2015/06/16 v0.5.1 IPOL article class

Published in Image Processing On Line on 2011–09–27.
Submitted on 2011–00–00, accepted on 2011–00–00.
ISSN 2105–1232 © 2011 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2011.g_lmii

Linear Methods for Image Interpolation

Pascal Getreuer

CMLA, ENS Cachan, France (getreuer@gmail.com)

Communicated by Gabriele Facciolo *Demo edited by* Pascal Getreuer

Abstract

We discuss linear methods for interpolation, including nearest neighbor, bilinear, bicubic, splines, and sinc interpolation. We focus on separable interpolation, so most of what is said applies to one-dimensional interpolation as well as N-dimensional separable interpolation.

Source Code

The source code (ANSI C), its documentation, and the online demo are accessible at the IPOL web page of this article[1].

Keywords: interpolation; linear methods; separable interpolation

1 Introduction

Given input image v with uniformly-sampled pixels $v_{m,n}$, the goal of interpolation is to find a function $u(x, y)$ satisfying

$$v_{m,n} = u(m, n) \quad \text{for all } m, n \in \mathbb{Z},$$

such that u approximates the underlying function from which v was sampled. Another way to interpret this is consider that v was created by subsampling, and interpolation attempts to invert this process.

We discuss linear methods for interpolation, including nearest neighbor, bilinear, bicubic, splines, and sinc interpolation. We focus on separable interpolation, so most of what is said applies to one-dimensional interpolation as well as N-dimensional separable interpolation.

[1] https://doi.org/10.5201/ipol.2011.g_lmii

2 Notations

We use the following normalization of the Fourier transform,

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) \mathrm{e}^{-2\pi i x \xi} dx,$$

the transform of a function f is denoted by hat. We will also use the bilateral Z-transform. The Z-transform of a sequence c_n is

$$c(z) := \sum_{n=-\infty}^{\infty} c_n z^{-n}.$$

We denote the sequence with a subscript c_n and its Z-transform as $c(z)$.

3 Interpolation Kernels

Linear interpolation can be abstractly described as a linear operator Z mapping samples v to a function $u := Z(v)$. We define two properties on Z.

1. Let $S_{k,l}$ denote shifting, $S_{k,l}(v)(m,n) := v_{m-k,n-l}$. Z is said to be *shift-invariant* if for every v and (k,l)
$$Z(S_{k,l}(v))(x,y) = Z(v)(x-k, y-l).$$

2. Let v^N denote the restriction of v to the set $\{-N, \ldots, +N\}^2$, then Z is said to be *local* if for every v and (x,y)
$$Z(v)(x,y) = \lim_{N\to\infty} Z(v^N)(x,y).$$

If Z is a linear, shift-invariant, local operator from $\ell^\infty(\mathbb{Z}^2)$ to $L^\infty(\mathbb{R}^2)$, then there exists a unique function $K \in L^1$ called the *interpolation kernel* such that $Z(v)$ can be written as a convolution [10]

$$Z(v)(x,y) = \sum_{m,n\in Z} v_{m,n} K(x-m, y-n).$$

The conditions for this result are mild and are satisfied by most linear interpolation methods of practical interest.

The properties of the kernel K, like symmetry, regularity, and moments, imply properties of the interpolation. For four-fold rotational symmetry, K is usually designed as a tensor product $K(x,y) = K_1(x)K_1(y)$, where K_1 is a symmetric function. Such a kernel also has the benefit that the interpolation is separable, the interpolation can be decomposed into one-dimensional interpolations along each dimension.

Similarly in three (or higher) dimensions, it is convenient to use a separable kernel $K(x,y,z) = K_1(x)K_1(y)K_1(z)$ so that the interpolation decomposes into one-dimensional interpolations. More generally, one can use different one-dimensional kernels along different dimensions, $K(x,y,z) = K_1(x)K_2(y)K_3(z)$. For instance, it may be more appropriate to use a different kernel for the temporal dimension than for the spatial dimensions in video interpolation.

In order to have the interpolant agree with the known samples $Z(v)(m,n) = v_{m,n}$ it is easy to see that the requirements are $K(m,n) = 0$ for all integer m and n, except at the origin where $K(0,0) = 1$. K is said to be an *interpolating* function if it satisfies these requirements.

4 Two-Step Interpolation

That K must be interpolating can be an inconvenient restriction. It complicates the problem of finding a kernel K that also satisfies other design objectives. It can also be an obstacle computationally. In some cases such as spline interpolation, K has infinite support, yet the interpolant can be expressed as a linear combination of compact support functions (the B-splines).

Suppose that one dimensional data f_m is to be interpolated. An alternative approach is to begin with a basis function $\varphi(t)$ and then express the interpolant as a linear combination of $\{\varphi(t-n)\}$

$$u(t) = \sum_{n\in\mathbb{Z}} c_n \varphi(t-n),$$

where the coefficients c_n are selected such that

$$f_m = \sum_{n\in\mathbb{Z}} c_n \varphi(m-n).$$

That is, interpolation is a two-step procedure: first, we solve for the coefficients c_n and second, the interpolant is constructed as $\sum c_n\varphi(t-n)$.

To solve for the coefficients, notice that $f_m = \sum c_n\varphi(m-n)$ is equivalent to a discrete convolution $f_n = (c * p)_m$ where $p_m = \varphi_m$. This suggests solving for c_n under the Z-transform as

$$f(z) = c(z)p(z) \quad \Rightarrow \quad c(z) = \frac{1}{p(z)} f(z).$$

Let $(p)^{-1}$ denote the convolution inverse of p, which is the sequence such that $p * (p)^{-1} = \delta$ where δ is the unit impulse ($\delta_0 := 1$ and $\delta_m := 0$ otherwise). Using $(p)^{-1}$ we obtain c_n by "prefiltering" as $c = (p)^{-1} * f$.

A unique convolution inverse exists in many practical cases of interest [7]. If it exists and φ is real and symmetric, then $(p)^{-1}$ can be factored into pairs of recursive (infinite impulse response) filters, allowing for an efficient in-place calculation. Finally, the interpolant is obtained as $\sum c_n\varphi(t-n)$.

It is possible to express this two-step interpolation procedure in terms of an interpolation kernel,

$$u(t) = \sum_{n\in\mathbb{Z}} ((p)^{-1} * f)_n \varphi(t-n)$$

$$u(t) = \sum_{m\in\mathbb{Z}} f_m \sum_{n\in\mathbb{Z}} ((p)^{-1})_{n-m} \varphi(t-n)$$

$$u(t) = \sum_{m\in\mathbb{Z}} f_m K_1(t-m), \quad K_1(t) = \sum_{n\in\mathbb{Z}} ((p)^{-1})_n \varphi(t-n)$$

For example, the figures 1 and 2 illustrate this relationship for cubic and septic (7th) B-spline interpolation where φ are the cubic and septic B-splines. In both cases, φ is not an interpolating function, but prefiltering with $(p)^{-1}$ yields a kernel K_1 that is interpolating.

5 Interpolation Properties

An interpolation method Z has *approximation order* J if it reproduces polynomials up to degree $(J-1)$.

Let Z and K_1 be defined from a function φ as in the previous section. The *Strang–Fix conditions* [2] provide the following equivalent conditions to determine the approximation order.

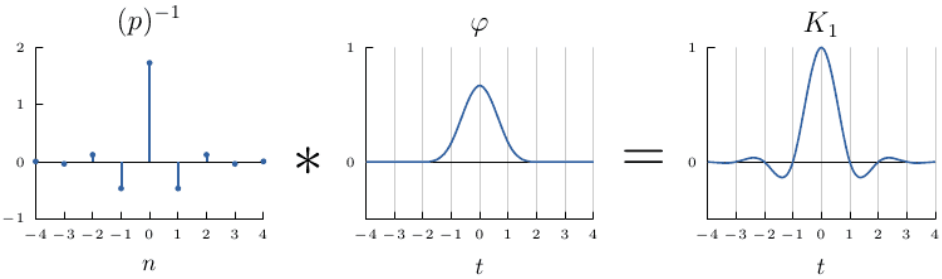

Figure 1: Two-step interpolation where φ is the cubic B-spline.

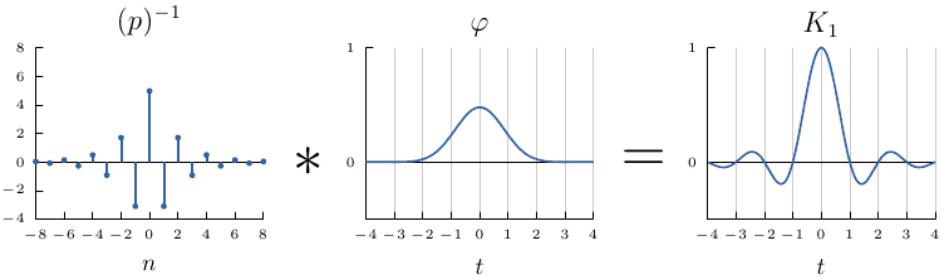

Figure 2: Two-step interpolation where φ is the septic B-spline.

1. Z has approximation order J,
2. $\hat{\varphi}(0) = 1, \quad \hat{\varphi}^j(2\pi k) = 0 \quad \forall k \in \mathbb{Z}_*, j = 0, \ldots, J-1,$
3. $\sum_{k\in\mathbb{Z}} (x-k)^j \varphi(x-k) = \mu_j \quad \forall x \in \mathbb{R}, j = 0, \ldots, J-1.$

where $\hat{\varphi}^j$ denotes the jth derivative of $\hat{\varphi}$ and $\mathbb{Z}_* := \mathbb{Z}/\{0\}$. Aside from approximation order, another property of interest is the L^2 interpolation error. Let $f \in L^2(\mathbb{R})$ and define $f_\delta(t) = \sum f(\delta n) K_1(t-n)$ for $\delta > 0$ The interpolation error is approximately

$$\|f - f_\delta\|_{L^2}^2 \approx \int |f(\xi)|^2 E_{int}(\xi\delta) d\xi,$$

$$E_{int}(\xi) := \frac{\left|\sum_{k\in\mathbb{Z}_*} \hat{\varphi}(\xi+k)\right|^2 + \sum_{k\in\mathbb{Z}_*} |\hat{\varphi}(\xi+k)|^2}{\left|\sum_{k\in\mathbb{Z}} \hat{\varphi}(\xi+k)\right|^2}.$$

This approximation is exact for bandlimited functions. Otherwise, it is equal to the average L^2 error over all translations of f. The function E is called the *interpolation error kernel*, and it can be interpreted as predicting the error made in interpolating $\sin(2\pi\zeta t)$. Asymptotically as $\delta \to 0$,

$$\|f - f_\delta\|_{L^2} \sim C_{int} \delta^J \|f^{(J)}\|_{L^2}.$$

The decay of the error is dominated by the approximation order J. When comparing methods of the same approximation order, the constant C_{int} is also of interest [7].

6 Kernel Normalization

A basic and desirable quality of an interpolation method is that it reproduces constants, which is equivalent to

$$\sum_{n\in\mathbb{Z}} K_1(t-n) = 1 \quad \forall t \in \mathbb{R}.$$

If the kernel does not reproduce constants, it can be the normalized to fix this as

$$\tilde{K}_1(t) = \frac{K_1(t)}{\sum_{m\in\mathbb{Z}} K_1(t-m)},$$

provided that the denominator does not vanish. Normalization ensures that the interpolation weights sum to 1. Interpolation with the normalized kernel reproduces constants: suppose $f_n = c$, then

$$u(t) = \sum_{n\in\mathbb{Z}} c\tilde{K}_1(t-n) = \frac{\sum_{n\in\mathbb{Z}} cK_1(t-n)}{\sum_{m\in\mathbb{Z}} K_1(t-m)} = c.$$

7 Nearest Neighbor

The most widely used methods for image interpolation are nearest neighbor, bilinear, and bicubic interpolation (see Figure 3).

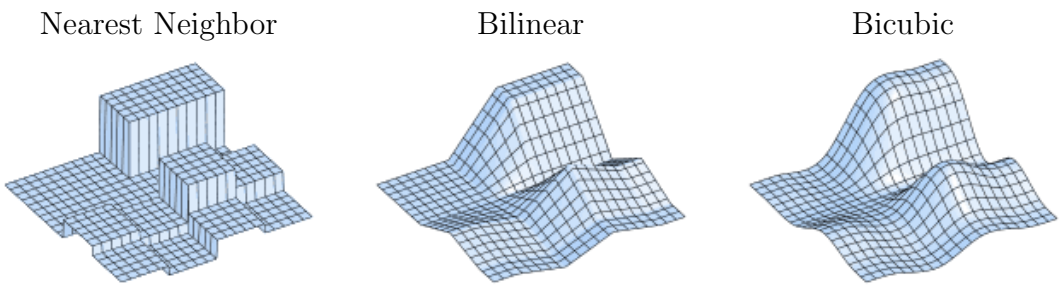

Figure 3: Nearest neighbor, bilinear, and bicubic applied to the same uniformly-spaced input data.

The nearest neighbor interpolation of v is the piecewise constant function

$$u(x,y) = v_{[x],[y]},$$

where [.] denotes rounding to the nearest integer. That is, $u(x,y)$ is defined as the value of the input sample that is closest to (x,y). For this reason, nearest neighbor interpolation is sometimes called "pixel duplication". The interpolation kernel for nearest neighbor is (Figure 4)

$$K(x,y) = K_1(x)K_1(y), \quad K_1(t) = \begin{cases} 1 & \text{if } -\frac{1}{2} \le t < \frac{1}{2}, \\ 0 & \text{otherwise.} \end{cases}$$

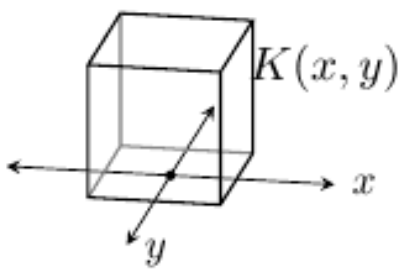

Figure 4: Nearest neighbor interpolation kernel.

The nearest neighbor idea can be applied to very general kinds of data, to any set of samples that has a notion of distance between sample locations. The interpolant is constant within the Voronoi cell around each sample location (Figure 5).

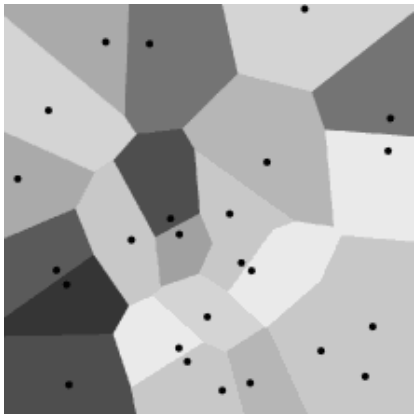

Figure 5: Nearest neighbor scattered data interpolation.

8 Bilinear

The bilinear interpolation of v is the continuous function

$$\begin{array}{rcrcr} u(x,y) & = & (1-\langle x\rangle)(1-\langle y\rangle)v_{\lfloor x\rfloor,\lfloor y\rfloor} & + & \langle x\rangle(1-\langle y\rangle)v_{\lfloor x\rfloor+1,\lfloor y\rfloor} \\ & + & (1-\langle x\rangle)\langle y\rangle v_{\lfloor x\rfloor,\lfloor y\rfloor+1} & + & \langle x\rangle\langle y\rangle v_{\lfloor x\rfloor+1,\lfloor y\rfloor+1} \end{array},$$

where $\lfloor . \rfloor$ denotes the floor function and $\langle x\rangle := x - \lfloor x\rfloor$ is the fractional part. The interpolation kernel is (Figure 6)

$$K(x,y) = K_1(x)K_1(y), \quad K_1(t) = (1-|t|)^+,$$

where $(.)^+$ denotes positive part.

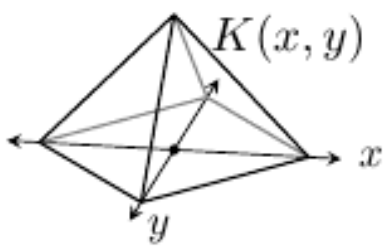

Figure 6: Bilinear interpolation kernel.

Within each cell $[m, m+1] \times [n, n+1]$, the interpolation is a convex combination of the samples located at the cell corners $v_{m,n}$, $v_{m+1,n}$, $v_{m,n+1}$, $v_{m+1,n+1}$. Because this combination is convex, the interpolation is bounded between $\min v$ and $\max v$ and does not produce overshoot artifacts. Bilinear interpolation reproduces affine functions: if $v_{m,n} = am + bn + c$ then $u(x,y) = ax + by + c$.

Bilinear interpolation is arguably the simplest possible separable method that produces a continuous function. It is extremely efficient and on many platforms available in hardware, making it practical for realtime applications.

A closely related method to bilinear interpolation is linear interpolation. Given a triangulation of the sample locations, each triangle is interpolated by the affine function that satisfies the data at the triangle vertices.

9 Bicubic

Bicubic interpolation [3] uses the interpolation kernel (Figures 7 and 8)

$$K_1(t) = \begin{cases} (\alpha+2)|t|^3 - (\alpha+3)|t|^2 + 1 & \text{if } |t| \le 1 \\ \alpha|t|^3 - 5\alpha|t|^2 + 8\alpha|t| - 4\alpha & \text{if } 1 < |t| < 2, \\ 0 & \text{otherwise.} \end{cases}$$

where α is a free parameter. This function is derived by finding a piecewise cubic polynomial with knots at the integers that is required to be symmetric, C^1 continuous, and have support in $-2 < t < 2$. These conditions leave one remaining degree of freedom represented by α. For any value of α, K_1 has extremal points at $t = 0$ and $\pm 4/3$.

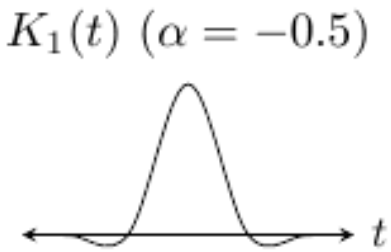

Figure 7: Cubic interpolation kernel.

The values -1, -0.75, and -0.5 have been proposed for α, motivated by various notions of optimality [9]. The choice $\alpha = -0.5$ is particularly compelling. Bicubic interpolation is third-order accurate with $\alpha = -0.5$ and only first-order accurate for any other α. Furthermore, $\alpha = -0.5$ is optimal in a sense of matching the sinc kernel and is also optimal in terms of E_{int}. All examples and comparisons with bicubic here use $\alpha = -0.5$.

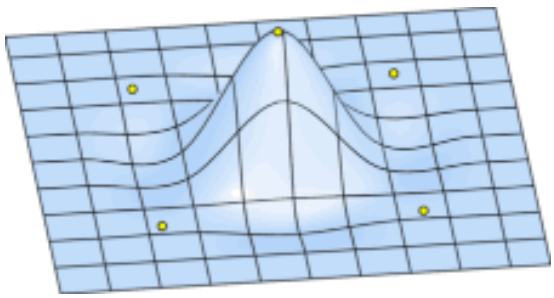

Figure 8: Surface plot of $K1(x)K1(y)$. Yellow circles indicate local maxima.

10 Sinc

The Whittaker–Shannon interpolation [1] of $v_{m,n}$ is

$$u(x,y) = \sum_{m,n\in\mathbb{Z}} v_{m,n} \operatorname{sinc}(x-m)\operatorname{sinc}(y-n),$$

where $sinc(t) := sin(\pi t)/(\pi t)$ for $t \neq 0$ and $sinc(0) := 1$ is the interpolation kernel (Figures 10 and 11). The interpolation is such that u is bandlimited, that is, its Fourier transform is zero for

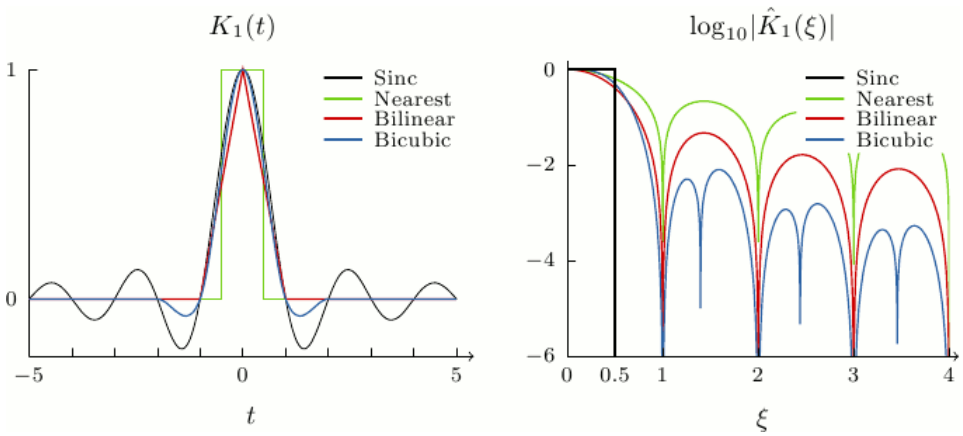

Figure 9: Comparison of the nearest neighbor, bilinear, and bicubic kernels and the sinc.

frequencies outside of $[-\frac{1}{2}, \frac{1}{2}]$. This interpolation is also often called "sinc interpolation" or "Fourier zero-padding." The extremal points of the sinc kernel are solutions of the equation $\pi t = tan(\pi t)$, which include $t \approx 0$, ± 1.4303, ± 2.4590, ± 3.4709, ± 4.4774.

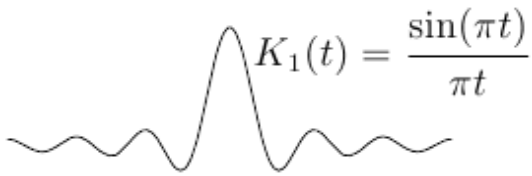

Figure 10: The sinc function.

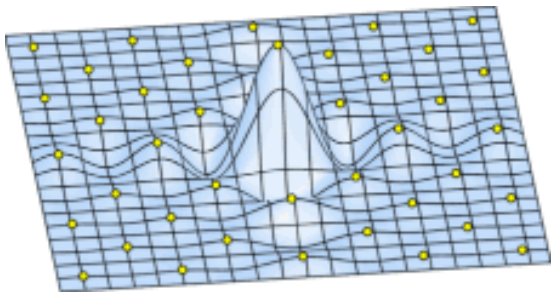

Figure 11: Surface plot of $sinc(x)sinc(y)$. Yellow circles indicate local maxima.

The powerful property of sinc interpolation is that it is exact for bandlimited functions, that is, if $v_{m,n} = f(m,n)$ with

$$\iint f(x,y)\exp^{-2\pi i(x\xi+y\eta)}\,dxdy = 0 \quad \text{for all} \quad |\xi| \quad \text{or} \quad |\eta| \geq \frac{1}{2},$$

then the interpolation reproduces f.

Proof. We can verify this result for $f \in L^1$ from two Fourier properties: first, the Fourier transform of sinc is the rectangular pulse

$$\int \mathbb{1}_{\{|\xi|<1/2\}} e^{2\pi i t\xi} dt = \frac{e^{\pi i t} - e^{-\pi i t}}{2\pi i t} = \frac{\sin \pi t}{\pi t},$$

and second, the Poisson formula

$$\sum_{m,n\in\mathbb{Z}} f(m,n)e^{-2\pi i(m\xi+n\eta)} = \sum_{m,n\in\mathbb{Z}} \hat{f}(\xi-m,\eta-n).$$

Applying these two properties, the Fourier transform of the interpolant is

$$\begin{aligned}
\hat{u}(\xi,\eta) &= \iint \sum_{m,n\in\mathbb{Z}} f(m,n)\ \mathrm{sinc}(x-m)\ \mathrm{sinc}(y-n)e^{-2\pi i(x\xi+y\eta)}dxdy \\
&= \sum_{m,n\in\mathbb{Z}} f(m,n) \iint \mathrm{sinc}(x-m)\ \mathrm{sinc}(y-n)e^{-2\pi i(x\xi+y\eta)}dxdy \\
&= \mathbb{1}_{\{|\xi|<1/2\}}\mathbb{1}_{\{|\eta|<1/2\}} \sum_{m,n\in\mathbb{Z}} f(m,n)e^{-2\pi i(m\xi+n\eta)} \\
&= \mathbb{1}_{\{|\xi|<1/2\}}\mathbb{1}_{\{|\eta|<1/2\}} \sum_{m,n\in\mathbb{Z}} \hat{f}(\xi-n,\eta-m).
\end{aligned}$$

In the second equality, the sum-integral interchange is justified by dominated convergence. The final quantity is equal to $\hat{f}(\xi,\eta)$ if $\hat{f}$ is zero outside of the region $[-\frac{1}{2}\frac{1}{2}]\times[-\frac{1}{2}\frac{1}{2}]$. □

In some ways, sinc interpolation is the ultimate interpolation. It is exact for bandlimited functions, so the method is very accurate on smooth data. Additionally, Fourier zero-padding avoids staircase artifacts, it is effective in reconstructing features at different orientations. Under an aliasing condition [10], Fourier interpolation reproduces cylindrical functions $f(x,y) = h(\alpha x+\beta y)$.

The disadvantage of sinc interpolation is that in aliased images it produces significant ripple artifacts (Gibbs phenomenon) in the vicinity of image edges (Figure 12). This is because $\mathrm{sinc}(x)$ decays slowly, at a rate of $1/x$, so the damage from meeting an edge is spread throughout the image. Moreover, bandlimitedness can be a distorted view of reality. Thévenaz et al. [7] give the following amusing example: consider the air/matter interface of a patient in a CT scan, then according to classical physics, this is an abrupt discontinuity and cannot be expressed as a bandlimited function. Antialias filtering is not possible on physical matter, and any attempt to do so would probably be harmful to the patient.

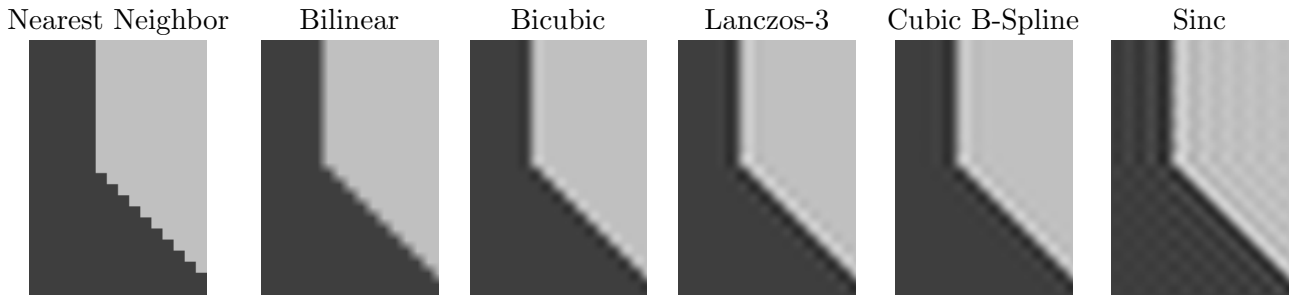

Figure 12: Linear interpolation of a step edge: a balance between staircase artifacts and ripples.

11 Windowed Sinc Approximations

A solution to limiting the ripple artifacts of the sinc kernel is to approximate it with a compactly-supported function

$$K_1(t) = w(t)\mathrm{sinc}(t),$$

where w is a *window function*. A popular choice in image processing is the Lanczos window,

$$w(t) = \begin{cases} \operatorname{sinc}(t/n) & \text{if } |t| < n, \\ 0 & \text{otherwise.} \end{cases}$$

where n is a positive integer usually set to 2 or 3. The Lanczos kernels (Figures 13, 14 and 15) do not reproduce constants exactly, but can be normalized to fix this as described in the section on Kernel Normalization.

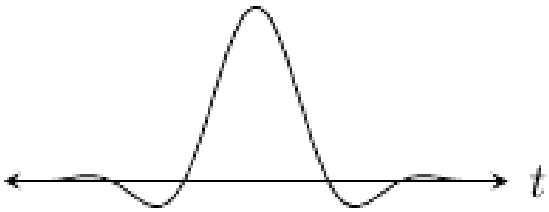

Figure 13: The Lanczos-3 kernel.

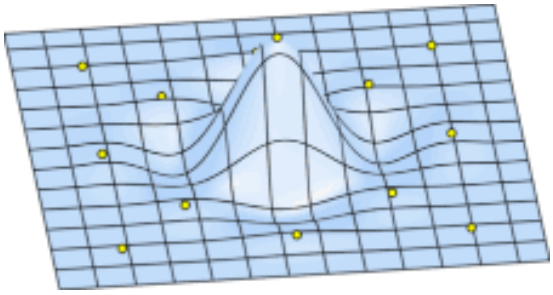

Figure 14: Surface plot of the Lanczos-3 kernel. Yellow circles indicate local maxima.

There are many other possibilities for the window, for example Hamming, Kaiser, and DolphChebyshev windows to name a few, each making different tradeoffs in frequency characteristics.

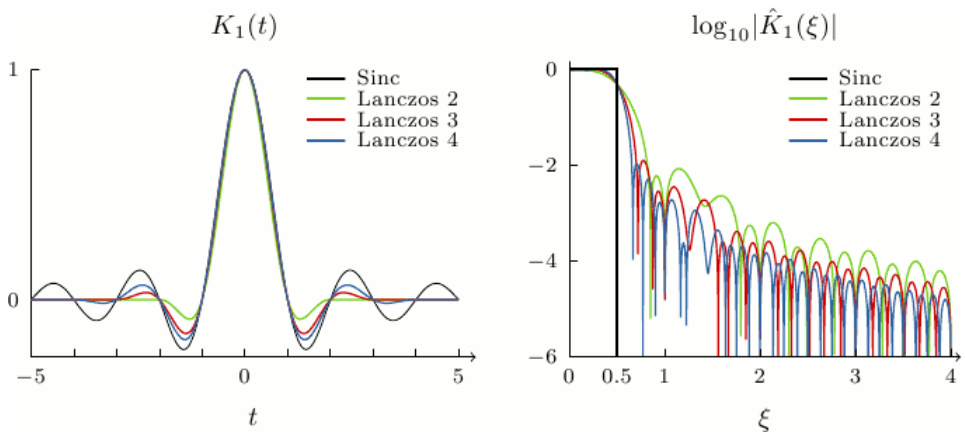

Figure 15: Comparison of the normalized Lanczos kernels and the sinc.

12 Splines

Define the B-splines

$$\beta^j(t) = \frac{1}{j!}\sum_{k=0}^{j+1}\binom{j+1}{k}(-1)^k\big((t+\frac{j+1}{2}-k)^+\big)^j.$$

The B-spline functions (see example in Figure 16) are optimal in the sense that, among all piecewise polynomials with uniformly spaced knots, the B-splines have the maximal approximation order and are maximally continuous for a given support.

The B-spline β^{J-1} has approximation order J. For $J > 2$, β^{J-1} is not interpolating, so prefiltering must be applied as described in the section on Two-Step Interpolation where β^{J-1} takes the role of φ.

As $J \to \infty$, B-spline interpolation converges to Whittaker–Shannon interpolation in a strong sense: the associated interpolation kernel K_1 converges to the sinc both in $1 \le p < \infty$, see [4],[6]. This convergence is illustrated Figure 17 with K_1 and its Fourier transform for degrees 1, 3, 5, and 7.

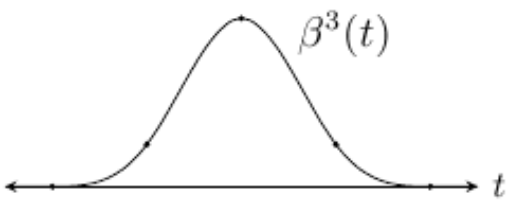

Figure 16: The cubic B-spline.

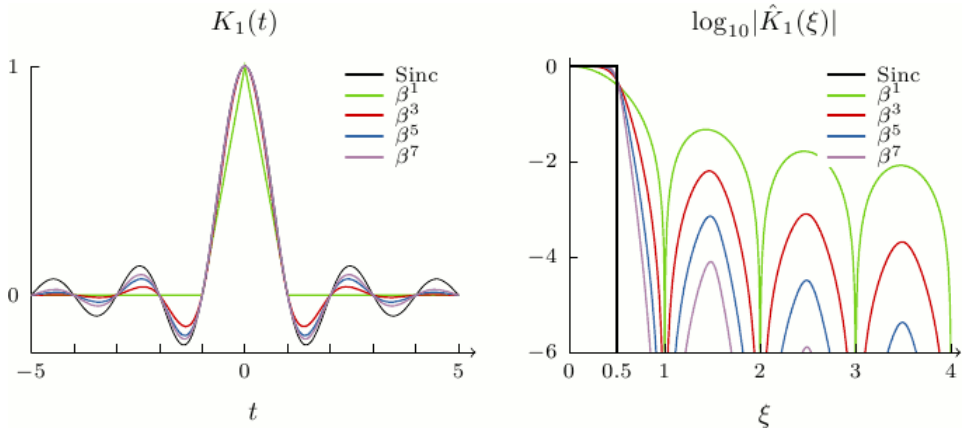

Figure 17: Comparison of the B-spline interpolation kernels and the sinc.

Aside from the B-splines, there are numerous other piecewise polynomial methods for interpolation in the literature. One example is the class of Maximum Order and Minimal Support (Moms) functions introduced by Blu, Thévenaz, and Unser [8], which are linear combinations of β^{J-1} and its derivatives

$$\varphi(t) = \sum_{k=0}^{J-1} c_k \frac{d^k}{dt^k}\beta^{J-1}(t).$$

The B-splines are a special case within the class of Moms. The B-splines are the Moms with maximum regularity, they are $J-2$ times continuously differentiable.

Within the Moms, the "o-Moms functions" are the Moms defined by minimizing the quantity

$$\frac{1}{J!}\sqrt{\sum_{k\in\mathbb{Z}_*}|\hat{\varphi}^(J)(k)|^2}.$$

The o-Moms functions have the same approximation order and support as the B-splines, but the C_{int} constant is smaller. So compared to the B-splines, the advantage of o-Moms is that they have lower asymptotic L^2 interpolation error. On the other hand, o-Moms are less regular than the B-splines.

The splines proposed by Schaum [5] are also within the class of Moms. Schaum splines have the property that they are interpolating, so prefiltering is not needed. However, for a given support size, the approximation constant C_{int} is worse than with the o-Moms of the same support.

The first few even-order o-Moms are

Order J	$\varphi(t)$
2	$\beta^1(t)$
4	$(1+\frac{1}{42}\frac{d^2}{dx^2})\beta^3(t)$
6	$(1+\frac{1}{33}\frac{d^2}{dx^2}+\frac{1}{7920}\frac{d^4}{dx^4})\beta^5(t)$
8	$(1+\frac{1}{30}\frac{d^2}{dx^2}+\frac{1}{4680}\frac{d^4}{dx^4}+\frac{1}{3603600}\frac{d^6}{\beta}^7(t)$
10	$(1+\frac{2}{57}\frac{d^2}{dx^2}+\frac{7}{25840}\frac{d^4}{dx^4}+\frac{1}{1627920}\frac{d^6}{dx^6}+\frac{1}{3047466240}\frac{d^8}{dx^8})\beta^9(t)$
12	$(1+\frac{5}{138}\frac{d^2}{dx^2}+\frac{1}{3220}\frac{d^4}{dx^4}+\frac{1}{1101240}\frac{d^6}{dx^6}+\frac{1}{1078334208}\frac{d^8}{dx^8}+\frac{1}{4151586700800}\frac{d^{10}}{dx^{10}})\beta^{11}(t)$

A note on numbering: Usually, with a piecewise polynomial method, its approximation order is L but its highest degree is $L-1$. We refer to methods by degree, for instance "o-Moms 3" and "β^3" are methods that are locally cubic polynomial and have approximation order 4.

13 Radial Basis Functions

A radial basis function (RBF) is a function φ such that

$$\varphi(x,y)=\varphi(r),\quad r:=\sqrt{x^2+y^2}.$$

RBF interpolation is useful for scattered data interpolation. Let (x_n,y_n) denote the location of the nth sample, then the interpolant is

$$u(x,y)=\sum_n c_n\varphi(x-x_n,y-y_n),$$

where the coefficients c_n are solved such that $u(x_n,y_n)=v_n$. If the samples are uniformly spaced, then the c_n can be found as described in the section on Two-Step Interpolation.

Popular choices for φ include

- Thin plate splines: $\varphi(r)=|\varepsilon r|^n\log|\varepsilon r|,\quad n$ even
- Bessel: $\varphi(r)=J_0(\varepsilon r)$
- Gaussian: $\varphi(r)=\exp(-(\varepsilon r)^2)$
- Multiquadric: $\varphi(r)=(1+(\varepsilon r)^2)^{\frac{1}{2}}$
- Inverse multiquadratic: $\varphi(r)=\varphi(r)=(1+(\varepsilon r)^2)^{-\frac{1}{2}}$

- Inverse quadratic: $\varphi(r) = \varphi(r) = (1 + (\varepsilon r)^2)^{-1}$

where ε is a positive parameter controlling the shape. A challenging problem is that smaller ε improves interpolation quality but worsens the numerical conditioning of solving the c_n. A topic of interest beyond the scope of this article is RBF interpolation in the limit $\varepsilon \to 0$. For example, [11] investigates numerically stable interpolation for small ε with Gaussian φ.

14 Methodology

14.1 Boundary handling

A technicality is how to handle the image boundaries. The interpolation formulas may refer to samples that are outside of the domain of definition, especially when interpolating at points near or on the image boundaries.

The usual approach is to extrapolate (pad) the input image. Several methods for doing this are

- Constant extension, $\ldots aaaabcdeeee\ldots$
- Half-sample symmetric extension, $\ldots cbaabcdeedc\ldots$
- Whole-sample symmetric extension, $\ldots dcbabcdedcb\ldots$

14.2 Image scaling

Suppose v is an $M \times N$ image with samples $v_{m,n}$, $m = 0, \ldots M-1$, $n = 0, \ldots N-1$ which are located on the integer grid $\mathbb{Z}^2$.

For scaling v by a (possibly non-integer) factor d, a choice for where to resample the function u is on a top-left-anchored grid of points (Figure 18)

$$(\frac{1}{d}m', \frac{1}{d}n'), \quad m' = 0, \ldots, M'-1, ; n' = 0, \ldots, N'-1,$$

with $M' = [dM]$, $N' = [dN]$, where $[.]$ is a rounding convention.

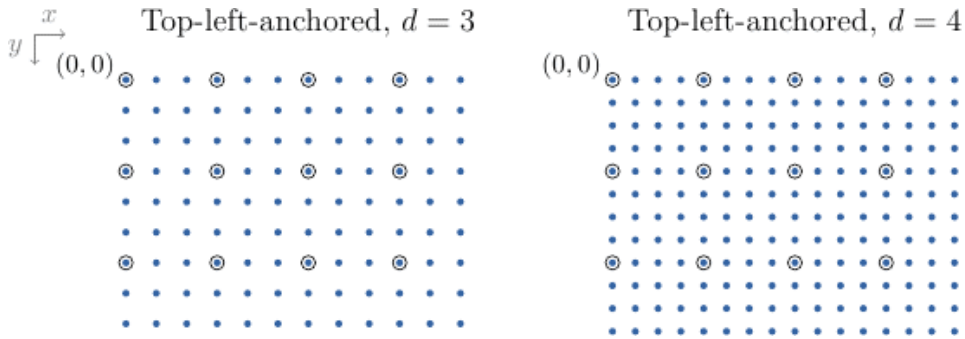

Figure 18: Example top-left-anchored grids. Open circles denote input samples and filled circles denote interpolation samples.

Another choice is the centered grid of points (Figure 19)

$$(\frac{1}{d}m' + s_{d,M,M'}, \frac{1}{d}n' + s_{d,N,N'}), \quad m' = 0, \ldots, M'-1, ; n' = 0, \ldots, N'-1,$$

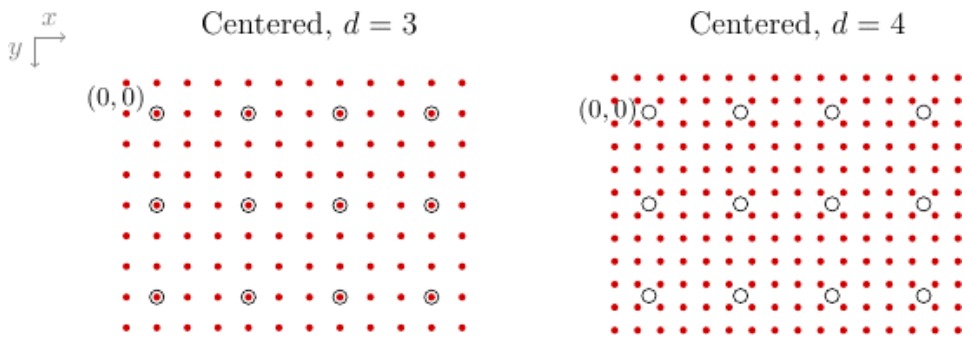

Figure 19: Example centered grids. Open circles denote input samples and filled circles denote interpolation samples.

where $s_{d,M,M'} = (1/d - 1 + M - M'/d)/2$. If d is integer, the offsets simplify to $s = (1/d - 1)/2$. Although more complicated, the centered grid has the advantage that interpolation with a symmetric kernel on this grid commutes with flipping the image.

For an odd integer scale factor and interpolation with a symmetric kernel, the two grids are related in a simple way through adding and removing border samples. Interpolation performed on the top-left-anchored grid is converted to an interpolation on the centered grid by removing $(d-1)/2$ rows and columns from the bottom and right borders and extrapolating (according to the boundary extension) the same number of rows and columns on the top and left.

14.3 Domain mapping

Aside from changing image resolution, another application is to deform or map the domain of the image. That is, the mapped coordinates (x', y') are related to the original coordinates (x, y) by

$$x = f(x', y'), \quad y = g(x', y').$$

Let $u(x, y)$ be an interpolation of v, then the deformed image is obtained by resampling as

$$\tilde{v}_{m,n} = u\big(f(m, n), g(m, n)\big).$$

For example, two simple mappings are

- Sub-pixel translation
  $x = x' - \Delta x$
  $y = y' - \Delta y$

- Rotation
  $x = x' \cos\theta - y' \sin\theta$
  $y = x' \sin\theta - y' \cos\theta$

Other applications include parallax correction, lens distortion correction, image registration, and texture mapping.

15 Algorithm

Linear interpolation amounts to evaluating the sum

$$u(x, y) = \sum_{m,n \in \mathbb{Z}} v_{m,n} K(x - m, y - n).$$

Choices of algorithms to do this efficiently depend on K and the sampling locations. Keep in mind that the sum is conceptually over the infinite integer grid, so $v_{n,m}$ should be replaced with an appropriate boundary extension when m, n is beyond the domain of the image.

15.1 Nearest neighbor, bilinear, and bicubic

Suppose the interpolation method has a separable kernel $K(x, y) = K_1(x)K_1(y)$ with compact support, $K_1(x) = 0$ for $x \geq R$. Then interpolation at point (x, y) can be implemented as

$$\begin{aligned} m_0 &= [x - R],\ n_0 = [y - R],\ L = 2R \\ u(x, y) &= \sum_{m=m_0}^{m_0+L-1} \sum_{n=n_0}^{n_0+L-1} v_{m,n} K_1(x - m) K_1(y - n), \end{aligned}$$

where there are $L \times L$ non zero terms in the sum. The values of K_1 can be reused so that K_1 only needs to be evaluated for $2L$ different arguments

For image scaling, it is possible to interpolate more efficiently by exploiting the separable structure of the sampling points. Let $v_{m,n}, m = 0, \ldots M - 1, n = 0, \ldots N - 1$ be the given image, which is to be interpolated on the grid of points (x'_n, y'_n) for $m' = 0, \ldots M' - 1$,n' = 0,...N'-1. Interpolation is performed by first interpolating along the y direction,

$$w_{m,n'} = \sum_{n=n_0}^{n_0+L-1} v_{m,n} K_1(y_{n'} - n), n_0 = \lceil y_{n'} - R \rceil, \quad \text{for } m = 0, \ldots M - 1, n' = 0, \ldots N' - 1,$$

and then along the x direction,

$$u(x_{m'}, y_{n'}) = \sum_{m=m_0}^{m_0+L-1} w_{m,n} K_1(x_{m'} - m), m_0 = \lceil x_{m'} - R \rceil, \quad \text{for } m' = 0, \ldots M' - 1, n' = 0, \ldots N' - 1.$$

An efficient way to implement the one dimensional interpolations is to express them as multiplication with a sparse matrix

$$\begin{aligned} w_{m,n'} &= \sum_{n=1}^{N} A_{n',n} v_{m,n}, \\ u(x_{m'}, y_{n'}) &= \sum_{m=1}^{M} B_{m',m} w_{m,n'}. \end{aligned}$$

A is a sparse matrix of size $N' \times N$ and B is a sparse matrix of size $M' \times M$. Away from the boundaries, the matrix entries are

$$A_{n',n} = K_1(y_{n'} - n), \quad B_{m',m} = K_1(x_{m'} - m).$$

Near the boundaries, the matrix entries need to be adjusted according to the boundary extension. Define

Half-sample symmetric: $\mathcal{E}(n) = \min\{n \mod 2N,\ (2N - 1 - n) \mod 2N\}$

Whole-sample symmetric: $\mathcal{E}(n) = \min\{n \mod 2N - 2,\ (2N - 2 - n) \mod 2N - 2\}$

Constant extension: $\mathcal{E}(n) = \begin{cases} 0 & \text{if } n < 0, \\ n & \text{if } 0 \leq n \leq N - 1, \\ N - 1 & \text{if } N - 1 < n, \end{cases}$

then the matrix entries near the boundaries are computed as

$$A_{n',n} = \sum_{k \in \mathcal{E}^{-1}(n)} K_1(y_{n'} - k),$$

and similarly for B. This sparse matrix approach is used by the libswscale library in FFmpeg [2] for scaling video frames. It is especially well-suited for video, as A and B only need to be constructed once to interpolate any number of frames.

If the interpolation locations are uniformly spaced with a rational period, $t_{n'} = t_0 + n'a/b$, then another approach is to compute one-dimensional interpolation through convolution. Given n', let $s = b\lfloor n'/b \rfloor$ and r be such that $n' = bs + r$, then

$$\sum_n f_n K_1(t_{n'} - n) = \sum_n f_n K_1(t_0 + r\frac{a}{b} + as - n) = (f * h^r{}_n)_{as}, \qquad h^r_n = K_1(t_0 + r\frac{a}{b} + n).$$

15.2 Lanczos

Interpolation with the Lanczos kernels does not exactly reproduce constants. To fix this, the Lanczos kernels should be normalized as described in the section on Kernel Normalization.

Kernel normalization can be incorporated efficiently into the algorithms discussed in the previous section. For the interpolation of a single point (x, y), the normalized interpolation is

$$u(x,y) = \frac{\sum_{m,n} v_{m,n} K_1(x-m) K_1(y-n)}{\sum_{m,n} K_1(x-m) K_1(y-n)}.$$

For the sparse matrix approach, kernel normalization is achieved by scaling each matrix row so that it sums to one. Similarly for the convolution approach, h^r should be scaled so that it sums to one.

15.3 Splines

For spline methods, implementation is as described in the section on Two-Step Interpolation. The method is determined by the choice of basis function φ, for example, φ may be a B-spline or an o-Moms function.

Given a basis function φ, define $p_m = \varphi(m)$. For the prefiltering step, the convolution inverse $(p)^{-1}$ is needed. We look first at the cubic B-spline as an example. For the cubic B-spline,

$$p(z) = (z + 4 + z^{-1})/6, \quad \text{and} \quad \frac{6}{z + 4 + z^{-1}} = 6\frac{1}{1 - rz^{-1}}\frac{-r}{1 - rz},$$

where $r = 3^{1/2} - 2.$. The first factor corresponds to the first-order causal recursive filter

$$c'_n = f_n + rc'_{n-1}, \quad n = 1, \ldots, N-1.$$

The left endpoint c'_0 can be computed depending on the boundary extension,

$$\text{Half-sample symmetric:} \qquad c'_0 = f_0 + r\sum_{n=0}^{\infty} r^n f_n,$$

$$\text{Whole-sample symmetric:} \quad c'_0 = \sum_{n=0}^{\infty} r^n f_n.$$

[2] http://www.ffmpeg.org.

Since $|r| < 1$, the terms of the sum are decaying, so in practice we only evaluate as many terms as are needed for the desired accuracy.

The second factor corresponds to the first-order anti-causal recursive filter

$$c''_n = r(c''_{n+1} - c'_n), \quad n = N-2, \ldots, 0.$$

The right endpoint c''_{N-1} can be computed according to the boundary extension as

$$\text{Half-sample symmetric:} \quad c''_{N-1} = \frac{r}{r-1} c'_{N-1}$$

$$\text{Whole-sample symmetric} \quad c''_{N-1} = \frac{r}{r^2-1}(c'_{N-1} + r c'_{N-2}).$$

Finally, the constant scale factor is applied $c_n = 6c''_n$. The cost of prefiltering is linear in the number of pixels. Prefiltering can be computed in-place so that the prefiltered values overwrite the memory used by the input. For prefiltering in two-dimensions, the image is first prefiltered along each column, then the column-prefiltered image is prefiltered along each row.

More generally, if φ is symmetric and compactly supported, then $p(z)$ has the form

$$p(z) = a_0 + \sum_{j=1}^{J} a_j (z^j + z^{-j}),$$

$$p(z) = a_J z^{-J} \prod_{j=1}^{J} (z - r_j)(z - \frac{1}{r_j}),$$

where the r_j and $1/r_j$ are the roots of the polynomial $z^{-J}p(z)$. We enumerate the roots so that $|r_j| < 1$. The Z-transform of $(p)^{-1}$ is

$$\frac{1}{p(z)} = \frac{1}{a_J} \prod_{j=1}^{J} \frac{1}{1 - r_j z^{-1}} \frac{-r_j}{1 - r_j z}$$

Therefore, prefiltering can be performed as a cascade of first-order recursive filters using the same algorithm as for the cubic B-spline. Table 1 lists the roots and the constant scale factor for several B-splines and o-Moms.

Once the image has been prefiltered $c = (p)^{-1} * v$, interpolation at a point (x,y) is computed as

$$u(x, y) = \sum_{m,n \in \mathbb{Z}} c_{m,n} \varphi(x - m) \varphi(y - n).$$

This second step is essentially the same formula as in the section on nearest neighbor, bilinear, and bicubic, and the same algorithms can be used to perform this step. The only differences are that the prefiltered image c is used in place of v and φ is used in place of K_1.

15.4 Sinc

For sinc interpolation with an integer scale factor, interpolation can be implemented using the fast Fourier transform (FFT). This approach follows from observing that the sinc interpolant is the unique image that is bandlimited and agrees with the input data.

The input image is first padded to twice its size in each dimension with half-symmetric extension and transformed with the FFT. The interpolation is then constructed in the Fourier domain by

Method	$1/a_J$	r_j
β^2	8	$-3+8^{1/2}$
β^3	3!	$3^{1/2}-2$
β^5	5!	$-4.309628820326465 \times 10^{-2}$, $-4.305753470999738 \times 10^{-1}$
β^7	7!	$-9.148694809608277 \times 10^{-3}$, $-1.225546151923267 \times 10^{-1}$, $-5.352804307964382 \times 10^{-1}$
β^9	9!	$-2.121306903180818 \times 10^{-3}$, $-4.322260854048175 \times 10^{-2}$, $-2.017505201931532 \times 10^{-1}$, $-6.079973891686259 \times 10^{-1}$
β^{11}	11!	$-5.105575344465021 \times 10^{-4}$, $-1.666962736623466 \times 10^{-2}$, $-8.975959979371331 \times 10^{-2}$, $-2.721803492947859 \times 10^{-1}$, $-6.612660689007345 \times 10^{-1}$

Method	$1/a_J$	r_j
o-Moms 3	21/4	$(105^{1/2}-13)/8$
o-Moms 5	7920/107	$-7.092571896868541 \times 10^{-2}$, $-4.758127100084396 \times 10^{-1}$
o-Moms 7	675675/346	$-1.976842538386140 \times 10^{-2}$, $-1.557007746773578 \times 10^{-1}$, $-5.685376180022930 \times 10^{-1}$

Table 1: Roots and constant scale factor for several B-splines and o-Moms.

copying the transform coefficients of the padded input image for the lower frequencies and filling zeros for the higher frequencies. The final interpolation is obtained by inverse FFT and removal of the symmetric padding. For real-valued input data, some computational savings can be made by using a real-to-complex FFT and complex-to-real inverse FFT to exploit the complex conjugate redundancy in the transform. The complexity of the interpolation is $O(N \log N)$ for N output pixels.

16 Examples

16.1 Scaling smooth data

The smooth function $\cos((x^2+y^2)/10)$ is sampled on the grid $\{0.5, 1.5, \ldots, 15.5\} \times \{-15.5, -14.5, \ldots, 15.5\}$ and interpolated on the centered grid with scale factor 4. Figure 20 compares the interpolation result using various linear methods, along with the root mean squared errors (RMSE) between the interpolation and the exact image.

For this example, higher-order methods tend to perform better. The o-Moms perform slightly better than the B-splines of the same order. Sinc interpolation yields the best result.

16.2 Scaling piecewise smooth data

We repeat the previous experiment with piecewise smooth data (Figure 21). The image is discontinuous at the edges, so the bandlimited hypothesis of sinc interpolation is severely violated. Sinc interpolation and other higher-order methods produce strong artifacts near edges. The bicubic and Lanczos-2 interpolations have the lowest RMSE in this case.

16.3 Texture mapping

A 3D scene of a plane and a sphere is texture mapped with an image (Figure 22). Texture mapping is an example of domain mapping, where in this case the image domain is mapped to the projected plane and sphere surfaces.

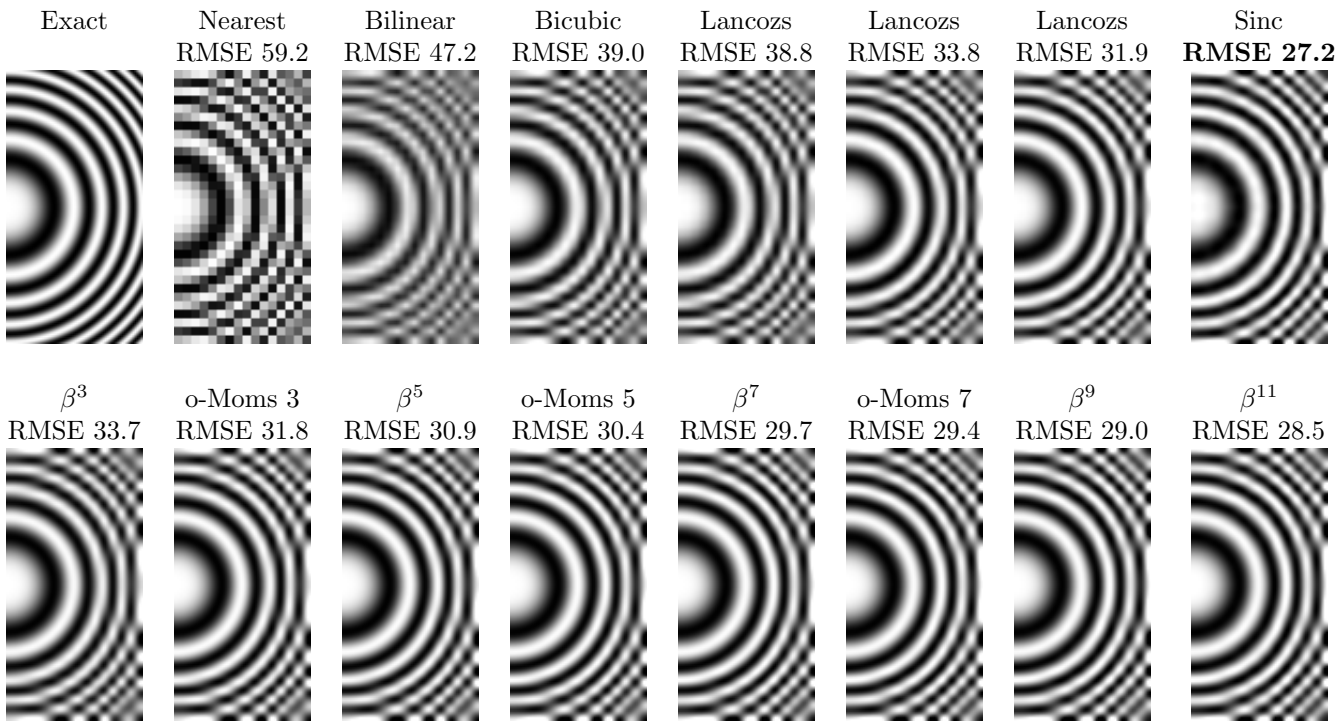

Figure 20: Interpolation examples with smooth data.

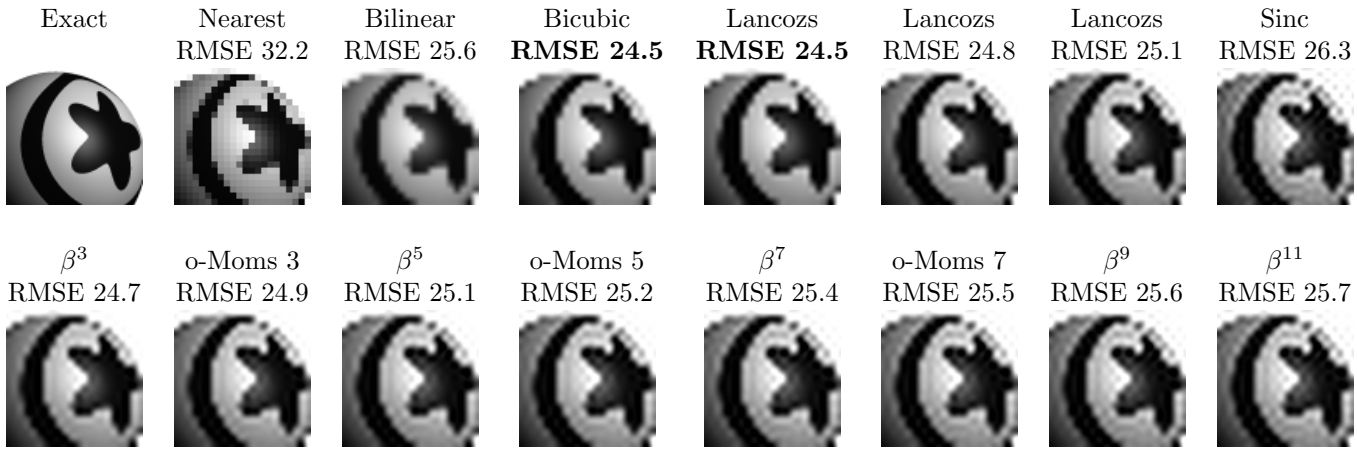

Figure 21: Interpolation examples with piecewise smooth data.

This mapping is derived through reverse ray tracing (see Figure 23). For each pixel of the rendering of the 3D scene, a ray of light is virtually traced in reverse. The light ray begins from the eye of the observer, passes through a pixel of the screen, and continues into the 3D scene. The ray may then intersect the plane or the sphere (or nothing, if the ray continues into the background).

The ray's intersection point $(X, Y,)$ with the 3D scene determines the color of its pixel. The plane is texture mapped as $x = X \mathrm{mod} M$, $y = Z \mathrm{mod} N$. The sphere is texture mapped as $x = \theta$, $y = \varphi$, where θ and φ are the spherical angles of the intersection.

16.4 Image rotation

Image rotation is another application that for a general rotation angle requires interpolation. The choice of interpolation method significantly affects the quality of the output image. In this exper-

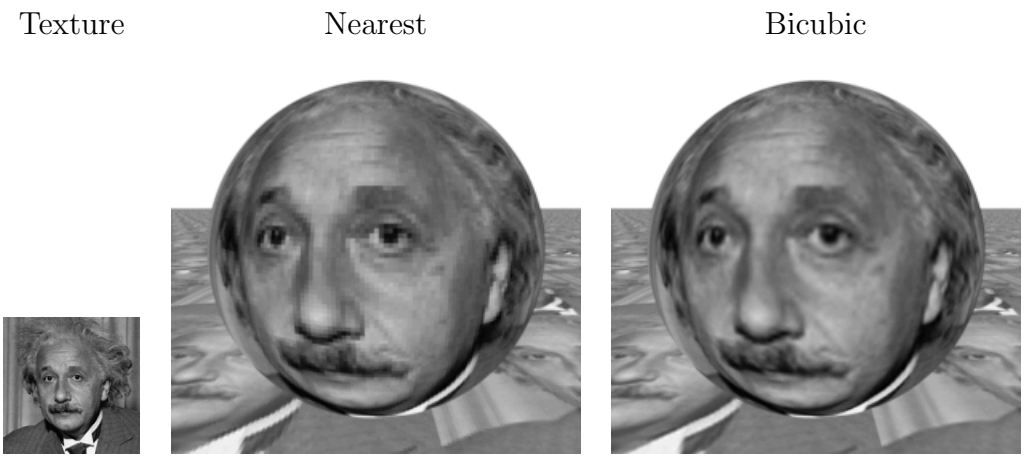

Figure 22: Example of texture mapping.

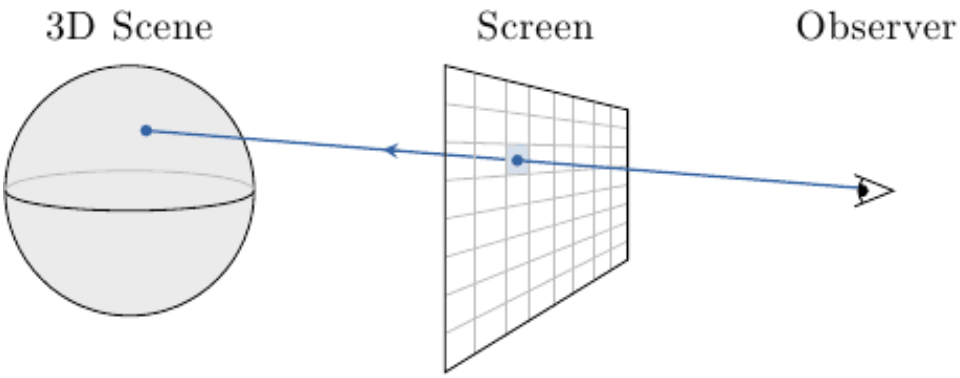

Figure 23: Reverse ray tracing. A ray of light is traced from the observer into the 3D scene.

iment, an image is successively rotated by 5 degrees increments using several linear methods. The initial image is rotated by 5 degrees, then the rotated image is rotated by another 5 degrees, and so on (see results in Figure 24).

Acknowledgment

This material is based upon work supported by the National Science Foundation under Award No.DMS-1004694. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. Work partially supported by the MISS project of Centre National d'Etudes Spatiales, the Office of Naval Research under grant N00014-97-1-0839 and by the European Research Council, advanced grant "Twelve labours".

Image Credits

Pascal Getreuer.

Standard test image.

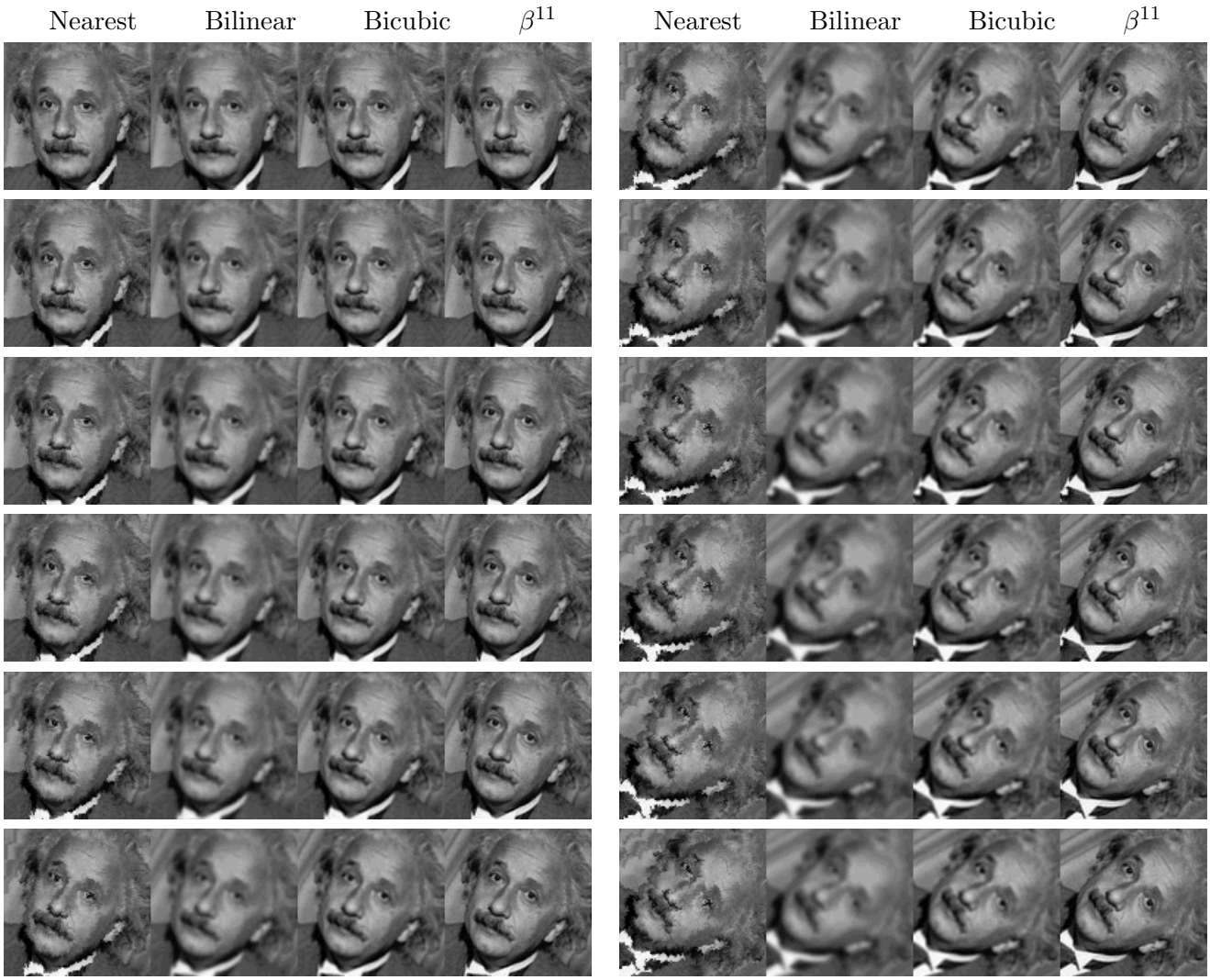

Figure 24: Rotated versions of the same image. Rotation angle increases from top to bottom and from left to right. Rotation using nearest neighbor produces jagged artifacts along edges. Rotation with bilinear is much better but blurs the image. Rotation with bicubic is sharper, but blurring is still noticeable. Finally, the sharpest rotation among these results is with the 11th order B-spline β^{11}. By interpolating with a high-order B-spline, it closely approximates rotation via sinc interpolation.

References

[1] C.E. Shannon, "A Mathematical Theory of Communication", Bell System Technical Journal, vol. 27, pp. 379–423, 623–656, 1948.

[2] G. Strang and G. Fix, "A Fourier analysis of the finite element variational method", Constructive Aspects of Functional Analysis, Edizioni Cremonese, Rome, pp. 795–840, 1973.

[3] R. Keys, "Cubic Convolution Interpolation for Digital Image Processing", IEEE Transactions on Acoustics, Speech, and Signal Processing, 29(6), 1981. http://dx.doi.org/10.1109/TASSP.1981.1163711.

[4] A. Aldroubi, M. Unser, and M. Eden, "Cardinal spline filters: stability and convergence to the ideal sinc interpolator," Signal Processing, vol. 28, no. 2, pp. 127–138, 1992.

[5] A. Schaum, "Theory and design of local interpolators", CVGIP: Graphical Models and Image Processing, vol. 55, no. 6, pp. 464–481, 1993. http://dx.doi.org/10.1006/cgip.1993.1035.

[6] M. Unser, "Splines: A Perfect Fit for Signal/Image Processing", IEEE Signal Processing Magazine, 16(6), pp. 22–38, 1999. http://dx.doi.org/10.1109/79.799930.

[7] P. Thévenaz, T. Blu, and M. Unser, "Interpolation Revisited," IEEE Transactions on Medical Imaging, vol. 19, no. 7, pp. 739–758, 2000.

[8] T. Blu, P. Thévenaz, and M. Unser, "MOMS: Maximal-Order Interpolation of Minimal Support", IEEE Transactions on Image Processing, vol. 10, no. 7, pp. 1069–1080, 2001. http://dx.doi.org/10.1109/83.931101.

[9] E. Meijering, "A Chronology of Interpolation: From Ancient Astronomy to Modern Signal and Image Processing", In Proceedings of the IEEE, 90, pp. 319–342, 2002. http://dx.doi.org/10.1109/5.993400.

[10] F. Malgouyres and F. Guichard, "Edge direction preserving image zooming: A mathematical and numerical analysis", SIAM Journal on Numerical Analysis, 39(1), pp. 1–37, 2002. http://dx.doi.org/10.1137/S0036142999362286.

[11] B. Fornberg, E. Larsson, N. Flyer, "Stable computations with Gaussian radial basis functions", SIAM Journal on Scientific Computing 33, pp. 869–892, 2011. http://dx.doi.org/10.1137/09076756X.