniform scale by factor s. Using explicit sampling rates as in (3.4) and (3.7), we first note a few convenient relations:

$$(f * g)_s = \tfrac{1}{s}\, f_s * g_s, \tag{5.9}$$

$$(\boldsymbol{c} * \varphi)_s = \boldsymbol{c} *_s \varphi_s, \tag{5.10}$$

$$[\![f_s]\!] = [\![f]\!]_{\frac{1}{s}}, \quad \text{and} \tag{5.11}$$

$$f * \delta_s = sf. \tag{5.12}$$

In scaling a discrete sequence, we distinguish between *magnification* (*upsampling*, $s > 1$) and *minification* (*downsampling*, $s < 1$). These cases are treated independently.

When magnifying, the prefilter becomes redundant in the presence of a good reconstruction filter. To see this, recall that a good reconstruction filter has a cut-off frequency $\approx \frac{1}{2}$ cycles per *input* sample, whereas a good prefilter has cut-off at $\approx \frac{1}{2}$ cycles per *output* sample. Since the scaling operation is such that the output sampling rate is s times the input sampling rate, the reconstruction filter's cut-off frequency is s times *lower* than the prefilter's. Therefore, common practice is to sample the scaled reconstruction without

any prefiltering (i.e., $\psi = \delta$ when $s \geq 1$):

$$\left[\tilde{f}_s * \delta^\vee\right] = s\left[(\tilde{f} * \delta^\vee_{\frac{1}{s}})_s\right] \tag{5.13}$$

$$= s\left[\tilde{f} * \delta^\vee_{\frac{1}{s}}\right]_{\frac{1}{s}} \tag{5.14}$$

$$= \left[\tilde{f}\right]_{\frac{1}{s}} \tag{5.15}$$

$$= \left[\boldsymbol{c} * \varphi\right]_{\frac{1}{s}}. \tag{5.16}$$

See function `upsample` in lines 198–216 of appendix A for source-code implementing this algorithm.

Conversely, when minifying, it is the effect of the reconstruction filter that is hidden by the prefilter. Accordingly, common practice is to prefilter the discrete signal without any reconstruction (i.e., $\varphi = \delta$ when $s \leq 1$):

$$\left[\tilde{f}_s * \psi^\vee\right] = \left[(\boldsymbol{c} * \delta)_s * \psi^\vee\right] \tag{5.17}$$

$$= \left[\boldsymbol{c} *_s \delta_s * \psi^\vee\right] \tag{5.18}$$

$$= s\left[\boldsymbol{c} *_s \psi^\vee\right] \tag{5.19}$$

$$= s\left[(\boldsymbol{c} * \psi^\vee_{\frac{1}{s}})_s\right] \tag{5.20}$$

$$= s\left[\boldsymbol{c} * \psi^\vee_{\frac{1}{s}}\right]_{\frac{1}{s}} \tag{5.21}$$

$$= \left[\boldsymbol{c} * (s\,\psi_{\frac{1}{s}})^\vee\right]_{\frac{1}{s}}. \tag{5.22}$$

Note that the factor s within $s\,\psi_{\frac{1}{s}}$ in (5.22) acts to preserve the integral of the kernel ψ as it is scaled. In general, if s is not an integer, $s\psi_{\frac{1}{s}}$ does not satisfy a partition of unity even when ψ does. It is therefore common practice to add a normalization step to the sampling process. To do so, we accumulate the weights assigned to each entry in $\boldsymbol{c}$, and divide the resulting sampled value by this number. The normalization ensures the resampled sequence has the same average level as the input. See function `downsample` in lines 218–247 of appendix A for the corresponding source-code.

Scaling using generalized sampling As we did for translation, we now derive an efficient algorithm for finding the coefficient array $\boldsymbol{c}_s$ of a

scaled signal, but employing generalized reconstruction and prefiltering:

$$\begin{aligned}
\boldsymbol{c}_s &= \left[\tilde{f}_s * (\boldsymbol{p} * \psi)^\vee\right] * \boldsymbol{r} && (5.23)\\
&= \left[\tilde{f}_s * \psi^\vee\right] * \boldsymbol{p}^\vee * \boldsymbol{r} && (5.24)\\
&= \left[\boldsymbol{c} *_s \varphi_s * \psi^\vee\right] * \boldsymbol{p}^\vee * \boldsymbol{r} && (5.25)\\
&= \left[\boldsymbol{c} *_s (s\,\varphi * \psi_{\frac{1}{s}}^\vee)_s\right] * \boldsymbol{p}^\vee * \boldsymbol{r} && (5.26)\\
&= s\left[\boldsymbol{c} * \varphi * \psi_{\frac{1}{s}}^\vee\right]_{\frac{1}{s}} * \boldsymbol{p}^\vee * \boldsymbol{r} && (5.27)\\
&= s\left[\boldsymbol{c} * a_{\varphi,\psi_{\frac{1}{s}}}\right]_{\frac{1}{s}} * \boldsymbol{p}^\vee * \boldsymbol{r}. && (5.28)
\end{aligned}$$

The traditional algorithms (5.16) and (5.22), are special cases of (5.28). Note that the mixed convolution and the sampling operation use different spacings in (5.16), (5.22), and (5.28), and relation (3.17) does not apply. Therefore, we may have to evaluate the cross-correlation term $a_{\varphi,\psi_{\frac{1}{s}}}$ at the arbitrary positions $i/s - j$, with $i, j \in \mathbb{Z}$. This is inconvenient, because any closed-form expression for $a_{\varphi,\psi_{\frac{1}{s}}}$ is specific to a given s, which may only be known at runtime. Nevertheless, since $a_{\varphi,\psi_{\frac{1}{s}}}$ has compact support, we can compute all required values once and reuse them for all rows and columns in the image. Furthermore, when s is rational, the values repeat and we can compute a single cycle. Finally, when $\frac{1}{s}$ is an integer, we can simply compute $\boldsymbol{c} * \left[a_{\varphi,\psi_{\frac{1}{s}}}\right]$ and decimate the results by $\frac{1}{s}$ or, better yet, compute only the elements in the discrete convolution that remain after decimation.

Repeated integration The computation required to evaluate a single sample from $\boldsymbol{c} * {\psi_{\frac{1}{s}}}^\vee$ in (5.22) or from $\boldsymbol{c} * a_{\varphi,\psi_{\frac{1}{s}}}$ in (5.28) increases proportionally to $\frac{1}{s}$ (i.e., with the kernel support). Fortunately, when downsampling an entire image, the increased computation per sample is cancelled by the corresponding reduction in the total number of samples required.

Nevertheless, many applications (such as texture filtering) demand repeated access to individual samples. Heckbert [1986] introduces a repeated integration strategy (based on integrated arrays [Crow, 1984, Perlin, 1985]) that enables this sampling in *constant time* for piecewise polynomial kernels. Furthermore, the method allows us to avoid computing the value of scaled kernels altogether. As we have seen, this is particularly convenient when dealing with $a_{\varphi,\psi_{\frac{1}{s}}}$.

Heckbert considers the case of a B-spline analysis filter $\psi = \beta^n$, and sets $\varphi = \delta$ since reconstruction does not play a significant role in minification. Muñoz et al. [2001] reach the same algorithm, while including reconstruction with $\varphi = \beta^m$ and a digital filtering stage. In our notation, (5.28) becomes:

$$\boldsymbol{c}_s = s\Big[\boldsymbol{c} * \varphi * (\beta^n_{\frac{1}{s}})^\vee\Big]_{\frac{1}{s}} * \boldsymbol{p}^\vee * \boldsymbol{r} \tag{5.29}$$

$$= s\Big[\boldsymbol{c} * \varphi * \beta^n_{\frac{1}{s}}\Big]_{\frac{1}{s}} * \boldsymbol{p}^\vee * \boldsymbol{r} \tag{5.30}$$

Before delving into the derivation, we first provide high-level intuition. Piecewise polynomial functions can be expressed as linear combinations of shifted one-sided power functions. The expression for the B-spline β^n can be seen in (3.34) and only involves $u^{*(n+1)}$. As is apparent from (3.35), scaling a one-sided power function is equivalent to a multiplication by a constant factor:

$$\big(u^{*(n+1)}\big)_{\frac{1}{s}} = s^n\, u^{*(n+1)}. \tag{5.31}$$

Finally, sampling a mixed convolution with a one-sided power function can be performed in constant time with a combination of precomputed running sums (i.e., higher-order summed-area tables). In the case of B-splines, it turns out that only the $(n+1)^{\text{th}}$-order sum is required.

Consider the expression for the scaled B-spline function $\beta^n_{\frac{1}{s}}$ from (3.34):

$$\beta^n_{\frac{1}{s}} = \big(\boldsymbol{\Delta}^{*(n+1)} * u^{*(n+1)} * \tau_{-(n+1)/2}\big)_{\frac{1}{s}} \tag{5.32}$$

$$= s\,\big(\boldsymbol{\Delta}^{*(n+1)} * u^{*(n+1)}\big)_{\frac{1}{s}} * \big(\tau_{-(n+1)/2}\big)_{\frac{1}{s}} \tag{5.33}$$

$$= s\,\boldsymbol{\Delta}^{*(n+1)} *_{\frac{1}{s}} \big(u^{*(n+1)}\big)_{\frac{1}{s}} * \big(\tau_{-(n+1)/2}\big)_{\frac{1}{s}} \tag{5.34}$$

$$= \boldsymbol{\Delta}^{*(n+1)} *_{\frac{1}{s}} \big(u^{*(n+1)}\big)_{\frac{1}{s}} * \tau_{-(n+1)/(2s)} \tag{5.35}$$

$$= s^n\,\boldsymbol{\Delta}^{*(n+1)} *_{\frac{1}{s}} u^{*(n+1)} * \tau_{-(n+1)/(2s)}. \tag{5.36}$$

On the right-hand side of the mixed convolution with spacing $\frac{1}{s}$ in (5.36), we can see *almost* all terms in the definition of the unscaled B-spline

$$\beta^n = \boldsymbol{\Delta}^{*(n+1)} * u^{*(n+1)} * \tau_{-(n+1)/2}. \tag{5.37}$$

To obtain the missing terms without changing results, convolve (5.36) on the right-hand side with[1]:

$$\delta = \tau_{-(n+1)/2} * \tau_{(n+1)/2} \quad \text{and} \quad \boldsymbol{\delta} = \boldsymbol{u}^{*(n+1)} * \boldsymbol{\Delta}^{*(n+1)}. \tag{5.38}$$

[1] We must be careful because the mixed convolutions involve different spacings.

Thus,

$$\beta^n_{\frac{1}{s}} = s^n \, \boldsymbol{\Delta}^{*(n+1)} *_{\frac{1}{s}} \beta^n * \boldsymbol{u}^{*(n+1)} * \tau_{-(n+1)(1-s)/(2s)}. \tag{5.39}$$

Substituting into (5.30), we finally get

$$\begin{aligned}
\boldsymbol{c}_s &= s^{n+1} \Big[\boldsymbol{c} * \big(\boldsymbol{\Delta}^{*(n+1)} *_{\frac{1}{s}} \varphi * \beta^n * \tau_h * \boldsymbol{u}^{*(n+1)} \big) \Big]_{\frac{1}{s}} * \boldsymbol{p}^\vee * \boldsymbol{r} && (5.40)\\
&= s^{n+1} \Big[\boldsymbol{\Delta}^{*(n+1)} *_{\frac{1}{s}} \big(\boldsymbol{c} * \varphi * \beta^n * \tau_h * \boldsymbol{u}^{*(n+1)} \big) \Big]_{\frac{1}{s}} * \boldsymbol{p}^\vee * \boldsymbol{r} && (5.41)\\
&= s^{n+1} \Big[\boldsymbol{c} * \boldsymbol{u}^{*(n+1)} * \varphi * \beta^n * \tau_h \Big]_{\frac{1}{s}} * \boldsymbol{\Delta}^{*(n+1)} * \boldsymbol{p}^\vee * \boldsymbol{r} && (5.42)\\
&= \big[\boldsymbol{d} * \phi \big]_{\frac{1}{s}} * \boldsymbol{q}, && (5.43)
\end{aligned}$$

where

$$\begin{aligned}
\boldsymbol{d} &= \boldsymbol{c} * \boldsymbol{u}^{*(n+1)}, && (5.44)\\
\phi &= s^{n+1} \, \varphi * \beta^n * \tau_h, \quad \text{with} \quad h = -\tfrac{(n+1)(1-s)}{2s}, \quad \text{and,} && (5.45)\\
\boldsymbol{q} &= \boldsymbol{\Delta}^{*(n+1)} * \boldsymbol{p}^\vee * \boldsymbol{r}. && (5.46)
\end{aligned}$$

To summarise, the steps of the algorithm are as follows:

1. Obtain a new sequence $\boldsymbol{d}$ by applying $\boldsymbol{u}^{*(n+1)}$ to the input array $\boldsymbol{c}$ (thus at input resolution). This is equivalent to $n+1$ successive running sums, which can also be implemented (more efficiently) as a single recursive filter of order $n+1$;

2. Reconstruct with $\varphi * \beta^n$ shifted by h, multiply by s^{n+1}, and sample with spacing $\frac{1}{s}$. The cost of computing each output sample is *independent* of s, requiring $n+1$ coefficient array accesses when $\varphi = \delta$ and $m+n+2$ when $\varphi = \beta^m$.

3. Apply the digital filter $\boldsymbol{q}$ to the output (thus at output resolution). This combines a direct convolution with filter $\boldsymbol{\Delta}^{*(n+1)}$, which has support $n+2$, and with filters $\boldsymbol{p}^\vee$ and $\boldsymbol{r}$.

As noted by Heckbert [1986] and Muñoz et al. [2001], the repeated integration framework can be extended beyond B-splines to arbitrary piecewise polynomial kernels such as the Catmull-Rom filter. However, this generalization may require the computation of multiple coefficient arrays $\boldsymbol{d}_i = \boldsymbol{u}^{*i} * \boldsymbol{c}$.

Least-squares scaling Unser et al. [1995a] explore the use of orthogonal projection for scaling. This is accomplished by setting $\psi * \boldsymbol{p} = \mathring{\varphi} = \varphi * \big[a_\varphi\big]^{-1}$ and $\boldsymbol{r} = \boldsymbol{\delta}$ in (5.28):

$$\boldsymbol{c}_s = s\Big[\boldsymbol{c} * a_{\varphi,\varphi_{\frac{1}{s}}}\Big]_{\frac{1}{s}} * \big[a_\varphi\big]^{-1} \quad \text{so that} \quad \Big[P_\varphi \tilde{f}_s\Big] = \boldsymbol{c}_s * \big[\varphi\big]. \tag{5.47}$$

Restricting φ to the family of B-splines, Unser et al. [1995a] provide explicit formulas for the cross-correlation function in the piecewise-constant $\varphi = \beta^0$ and piecewise-linear $\varphi = \beta^1$ cases, and note that for higher orders the cross-correlation function $a_{\beta^n,\beta^n_{\frac{1}{s}}}$ quickly converges to a Gaussian (by the Central Limit Theorem).

In contrast, Lee et al. [1998] approach the problem from the consistent sampling perspective. The idea is to reconstruct with φ but prefilter with a kernel ψ that simplifies the computations. Lee et al. describe how to efficiently implement the case where $\varphi = \beta^n$ and $\psi = \beta^0$. Recall from (4.41) in section 4.3 that when reconstructing with φ and prefiltering with ψ, the digital correction filter that achieves oblique projection is simply $\boldsymbol{q} = \big[\varphi * \psi^\vee\big]^{-1}$. Applying this correction to (5.28) and setting $\boldsymbol{p} = \boldsymbol{r} = \boldsymbol{\delta}$ we obtain:

$$\boldsymbol{c}_s = s\Big[\boldsymbol{c} * a_{\varphi,\psi_{\frac{1}{s}}}\Big]_{\frac{1}{s}} * \big[a_{\varphi,\psi}\big]^{-1} \quad \text{so that} \quad \Big[P_{\varphi\perp\psi} \tilde{f}_s\Big] = \boldsymbol{c}_s * \big[\varphi\big]. \tag{5.48}$$

Muñoz et al. [2001] apply the repeated integration strategy to the problem of oblique projection of scaled signals, essentially combining the work of Lee et al. [1998] and of Heckbert [1986]. To do so, simply set $\varphi = \beta^m$, $\boldsymbol{p} = \boldsymbol{\delta}$, and $\boldsymbol{r} = \big[a_{\varphi,\beta^n}\big]^{-1} = \big[\beta^{m+n+1}\big]^{-1}$ in (5.42) to reach:

$$\boldsymbol{c}_s = s^{n+1}\Big[\boldsymbol{c} * \boldsymbol{u}^{*(n+1)} * \beta^{m+n+1} * \tau_h\Big]_{\frac{1}{s}} * \boldsymbol{\Delta}^{*(n+1)} * \big[\beta^{m+n+1}\big]^{-1}.$$

As Muñoz et al. [2001] note, the extension of the repeated integration method to linear combinations of B-splines and their derivatives allows the method to support the entire MOMS family [Blu et al., 2001].

Nearest-neighbor and linear filtering We can now give precise definitions for the terms "linear" and "nearest-neighbor" that are widely used in relation to magnification and minification. Using (5.28) as a starting point, we first eliminate the digital filtering by setting $\boldsymbol{p} = \boldsymbol{q} = \boldsymbol{\delta}$ since these are traditional algorithms. Intuitively, nearest-neighbor magnification means directly

sampling the box reconstruction ($\varphi = \beta^0$, $\psi = \delta$):

$$\boldsymbol{c}_s = \left[\boldsymbol{c} * \beta^0\right]_{\frac{1}{s}}. \tag{5.49}$$

Conversely, linear magnification means directly sampling the hat reconstruction ($\varphi = \beta^1$, $\psi = \delta$):

$$\boldsymbol{c}_s = \left[\boldsymbol{c} * \beta^1\right]_{\frac{1}{s}}. \tag{5.50}$$

In a perhaps counter-intuitive fashion, linear minification means directly prefiltering with the *box* kernel ($\varphi = \delta$, $\psi = \beta^0$):

$$\boldsymbol{c}_s = s\left[\boldsymbol{c} * \beta^0_{\frac{1}{s}}\right]_{\frac{1}{s}}. \tag{5.51}$$

This breaks the analogy to linear magnification, which uses the *hat* filter. Finally, nearest-neighbor minification uses the exact same equation as nearest-neighbor magnification. But since $s \leq 1$ for minification, the absence of a prefilter potentially leads to severe aliasing.

Pyramid specialization (Mipmaps) Equation (5.28) describes a family of downscaling algorithms parametrized by the choices of generalized analysis and reconstruction kernels, and by the scaling factor s. Here we describe a specialization that is useful in the generation of dyadic pyramids or mipmaps (i.e., $s = \frac{1}{2}$).

As discussed later in section 10, the cardinal cubic B-spline, i.e. $\psi * \boldsymbol{p} = (\beta^3)_{\text{int}}$, is an excellent prefilter. Using the traditional approach of reconstructing with the impulse $\varphi * \boldsymbol{r} = \delta$ and substituting in (5.28), we reach the following algorithm for the recursive generation of level i in an image pyramid, starting from an input image $\boldsymbol{f}_1$:

$$\boldsymbol{f}_i = \tfrac{1}{2}\left[\beta^3\right]^{-1} * \left[\boldsymbol{f}_{i-1} * \left[\beta^3_2\right]\right]_2. \tag{5.52}$$

The direct convolution is with kernel $\left[\beta^3_2\right]$, which has support 7. In the case of a dual-grid mipmap structure (see section 8), the direct convolution kernel is replaced by $\left[\beta^3_2 * \tau_{1/2}\right]$, which has support 8. Only the elements surviving decimation must be computed. These elements undergo inverse convolution with kernel $\left[\beta^3\right]$, which is symmetric with support 3, as per section 4.2. Exploiting separability and normalization, the full 2D process requires only 35

floating-point operations per output pixel, per channel (only 27 with multiply-add instructions (MADs)). In comparison, Catmull-Rom kernel requires 30 (22 with MADs), and a Lanczos windowed sinc ($W = 6$) requires 49 (34 with MADs).

6

Approximation of derivatives

In computer graphics and visualization, it is often desired to reconstruct not only the function but also its derivatives. For example, first derivatives contribute tangent vectors for surface shading, and second derivatives allow edge detection with subpixel accuracy.

The theory presented so far applies equally well to the n^{th} derivative $f^{(n)}$ of a given function f, so long as we have access to $f^{(n)}$. In that case, we can subject it to the generalized sampling pipeline and find its approximation in the order-L space V_φ of our choice:

$$\widetilde{f^{(n)}} = \left[f^{(n)} * \psi^\vee\right] * \boldsymbol{q} * \varphi, \tag{6.1}$$

in the same way we would have obtained an approximation to f:

$$\tilde{f} = \left[f * \psi^\vee\right] * \boldsymbol{q} * \varphi. \tag{6.2}$$

However, our access is often restricted to $\tilde{f}$. Given only the coefficient array $\left[f * \psi^\vee\right]$, we wish to obtain an approximation to $f^{(n)}$. The most obvious approach is to differentiate $\tilde{f}$, which is equivalent to reconstructing with the derivative $\varphi^{(n)}$ of the original kernel:

$$(\tilde{f})^{(n)} = \left(\left[f * \psi^\vee\right] * \boldsymbol{q} * \varphi\right)^{(n)} = \left[f * \psi^\vee\right] * \boldsymbol{q} * \varphi^{(n)}. \tag{6.3}$$

Unfortunately, reasoning about the approximation order of this scheme is not easy. We may even fail to obtain an approximation to $f^{(n)}$ as we reduce the sample spacing. For one thing, the approximation order cannot be higher than the order of the new approximation space $V_{\varphi^{(n)}}$ where $(\tilde{f}\,)^{(n)}$ lives.

Some important tools for understanding and solving this problem are presented in the work of Condat and Möller [2011]. The insight is to use properties (3.29) and (3.34):

$$\begin{aligned}
\big[f * \psi^\vee\big] &= \big[f * \big((\delta')^{*n} * u^{*n}\big) * \psi^\vee\big] && (6.4)\\
&= \big[f^{(n)} * u^{*n} * \psi^\vee\big] && (6.5)\\
&= \big[f^{(n)} * \boldsymbol{u}^{*n} * \beta^{n\text{-}1} * \tau_{n/2} * \psi^\vee\big] && (6.6)\\
&= \big[f^{(n)} * \beta^{n\text{-}1} * \tau_{n/2} * \psi^\vee\big] * \boldsymbol{u}^{*n} && (6.7)
\end{aligned}$$

and therefore

$$\begin{aligned}
\big[f * \psi^\vee\big] * \boldsymbol{\Delta}^{*n} &= \big[f^{(n)} * \beta^{n\text{-}1} * \tau_{n/2} * \psi^\vee\big] && (6.8)\\
&= \big[f^{(n)} * \big(\beta^{n\text{-}1} * \tau_{-n/2} * \psi\big)^\vee\big] && (6.9)\\
&= \big[f^{(n)} * \xi^\vee\big], \quad \text{with} \quad \xi = \beta^{n\text{-}1} * \tau_{-n/2} * \psi. && (6.10)
\end{aligned}$$

In other words, starting from the samples of f filtered with ψ, and after applying the discrete derivative $\boldsymbol{\Delta}^{*n}$, we obtain samples of $f^{(n)}$, but these samples come filtered with ξ instead. We can now apply our standard tools to reason about the asymptotic behavior of the approximation

$$\begin{aligned}
\widetilde{f^{(n)}} &= \big[f^{(n)} * \xi^\vee\big] * \boldsymbol{q} * \varphi\\
&= \big[f * \psi^\vee\big] * \boldsymbol{\Delta}^{*n} * \boldsymbol{q} * \varphi.
\end{aligned} \qquad (6.11)$$

For example, to guarantee that we achieve the optimal approximation order in V_φ, we can use oblique projection and set $\boldsymbol{q} = \big[a_{\varphi,\xi}\big]^{\text{-}1}$. Another remarkable consequence from this analysis is that we can obtain from samples of f the exact orthogonal projection of $f^{(n)}$ in space $V_{\beta^{n-1}(\,\bullet + \frac{n}{2})}$, since in that case the oblique projection reduces to orthogonal projection.

Concrete examples Consider the typical case in which we are provided with samples from f and we assume $\psi = \delta$. As an example, we derive a 4^{th}-order approximation to the second derivative $f^{(2)}$ of f in the space V_{β^3} of

cubic B-splines. To do so, we simply use oblique projection by setting $\psi = \delta$ and $\varphi = \beta^3$ in (6.11), from which $\xi = \beta^1 * \tau_1$:

$$\widetilde{f^{(2)}} = \left[f^{(2)} * \xi^\vee\right] * \left[\varphi * \xi^\vee\right]^{-1} * \varphi \tag{6.12}$$

$$= \left[f\right] * \boldsymbol{\Delta}^{*2} * \left[\beta^3 * \beta^1 * \tau_1\right]^{-1} * \beta^3 \tag{6.13}$$

$$= \left[f\right] * \left[\underset{\cdot}{1}, -2, 1\right] * \left[\beta^5 * \tau_1\right]^{-1} * \beta^3 \tag{6.14}$$

$$= \left[f\right] * \left[1, \underset{\cdot}{-2}, 1\right] * \left[\beta^5\right]^{-1} * \beta^3. \tag{6.15}$$

Incidentally, this is equivalent to taking the second derivative of the interpolation of f in the space of quintic B-splines, since

$$(\beta^5)^{(2)} = (\beta^1 * \beta^3)^{(2)} \tag{6.16}$$

$$= \left(\boldsymbol{\Delta}^{*2} * u^{*2} * \tau_{-2/2}\right)^{(2)} * \beta^3 \tag{6.17}$$

$$= \boldsymbol{\Delta}^{*2} * \tau_{-1} * \beta^3 \tag{6.18}$$

$$= \left[1, \underset{\cdot}{-2}, 1\right] * \beta^3. \tag{6.19}$$

Conversely, the naïve idea of differentiating the interpolation of f in space V_{β^3} is equivalent to performing the oblique projection of $f^{(2)}$ into space V_{β^1}; it is therefore only a second-order approximation strategy.

Similarly, for the 4th-order approximation to the first derivative, we have:

$$\widetilde{f}' = \left[f\right] * \boldsymbol{\Delta} * \left[\beta^4 * \tau_{1/2}\right]^{-1} * \beta^3. \tag{6.20}$$

Note the half-pixel shift that breaks the analogy with the derivative of the interpolation in space V_{β^4}, which would result in:

$$\widetilde{f}' = \left[f\right] * \boldsymbol{\Delta} * \left[\beta^4\right]^{-1} * \beta^3 * \tau_{-1/2}. \tag{6.21}$$

The latter is, of course, what we would obtain by performing the oblique projection of f' in the space of half-shifted cubic B-splines $V_{\beta^3(\cdot+\frac{1}{2})}$, and therefore is also a 4th-order approximation scheme.

Often, we must reconstruct estimates for both f and $f^{(n)}$. The concrete methods we described so far assume we have independent coefficient arrays for $\tilde{f}$ and $\widetilde{f^{(n)}}$. This is because the digital filters must be precomputed, since they are inverse discrete convolutions. In volume-rendering applications that require gradients, keeping 4 versions of the coefficient volume (for f and its 3 partial derivatives) can be prohibitive in terms of memory consumption. Fortunately, we are not required to use oblique projection to guarantee an optimal

approximation order to $f^{(n)}$. (See section 9 for details.) In recent follow-up work, Alim et al. [2010] used this additional freedom to derive FIR filters that can be applied locally, on demand, on the interpolating coefficients for f. This allows sharing the same coefficient vector for optimal-order approximations to f and its partial derivatives.

7

Generalized prefiltering and estimator variance

We previously explored how images can be downscaled using a generalized prefilter, expressed as a mixed convolution of a compact kernel ψ and a digital filter $\boldsymbol{p}$ (section 5.2). This strategy attains efficiency due to the compactness of kernel ψ yet reduces aliasing and maintains sharpness due to the improved frequency response of the effective kernel $\psi * \boldsymbol{p}$.

We now apply this same generalized prefiltering strategy to the case where the input function f is defined procedurally rather than sampled. This is particularly relevant to computer graphics because procedural definitions abound in the rendering process:

- Vector representations define shapes with intricate boundaries;
- Procedural texturing assigns material properties to surfaces;
- Procedural lighting defines local illumination over the surface;
- Procedural shading evaluates color from materials and lighting;
- Perspective projection maps scenes surfaces to the rendered image.

An important challenge in rendering is that any of these procedures may introduce high frequencies into the rendered image f. Troublesome cases

include sharp outlines, noise textures, shadow discontinuities, specular highlights, surface silhouettes, and perspective distortions near vanishing points.

The general approach to overcome this problem of discontinuities is to prefilter the function f, typically by *supersampling*. We evaluate f at several positions per pixel and integrate these samples using a prefilter kernel. The resulting estimate at each pixel suffers from aliasing and noise. Aliasing is attenuated by improving the frequency response of the prefilter, and noise is attenuated by reducing the variance of the supersampling estimate. We next analyze these characteristics in the generalized sampling pipeline. In particular, we investigate whether it is advantageous to reuse samples across pixels when evaluating prefilter kernels.

Computing the prefiltered sample at pixel k from signal f entails evaluating the integral:

$$b_k = (f * \psi^\vee)(k) = \int_{-\infty}^{\infty} f(x)\, \psi(x-k)\, \mathrm{d}x. \tag{7.1}$$

When exact evaluation is impractical, the integral is approximated by Monte Carlo integration (i.e, by *supersampling*). The idea is to rewrite the integral as an expectation, and to approximate it with a sample mean. Although changing the sample distribution can reduce the variance of the estimator (see *quasi*-Monte Carlo, importance sampling), or even reshape its spectral properties [Dippé and Wold, 1985], here our interest is analyzing the effect of the generalized prefilter $\psi * \boldsymbol{p}$ on the variance of the estimator.

Sharing samples To share samples between pixels, we start by tiling the real line with the support of uniformly distributed random variables. To that end, let $X_i \sim U(-.5, .5) + i - o_W$, where o_W takes value .5 when the support width W of ψ is even, and 0 otherwise. Using X_i, we can rewrite (7.1) as a sum of expectations:

$$b_k = \sum_{i \in \mathbb{Z}} \mathrm{E}_{X_i}\big(f(\cdot)\, \psi(\cdot - k)\big), \tag{7.2}$$

since $\mathrm{E}_{X_i}(g) = \int g(x)\, p_{X_i}(x)\, \mathrm{d}x$ and the uniform probability density function $p_{X_i} = 1$. To make the analysis practical, we assume that f is stationary and $f(X_i)$ is uncorrelated to X_i. This is reasonable if the frequency content of f is much higher than the supersampling rate. Therefore, $\mathrm{E}_{X_i}(f) = \mathrm{E}_{X_j}(f)$ for all i and j, and we denote the common value by $\mathrm{E}_X(f)$.

The unbiased Monte Carlo estimator for b_k is:

$$\hat{b}_k = \frac{1}{m} \sum_{j=1}^{m} \sum_{i \in \mathbb{Z}} f(x_{i,j})\, \psi(x_{i,j} - k), \tag{7.3}$$

where m variates $x_{i,j}$ are drawn from each X_i. Because samples are shared between $\hat{b}_k$ and $\hat{b}_{k+w}$ whenever $|w| < W$, the estimators $\hat{b}_k$ and $\hat{b}_{k+w}$ are not independent random variables. The covariance is:

$$\begin{aligned} \operatorname{Cov}(\hat{b}_k, \hat{b}_{k\text{-}w}) = {} & \tfrac{1}{m} \mathrm{E}_X(f^2) \textstyle\sum_i \mathrm{E}_{X_i}\big(\psi(\cdot)\, \psi(\cdot + w)\big) \\ & - \tfrac{1}{m} \mathrm{E}_X^2(f) \textstyle\sum_i \mathrm{E}_{X_i}(\psi)\, \mathrm{E}_{X_i}\big(\psi(\cdot + w)\big), \end{aligned} \tag{7.4}$$

and the variance (i.e. $w = 0$) simplifies to:

$$\operatorname{Var}(\hat{b}_k) = \tfrac{1}{m} \mathrm{E}_X(f^2) \textstyle\sum_i \mathrm{E}_{X_i}(\psi^2) - \tfrac{1}{m} \mathrm{E}_X^2(f) \textstyle\sum_i \mathrm{E}_{X_i}^2(\psi). \tag{7.5}$$

In other words, the kernels that reduce the variance the most are those with small $\sum_i \mathrm{E}_{X_i}(\psi^2)$ and large $\sum_i \mathrm{E}_{X_i}^2(\psi)$.

As for the digital filter $\boldsymbol{p}$, it acts by computing a weighted sum of $\hat{b}_k$:

$$\boldsymbol{c} = \boldsymbol{p} * \boldsymbol{b} \quad \Rightarrow \quad \hat{c}_k = \sum_{i \in \mathbb{Z}} p_i \hat{b}_{k-i}. \tag{7.6}$$

Keeping in mind that the b_k are not independent,

$$\operatorname{Var}(\hat{c}_k) = \sum_{w \in \mathbb{Z}} \sum_{i \in \mathbb{Z}} p_i p_{i+w} \operatorname{Cov}(\hat{b}_{k\text{-}i}, \hat{b}_{k\text{-}i\text{-}w}) \tag{7.7}$$

$$= \sum_{w=-W+1}^{W-1} (\boldsymbol{a}_p)_{\text{-}w} \operatorname{Cov}(\hat{b}_k, \hat{b}_{k\text{-}w}). \tag{7.8}$$

Only $2W - 1$ elements from sequence $\boldsymbol{a}_p = \boldsymbol{p} * \boldsymbol{p}^\vee$ are associated to nonzero covariances. Furthermore, since $\boldsymbol{a}_p$ is symmetric, only W of them are distinct. These can be computed exactly via integrals of the DTFT of $\boldsymbol{p}$ (using Parseval's theorem and the time-shift property). Substituting (7.4) and (7.5) into (7.8), and collecting together the terms multiplying $\mathrm{E}_X(f^2)$ and $\mathrm{E}_X^2(f)$ into coefficients $S_{\psi,\boldsymbol{p}}$ and $s_{\psi,\boldsymbol{p}}$, respectively, we reach our final expression for the variance:

$$\operatorname{Var}(\hat{c}_k) = \tfrac{1}{m} \big(S_{\psi,\boldsymbol{p}}\, \mathrm{E}_X(f^2) - s_{\psi,\boldsymbol{p}}\, \mathrm{E}_X^2(f)\big). \tag{7.9}$$

Table 7.1: Analysis of variance when supersampling with various prefilters. Constants $S_{\psi,\boldsymbol{p}}$, $s_{\psi,\boldsymbol{p}}$, $N_{\psi,\boldsymbol{p}}$, and $n_{\psi,\boldsymbol{p}}$, are the factors from (7.9) and (7.13). Column σ_U gives the standard deviation of estimator $\hat{c}_k$ using $m = 1$ sample per unit pixel area and signal $f \sim U(0,1)$ uniformly distributed. (β^N_{int} the cardinal B-splines; K the Catmull-Rom filter; M the Mitchell-Netravali filter.)

Prefilter	Sharing samples						No sample sharing					
	$S_{\psi,\boldsymbol{p}}$		$s_{\psi,\boldsymbol{p}}$		σ_U		$N_{\psi,\boldsymbol{p}}$		$n_{\psi,\boldsymbol{p}}$		σ_U	
	$1D$	$2D$	$1D$	$2D$	$1D$	$2D$	$1D$	$2D$	$1D$	$2D$	$1D$	$2D$
β^5_{int}	.92	.84	.70	.49	.36	.40	29.3	856	12.4	153	2.6	15.8
β^3_{int}	.87	.76	.66	.43	.36	.38	6.64	44.1	3.46	12.0	1.16	3.42
Lanczos	.89	.79	.67	.45	.36	.39	5.33	28.4	.994	.988	1.24	3.03
K	.81	.66	.59	.35	.35	.37	3.28	10.6	1	1	.914	1.81
M	.68	.46	.53	.28	.31	.29	2.73	7.43	1	1	.812	1.49
β^0	1	1	1	1	.29	.29	1	1	1	1	.29	.29

No sample sharing The analysis is simpler when no sample-sharing is involved. We use independent variables Y, uniformly distributed over the support Ω of ψ (with area A_Ω). Equation (7.3) simplifies to:

$$\hat{b}_k = \frac{A_\Omega}{m} \sum_{j=1}^{m} f(y_j)\, \psi(y_j - k). \tag{7.10}$$

Independence also leads to simpler versions of (7.5) and (7.8):

$$\operatorname{Var}(\hat{b}_k) = \tfrac{A_\Omega^2}{m} \operatorname{E}_Y(f^2) \operatorname{E}_Y(\psi^2) - \tfrac{A_\Omega^2}{m} \operatorname{E}_Y^2(f) \operatorname{E}_Y^2(\psi), \text{ and} \tag{7.11}$$

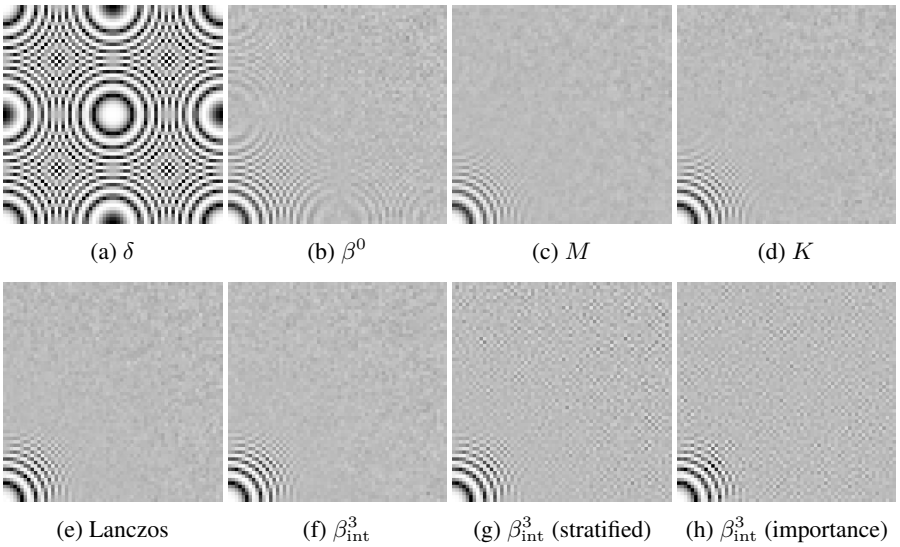
$$\operatorname{Var}(\hat{c}_k) = \|\boldsymbol{p}\|^2 \operatorname{Var}(\hat{b}_k). \tag{7.12}$$

Collecting the terms on $\operatorname{E}_Y(f^2)$ and $\operatorname{E}_Y^2(f)$ into coefficients $N_{\psi,\boldsymbol{p}}$ and $n_{\psi,\boldsymbol{p}}$, respectively, we reach an expression analogous to (7.9):[1]

$$\operatorname{Var}(\hat{c}_k) = \tfrac{1}{m}\left(N_{\psi,\boldsymbol{p}} \operatorname{E}_Y(f^2) - n_{\psi,\boldsymbol{p}} \operatorname{E}_Y^2(f)\right). \tag{7.13}$$

In the second term of (7.11), note that $\operatorname{E}_Y(\psi) = 1/A_\Omega$ if the kernel ψ has unit integral. Therefore this second term simplifies to $-\frac{1}{m}\operatorname{E}_Y^2(f)$, which is independent of the choice of kernel ψ. Consequently, we get $n_{\psi,\boldsymbol{p}} \approx 1$ for all the traditional kernels, i.e. those for which the digital filter $\boldsymbol{p}$ is the identity. (The Lanczos kernel does not precisely have unit integral, so its value $n_{\psi,\boldsymbol{p}}$ differs slightly from 1.)

[1] Note that $\operatorname{E}_Y^2(\psi) = 1$ for normalized prefilters.



(a) δ (b) β^0 (c) M (d) K

(e) Lanczos (f) β^3_{int} (g) β^3_{int} (stratified) (h) β^3_{int} (importance)

Figure 7.1: A 128^2 zonal plate, rendered with 5^2 stratified samples per pixel spacing area. (a-f) Various prefilters show particular sharpness and aliasing-suppression properties, but lead to similar noise reduction. (g,h) Sharing of samples is crucial to prevent the digital filtering stage from boosting noise, as seen when we independently integrate each pixel with 20^2 stratified samples (g), or even when using 400 importance-samples (h).

Discussion The values of constants $S_{\psi,\boldsymbol{p}}$, $s_{\psi,\boldsymbol{p}}$ $N_{\psi,\boldsymbol{p}}$ and $n_{\psi,\boldsymbol{p}}$ for a variety of filtering strategies are shown in table 7.1. As a measure of visible noise, the table also lists the standard deviation of estimators ($m = 1$) when sampling from a white-noise function.

As expected, the prefilter that gives the greatest factor in variance reduction is the box filter, but its low-pass properties are lacking (see figure 7.1b). The Mitchell-Netravali filter is noteworthy in having similar variance reduction and greatly improved response.

The most important takeaway from this analysis is that the remaining filters (in particular, the nice cardinal splines highlighted in blue) are not that much worse in terms of variance reduction, *as long as samples are shared*. The correlations due to sharing act to attenuate the effect of the digital filter $\boldsymbol{p}$ over the variance, which is otherwise large (as highlighted in red in the table). In fact, sharing samples performs better then not sharing even when the number of samples integrated under each kernel support is the same in both cases (compare figure 7.1f,g), and even when using importance sampling

(compare figure 7.1f,h). And of course, sharing samples is a huge benefit computationally, as it significantly reduces the number of evaluations of the input signal.

8

Practical considerations

8.1 Grid structure

Given a function f defined over domain $[0, 1]$, there are several conventions for where to position n uniformly spaced samples. The simplest may be the *primal grid structure*, which places sample c_k at position k/n. However, this leads to asymmetry at the boundaries since the first sample is at 0 and the last sample is at $(n-1)/n \neq 1$. Samples can also be placed at positions $k/(n-1)$ but this requires non-power-of-two resolutions in an image pyramid. For these reasons, computer graphics usually prefers the *dual grid structure*, which places sample c_k at position $(k + \frac{1}{2})/n$. We have adopted the primal structure during our derivations, but it is easy to transition between the two. Given the sample spacing $T = 1/n$, the primal-grid operations of sampling and reconstruction are:

$$\boldsymbol{c}_T = \left[f * \psi_T^\vee \right]_T \quad \Longleftrightarrow \quad \tilde{f} = \boldsymbol{c}_T *_T \varphi_T, \tag{8.1}$$

whereas the dual-grid counterparts are:

$$\boldsymbol{c}_T = \left[f * \psi_T^\vee(\cdot - \tfrac{T}{2}) \right]_T \quad \Longleftrightarrow \quad \tilde{f} = \boldsymbol{c}_T *_T \varphi_T(\cdot - \tfrac{T}{2}). \tag{8.2}$$

In a multiresolution representation, it is useful to obtain a nesting of the function spaces defined at the various resolutions, so that any reconstruction

at a coarser level can be represented exactly in the space of reconstructions at finer levels. The B-spline family of kernels offers this refinability. The main constraint is that even-degree kernels must follow a primal structure, whereas odd-degree kernels must follow a dual structure. Thus, for compatibility with the dual-grid structure, we often use B-splines of degree 3 and 5.

8.2 Efficient use of piecewise-polynomial kernels

To illustrate the efficient use of piecewise-polynomial kernels, we examine the common operations of sequence magnification and minification discussed in section 5.2. We start with an intuitive implementation.

The task in magnification is to obtain the sequence $\boldsymbol{c}_s = \big[\boldsymbol{c} * \varphi\big]_{\frac{1}{s}}$ expressed in (5.16). Assuming a reconstruction kernel φ with support W:

$$\begin{gathered}(\boldsymbol{c}_s)_j = \sum_{i\in\mathbb{Z}} \boldsymbol{c}_i\, \varphi(j/s - i) = \sum_{i=\ell_j}^{r_j} \boldsymbol{c}_i\, \varphi(j/s - i), \quad \text{with} \\ \ell_j = \lceil j/s - W/2 \rceil \quad \text{and} \quad r_j = \lfloor j/s + W/2 \rfloor.\end{gathered} \tag{8.3}$$

This is the intuitive algorithm implemented by the `upsample` function in appendix A, lines 198–216 (except for using the dual grid structure).

The minification formula starts from $\boldsymbol{c}_s = \big[\boldsymbol{c} * (s\,\psi_{\frac{1}{s}})^\vee\big]_{\frac{1}{s}}$ as in (5.21). Assuming this time an analysis filter ψ with support W:

$$\begin{gathered}(\boldsymbol{c}_s)_j = s \sum_{i\in\mathbb{Z}} \boldsymbol{c}_i\, \psi^\vee_{\frac{1}{s}}(j/s - i) = s \sum_{i=\ell_j}^{r_j} \boldsymbol{c}_i\, \psi^\vee(j - is), \quad \text{with} \\ \ell_j = \lceil j/s - W/(2s) \rceil \quad \text{and} \quad r_j = \lfloor j/s + W/(2s) \rfloor.\end{gathered} \tag{8.4}$$

This is the intuitive algorithm implemented by the `downsample` function in appendix A, lines 218–247 (again except for the dual grid structure).

Equations (8.3) and (8.4) allow direct random-access evaluation of any sample $(\boldsymbol{c}_s)_j$. While this freedom is useful in a variety of applications, it has its costs. First, because the kernels are piecewise polynomial, evaluating them at arbitrary offsets $\varphi(j/s - i)$ or $\psi^\vee(j - is)$ requires branching on the correct polynomial piece for each loop iteration i. Second, the algorithms often read each input sample multiple times. Instead, it is preferable to traverse the input only once to reduce expensive memory accesses and indexing computations near boundaries. When the task at hand is to magnify or minify an entire image

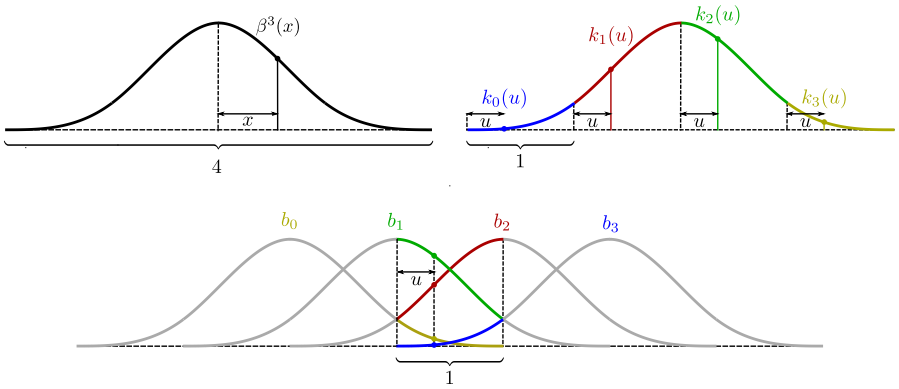


Figure 8.1: Efficient piecewise-polynomial kernel evaluation. The cubic B-spline kernel defined over the domain $x \in (-2, 2]$ (top left) is divided into its 4 polynomial pieces, each defined inside a unit domain $u \in (0, 1]$ (top right). During magnification, we keep track of the value of u associated to each output sample. During minification, we keep track of the value of u associated to each input sample. That way we always know which polynomial piece to use.

(a common case), it is possible to eliminate these inefficiencies, leading to performance improvements of 4–10×.

As an example, consider the top of figure 8.1. It shows the cubic B-spline kernel, with support $W = 4$, divided into its 4 polynomial pieces, k_0, k_1, k_2, k_3, each covering a unit interval. We set up the computation so that we always know which piece to use, as shown in the bottom of figure 8.1.

In the case of magnification, consider b_0, b_1, b_2, b_3 as input samples stored in a FIFO buffer. It is clear from figure 8.1 that to evaluate any output sample f_u situated at u along the way between b_1 and b_2, we can use the expression:

$$f_u \leftarrow b_0 k_3(u) + b_1 k_2(u) + b_2 k_1(u) + b_3 k_0(u). \tag{8.5}$$

By keeping these four samples in fast local variables, we can generate all output samples in the interval by incrementally adding $\frac{1}{s}$ to u and evaluating the exact same expression. When $u > 1$, we advance the input by shifting a new sample into the FIFO buffer, subtract 1 from u, and repeat the process. This incremental algorithm implemented within function `upsample2` in lines 256–274 of appendix A.

The incremental minification algorithm is very similar. The difference is that now b_0, b_1, b_2, b_3 each represent an *output* sample accumulating weighted input samples. From figure 8.1, the contribution of an input sample f_u situ-

ated u along the way between b_1 and b_2 is simply:

$$b_0 \leftarrow b_0 + f_u\, k_3(u), \qquad b_1 \leftarrow b_1 + f_u\, k_2(u), \tag{8.6}$$

$$b_2 \leftarrow b_2 + f_u\, k_1(u), \qquad b_3 \leftarrow b_3 + f_u\, k_0(u). \tag{8.7}$$

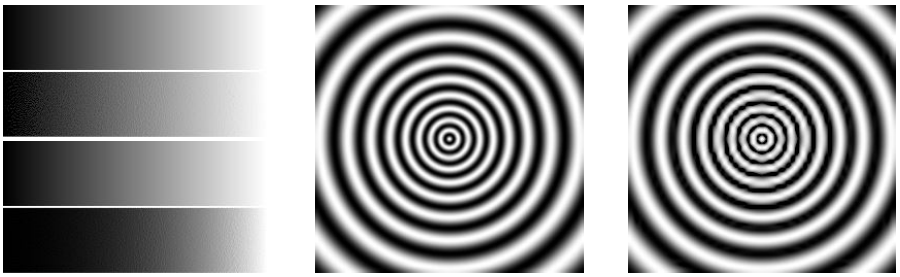
We can therefore iterate over the input, incrementally adding s to u and updating the partial output samples in the buffer using the expressions above. Whenever $u > 1$, we subtract 1 from u, advance the output by saving the value $s\, b_0$, shift a 0 into the FIFO buffer, and repeat the process. This incremental algorithm is implemented within function `downsample2` in lines 276–303 of appendix A.

GPU reconstruction Texture samplers on GPUs often have custom hardware to perform bilinear reconstruction at reduced cost. Moving to higher-quality reconstruction may incur a performance penalty if these bilinear samplers are underutilized. For instance, an implementation of the bicubic Mitchell-Netravali filter requires $4\times4=16$ texture reads, versus 1 read for bilinear filtering. As shown by Sigg and Hadwiger [2005], even though the bicubic B-spline basis has the same 4×4 support, the fact that it is non-negative and separable allows it to be evaluated by combining just 4 bilinear reads at appropriately computed locations. Ruijters et al. [2008] also describe a similar trick. Depending on the GPU, our implementation of this idea runs 3–6$\times$ slower than bilinear filtering, but about 2$\times$ faster than Catmull-Rom or Mitchell-Netravali filters.

8.3 Prefiltering, reconstruction, and color spaces

In reconstruction, the average between two function values should be *perceived* to have the intermediate value between them. Luminance values do not have this property due to a nonlinear response in the human visual system. This nonlinear perceptual response to luminance is called *lightness*. The gamma correction in the sRGB standard approximates the conversion of luminance to lightness. Therefore, when processing visual data we recommend performing reconstruction directly in sRGB space to achieve better perceptual results.

In stark contrast, prefiltering is an area integral of radiant energy. It must be performed over linear intensities and *only then* can results be moved to a



(a) (b) (c)

Figure 8.2: Prefiltering, reconstruction, and color spaces. (a) A large image with two ramps (one continuous and one dithered) was downsampled by prefiltering linear intensities (top ramps). Downsampled ramps look the same. When prefiltering is performed in lightness space, the results do not match (bottom ramps). (b) An image upsampled with quintic B-spline interpolation in lightness space. (c) The same image upsampled in linear intensities appears incorrect.

perceptual color space [Blinn, 1998]. As figure 8.2 demonstrates, prefiltering is best done in luminance space and reconstruction is best done in lightness space.

8.4 Range constraints

An important consideration is that the digital filter $\boldsymbol{q}$ may extend the range of values in an image. For example, a signal f with range bounded to $[0, 1]$ may require a coefficient vector $\boldsymbol{c}$ with values outside this range, especially if f has high-frequency content. While this larger range is likely acceptable for floating-point or HDR image representations, it is a concern for common 8-bit fixed-point formats.

To find an approximation for interpolation coefficients $\boldsymbol{c}$ within an 8-bit image format, we set its quantized range to a somewhat larger interval $[\ell, h]$, and solve the constrained optimization:

$$\bar{\boldsymbol{c}} = \arg\min_{\boldsymbol{c}} \left\| \boldsymbol{c} * \boldsymbol{[}\varphi\boldsymbol{]} - \boldsymbol{f} \right\|^2, \quad \text{s.t.} \quad \ell \leq c_k \leq h. \tag{8.8}$$

This is a least squares problem with simple bounds constraints. Efficient direct solvers exist, e.g. the Matlab function lsqlin. Alternatively, we obtain good approximations using just five iterations of Gauss-Seidel relaxation, wherein the update of each pixel value is clamped to the bounds constraints.

Experimenting on natural images, we find that, for bicubic B-splines, setting the expanded range $[\ell, h] = [-0.5, 1.5]$ gives good results with respect to both an $\mathcal{L}_2$ and SSIM metric. One might expect that the sampled reconstruction $\mathbf{[}\tilde{f}\mathbf{]}$ may only interpolate a given input f with 7 bits of precision, because the quantized $\boldsymbol{c}$ values now span twice the range. Surprisingly, this is not the case for natural images. With the bicubic B-spline basis, each pixel is reconstructed by a weighted combination of 16 coefficients of $\boldsymbol{c}$. This combination effectively provides sub-quantum precision. Even with the simple constrained optimization (8.8), which is unaware of the subsequent quantization process, a reconstruction of the original image of figure 10.5 from an 8-bit-quantized vector $\boldsymbol{c}$ has an rms error of 0.07% and a maximum error of 12%, with about 93% of pixel values reproduced exactly. Intuitively, the bicubic B-spline gives more precision to low frequencies.

As future work one could explore a combinatorial optimization to let the 8-bit-quantized $\boldsymbol{c}$ values approximate an image with more than 8 bits of precision. Such super-precision may be possible only in low-frequency regions, but these smooth regions are precisely where precision is most perceptible.

9

Theoretical considerations

In this section, we delve deeper into theoretical considerations on the approximation order of a complete sampling pipeline. Recall that we obtain an approximation $\tilde{f}_T$ to a signal f as follows:

$$\tilde{f}_T = \Big[f * (\boldsymbol{q}^\vee * \psi)^\vee_T\Big]_T *_T \varphi_T = \Big[f * \psi^\vee_T\Big]_T *_T (\boldsymbol{q} * \varphi)_T. \tag{9.1}$$

Any given choice of generating function φ defines the shift-invariant space $V_{\varphi,T}$ of representable approximations:

$$V_{\varphi,T} = \{\tilde{f}_T : \mathbb{R} \ \to \ \mathbb{R} \mid \tilde{f}_T = \boldsymbol{c}_T *_T \varphi_T, \ \forall \boldsymbol{c}_T \in l_2\}. \tag{9.2}$$

The roles of the digital filter $\boldsymbol{q}$ and the analysis filter ψ are to select a particular approximation $\tilde{f}_T$ in this space, i.e., they select the coefficient vector $\boldsymbol{c}_T$.

We say that space $V_{\varphi,T}$ has approximation order L when we can find a constant $C > 0$ such that the following relation holds:

$$\|f - P_{\varphi,T} f\|_{\mathcal{L}_2} = C\, T^L \, \|f^{(L)}\|_{L_2} \quad \text{as} \quad T \to 0. \tag{9.3}$$

In other words, the norm of the residual between f and its orthogonal projection $P_{\varphi,T} f$ into $V_{\varphi,T}$ falls to zero with order L when we progressively reduce sample spacing T.

We will see the connection between the approximation order and the ingredients φ, ψ, and $\boldsymbol{q}$ of the sampling pipeline.

On admissible kernels In formal presentations, it is common to open with admissibility restrictions on the generating function φ. That is, we must ensure that $V_{\varphi,T}$ is a closed subspace of $\mathcal{L}_2$ before we can discuss approximation in this particular signal subspace. We require that the set of shifted generating functions $\varphi(\cdot/T-k), k \in \mathbb{Z}$ form a Riesz basis of $V_{\varphi,T}$ so that any signal in $V_{\varphi,T}$ is uniquely determined by $\boldsymbol{c}_T$. This condition is equivalent to the orthogonal projection being well-defined, which in turn is equivalent to the requirement that convolution with the sampled auto-correlation $\boldsymbol{[}a_\varphi\boldsymbol{]} = \boldsymbol{[}\varphi * \varphi^\vee\boldsymbol{]}$ of φ must be a bounded invertible operator from l_2 into itself. Interestingly, if $\boldsymbol{p}$ is an invertible convolution operator then $V_{\boldsymbol{p}*\varphi} = V_\varphi$ and therefore $\boldsymbol{p}*\varphi$ and φ are *equivalent generating functions*. The important observation is that these safeguards are satisfied by most generating functions used in practice [Aldroubi and Unser, 1994]. A key exception is that derivatives of admissible kernels are *not* themselves admissible.

On approximation order The Strang-Fix conditions [Strang and Fix, 1973] state that a space V_φ has approximation order L if and only if all polynomials of degree up to $L-1$ belong to V_φ. Any projection operator finds the unique representation of the polynomials in V_φ. In particular, the orthogonal projection of section 4.3. The Strang-Fix conditions are therefore equivalent to:

$$P_\varphi\,(\cdot)^n = \boldsymbol{[}(\cdot)^n * \mathring{\varphi}^\vee\boldsymbol{]} * \varphi = (\cdot)^n, \quad n \in \{0,\ldots,L-1\}. \tag{9.4}$$

More generally, the oblique projection of section 4.4 also finds the representation of polynomials of degree up to $L-1$ in V_φ if they exist, so long as the analysis filter ψ is such that $\boldsymbol{[}a_{\varphi,\psi}\boldsymbol{]}$ is invertible:

$$P_{\varphi\perp\psi}\,(\cdot)^n = \boldsymbol{[}(\cdot)^n * \psi^\vee\boldsymbol{]} * \boldsymbol{[}a_{\varphi,\psi}\boldsymbol{]}^{-1} * \varphi = (\cdot)^n, \quad n \in \{0,\ldots,L-1\}. \tag{9.5}$$

A simpler formulation comes from the interpolation strategy of section 4.1. It is the projection operator that results from setting $\psi = \delta$ above:

$$\boldsymbol{[}(\cdot)^n\boldsymbol{]} * \varphi_{\text{int}} = \boldsymbol{[}(\cdot)^n\boldsymbol{]} * \boldsymbol{[}\varphi\boldsymbol{]}^{-1} * \varphi = (\cdot)^n, \quad n \in \{0,\ldots,L-1\}. \tag{9.6}$$

The conditions on approximation order describe the asymptotic behavior of the orthogonal projection. Other choices of analysis filter and digital filter necessarily lead to poorer approximations with respect to the $\mathcal{L}_2$ norm, but

may attain the same approximation order L as the orthogonal projection. This is the case if the approximation scheme reproduces all polynomials of degree up to $L-1$ exactly [de Boor, 1989].

Any projection into a space V_φ of order L satisfies this property. These include interpolation, orthogonal projection, and oblique projection. Enforcing the projection property is equivalent to requiring the combination of analysis filter ψ and digital filter $\boldsymbol{q}$ to be biorthogonal to φ:

$$\left[\varphi * (\boldsymbol{q}^\vee * \psi^\vee)\right] = \boldsymbol{\delta}. \tag{9.7}$$

Although sufficient, this is not a necessary condition. A projection exactly reproduces *all* functions in V_φ. This includes many other functions besides the polynomials of degree up to $L-1$ that we care about. The necessary and sufficient condition is simply

$$\left[(\cdot)^n * (q^\vee * \psi)^\vee\right] = \left[(\cdot)^n * \mathring{\varphi}^\vee\right], \quad n \in \{0, \ldots, L-1\}. \tag{9.8}$$

This is in fact equivalent to requiring equality only at the sample placed at 0. It is enough for the effective analysis filter to have same moments as the dual up to order $L-1$ [Blu and Unser, 1999b]:

$$\left\langle (\cdot)^n,\ \boldsymbol{q}^\vee * \psi \right\rangle = \left\langle (\cdot)^n,\ \mathring{\varphi} \right\rangle, \quad n \in \{0, \ldots, L-1\}. \tag{9.9}$$

This milder condition is called *quasibiorthogonality of order* L, and the resulting schemes are called *quasiinterpolators of order* L.

Frequency domain characterizations For many of the properties we discussed in this section, the frequency domain characterizations bring valuable insight. For example, the admissibility criterion on V_φ is equivalent to the sampled auto-correlation $\boldsymbol{a}_\varphi = \left[a_\varphi\right]$ of φ being a bounded, invertible discrete convolution operator. In the frequency domain

$$\boldsymbol{b} * \boldsymbol{a}_\varphi \quad \xrightarrow{\mathcal{F}} \quad \hat{\boldsymbol{b}}\,\widehat{\boldsymbol{a}_\varphi} \qquad \text{and} \qquad \boldsymbol{c} * (\boldsymbol{a}_\varphi)^{-1} \quad \xrightarrow{\mathcal{F}} \quad \frac{\hat{\boldsymbol{c}}}{\widehat{\boldsymbol{a}_\varphi}}. \tag{9.10}$$

Boundedness and invertibility of the operator are equivalent to the conditions

$$A \leq \widehat{\boldsymbol{a}_\varphi} \leq B, \quad \text{for} \quad 0 < A \leq B < \infty, \tag{9.11}$$

which are easier to verify in practice.

In more theoretical texts, we often find the sampled auto-correlation function defined directly in the frequency domain. Recall

$$\boldsymbol{a}_{\boldsymbol{\varphi}} = \big[\!\big[a_{\varphi}\big]\!\big] = \big[\!\big[\varphi * \varphi^{\vee}\big]\!\big] = \text{Ш} \cdot (\varphi * \varphi^{\vee}). \tag{9.12}$$

Taking the Fourier transform,

$$\widehat{\text{Ш}} * \big(\hat{\varphi} \cdot \widehat{(\varphi^{\vee})}\big) = \text{Ш} * \big(\hat{\varphi} \cdot (\hat{\varphi})^{*}\big) = \text{Ш} * |\hat{\varphi}|^2 = \sum_{k \in \mathbb{Z}} \big|\hat{\varphi}(\cdot + k)\big|^2, \tag{9.13}$$

so that

$$\widehat{\boldsymbol{a}_{\boldsymbol{\varphi}}} = \sum_{k \in \mathbb{Z}} \big|\hat{\varphi}(\cdot + k)\big|^2. \tag{9.14}$$

An equivalent formulation for (9.6) is given in [Unser et al., 1995b]:

$$\text{Ш} * \big((\cdot)^n \varphi_{\text{int}}(\cdot)\big) = \begin{cases} 1 & n = 0, \quad \text{(partition of unity)} \\ 0 & n \in \{1, \dots, L-1\}. \end{cases} \tag{9.15}$$

Now if we recall the Fourier transform rule for derivatives

$$(\cdot)^n f \quad \xrightarrow{\mathcal{F}} \quad (\tfrac{i}{2\pi})^n (\hat{f}\,)^{(n)}, \tag{9.16}$$

we reach the frequency-domain formulation for (9.6):

$$\big[\!\big[\widehat{(\varphi_{\text{int}})}^{(n)}\big]\!\big] = \begin{cases} \boldsymbol{\delta} & n = 0, \\ \mathbf{0} & n \in \{1, \dots, L-1\}, \end{cases} \tag{9.17}$$

The form in (9.17) indicates that, as the approximation order increases, the frequency response of the cardinal φ_{int} gets closer to the ideal low-pass filter.

The quasibiorthogonality conditions in (9.9) are also difficult to verify as stated. Recall that both the effective analysis filter and the dual are likely to have infinite support, so the resulting integrals are improper and involve infinite summations. Fortunately, the expressions in frequency domain are much simpler. If we recall that:

$$\hat{f}(0) = \int_{-\infty}^{\infty} f(t)\,\mathrm{d}t, \tag{9.18}$$

we see that the conditions in (9.9) are equivalent to

$$\big(\widehat{\boldsymbol{q}^{\vee} * \psi}\big)^{(n)}(0) = \big(\widehat{\hat{\varphi}}\big)^{(n)}(0), \quad n \in \{0, \dots, L-1\}. \tag{9.19}$$

In other words, the Maclaurin series of the Fourier transforms of the effective analysis filter and the dual must be the same up to order $L-1$. The formulation in the frequency domain is again easier to verify or enforce.

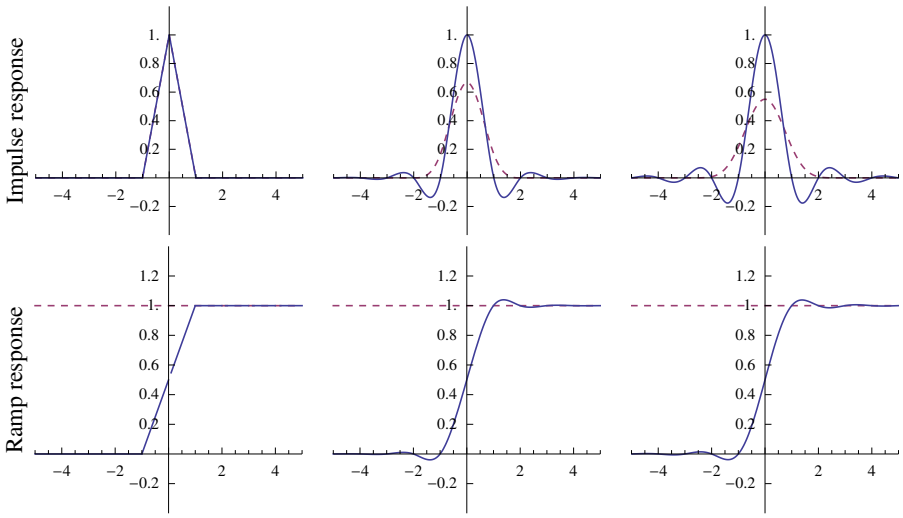
10

Experiments and analyses

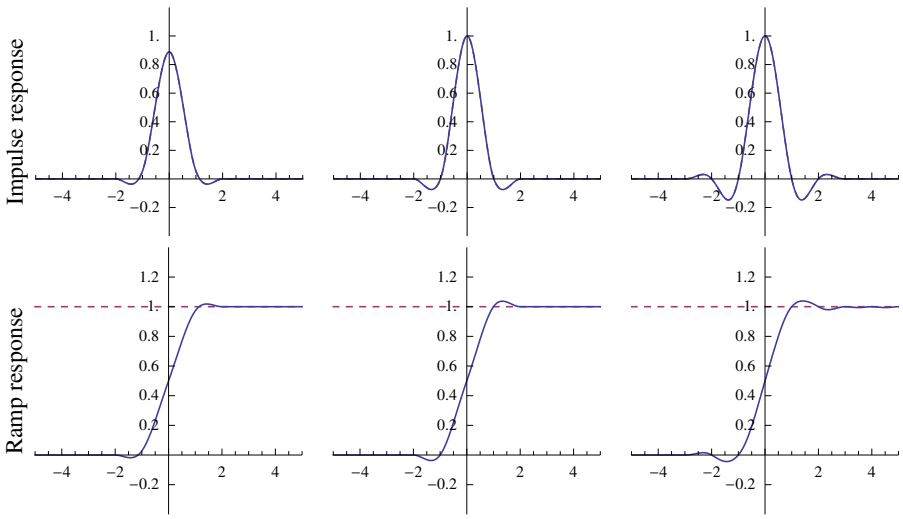
We performed experiments on about 50 kernels described in the literature. Here we present a subset of these results. More details are available in the supplemental material.

Transient responses Figure 10.1 shows the impulse and ramp responses for a variety of reconstruction kernels. The impulse responses for the B-spline kernels show the generator φ as a dashed line, and the cardinal $\varphi * \mathbf{[}\varphi\mathbf{]}^{-1}$ as a solid line. The presence of negative lobes in an impulse response is often cited as a concern for possible ringing artifacts in reconstruction. However, images with good signal-to-noise ratio and proper antialiasing rarely contain impulses or even sharp step-discontinuities. Instead, we think that the ramp response is more representative of content present in natural images, and it predicts significantly reduced ringing for all kernels.

Continuous frequency response Another popular analysis tool for prefiltering and reconstruction kernels is the frequency response. In traditional sampling, this is simply the magnitude of the Fourier transform of ψ or φ. In generalized sampling, we must account for the effect of the digital filter $\boldsymbol{q}$ by multiplying-in its DTFT. Figure 10.2 shows the resulting responses for typical

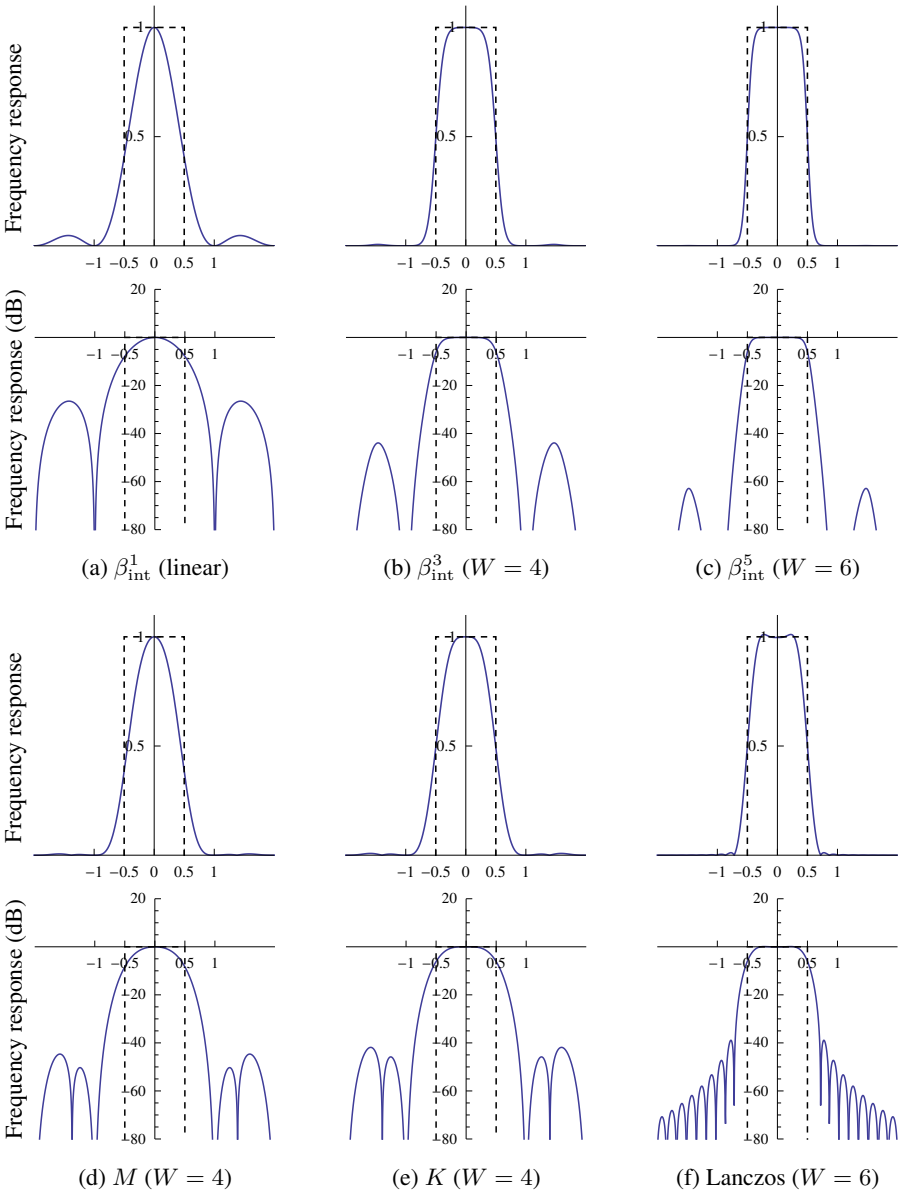


(a) β^1_{int} (linear) (b) β^3_{int} ($W = 4$) (c) β^5_{int} ($W = 6$)



(d) M ($W = 4$) (e) K ($W = 4$) (f) Lanczos ($W = 6$)

Figure 10.1: Transient responses. Any reconstruction kernel containing negative lobes results in a certain amount of ringing (b–f). Mitchell and Netravali [1988] designed their kernel to minimize such problems (d), as seen in the comparison with the impulse responses of other kernels, in particular B-spline interpolation (b,c) and Lanczos (f). The ramp response, which is more characteristic of properly antialiased images, predicts significantly reduced ringing across all kernels.



(a) β^1_{int} (linear) (b) β^3_{int} ($W = 4$) (c) β^5_{int} ($W = 6$)

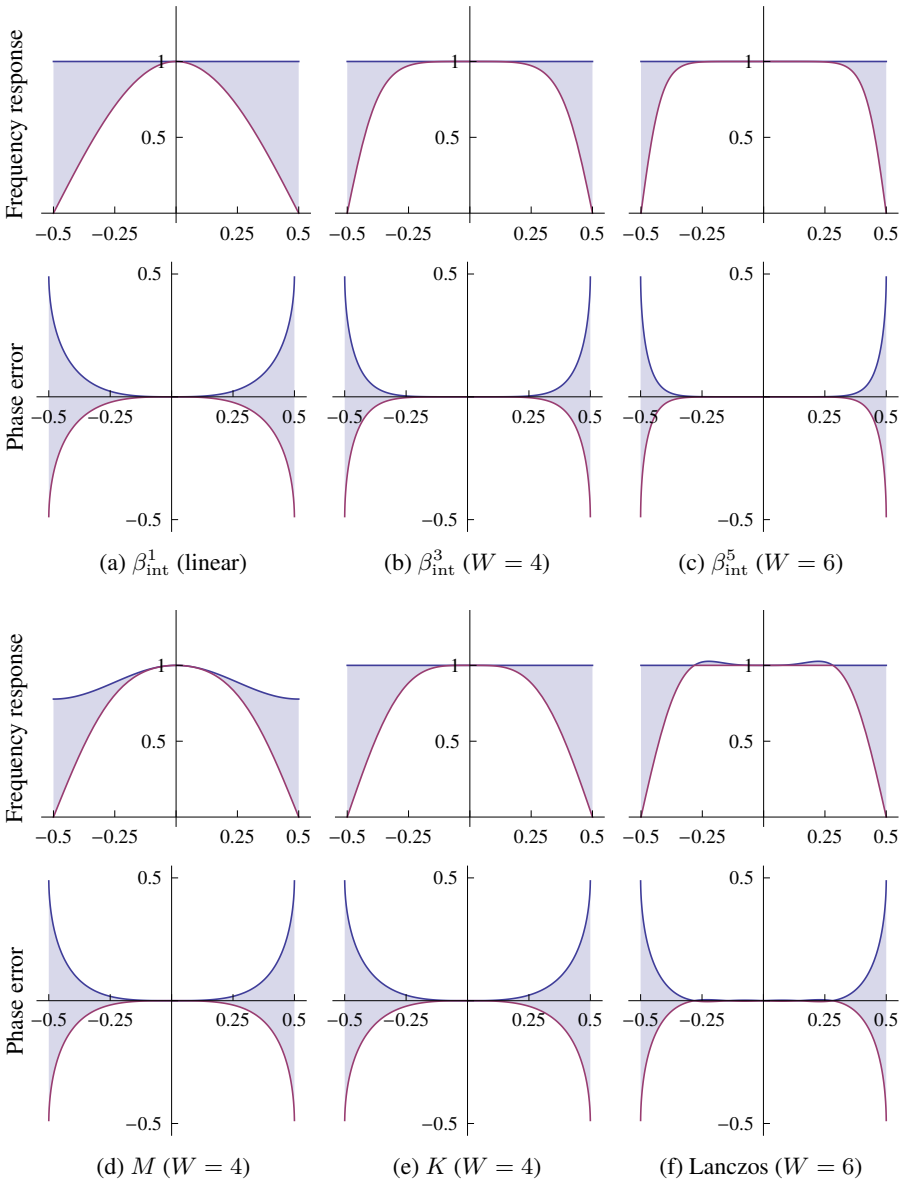
(d) M ($W = 4$) (e) K ($W = 4$) (f) Lanczos ($W = 6$)

Figure 10.2: Frequency responses. (a–c) The frequency response of B-spline interpolation converges towards that of the ideal low-pass filter as the degree increases. (b) $(\beta^3)_{\text{int}}$ outperforms the popular cubic kernels (d) Mitchell and Netravali [1988] and (e) Keys [1981], and its low-pass behavior is at least as good as the more expensive Lanczos kernel (f). Using the same support, $(\beta^5)_{\text{int}}$ in (c) performs significantly better.

kernels used in computer graphics. For each filter, the top graph uses a linear vertical scale and the bottom graph uses a log vertical scale (in decibels) to more clearly reveal the attenuation of unwanted high frequencies. The "ideal" frequency response corresponds to the dashed line depicted in each plot. The progression towards ideal low-pass behavior for increasing approximation order, as indicated by equation (9.17), is clearly confirmed in the plots.

Frequency response in resampling Figure 10.3 shows the effects of a resampling strategy on the amplitude of each frequency in the discrete input signal, as well as the phase errors introduced. The work of Parker et al. [1983] served as an inspiration for this visualization. Recall from section 5.1 that naïve translation by an offset h after reconstruction with $\boldsymbol{r} * \varphi$ is equivalent to discrete convolution with $\boldsymbol{r} * \big[\tau_h * \varphi\big]$. Shaded areas represent the span of behaviors of the discrete kernels associated to $h \in [-0.5, 0.5]$. Thus, the frequency response can vary significantly as we vary the alignment between the input and output sample grids. It is clear that generalized kernels result in better amplitude preservation and lower phase errors. The lower phase errors may be useful when extracting coordinates of matching features from resampled images (e.g., in computer vision applications involving rectification). More research is needed.

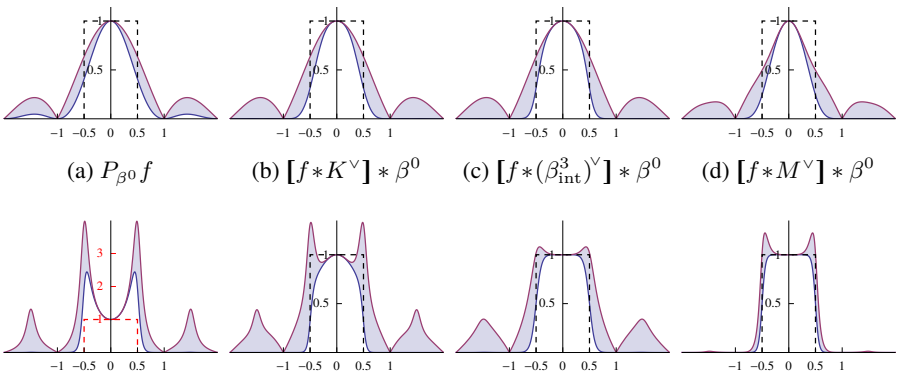
Response of complete pipeline The reconstruction response plots in figures 10.2, though informative, show the response of only part of the pipeline. In contrast, figure 10.4 shows a visualization of the effect of the *entire* pipeline, for a variety of prefiltering and reconstruction strategies, including the effect of the sampling operation between them. In each plot, the blue curve shows what is left of the original f in the reconstructed $\tilde{f}$. The purple curve includes the effect of all aliases generated by the sampling procedure. The shaded area between the curves therefore represents the amount of aliasing in the reconstructed signal. We focus on reconstruction with the box kernel, which is a reasonable approximation of typical LCD displays. One takeaway message is that a sharper prefilter preserves more of the original signal. Nonetheless, a softer prefilter may be preferable for reducing post-aliasing when a poor reconstruction filter shows excessive artifacts. Another interesting observation is that orthogonal projection with B-splines is very effective, except of course



(a) β^1_{int} (linear) (b) β^3_{int} ($W = 4$) (c) β^5_{int} ($W = 6$)

(d) M ($W = 4$) (e) K ($W = 4$) (f) Lanczos ($W = 6$)

Figure 10.3: Frequency response and phase errors in resampling. The shaded regions indicate the range of amplitude attenuation (or magnification) and phase errors introduced when resampling a signal under all possible translation offsets. These results corroborate those of figure 10.2. The cubic B-spline interpolation (b) performs better than other cubic kernels (d,e), and the quintic B-spline (c) performs significantly better than Lanczos (f).



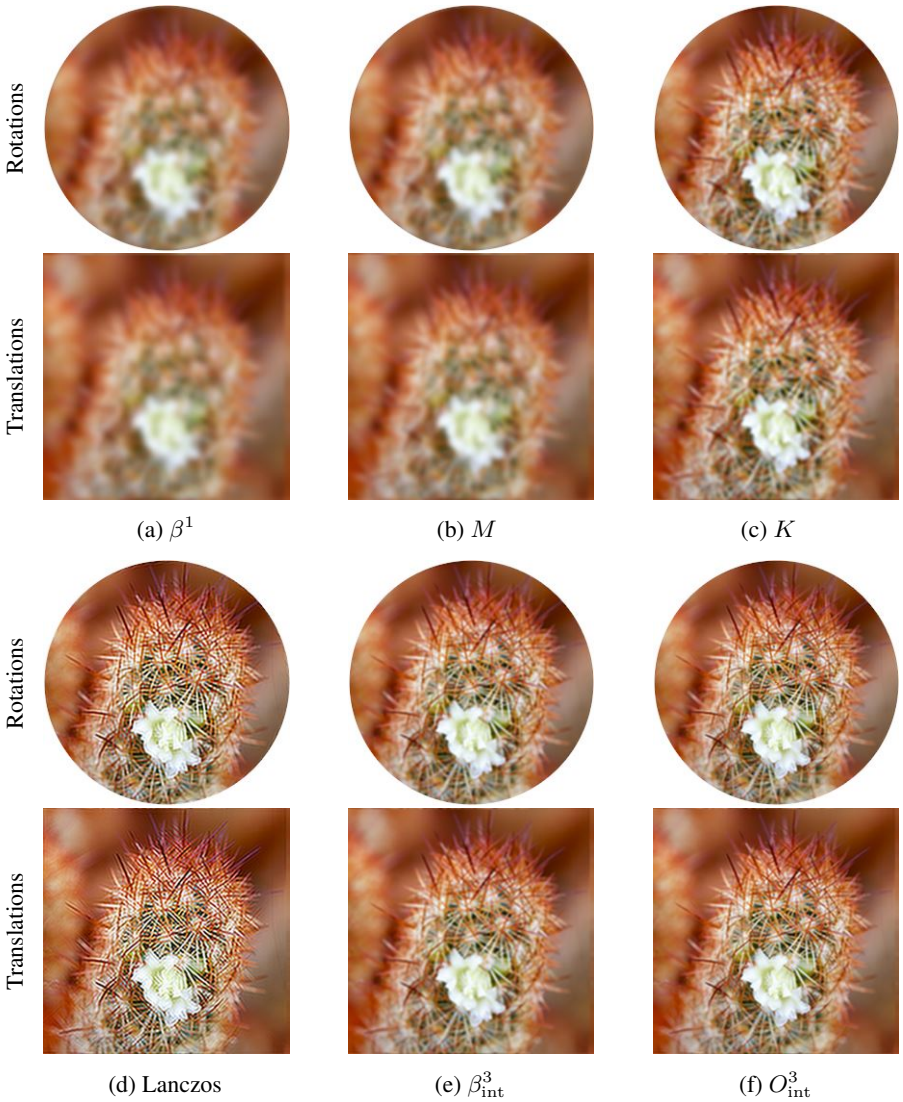
(a) $P_{\beta^0} f$ (b) $\big[f * K^\vee\big] * \beta^0$ (c) $\big[f * (\beta^3_{\text{int}})^\vee\big] * \beta^0$ (d) $\big[f * M^\vee\big] * \beta^0$

(e) $\big[f * (\mathring{\beta}^3)^\vee\big] * \beta^0$ (f) $\big[P_{\beta^3} f\big] * \beta^0$ (g) $P_{\beta^0 \perp \beta^3} f$ (h) $P_{\beta^3} f$

Figure 10.4: Frequency response for the complete pipeline. In each plot, the blue curve marks the combined effect of the prefilter and the reconstruction filter on a broad-spectrum function f when *no* sampling is involved. The purple curve marks the effect of the entire pipeline, including sampling operations. The shaded region between them therefore represents aliasing. (a–g) reconstruct with the box filter, which is a good approximation for current LCD displays. (a) is orthogonal projection in space V_{β^0}. In (a–c), more content is preserved as prefilters become sharper, from the box to the cardinal cubic B-spline. (d) shows how Mitchell and Netravali [1988] softens results but *also* attenuates aliasing relative to (a–c). (e) shows the effect of directly sending the coefficients of orthogonal projection in V_{β^3} for reconstruction with box, (f) shows the less objectionable idea of sampling the projection first, and (g) shows the more principled oblique projection from the cubic B-spline space to the box space. Finally, (h) shows the excellent performance of orthogonal projection in the cubic B-spline space. Note, however, the aliasing concentrated around the Nyquist rate.

when reconstructing with a different basis, in which case oblique projection is preferred. These experiments provide a glimpse of what is possible when orthogonal projection is realized in a space with good approximation properties. Example renderings for each of these pipelines are shown in the supplemental material.

Effect of repeated resampling The importance of good approximation properties in a sampling scheme is typically demonstrated using the cumulative effect of repeated resampling operations. Figure 10.5 shows the result of repeatedly translating or rotating an image 60 times. This challenging test may seem a somewhat artificial experiment. However, lower degradation rates are useful in real-time applications that amortize computations across subsequent frames (e.g., using reprojection). Furthermore, the test helps reveal filtering



(a) β^1 (b) M (c) K

(d) Lanczos (e) β^3_{int} (f) O^3_{int}

Figure 10.5: Repeated resampling. A sharp image was rotated 60 times by 6° (top) and translated 60 times around a 5-pixel radius circle (bottom). We see that the amount of detail preserved is related to the approximation order. (a,b) The second-order hat and Mitchell-Netravali kernels perform worst, followed by the third-order Catmull-Rom (c). (d) Lanczos reconstruction ($L = 1$ with normalization, $W = 6$) causes certain frequencies to explode (see figures 10.2f and 10.3f). (e) The optimal-order cardinal cubic B-spline ($L = 4$) performs significantly better, but is surpassed by the remarkable cardinal cubic O-MOMS (f), which has the same approximation order but a smaller approximation constant.

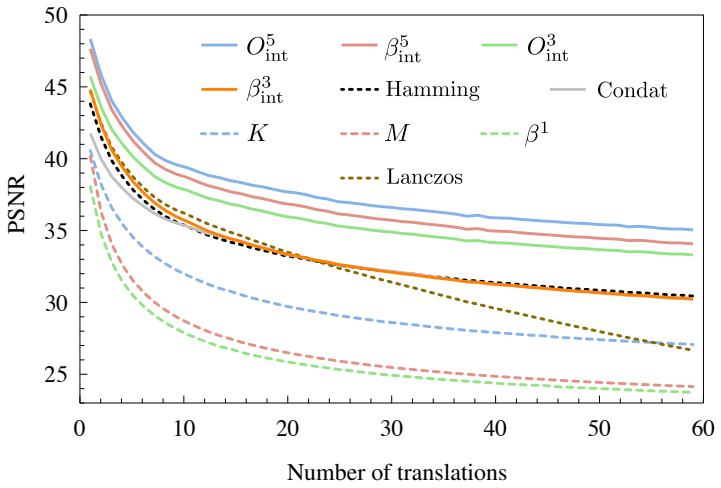


Figure 10.6: Effect of repeated translations on PSNR. The top performing methods *all* include a digital filter $\boldsymbol{q}$. We note the excellent behavior of interpolation with the cubic O-MOMS kernel ($W = 4$), and of approximation by the *quadratic* scheme of Condat et al. [2005] ($W = 3$). The Lanczos window is outperformed by the Hamming window ($W = 6$), and both are outperformed by the more efficient, high-approximation-order generalized kernels. Traditional linear interpolation, Mitchell and Netravali [1988], and Keys [1981] lag far behind.

weaknesses that may not be visible in a single processing step but become apparent as part of a larger pipeline. Also, any blurring or contrast reduction is greatly accentuated and thus easier to visualize. It is remarkable how well the generalized methods perform relative to traditional approaches. The results also show the advantage of dropping the regularity constraint in favor of a lower approximation coefficient (see O-MOMS vs. B-spline).

Quantitative experiments In additional to providing results for visual inspection, we also present quantitative comparisons using both the traditional $\mathcal{L}_2$ metric and the mean structural similarity (MSSIM) metric of Wang et al. [2004]. MSSIM is closer to the perceptual characteristics of the human visual system. Figure 10.6 shows a plot of the accumulated degradation, in terms of PSNR, as operations are repeated with a range of resampling strategies. Larger values of PSNR are ideal, and of course some loss in signal is inevitable due to the resampling operations. The MSSIM results are also summarized in table 10.1. Again, the generalized resampling strategies stand out for their improved reconstruction quality (for a given support size).

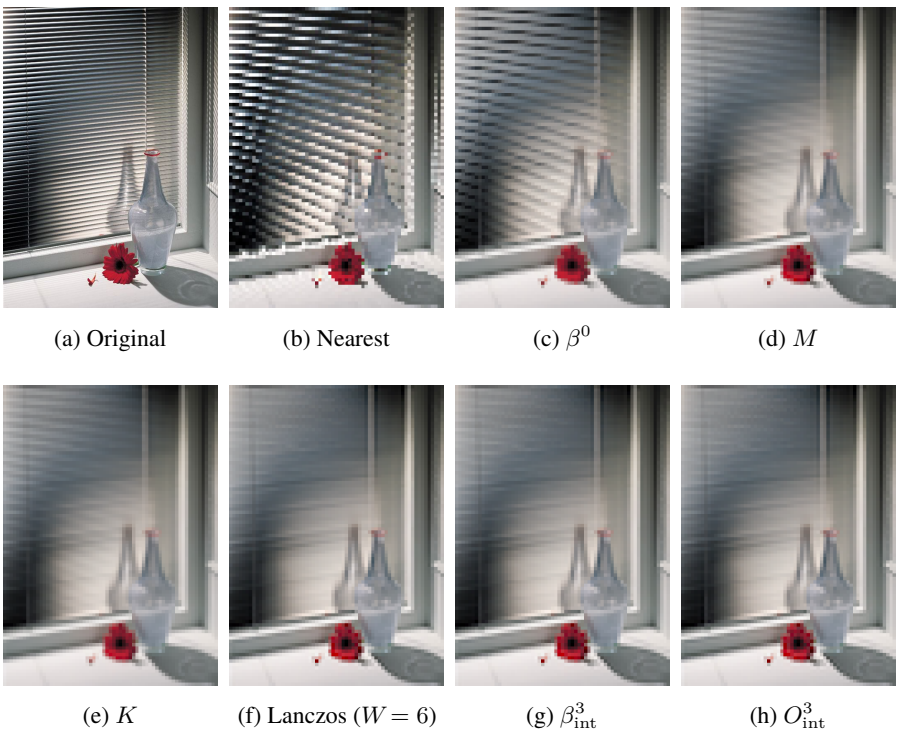
Table 10.1: Quantitative analysis of reconstruction quality. Kernel properties are degree N, width W, and approximation order L. The main columns report the mean structural similarity (MSSIM) between reference images (concentric circles as in figure 8.2, and average of four Kodak benchmarks) and their reconstructions for three types of experiments (repeated translations or rotations, and single upsampling). The kernels are sorted in descending order of average quality across all experiments. The interpolating B-splines `bspline*i` consistently outperform the more traditional filters for the same N and W. The O-MOMS kernels `omoms*` offer even slightly higher quality but at the expense of differentiability. The quasi-interpolant `condat2`, which has degree 2 and support 3, also performs remarkably well.

				Comparisons against ground truth (MSSIM)					
	Properties			Translations		Rotations		Upscaling	Average
Kernel	N	W	L	CIR	Kodak	CIR	Kodak	CIR	AVG
`omoms5`	5	6	6	0.993	0.965	0.999	0.983	0.886	0.979
`bspline5i`	5	6	6	0.990	0.956	0.998	0.980	0.886	0.973
`omoms3`	3	4	4	0.981	0.940	0.997	0.977	0.886	0.964
`quasiblu35`	3	4	4	0.965	0.919	0.994	0.973	0.885	0.946
`condat3`	3	4	4	0.962	0.915	0.991	0.969	0.885	0.943
`hamming6`	–	6	1	0.960	0.912	0.977	0.956	0.885	0.941
`bspline3i`	3	4	4	0.948	0.900	0.977	0.957	0.885	0.935
`condat2`	2	3	3	0.930	0.884	0.989	0.966	0.884	0.927
`lanczos6`	–	6	1	0.964	0.828	0.987	0.960	0.884	0.910
`bspline2i`	2	3	3	0.901	0.860	0.955	0.942	0.884	0.906
`omoms2`	2	3	3	0.814	0.802	0.961	0.945	0.885	0.876
`schaum2`*	2	3	3	0.822	0.804	0.921	0.921	0.877	0.864
`lanczos4`	–	4	1	0.822	0.804	0.896	0.911	0.882	0.857
`keys`†	3	4	3	0.822	0.804	0.894	0.909	0.882	0.857
`schaum3`‡	3	4	4	0.822	0.804	0.876	0.902	0.883	0.852
`dalai1`	1	2	2	0.657	0.716	0.956	0.941	0.865	0.828
`linrev`	1	2	2	0.686	0.716	0.960	0.928	0.864	0.822
`condat1`	1	2	2	0.651	0.713	0.947	0.936	0.866	0.824
`hamming4`	–	4	1	0.663	0.718	0.822	0.878	0.879	0.787
`mitchell`	3	4	2	0.581	0.680	0.625	0.806	0.881	0.715
`linear`	1	2	2	0.391	0.600	0.540	0.779	0.864	0.644
`nearest`	0	1	1	0.042	0.279	0.547	0.717	0.586	0.457

*Same as IMOMS-2. †Same as Catmull-Rom. ‡Same as IMOMS-3. The kernels `omoms*` are from [Blu et al., 2001]; `quasiblu35` is from [Blu and Unser, 1999a]; `bspline*i` are the cardinal B-splines β_{int}; `schaum*` are from [Schaum, 1993]; `linrev` is from [Blu et al., 2004]; `condat*` are from [Condat et al., 2005]; `dalai1` is from [Dalai et al., 2006].

Discrete downscaling Figure 10.7 shows a challenging example for discrete image downscaling using equation (5.28) (i.e., with no reconstruction). At the selected output resolution, the image is extremely prone to aliasing, which is clearly visible even with the soft Mitchell-Netravali kernel. In contrast, the cardinal cubic B-spline and O-MOMS kernels virtually eliminate aliasing, while maintaining sharpness indistinguishable from the wider Lanczos kernel.

Animation tests In animation sequences, the effect of the resampling frequency response of section 10 is often visible. Phase errors cause high frequencies to oscillate in position, whereas amplitude errors cause high frequencies to oscillate in brightness. To demonstrate these effects, we use simple animation sequences in which each frame is the result of translating an input image around a circle with a 5 pixel radius, according to the naïve scheme of section 5.1. These results are also available in the supplemental material.



(a) Original (b) Nearest (c) β^0 (d) M

(e) K (f) Lanczos ($W = 6$) (g) β^3_{int} (h) O^3_{int}

Figure 10.7: Image downscaling using generalized sampling as in equation (5.28), using impulse reconstruction (i.e., $\boldsymbol{r} * \varphi = \delta$), for various prefilters $\boldsymbol{p} * \psi$. The image shown in (a) was downscaled from its original (500×700) to an alias-prone resolution (65×91). It was then upscaled back to the original size using nearest neighbors to minimize any reproduction artifacts. (b,c) Note that nearest and box filtering result in a substantial amount of aliasing. (d,e) Mitchell-Netravali and Catmull-Rom (Keys) alleviate the problem, but aliasing is still clearly visible. (f) A wider support allows the Lanczos windowed-sinc to virtually eliminate aliasing while preserving image sharpness. (g,h) However, similar results are obtained with the cardinal cubic B-spline and O-MOMS filters, which have smaller support but include a fast digital filtering stage.

11

Conclusions

The framework of generalized sampling offers practical improvements in several areas of graphics. Our analysis and experiments confirm that it enables significantly higher quality in upsampling and resampling operations. We have adapted its concepts to derive a variety of supersampling techniques, and shown that these offer comparable quality to the best conventional techniques, but at a reduced computational cost. Similarly, we have explored a new family of image downscaling techniques, some of which outperform traditional algorithms in terms of blurring, ringing, and aliasing. Although generalized sampling does require solving discrete inverse filters, this computation is efficient and scales well to multicore and GPU architectures. Pedagogically, one of our contributions is a new parameterless notation that moves easily between discrete and continuous functions. By avoiding tedious index manipulations, this algebraic notations lets us to reason more intuitively about otherwise complex operations. We hope that these contributions may aid the dissemination of generalized sampling methods and attract broader interest from graphics researchers.

Acknowledgements

We would like to thank Leonardo Sacht for carefully reading the manuscript multiple times and providing insightful comments and corrections.

Appendices

A

Source-code

The C++11 sample code in this appendix demonstrates the following set of functionalities:

Fast inverse discrete convolution in lines 12–32.
Safe indexing with reflection at boundaries in lines 34–38.
Box kernel in lines 66–77.
Hat kernel in lines 79–92.
Mitchell-Netravali kernel in lines 113–133.
Catmull-Rom kernel in lines 136–147.
Cardinal Cubic B-spline, a generalized kernel, in lines 157–168.
Cardinal Cubic O-MOMS, a generalized kernel, in lines 186–196.
Simple, intuitive upsampling in lines 198–216.
Simple, intuitive downsampling in lines 218–247.
Fast, incremental upsampling in lines 256–274.
Fast, incremental downsampling in lines 276–303.

```cpp
#include <iostream>    // std::ostream, std::cout
#include <cmath>       // std::abs(), std::ceil(), std::floor()
#include <algorithm>   // std::min()
#include <vector>      // std::vector<>
#include <array>       // std::array<>
#include <string>      // std::string
#include <cassert>     // assert()
#include <chrono>      // std::chrono for timing benchmarks
using namespace std;

// Apply to sequence f the inverse discrete convolution given by
// a pre-factored LU decomposition
template<size_t M>
void linear_solve(const array<float,M>& L, vector<float>& f) {
  const int m = M, n = int(f.size());
  const float p_inv = 1.f; // Optimized for prescaled kernel where p = 1
  // Pre-factored decomposition only works when n>m. Grow sequence f if needed.
  while (f.size() <= m) {
    f.reserve(2*f.size()); // Prevent reallocation during insertions
    f.insert(f.end(), f.rbegin(), f.rend()); // Append a reflection
  }
  int nn = int(f.size()); // New size
  const float L_inf = L[m-1], v_inv = L_inf/(1.f+L_inf);
  // Forward pass: solve Lc' = f in-place
  for (int i=1; i<m;  i++)      f[i] -= L[i-1]*f[i-1];
  for (int i=m; i<nn; i++)      f[i] -= L_inf *f[i-1];
  // Reverse pass: solve Uc = c' in-place
  f[nn-1] *= p_inv*v_inv;
  for (int i=nn-2; i>=m-1; i--) f[i] = L_inf*(p_inv*f[i]-f[i+1]);
  for (int i=m-2;  i>=0;   i--) f[i] = L[i] *(p_inv*f[i]-f[i+1]);
  f.resize(n); // Truncate back to original size if grown
}

// Given sequence f, access f[i] using reflection at boundaries
static inline float get(const vector<float>& f, int i) {
  return i<0? get(f,-i-1): i>=int(f.size())?
    get(f,2*int(f.size())-i-1): f[i];
}

// Kernel interface
template<size_t N>
class KernelBase {
  public:
    KernelBase() { b.fill(0.f); }
    // Evaluate kernel at coordinate x
    virtual float operator()(float x) const = 0;
```

```
47      // Apply the kernel's associated digital filter to sequence f
48      virtual void digital_filter(vector<float>& f) const = 0;
49      // Kernel support (-support/2, support/2]
50      int support() const { return N; }
51      // Shift the incremental buffer
52      float shift_buffer(float a) {
53        rotate(b.begin(), b.begin()+1, b.end());
54        swap(b.back(), a);
55        return a;
56      }
57      // Incrementally accumulate a sample into buffer
58      virtual void accumulate_buffer(float fu, float u) = 0;
59      // Incrementally reconstruct the function from samples in buffer
60      virtual float sample_buffer(float u) const = 0;
61      virtual string name() const = 0;
62      // How much does the kernel integrate to?
63      virtual float integral() const { return 1.f; }
64    protected:
65      array<float,N> b; // Incremental buffer
66  };
67
68  // Simple box kernel
69  struct Box final: KernelBase<1> {
70      float operator()(float x) const override {
71        return x<=-0.5f || x>0.5f ? 0.f : 1.f;
72      }
73      void accumulate_buffer(float fu, float) override {
74        b[0] += fu;
75      }
76      float sample_buffer(float) const override { return b[0]; }
77      void digital_filter(vector<float>&) const override { }
78      string name() const override { return "Box"; }
79  };
80
81  // Simple hat kernel
82  struct Hat final: KernelBase<2> {
83      float operator()(float x) const override {
84        x = abs(x); return x>1.f ? 0.f : 1.f-x;
85      }
86      void accumulate_buffer(float fu, float u) override {
87        b[0] += fu*(1.f-u); b[1] += fu*u;
88      }
89      float sample_buffer(float u) const override {
90        return b[0]*(1.f-u)+b[1]*u;
91      }
92      void digital_filter(vector<float>&) const override { }
```

```
93      string name() const override { return "Hat"; }
94  };
95
96  // Most cubics are C^1-continuous, symmetric, and have support 4.
97  // Factor out common functionality into a class.
98  template<typename Pieces>
99  class Symmetric4Pieces: public KernelBase<4> {
100   public:
101     float operator()(float x) const override final {
102       x=abs(x); return x>2.f ? 0.f : x>1.f ? p.k0(2.f-x) : p.k1(1.f-x);
103     }
104     void accumulate_buffer(float fu, float u) override final {
105       b[0] += fu*p.k3(u); b[1] += fu*p.k2(u);
106       b[2] += fu*p.k1(u); b[3] += fu*p.k0(u);
107     }
108     float sample_buffer(float u) const override final {
109       return b[0]*p.k3(u)+b[1]*p.k2(u)+b[2]*p.k1(u)+b[3]*p.k0(u);
110     }
111   private:
112     Pieces p; // Polynomial pieces of kernel (k0:[-2,-1], k1:[-1,0], k2:[0,1], k3:[1,2]
113 };
114
115 // Traditional Mitchell-Netravali kernel
116 struct MitchellNetravaliPieces {
117   static float k0(float u) {
118     return (((7/18.f)*u-1/3.f)*u)*u;
119   }
120   static float k1(float u) {
121     return (((-7/6.f)*u+1.5f)*u+.5f)*u+1/18.f;
122   }
123   static float k2(float u) {
124     return (((7/6.f)*u-2.f)*u)*u+8/9.f;
125   }
126   static float k3(float u) {
127     return (((-7/18.f)*u+5/6.f)*u-.5f)*u+1/18.f;
128   }
129 };
130
131 struct MitchellNetravali final:
132   Symmetric4Pieces<MitchellNetravaliPieces> {
133     void digital_filter(vector<float>&) const override { }
134     string name() const override { return "Mitchell-Netravali"; }
135 };
136
137
138 // Traditional Catmull-Rom kernel
```

```
139 struct CatmullRomPieces {
140   static float k0(float u) { return ((.5f*u-.5f)*u)*u; }
141   static float k1(float u) { return ((-1.5f*u+2.f)*u+.5f)*u; }
142   static float k2(float u) { return ((1.5f*u-2.5f)*u)*u+1.f; }
143   static float k3(float u) { return ((-.5*u+1.f)*u-.5f)*u; }
144 };
145
146 struct CatmullRom final: Symmetric4Pieces<CatmullRomPieces> {
147     void digital_filter(vector<float>&) const override { }
148     string name() const override { return "Catmull-Rom"; }
149 };
150
151 // Cubic B-spline kernel pieces (multiplied by 6)
152 struct Bspline3Pieces {
153   static float k0(float u) { return ((u)*u)*u; }
154   static float k1(float u) { return ((-3.f*u+3.f)*u+3.f)*u+1.f; }
155   static float k2(float u) { return ((3.f*u-6.f)*u)*u+4.f; }
156   static float k3(float u) { return ((-u+3.f)*u-3.f)*u+1.f; }
157 };
158
159 // Generalized Cardinal Cubic B-spline kernel
160 struct CardinalBspline3 final: Symmetric4Pieces<Bspline3Pieces> {
161     void digital_filter(vector<float>& f) const override {
162       // Pre-factored L U decomposition of digital filter
163       const array<float,8> L{.2f, .26315789f, .26760563f,
164         .26792453f, .26794742f, .26794907f, .26794918f, .26794919f};
165       linear_solve(L, f);
166     }
167     string name() const override {
168       return "Cardinal Cubic B-spline";
169     }
170     float integral() const override { return 6.f; }
171 };
172
173 // Cubic OMOMS kernel pieces (multiplied by 5.25)
174 struct OMOMS3Pieces {
175   static float k0(float u) {
176     return ((.875f*u)*u+.125f)*u;
177   }
178   static float k1(float u) {
179     return ((-2.625f*u+2.625f)*u+2.25f)*u+1.f;
180   }
181   static float k2(float u) {
182     return ((2.625f*u-5.25f)*u+.375f)*u+3.25f;
183   }
184   static float k3(float u) {
```

```
185     return ((-.875f*u+2.625f)*u-2.75f)*u+1.f;
186   }
187 };
188
189 // Generalized Cardinal Cubic O-MOMS3 kernel
190 struct CardinalOMOMS3 final: Symmetric4Pieces<OMOMS3Pieces> {
191     void digital_filter(vector<float>& f) const override {
192       // Pre-factored L U decomposition of digital filter
193       const array<float,9> L{.23529412f, .33170732f, .34266611f,
194         .34395774f, .34411062f, .34412872f, .34413087f, .34413112f,
195         .34413115f};
196       linear_solve(L, f);
197     }
198     string name() const override { return "Cardinal Cubic OMOMS"; }
199     float integral() const override { return 5.25f; }
200 };
201
202 // Simple, intuitive implementation of upsampling
203 template<typename Kernel> static vector<float>
204 upsample(vector<float> f, int m, Kernel& k) {
205   assert(m >= int(f.size())); // Ensure we are upsampling
206   vector<float> g(m); // New sequence of desired size m>f.size()
207   k.digital_filter(f); // Apply kernel's associated digital filter
208   const float kr = .5f*float(k.support());
209   for (int j=0; j<m; j++) { // Index of sample in g
210     float x = (j+.5f)/m; // Position in domain [0,1] of both f and g
211     float xi = x*f.size()-.5f; // Position in input sequence f
212     int il = int(ceil(xi-kr)); // Leftmost sample under kernel support
213     int ir = int(floor(xi+kr)); // Rightmost sample under kernel support
214     double sum = 0.;
215     for (int i=il; i<=ir; i++)
216       sum += get(f,i)*k(xi-i);
217     g[j] = float(sum);
218   }
219   return g;
220 }
221
222 // Simple, intuitive implementation of downsampling
223 template<typename Kernel> static vector<float>
224 downsample(const vector<float>& f, int m, Kernel& k) {
225   assert(m <= int(f.size()));  // Ensure we are downsampling
226   float s = float(m)/f.size(); // Scale factor
227   const int n = int(f.size());
228   const float kr = .5f*float(k.support());
229   const bool should_normalize = (f.size()%m != 0);
230   vector<float> g(m); // New sequence of desired size m<f.size()
```

```
231   for (int j=0; j<m; j++) { // Index of sample in g
232     float x = (j+.5f)/m; // Position in domain [0,1] of both f and g
233     int il = int(ceil ((x-kr/m)*n-.5f)); // Leftmost sample under kernel
234     int ir = int(floor((x+kr/m)*n-.5f)); // Rightmost sample under kernel
235     if (should_normalize) { // Should normalize?
236       double sum = 0., sumw = 0.;  // Sums of values and weights
237       for (int i=il; i<=ir; i++) { // Loop over input samples
238         float w = k((x-(i+.5f)/n)*m); // Weight for sample
239         sum += w*get(f,i); sumw += w; // Accumulate values and weights
240       }
241       g[j] = k.integral()*float(sum/sumw); // Normalize by summed weights
242     } else {
243       for (int i=il; i<=ir; i++) {
244         g[j] += k((x-(i+.5f)/n)*m)*get(f,i);
245       }
246       g[j] *= s;
247     }
248   }
249   k.digital_filter(g); // Apply kernel's associated digital filter
250   return g;
251 }
252
253 // Advance to next sample
254 inline bool advance(int& i, int& j, double& u, double inv_s) {
255   ++i; u += inv_s;
256   if (u < 1.) return false;
257   u -= 1.; ++j; return true;
258 }
259
260 // Faster, incremental implementation of upsampling
261 template<typename Kernel> vector<float>
262 upsample2(vector<float> f, int m, Kernel& k) {
263   k.digital_filter(f);
264   double inv_s = double(f.size())/m; // Inverse scale factor
265   // Output sample position between input samples
266   double u = .5*(inv_s+(k.support()+1)%2);
267   int fi = -k.support()/2-1;
268   assert(f.size() <= m); // Ensure we are upsampling
269   vector<float> g(m); // New sequence of desired size m>f.size()
270   for (int i=0; i<k.support(); i++) // Initialize incremental buffer
271     k.shift_buffer(get(f, ++fi));
272   for (int gi=0; gi<m; ) { // Sample reconstruction of f into g[gi]
273     g[gi] = k.sample_buffer(u);
274     if (advance(gi, fi, u, inv_s))
275       k.shift_buffer(get(f,fi));
276   }
```

```
277   return g;
278 }
279
280 // Faster, incremental implementation of downsampling
281 template<typename Kernel> vector<float>
282 downsample2(const vector<float>& f, int m, Kernel& k) {
283   const int n = f.size();
284   double s = double(m)/n; // Scale factor
285   double kr = .5*k.support();
286   int fi = int(ceil(((.5-kr)/m)*n-.5));
287   // Input sample position between output samples
288   double u = ((fi+.5)/n)*m -(.5-kr);
289   assert(f.size() >= m); // Ensure we are downsampling
290   vector<float> g(m); // New sequence of desired size m>f.size()
291   int gi = -k.support();
292   for (int i=0; i < k.support(); i++) // Initialize incremental buffer
293     k.shift_buffer(0.f);
294   while (gi < -1) {
295     k.accumulate_buffer(get(f, fi), u);
296     if (advance(fi, gi, u, s)) k.shift_buffer(0.f);
297   }
298   while (1) { // Accumulate weighted f samples into g
299     k.accumulate_buffer(get(f, fi), u);
300     if (advance(fi, gi, u, s)) {
301       if (gi >= m) break;
302       g[gi] = s*k.shift_buffer(0.f);
303     }
304   }
305   k.digital_filter(g);
306   return g;
307 }
308
309 // Output a sequence
310 static ostream& operator<<(ostream& out, const vector<float>& f) {
311   for (int i=0; i<f.size(); i++)
312     out << (i+.5)/f.size() << '\t' << f[i] << '\n';
313   return out;
314 }
315
316 // Report elapsed lifetime of object
317 class Timing {
318   public:
319     Timing(const string& name) : m_name(name), m_start(now()) { }
320     ~Timing() {
321       using namespace std::chrono;
322       double t = duration_cast<microseconds>(now()-m_start).count();
```

```
323       cout << m_name << " in " << t/1000. << " ms\n";
324     }
325   private:
326     using clock = chrono::high_resolution_clock;
327     using time_point = chrono::time_point<clock>;
328     static time_point now() { return clock::now(); }
329     string m_name;
330     time_point m_start;
331 };
332
333 static double
334 maxerr(const vector<float>& f, const vector<float>& g) {
335   double m = 0.;
336   assert(f.size() == g.size());
337   for (size_t i=0; i<f.size(); i++) {
338     m = max(m, abs(double(g[i])-f[i]));
339   }
340   return m;
341 }
342
343 template<typename Kernel>
344 void test_performance(int up, int down) {
345   Kernel k;
346   vector<float> f{0.f, 3.f, 1.f, .5f, 4.f, 2.f}, g(f);
347   { Timing t(k.name()+" up");    f = upsample(f, up, k); }
348   { Timing t(k.name()+" up2");   g = upsample2(g, up, k); }
349   cout << "  " << maxerr(f, g) << "  max error\n";
350   { Timing t(k.name()+" down");  f = downsample(f, down, k); }
351   { Timing t(k.name()+" down2"); g = downsample2(g, down, k); }
352   cout << "  " << maxerr(f, g) << "  max error\n";
353 }
354
355 // Plot kernel impulse response. Pipe through gnuplot -persist
356 template <typename Kernel>
357 static void
358 gnuplot(const vector<float>& f, int window, Kernel& k, int w) {
359   cout << "set terminal aqua " << window << '\n'; // Change to your needs
360   cout << "set title \"" << k.name() << "\"\n";
361   cout << "set xrange [" << -.5f*w << ":" << .5f*w << "]\n";
362   cout << "plot \"-\" u (" << w << "*($1-0.5)):2 w l t \"\"\n";
363   cout << f << "e\n";
364 }
365
366 template <typename Kernel>
367 void plot(int up, int down, int& window) {
368   Kernel k;
```

```
369   // Trick to see impulse response
370   vector<float> f{0.f, 0.f, 0.f, 0.f, 1.f, .0f, 0.f, 0.f, 0.f};
371   const int w = f.size();
372   f = upsample(f, up, k);
373   gnuplot(f, window++, k, w);
374   f = downsample(f, down, k);
375   gnuplot(f, window++, k, w);
376 }
377
378 template<typename Kernel>
379 static void check_interpolation() {
380   Kernel k;
381   cout << "checking " << k.name() << '\n';
382   vector<float> f{.5f, 2.f, 1.f, 0.f, 5.f};
383   vector<float> g = upsample(f, f.size(), k);
384   cout << "  " << maxerr(f, g) << "  max error\n";
385   g = downsample(f, f.size(), k);
386   cout << "  " << maxerr(f, g) << "  max error\n";
387 }
388
389 int main() {
390   if (1) {
391     const int up = 1000001, down = 101;
392     test_performance<Box>(up, down);
393     test_performance<Hat>(up, down);
394     test_performance<CatmullRom>(up, down);
395     test_performance<MitchellNetravali>(up, down);
396     test_performance<CardinalBspline3>(up, down);
397     test_performance<CardinalOMOMS3>(up, down);
398   } else if (0) {
399     const int up = 1001, down = 51;
400     int window = 1;
401     plot<Box>(up, down, window);
402     plot<Hat>(up, down, window);
403     plot<CatmullRom>(up, down, window);
404     plot<MitchellNetravali>(up, down, window);
405     plot<CardinalBspline3>(up, down, window);
406     plot<CardinalOMOMS3>(up, down, window);
407   } else {
408     check_interpolation<Box>();
409     check_interpolation<Hat>();
410     check_interpolation<CatmullRom>();
411     check_interpolation<CardinalBspline3>();
412     check_interpolation<CardinalOMOMS3>();
413   }
414 }
```

Bibliography

A. Aldroubi and M. Unser. Sampling procedures in function spaces and asymptotic equivalence with Shannon's sampling theory. *Numerical Functional Analysis and Optimization*, 15(1–2):1–21, 1994.

U. Alim, T. Möller, and L. Condat. Gradient estimation revitalized. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1495–1504, 2010.

J. F. Blinn. Return of the jaggy. *IEEE Computer Graphics and Applications*, 9(2): 82–89, 1989.

J. F. Blinn. A ghost in a snowstorm. *IEEE Computer Graphics and Applications*, 18 (1):79–84, 1998.

T. Blu and M. Unser. Quantitative Fourier analysis of approximation techniques: Part I—Interpolators and projectors. *IEEE Transactions on Signal Processing*, 47(10): 2783–2795, 1999a.

T. Blu and M. Unser. Approximation error for quasi-interpolators and (multi-)wavelet expansions. *Applied and Computational Harmonic Analysis*, 6(2):219–251, 1999b.

T. Blu, P. Thénavaz, and M. Unser. Generalized interpolation: Higher quality at no additional cost. In *Proceedings of the IEEE International Conference on Image Processing*, volume 3, pages 667–671, 1999.

T. Blu, P. Thévenaz, and M. Unser. MOMS: Maximal-order interpolation of minimal support. *IEEE Transactions on Image Processing*, 10(7):1069–1080, 2001.

T. Blu, P. Thévenaz, and M. Unser. Linear interpolation revitalized. *IEEE Transactions on Image Processing*, 13(5):710–719, 2004.

E. Catmull and R. Rom. A class of local interpolating splines. In *Computer Aided Geometric Design*, pages 317–326, 1974.

L. Condat and T. Möller. Quantitative error analysis for the reconstruction of derivatives. *IEEE Transactions on Image Processing*, 59(6):2965–2969, 2011.

L. Condat, T. Blu, and M. Unser. Beyond interpolation: optimal reconstruction by quasi-interpolation. In *Proceedings of the IEEE International Conference on Image Processing*, volume 1, pages 33–36, 2005.

Franklin Crow. Summed-area tables for texture mapping. *Computer Graphics (Proceedings of ACM SIGGRAPH 84)*, 18(3):207–212, 1984.

M. Dalai, R. Leonardi, and P. Migliorati. Efficient digital pre-filtering for least-squares linear approximation. In *Visual Content Processing and Representation*, volume 3893 of *Lecture Notes in Computer Science*, pages 161–169. 2006.

C. de Boor. Quasiinterpolants and the approximation power of multivariate splines. Technical Report #99-12, University of Wisconsin-Madison, 1989.

M. A. Z. Dippé and E. H. Wold. Antialiasing through stochastic sampling. *Computer Graphics (Proceedings of ACM SIGGRAPH 1985)*, 19(3):69–78, 1985.

N. A. Dodgson. Quadratic interpolation for image resampling. *IEEE Transactions on Image Processing*, 6(9):1322–1326, 1997.

C. E. Duchon. Lanczos filtering in one and two dimensions. *Journal of Applied Meteorology*, 18(8):1016–1022, 1979.

I. German. Short kernel fifth-order interpolation. *IEEE Transactions on Signal Processing*, 45(5):1355–1359, 1997.

R. W. Hamming. *Digital Filters*. Prentice Hall, 1977.

P. S. Heckbert. Filtering by repeated integration. *Computer Graphics (Proceedings of ACM SIGGRAPH 1986)*, 20(4):315–321, 1986.

H. S. Hou and H. C. Andrews. Cubic splines for image interpolation and digital filtering. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 25(6): 508–517, 1978.

R. Hummel. Sampling for spline reconstruction. *SIAM Journal on Applied Mathematics*, 43(2):278–288, 1983.

J. Kajiya and M. Ullner. Filtering high quality text for display on raster scan devices. *Computer Graphics (Proceedings of ACM SIGGRAPH 1981)*, 15(3):7–15, 1981.

R. G. Keys. Cubic convolution interpolation for digital image processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(6):1153–1160, 1981.

C. Lee, M. Eden, and M. Unser. High-quality image resizing using oblique projection operators. *IEEE Transactions on Image Processing*, 7(5):679–692, 1998.

M. A. Malcolm and J. Palmer. A fast method for solving a class of tridiagonal linear systems. *Communications of the ACM*, 17(1):14–17, 1974.

M. D. McCool. Analytic antialiasing with prism splines. In *Proceedings of ACM SIGGRAPH 1995*, pages 429–436, 1995.

E. H. W. Meijering. A chronology of interpolation: From ancient astronomy to modern signal processing. *Proceedings of the IEEE*, 90(3):319–342, 2002.

E. H. W. Meijering, W. J. Niessen, J. P. W. Pluim, and M. A. Viergever. Quantitative comparison of sinc-approximating kernels for medical image interpolation. In *Medical Image Computing and Computer-Assisted Intervention*, volume 1679 of *Lecture Notes in Computer Science*, pages 210–217. 1999a.

E. H. W. Meijering, K. J. Zuiderveld, and M. A. Viergever. Image reconstruction by convolution with symmetrical piecewise nth-order polynomial kernels. *IEEE Transactions on Image Processing*, 8(2):192–201, 1999b.

E. H. W. Meijering, W. J. Niessen, and M. A. Viergever. Quantitative evaluation of convolution-based methods for medical image interpolation. *Medical Image Analysis*, 5(2):111–126, 2001.

D. P. Mitchell and A. N. Netravali. Reconstruction filters in computer graphics. *Computer Graphics (Proceedings of ACM SIGGRAPH 1988)*, 22(4):221–228, 1988.

A. Muñoz, T. Blu, and M. Unser. Least-squares image resizing using finite differences. *IEEE Transactions on Image Processing*, 10(9):1365–1378, 2001.

D. Nehab, A. Maximo, R. S. Lima, and H. Hoppe. GPU-efficient recursive filtering and summed-area tables. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2011)*, 30(6):176, 2011.

S. K. Park and R. A. Schowengerdt. Image reconstruction by parametric cubic convolution. *Computer Vision, Graphics & Image Processing*, 23(3):258–272, 1983.

J. A. Parker, R. V. Kenyon, and D. E. Troxel. Comparison of interpolating methods for image resampling. *IEEE Transactions on Medical Imaging*, MI-2(1):31–39, 1983.

K. Perlin. State of the art in image synthesis, 1985. SIGGRAPH 1985 Course Notes.

Pixar. *The RenderMan Interface*, 2005. Version 3.2.1.

D. Ruijters, B. M. ter Haar Romeny, and P. Suetens. Efficient GPU-based texture interpolation using uniform B-splines. *Journal of Graphics, GPU & Game Tools*, 13(4):61–69, 2008.

R. W. Schafer and L. R. Rabiner. A digital signal processing approach to interpolation. *Proceedings of the IEEE*, 61(6):692–702, 1973.

A. Schaum. Theory and design of local interpolators. *Computer Vision, Graphics & Image Processing*, 55(6):464–481, 1993.

C. E. Shannon. Communication in the presence of noise. *Proceedings of the Institute of Radio Engineers*, 37(1):10–21, 1949.

C. Sigg and M. Hadwiger. Fast third-order texture filtering. In M. Pharr, editor, *GPU Gems 2*, chapter 20, pages 313–329. Addison Wesley Professional, 2005.

G. Strang and G. Fix. A Fourier analysis of the finite element variational method. In *Constructive Aspects of Functional Analysis*, pages 793–840, 1973.

P. Thévenaz, T. Blu, and M. Unser. Interpolation revisited. *IEEE Transactions on Medical Imaging*, 19(17):739–758, 2000.

M. Unser. Sampling—50 years after Shannon. *Proceedings of the IEEE*, 88(4): 569–587, 2000.

M. Unser and A. Aldroubi. A general sampling theory for nonideal acquisition devices. *IEEE Transactions on Signal Processing*, 42(11):2915–2925, 1994.

M. Unser, A. Aldroubi, and M. Eden. Fast B-spline transforms for continuous image representation and interpolation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3):277–285, 1991.

M. Unser, A. Aldroubi, and M. Eden. Enlargement or reduction of digital images with minimum loss of information. *IEEE Transactions on Image Processing*, 4(3): 247–258, 1995a.

M. Unser, P. Thénevaz, and L. Yaroslavsky. Convolution-based interpolation for fast, high-quality rotation of images. *IEEE Transactions on Image Processing*, 4(10): 1371–1381, 1995b.

Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13 (4):600–612, 2004.