Quadrature Prefiltering for High Quality Antialiasing

BRIAN GUENTER
Microsoft Corp.
and
JACK TUMBLIN
Georgia Institute of Technology

This article introduces quadrature prefiltering, an accurate, efficient, and fairly simple algorithm for prefiltering polygons for scanline rendering. It renders very high quality images at reasonable cost, strongly suppressing aliasing artifacts. For equivalent RMS error, quadrature prefiltering is significantly faster than either uniform or jittered supersampling. Quadrature prefiltering is simple to implement and space-efficient; it needs only a small two-dimensional lookup table, even when computing nonradially symmetric filter kernels. Previous algorithms have required either three-dimensional tables or a restriction to radially symmetric filter kernels. Though only slightly more complicated to implement than the widely used box prefiltering method, quadrature prefiltering can generate images with much less visible aliasing artifacts.

Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation

Additional Key Words and Phrases: Antialiasing, prefiltering

INTRODUCTION

Good spatial antialiasing is computationally expensive, a significant consumer of computing resources when rendering images with very high spatial frequency patterns. Fast, reliable, antialiasing methods are crucial, yet the most popular methods still fail occasionally in annoying and unpredictable ways. They fail spectacularly on a few pathological images, such as checkerboards (Plates 8–10), arrays of very thin, sliver-like polygons, or zone plate patterns (Plates 1–7). Antialiasing failures may also appear in more subtle, pernicious ways; a few images may have a noisy twinkling horizon, a flashing, wobbly-looking, textured region, or some

Authors' addresses: B. Guenter, Microsoft Research, Microsoft Corp., One Microsoft Way, Redmond, WA 98052; J. Tumblin, Georgia Institute of Technology, Atlanta, GA 30332.

visibly noisy edges. We briefly review the cause of these failures and then describe a new algorithm, quadrature prefiltering, as an efficient solution.

Aliasing is a consequence of sampling, a penalty for converting a continuous image to a discrete image. Rendering algorithms create a discrete or pixel image $\mathbf{I}_{\text{pix}}(n_1, n_2)$ by sampling an ideal continuous image $\mathbf{I}_{\text{ideal}}(x, y)$, an abstraction built from geometric primitives such as polygons or polynomial patches. Regard this abstraction as an explicit two-dimensional image intensity function $\mathbf{I}_{\text{ideal}}(x, y)$, with a finite value for all real (x, y). To avoid aliasing when sampling, one must ensure that all significant frequency components of $\mathbf{I}_{\text{ideal}}(x, y)$ are below the Nyquist spatial frequencies (f_x, f_y), defined as exactly half the sampling rates [Oppenheim and Schafer 1989; Mersereau and Dudgeon 1984]. Unfortunately, $\mathbf{I}_{\text{ideal}}(x, y)$ often contains sharp discontinuities at the boundaries of geometric primitives that cause high amplitude spectral components at extremely high frequencies.

Three techniques are now commonly used to reduce aliasing: prefiltering, uniform super sampling, and stochastic sampling. Prefiltering methods change the spectrum of $\mathbf{I}_{\text{ideal}}(x, y)$, attenuating all high-frequency components that may alias by low-pass filtering before sampling at pixel rates. Both supersampling methods leave the spectrum of $\mathbf{I}_{\text{ideal}}(x, y)$ unchanged, but increase the sampling rates (and hence the Nyquist rates) to some small multiple of pixel rates, then form each pixel of $\mathbf{I}_{\text{pix}}(n_1, n_2)$ from a weighted average of neighboring samples. Stochastic supersampling randomly displaces the supersampling positions so that any aliased components appear as uncorrelated noise in $\mathbf{I}_{\text{pix}}(n_1, n_2)$.

Only prefiltering methods can completely eliminate aliasing artifacts, because only prefiltering imposes limits on the spectrum of $\mathbf{I}_{\text{ideal}}(x, y)$ before sampling. Although supersampling methods can often reduce aliasing artifacts to the point of invisibility, they cannot guarantee good results for all images. Because the $\mathbf{I}_{\text{ideal}}(x, y)$ spectrum is unbounded, no supersampling rate is ever high enough to avoid all possible aliasing; small geometric primitives can always squeeze between the supersampling positions and cause aliasing errors. Higher supersampling rates reduce the visibility of aliasing but are costly, with quadratic time complexity as a function of the highest spatial frequency component f in $\mathbf{I}_{\text{ideal}}(x, y)$ that will not be aliased. Stochastic or jittered supersampling methods disperse the aliasing of high-frequency components of $\mathbf{I}_{\text{ideal}}(x, y)$ as noise in $\mathbf{I}_{\text{pix}}(n_1, n_2)$. (See Dippé and Erling [1992] for a good tutorial and bibliography on stochastic sampling.) This noise degrades the final image, and the signal-to-noise ratio grows only as the square root of the number of samples taken at each pixel. High signal-to-noise ratios require high supersampling rates.

Supersampling methods are often far simpler and more flexible than prefiltering methods and this probably accounts for their widespread use. Prefilter methods evaluate continuous area integrals at each pixel, and need detailed boundary information from $\mathbf{I}_{\text{ideal}}(x, y)$ to find the integration limits. Published methods require unoccluded polygonal primitives; that is, the hidden surface problem must be solved completely and not just at sample points in the image (see Catmull [1984] for an efficient algorithm

for solving this problem). Supersampling methods can easily be used on a mixture of complex primitives in $\mathbf{I}_{\text{ideal}}(x, y)$, such as polygons, implicit equations, and parametric surfaces. Computing boundary information directly for nonpolygonal surfaces is a difficult problem and limits the use of prefiltering algorithms primarily to polygonal object representations. However, many commercially available image synthesis programs use scanline algorithms that are well suited for prefiltering, and convert nonpolygonal surface primitives into polygonal approximations before rendering. For these algorithms, prefiltering can provide a level of antialiasing performance only achievable with very high supersampling rates.

Most published prefiltering methods are slow, costly, and complex, so only one simple prefiltering method has found wide acceptance. This special case is known as box filtering or area sampling. This algorithm clips visible polygons in $\mathbf{I}_{\text{ideal}}(x, y)$ to a 1×1 pixel-sized region to form polygon fragments, then sets pixel color as the sum of each fragment's shaded color weighted by its area. This is equivalent to prefiltering $\mathbf{I}_{\text{ideal}}(x, y)$ with a constant, box-shaped, continuous filter kernel $\mathbf{h}(x, y)$ where

$$\text{Box filter: } \mathbf{h}(x, y) = \begin{cases} 1 & \text{for all } -0.5 \le x < 0.5 \text{ and } -0.5 \le y < 0.5 \\ 0 & \text{otherwise} \end{cases}$$

before sampling the result at each integer (x, y).

The box filter performs poorly: its frequency response is $\mathbf{H}(\omega_x, \omega_y) = sinc(\omega_x) \cdot sinc(\omega_y)$, with Nyquist rates at $(\pm 1, \pm 1)$. It blurs the image by attenuating spatial frequencies just below the Nyquist rates, yet still allows significant aliasing because its response shrinks slowly with increasing frequency.

One of the earliest sophisticated prefiltering algorithms [Feibush et al. 1980] finds the convolution of right triangle primitives with a continuous filter kernel $\mathbf{h}(x, y)$ by using a polar-coordinate lookup table. For each pixel, all visible portions of the polygon primitives in $\mathbf{I}_{\text{ideal}}(x, y)$ are first clipped against the rectangular extent of the filter kernel, then subdivided into triangles, then further decomposed into right triangles with a vertex at the pixel center. Decomposing an arbitrary polygon into these oriented right triangles requires a fairly complex set of geometric computations, repeated for each affected pixel. This algorithm requires a two-dimensional lookup table if $\mathbf{h}(x, y)$ is radially symmetric; nonradially symmetric filter kernels require a three-dimensional table, which may be unacceptably large.

Others have extended this early work. Abram et al. [1985] devised a more efficient implementation of the algorithm that avoids the full complexity of the general triangular decomposition except in a few special cases. However, the algorithm is still quite complex and has the same three-dimensional table requirements for nonradially symmetric filter kernels. Catmull [1984] extended the Feibush et al. [1980] algorithm to temporal antialiasing, and developed an efficient algorithm for solving the hidden surface problem at the

pixel level, a necessary step in prefiltering. Another prefiltering algorithm that constrains filter kernels to be radially symmetric is described in Turkowski [1982].

A very elegant analytic prefiltering algorithm using prism splines is described in McCool [1995]. This algorithm does not require lookup tables. However, efficiently evaluating arbitrary filter kernels is computationally expensive and relatively complicated.

The prefiltering algorithm described in this article, quadrature prefiltering, provides several capabilities not found in previous work.

(1) Arbitrary filter kernels are computed with only a two-dimensional lookup table, eliminating the radial symmetry constraint. This is useful, for example, in exploiting the oblique effect in human vision [Essock 1982]. Humans have significantly lower spatial resolution along the diagonal than along the horizontal or vertical axes. By using a prefilter with a diamond-shaped passband and sampling the image on a hexagonal rather than a rectangular lattice, the number of data points needed to sample an image without aliasing artifacts can be reduced by a factor of two. The diamond-shaped prefilter passband required for this has a nonradially symmetric filter kernel.

(2) This algorithm lifts the restriction of constant $\mathbf{I}_{\text{ideal}}(x, y)$ in the region of integration present in many previous prefiltering algorithms, albeit at greater computational cost. For constant $\mathbf{I}_{\text{ideal}}(x, y)$ in the region of integration, the complexity of the new algorithm is $\mathbf{O}(kn)$ where k is the number of edges in the polygon and n is the order of quadrature used. For $\mathbf{I}_{\text{ideal}}(x, y)$ not constant in the region of integration, the new algorithm is $\mathbf{O}(kn^2)$. In practice we have encountered few situations where the nonconstant $\mathbf{I}_{\text{ideal}}(x, y)$ method is necessary.

(3) Quadrature prefiltering does not require the complicated geometric operations of Feibush et al. [1980]. The new algorithm is efficient and simple and the rendering time part of the prefiltering algorithm, as opposed to the table construction portion, can be implemented in a few tens of lines of code.

Quadrature prefiltering is most easily applied to polygonal surface primitives because their boundaries are straight line segments. Other primitives may be used, but curved boundaries increase the complexity. Only the polygonal surface primitive case is discussed here. This should not be a serious limitation because many surface representations can be efficiently converted to approximate polygonal representations at rendering time.

SIMPLIFYING THE CONVOLUTION INTEGRAL

Prefiltering consists of continuous convolution followed by sampling: the ideal image $\mathbf{I}_{\text{ideal}}(x, y)$ is convolved with the filter kernel $\mathbf{h}(x, y)$, then sampled at integer (x, y) to form the discrete pixel output image $\mathbf{I}_{\text{pix}}(n_1, n_2)$.

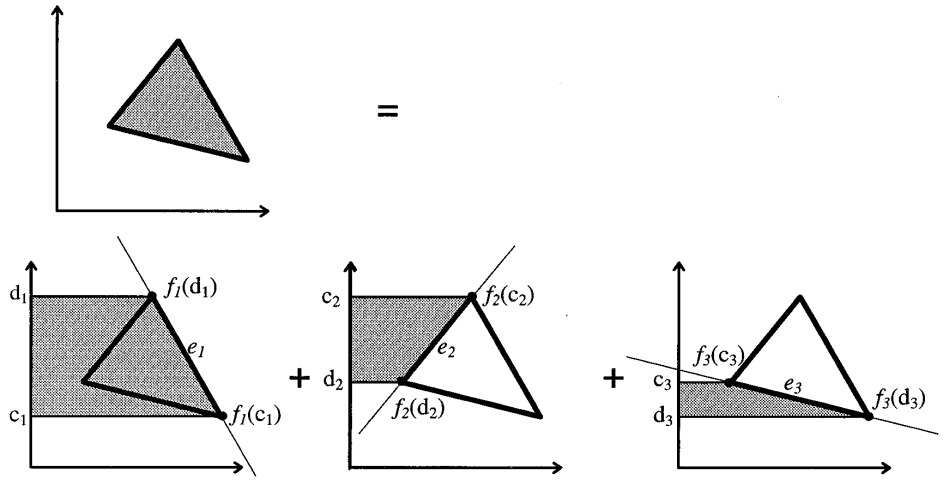

Fig. 1. Decomposition of each polygon integral into edge integrals e_1, e_2, and e_3.

Equivalently, prefiltering is the evaluation of an area integral at each pixel: each pixel in $\mathbf{I}_{\text{pix}}(n_1, n_2)$ equals the volume under the filter kernel $\mathbf{h}(x, y)$ in the area defined by the inverted, displaced ideal image $\mathbf{I}_{\text{ideal}}(n_1 - x, n_2 - y)$.

$$\mathbf{I}_{\text{pix}}(n_1, n_2) = \int\int \mathbf{h}(n_1 - x, n_2 - y)\mathbf{I}_{\text{ideal}}(x, y)\mathrm{d}x\ \mathrm{d}y$$

$$= \int\int \mathbf{h}(x, y)\mathbf{I}_{\text{ideal}}(n_1 - x, n_2 - y)\mathrm{d}x\ \mathrm{d}y. \quad (1)$$

This integral is zero outside the region of support of the filter kernel. We can take advantage of this fact to limit computation by clipping polygons to the rectangular region of support of the filter kernel and only evaluating the integral inside this region.

The convolution integral is difficult to evaluate in this form, so we decompose it into several simpler pieces. First, split it into a sum of individual integrals, one for each visible polygon fragment in $\mathbf{I}_{\text{ideal}}$ that may contribute to the pixel at (n_1, n_2). Then split each fragment's integral into a sum of simpler edge integrals e_i, as shown in Figure 1. Form a trapezoid by projecting each polygon edge to the y axis. Edge integrals integrate within these gray trapezoidal regions; the sum of edge integrals equals the integral within the polygonal region. We evaluate the edge integrals within trapezoidal limits, using the trapezoid extending from an edge of a fragment to the y axis (using the x axis is equally valid). The sum of the edge

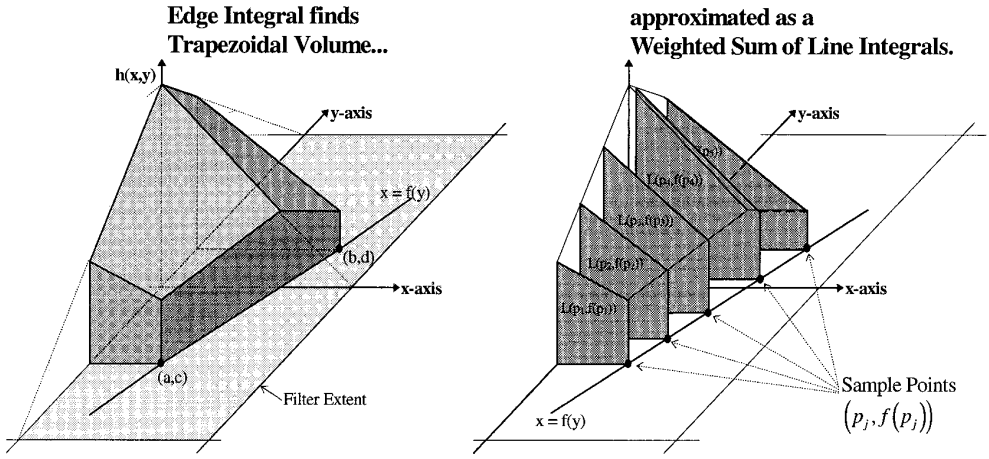

Fig. 2. Approximation of each edge integral as a weighted sum of line integrals. Form a trapezoid by projecting each polygon edge to the y axis. Edge integrals integrate within these gray trapezoidal regions; the sum of edge integrals equals the integral within the polygonal region.

integrals is the integral over the fragment.

$$\mathbf{I}_{\text{pix}}(n_1, n_2) = \sum_i e_i,$$

$$\text{where } e_i = \int_c^d \int_0^{f_i(y)} \mathbf{h}(x, y) \cdot \mathbf{I}_{\text{ideal}}(n_1 - x, n_2 - y) \mathrm{d}x \, \mathrm{d}y. \quad (2)$$

Define the fragment edge endpoints as (a, c) and (b, d). Function $f_i(y) = x$ describes a line through these endpoints, giving the distance from the y axis to the polygon fragment edge at height y, as shown in Figure 2.
The edge integrals are further simplified by assuming that $\mathbf{I}_{\text{ideal}}(x, y)$ is equal to a constant K inside the polygonal fragment's region of integration. The constant value, K, can be computed in a variety of ways. We use the average of the image intensities at the vertices of the polygon fragment. This is not a necessary assumption inasmuch as our algorithm will work with nonconstant $\mathbf{I}_{\text{ideal}}(x, y)$ but with higher computational cost because the image function must be evaluated at more points in the integration interval. Our experience has been that with the relatively small filter kernels required for good antialiasing performance, the constant $\mathbf{I}_{\text{ideal}}(x, y)$ is reasonable for the vast majority of images.

The edge integral with the constant $\mathbf{I}_{\text{ideal}}(x, y)$ assumption is

$$e_i = K \int_c^d \int_0^{f_i(y)} \mathbf{h}(x, y) \mathrm{d}x \, \mathrm{d}y. \quad (3)$$

Even this simplified integral does not generally have a closed form solution, but it can be closely approximated using Gaussian quadrature, a numerical integration method found in standard texts such as Burden and Faires [1989].

GAUSSIAN QUADRATURE APPROXIMATION

Gaussian quadrature is a way to quickly and accurately approximate the definite integral of a smooth function $r(v)$. The solution is a weighted sum of $r(v)$'s value at n carefully chosen "sample" points v_j, using weights w_j. If $r(v)$ is a polynomial of degree $2n - 1$ or less, then the Gaussian quadrature approximation is precise. The simplest form of Gaussian quadrature estimates the definite integral of $r(v)$ over the fixed interval $(-1, 1)$:

$$\int_{-1}^{1} r(v)dv \cong \sum_{j=1}^{n} w_j r(v_j), \tag{4}$$

where the order of quadrature is n, and the points v_j and the weights w_j are constants, as explained and tabulated in the Appendix. Accuracy grows quickly with order of quadrature n; no tested filter showed any improvement beyond $n = 5$.

A linear transformation extends this simple form to integrate $r(v)$ over an arbitrary interval (c, d). Change the variable v to the new variable t using the linear relations

$$t = \frac{2v - (d + c)}{(d - c)}, \quad v = \frac{(d - c)t + (d + c)}{2}, \tag{5}$$

so that any v in the interval (c, d) maps to a value of t in the interval $(-1, 1)$:

$$\int_{c}^{d} r(v)dv = \frac{d - c}{2} \int_{-1}^{1} r\left(\frac{(d - c)t + (d + c)}{2}\right) \mathrm{d}t. \tag{6}$$

Similarly, each sample point v_j in the range $(-1, 1)$ is mapped to a corresponding sample point p_j in the original range (c, d). Combine Equations (4) and (6) for a more general form of Gaussian quadrature:

$$\int_{c}^{d} r(v)dv \cong \frac{d - c}{2} \sum_{j=1}^{n} w_j r(p_j), \qquad \text{where } p_j = \frac{(d - c)v_j + (d + c)}{2}. \tag{7}$$

QUADRATURE PREFILTERING

Now we can use Gaussian quadrature to solve the unwieldy edge integral e_i of Equation (3). Rewrite the inner integral of Equation (3) by defining a line

integral function $\mathbf{L}(x, y)$, which integrates the filter kernel $\mathbf{h}(x, y)$ along a line of constant y between $(0, y)$ and (x, y):

$$e_i = K \int_c^d \mathbf{L}(f_i(y), y)\mathrm{d}y \qquad \text{where } \mathbf{L}(x, y) = \int_0^x \mathbf{h}(x, y)\mathrm{d}x. \tag{8}$$

Approximate this form of the e_i integral using Gaussian quadrature as shown in Equation (7):

$$e_i = K \int_c^d \mathbf{L}(f_i(y), y)dy \cong K\frac{d-c}{2} \sum_{j=1}^{n} w_j \mathbf{L}(f_i(p_j), p_j). \tag{9}$$

This is the final usable form of the edge integral equation. Each e_i integral is approximated as a weighted sum of a few line integrals $\mathbf{L}(f_i(p_j), p_j)$. The extent of the line integral function is no larger than the filter kernel $\mathbf{h}$, and is a two-dimensional function regardless of any asymmetry in $\mathbf{h}$. In Figure 2, each edge integral finds the volume of the filter kernel within trapezoidal limits defined by an edge's endpoints (a, c) and (b, d). Gaussian quadrature in Equation (9) approximates volume as the weighted sum of n line integrals, each spanning a line between a sample point at $(p_j, f(p_j))$ and the y-axis. Place sample points along the line between endpoints (a, c), (b, d), as given in Equation (7).

SPEEDUP WITH LOOKUP TABLES

Directly evaluating Equation (9) is very inefficient even for low orders of quadrature, but lookup tables boost the calculation speed with very little loss of accuracy. We use a line integral table $\mathbf{LT}(s, t)$ to store values of the line integral function $\mathbf{L}(x, y)$ at quantized (x, y) positions:

$$\mathbf{LT}(s, t) = \mathbf{L}(x_s, y_t) = \int_0^{x_s} \mathbf{h}(x, y_t)\mathrm{d}x. \tag{10}$$

To build the **LT** table for the filter kernels used in this article we evaluated the line integral of Equation (10) at every quantized position in the region of support of the filter kernel. We used adaptive numerical integration to get table entries accurate to at least five significant digits.

Quadrature prefiltering has two error sources: inaccuracies in the Gaussian quadrature approximation, and input quantization error in the lookup tables. Gaussian quadrature precision increases rapidly with the order of quadrature, n; its solution is exact for any polynomial of degree $2n + 1$, so even discontinuous kernels, such as the box filter, are well approximated with n as small as 5. Errors due to sampling the line integral function **L** to form the lookup table **LT** are more serious inasmuch as table size grows as the square of the sampling rate. However, total error varies gracefully with

the number of entries in the line integral table **LT**. Empirical tests on several filter kernels showed **LT** tables of 32 × 32 entries per pixel were sufficient for even very challenging images. Larger tables yielded negligible improvements. As we reduced table size below 32 × 32 entries the RMS error gradually increased.

Horizontal interpolation can also improve the accuracy of the **LT** table output. Horizontally adjacent entries in the **LT** store results of line integrals whose limits largely overlap; linearly interpolating between these values can lead to a slight improvement in accuracy, especially for filter kernels with regions of high gradient. Vertical interpolation is not recommended because vertically adjacent entries in the **LT** table store line integrals along nonoverlapping paths.

Computational Complexity

Computational complexity for quadrature prefiltering is quite favorable. Quadrature of order n for each edge integral requires n linear interpolations to find the sample points (symmetry in v_j values can reduce this), n lookups in the **LT** table, and an n-term weighted sum with fixed weights w_j. This is approximately n times as much arithmetic as required for box prefiltering. For a polygon with k edges, computation is $\mathbf{O}(kn)$. As mentioned in the Introduction, quadrature prefiltering can be extended to the case where $\mathbf{I}_{\text{ideal}}(x, y)$ is not assumed constant in the region of integration. For nonconstant $\mathbf{I}_{\text{ideal}}(x, y)$, computing each edge integral takes n^2-evaluations of the filter kernel **h**, efficiently performed with table lookup, n^2 multiplications of these values with $\mathbf{I}_{\text{ideal}}(x, y)$, and $n^2 - 1$ additions. For a polygon with k edges and nonconstant $\mathbf{I}_{\text{ideal}}(x, y)$, complexity is $\mathbf{O}(kn^2)$.

TESTING METHODS

Each prefilter method was tested for antialiasing performance with various renderings of zone plate and checkerboard images. Plates 1–12 show 256 × 256 pixel images magnified by pixel replication to display distinct pixel values. Test images were rendered with either box or sharp spline filter kernels. The sharp spline filter is a 4 × 4 pixel continuous piecewise cubic filter kernel first described by Parker et al. [1983] and exactly equivalent to the parameterized filter of Mitchell and Netravali [1988] with settings B = 0.0, C = 1.0. The sharp spline filter is a good example of the superior antialiasing kernels made practical by quadrature prefiltering. Mitchell and Netravali's piecewise cubic filter is intended for reconstruction filtering (converting sampled data back to a continuous signal), but it also works well for prefiltering. The filter is

—small: 4 × 4 pixel extent;

—smooth: continuous first derivative everywhere;

—adjustable: constants B and C can be chosen for best compromise between aliasing, blur, anisotropy, and ringing;

—separable: the filter kernel $\mathbf{h}(x, y) = k(x) \cdot k(y)$ where

$$k(t) = \begin{cases} (12 - 9B - 6C)|t|^3 + (-18 + 12B + 6C)|t|^2 \\ \quad + (6 - 2B) \quad \text{when } |t| \le 1, \\ (-B - 6C)|t|^3 + (6B + 30C)|t^2| + (-12B - 48C)|t| \\ \quad + (8B + 24C) \quad \text{when } 1 \le |t| \le 2, \\ 0 \quad \text{when } |t| > 2 \end{cases}$$

—exactly equivalent to the sharp spline filter of Parker et al. [1983] when $B = 0.0$ and $C = 1.0$.

The zone plate test pattern is an analytic two-dimensional chirp-like function [Eq. (11)]. It is a circularly symmetric sinusoid whose frequency term is zero at the center and which increases linearly with radial distance to some maximum $f_{\max}$. Its spatial frequency is essentially constant when measured over any small neighborhood, and such neighborhoods contain all orientations of all spatial frequencies between 0 and $f_{\max}$ at uniform amplitude, as shown in Plate 1.

$$\text{zone}(r, \theta) = \left\{ \begin{array}{l} \text{for } r \le r_{\max} = 0.5\left[1 + \cos\left(\pi f_{\max}\left(\dfrac{r}{r_{\max}} \right)^2 \right) \right], \\ \text{else} = 0 \end{array} \right\}. \tag{11}$$

Filtering a zone plate function with a small filter kernel vividly displays the spectral response of the filter. Spatial frequencies strongly attenuated by the filter appear neutral gray, and the distinctive zone plate pattern appears as attenuation weakens, as shown in Plates 3 and 4.

Because our prefilter method does not accept continuous-valued functions when applying the constant $\mathbf{I}_{\text{ideal}}(x, y)$ assumption, Plates 1–7 used a carefully constructed polygonal approximation of the zone plate pattern. Each cycle of the *zone*( ) sinusoid is represented by eight concentric rings of constant intensity that match the integral of the *zone*( ) function over the extent of each ring. Each ring thickness is chosen to minimize intensity deviations between the ring and the *zone*( ) function evaluated at ring boundaries. Each ring consists of enough four- or five-sided polygons that the radial deviation from a perfectly circular ring is always less than one fourth of the ring thickness. The resulting approximation contained 148,000 polygons, and is virtually indistinguishable from the analytic *zone*( ) function in the renderings.

The checkerboard test images (Plates 8–10) show a 3D-perspective view of a planar mesh of square polygons embedded in the x–y plane. Each polygon measures 1.0×1.0, each square intensity is either 0.0 or 1.0, and the most distant row of polygons is at $x = 600$. The images show a 43.6 degree field of view from a point 5 units above the xy origin looking along the $+x$-axis, tilted slightly downwards. The precise viewing parameters are: view reference point (0.0, 0.0, 5.0), view plane normal (−0.951495, 0.0,

Table I. RMS Error for antialiasing methods

Antialiasing Method	Error with respect to best prefiltering result
Nominal Quadrature Prefiltering	−25.82 dB
1 × 1 uniform Supersampling	−5.49 dB
2 × 2 " " "	+1.52** dB
4 × 4 " " "	−19.47 dB
8 × 8 " " "	−22.49 dB
16 × 16 " " "	−25.71 dB
2 × 2 jittered Supersampling	−0.583 dB
4 × 4 " " "	−13.65 dB
8 × 8 " " "	−18.83 dB
16 × 16 " " "	−23.79 dB

** Aliased components had 180 degree phase shift; error was often larger than signal!

0.307665), perspective reference point (0.0, 0.0, 1.25), view up vector (0.0 0.0 1.0), and image plane extent (±0.5, ±0.5).

RESULTS

Error measurements on zone plate images strongly support our claims for improved accuracy and reduced aliasing with quadrature prefiltering. Table I gives error measurements for 10 different renderings of zone plate patterns with $f_{max} = 2.5$ cycles per pixel, all using the sharp spline filter kernel. Note that smaller errors give more negative values in the table. Boldface entries show that nominal quadrature prefiltering ($n = 5$, **LT** table of 32 × 32 entries) has less error than either uniform or jittered supersampling for all supersampling rates up to 256 samples per pixel. More reasonable supersampling rates are much less accurate.

Each of the error measurements in Table I resulted from comparing a test image with a reference image. The reference image was computed by the most accurate method available; we used quadrature prefiltering with an extreme order of quadrature, $n = 10$, and a large **LT** lookup table holding 128 × 128 entries per pixel of filter kernel. We chose these values by computing images with many different n and **LT** table size values, then selecting the smallest values that showed no further image improvements. All images were stored with excess precision, using 24 bits per color component.

Images stored as RGB bytes (8 bits per color component) hid significant amounts of error. Such images showed no significant improvement for n greater than 5, or **LT** tables larger than 32 × 32, hence we consider these values "nominal" quadrature prefiltering.

To compute RMS error, first find the pixel-by-pixel difference between

the reference and test images, and remove its mean value

$$diff(n_1, n_2) = reference(n_1, n_2) - test(n_1, n_2);$$

$$err(n_1, n_2) = diff(n_1, n_2) - \sum \sum diff(n_1, n_2)/(pixel\_count).$$

Then compute the average squared error, and take its $\log_{10}$ multiplied by 10

$$avg = \sum \sum (err(n_1, n_2))^2/(pixel\_count) RMS\_error\_in\_dB = 10 \cdot \log_{10}(\sqrt{avg}).$$

Images stored as RGB bytes (8 bits per color channel) hide a significant amount of error. An error of only 1/254 at every pixel causes −24 dB image error, but can be completely obscured by 8-bit quantizing. We used images with 24-bit integer accuracy per pixel color field so that the error reported in the tables is solely due to filtering—quantization errors are negligible.

Plates 1 and 2 display the zone plate pattern and demonstrate aliasing. Both display a polygonal approximation of a zone plate pattern rendered by point sampling at one sample per pixel. Plate 1 has f_{max} equal to the Nyquist rates, or 0.5 cycles per pixel in both x and y directions; thus the top, bottom, and sides of the pattern contain the highest frequency components that can be point sampled without aliasing. Plate 2 shows a zone plate sampled uniformly at one sample per pixel with f_{max} at 2.5 cycles per pixel, or 5 times the Nyquist rate. Only the center circular pattern is part of the original signal; all others are aliasing errors, and would be uniform gray in an ideal antialiasing scheme. Plates 3–7 show the same zone plate filtered with two types of prefilters, box and sharp spline, and with uniform and jittered sampling.

Plates 3 and 4 display the shortcomings of box prefiltering when compared to the sharp spline filter. Plates 3 and 4 use the same zone plate as Plate 2. Plate 3 was rendered with box prefiltering that removed much of the aliasing energy that dominates Plate 2, but still contains clearly visible aliasing artifacts above the Nyquist rates; these fade slowly with increasing frequency. Plate 4 shows the benefits of the more sophisticated sharp spline filter of Parker et al. [1983] and Mitchell and Netravali [1988], and was rendered with nominal quadrature prefiltering ($n = 5$; 32×32 entry **LT** table). The high frequency aliasing artifacts are dramatically reduced by the filter's superior stop band performance, yet high frequency components below the Nyquist rates are clear and strong.

Plates 5–7 show the aliasing inherent in supersampling methods; compare them with the quadrature prefiltered image in Plate 4. Plate 5 was rendered using uniform supersampling, and jittered supersampling was used in Plate 6. Both use four samples per pixel in a 2×2 pattern, decimated to pixel rate using a discrete version of the sharp spline filter. At frequencies around four times the Nyquist rates, Plate 5 shows strong aliasing errors, which would also appear at all other multiples of 4X the Nyquist rates (8X, 12X, 16X. . .) if the zone plate were larger. Any patho-

Table II. Computation time for prefiltering versus supersampling

Image	Jittered Supersampling			Nominal Prefiltering ($n = 5$, 32×32 **LT** table)	
	number of samples/pixel	time (secs)	RMS error (dB)	time	RMS error (dB)
skull	36	64	−22.093	49	−23.189
checkerboard	100	241	−15.211	115	−15.748

logical images with high energy at any of these spatial frequencies would be badly aliased if rendered with 2×2 supersampling, regardless of the size or quality of the antialiasing filter. The energy of the strong, coherent aliasing patterns shown in Plate 5 is dispersed throughout the spectrum as high-frequency noise by the jittered supersampling used in Plate 6; the amount of noise is surprisingly large when compared to quadrature prefiltering in Plate 4. Plate 7 was rendered using jittered supersampling at 256 samples per pixel. This horribly expensive rendering was at the lowest supersampling rate that would reduce the image RMS error below that of nominal quadrature filtering in Plate 4.

Plates 8–10 compare antialiasing methods on the commonly used checkerboard pattern. The reference image was rendered with the sharp spline filter kernel and quadrature prefiltering ($n = 10$, 128×128 **LT** table). All error measurements are with respect to this reference image. Plate 8 was rendered using box prefiltering; it blurs detail near the horizon yet still allows clear aliasing artifacts. Plate 9 was rendered with jittered supersampling at 16 samples per pixel and the sharp spline filter; it recovers some detail lost in Plate 8, but shows strong noise at the horizon. The RMS error for this image was −8.74 dB. Plate 9(a) was rendered with 4×4 uniform supersampling and the sharp spline filter. The RMS error was also −8.74 dB for this image. Plate 10 was rendered via nominal quadrature prefiltering and the sharp spline filter ($n = 5$, 32×32 **LT** table); it shows fine detail near the horizon without the noise of Plate 9; its RMS error was only −18.65 dB.

The superior antialiasing performance of quadrature prefiltering comes at a surprisingly low computational cost. Plates 11 and 12 show images computed with both nominal quadrature prefiltering and with jittered supersampling, using a supersampling rate high enough to give RMS error very close to that of quadrature prefiltering. The checkerboard in Plate 12 has fewer rows than those of Plates 8–10 to reduce the size of the geometric database enough to eliminate virtual memory paging. This allowed us to achieve more reliable timings. Table II shows the computation times for each set of images using both types of filtering. For both images the RMS error for prefiltering is lower than for supersampling but computation time for prefiltering is still significantly less. All timings were performed on a 90 MHz Pentium processor.

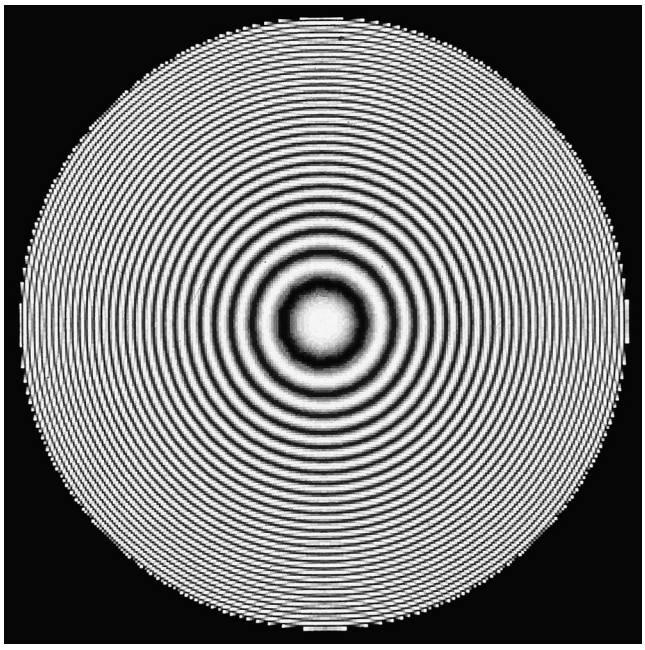

Plate 1. The zone plate pattern. $f_{\max} = .5$ cycles/pixel, point sampled once per pixel.

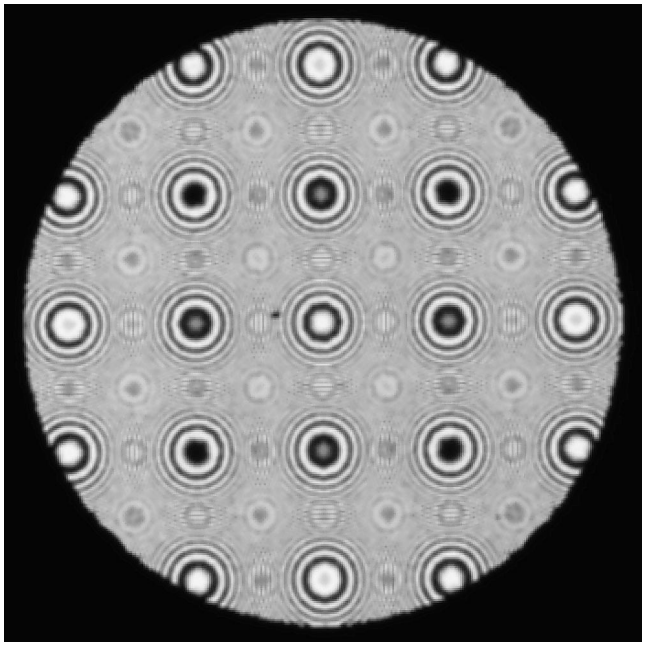

Plate 2. Zone plate aliasing artifacts. Zone plate with $f_{\max}$ of 2.5 cycles/pixel, or 5 times the Nyquist rates, point sampled once per pixel. Only the center circular pattern is part of the original signal; all others are aliasing errors, and would be replaced by uniform gray in an ideal antialiasing scheme.

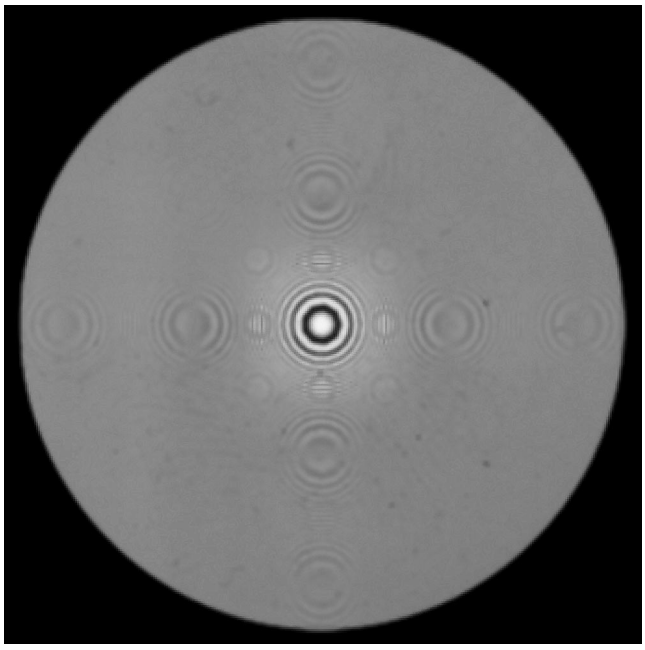

Plate 3. Box prefiltering. Zone plate with f_{max} of 2.5 cycles/pixel rendered with box prefiltering; note poor high frequency response and persistent aliased components.

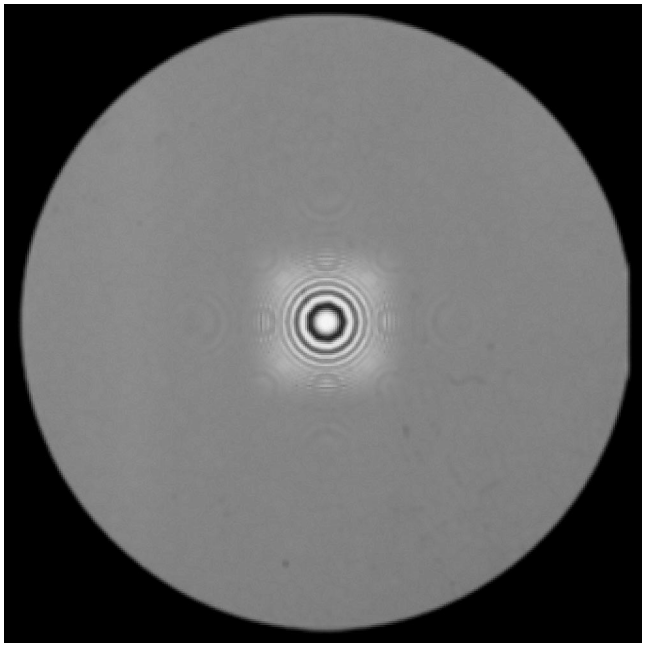

Plate 4. Sharp spline prefiltering. Zone plate with f_{max} of 2.5 cycles/pixel, nominal quadrature prefiltering (n = 5, 32 × 32 **LT** table) with sharp spline prefilter; note strong high frequency response and strong aliasing suppression.

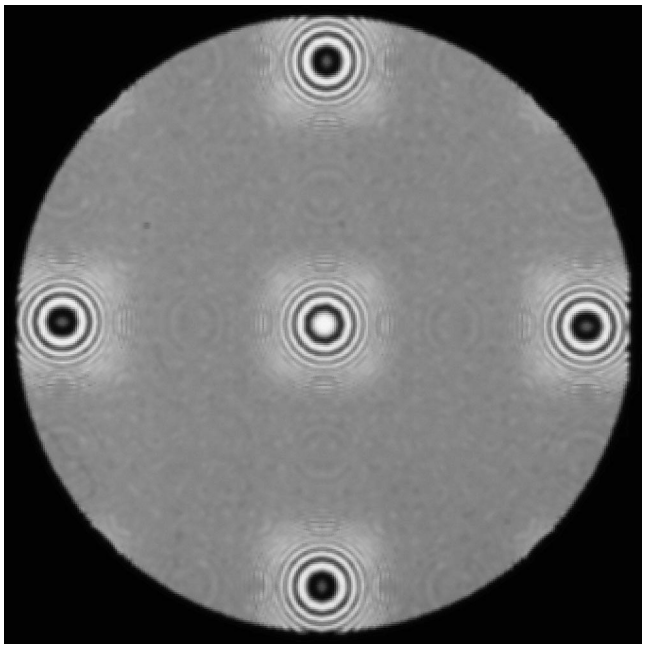

Plate 5. Aliasing in uniform supersampling (4 samples/pixel). Zone plate with f_{max} = 2.5 cycles/pixel, rendered with uniform supersampling and discrete sharp spline filter, allows strong aliasing of all frequencies near any multiple of 2.0 cycles/pixel.

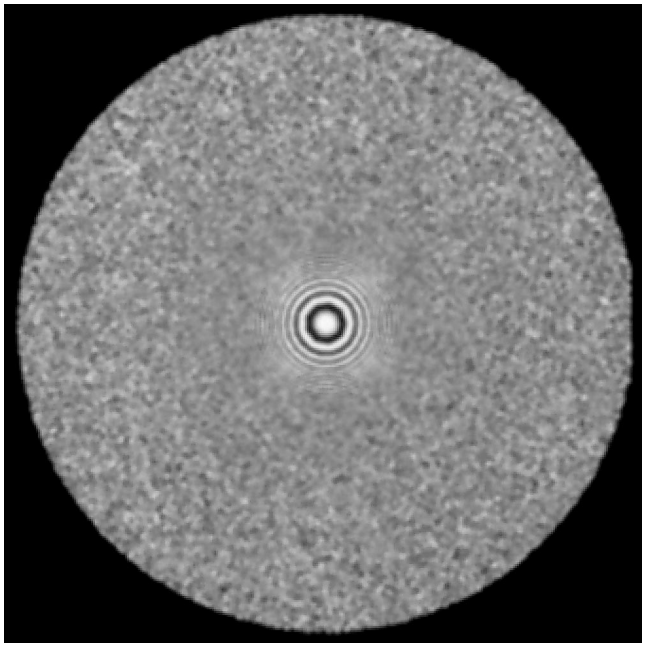

Plate 6. Aliasing in jittered supersampling (4 samples/pixel). Zone plate with f_{max} = 2.5 cycles/pixel, rendered with jittered supersampling and discrete sharp spline filter converts aliasing energy in Plate 5 to high frequency noise.

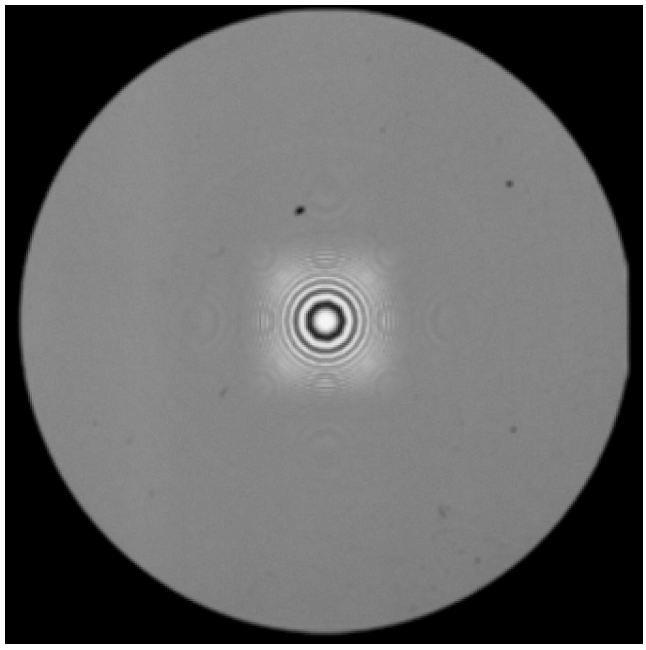

Plate 7. Jittered supersampling (256 samples/pixel). Zone plate with $f_{max} = 2.5$ cycles/pixel; this matches the SNR of quadrature prefiltering in Plate 4.

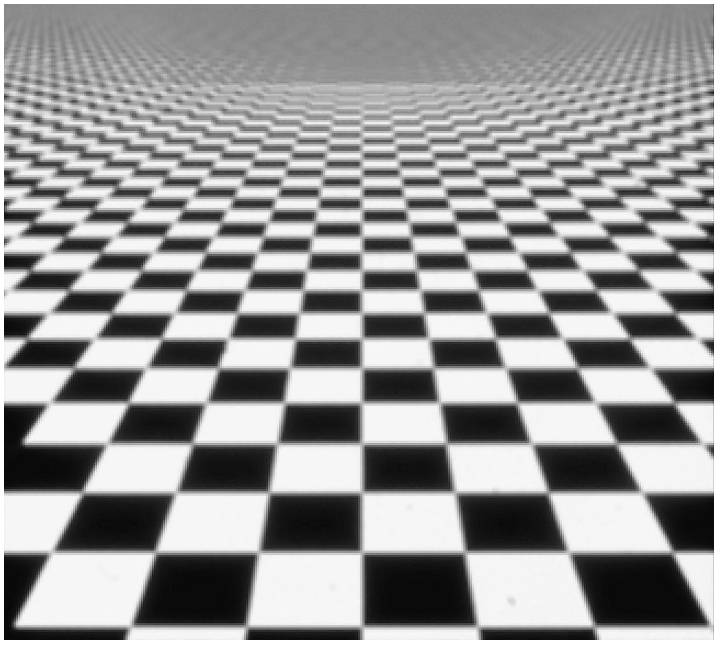

Plate 8. Box prefiltering a checkerboard. Box prefiltering causes aliasing and blurring at horizon.

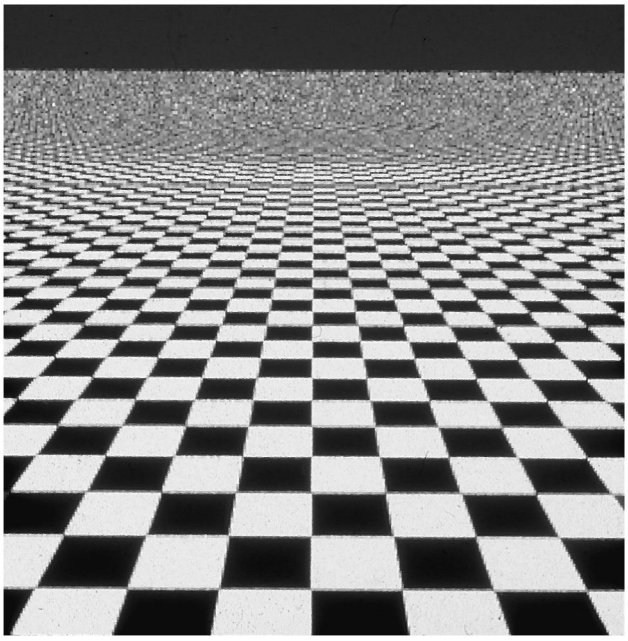

Plate 9. Jittered supersampling (16 samples/pixel).

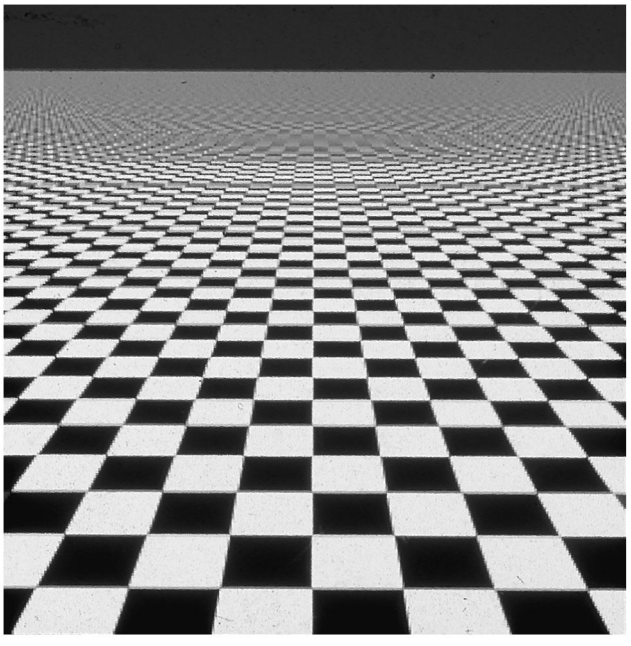

Plate 9a. Uniform sampling (16 samples/pixel). 4 × 4 uniform supersampling with sharp spline filter.

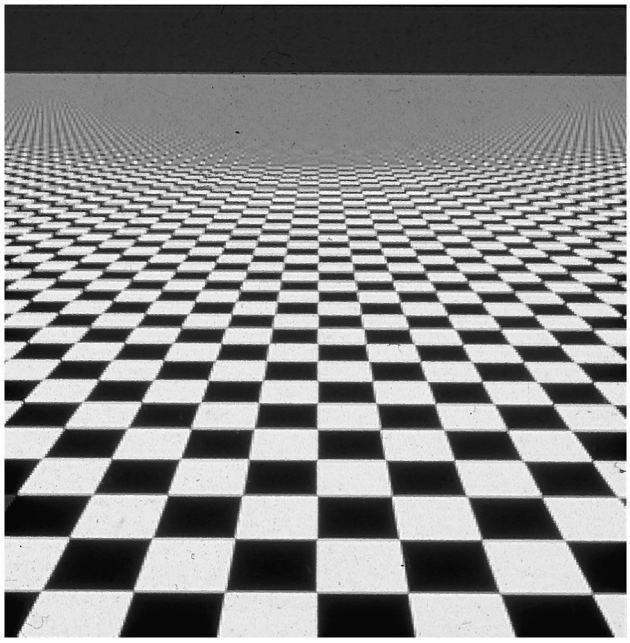

Plate 10. Quadrature prefiltering. Nominal quadrature prefiltering ($n = 5$, 32×32 LT table) with sharp spline filter is sharper and less noisy than either jitter or uniform supersampling.

Plate 11. Skull image used in timing comparisons of prefiltering and jittered supersampling.

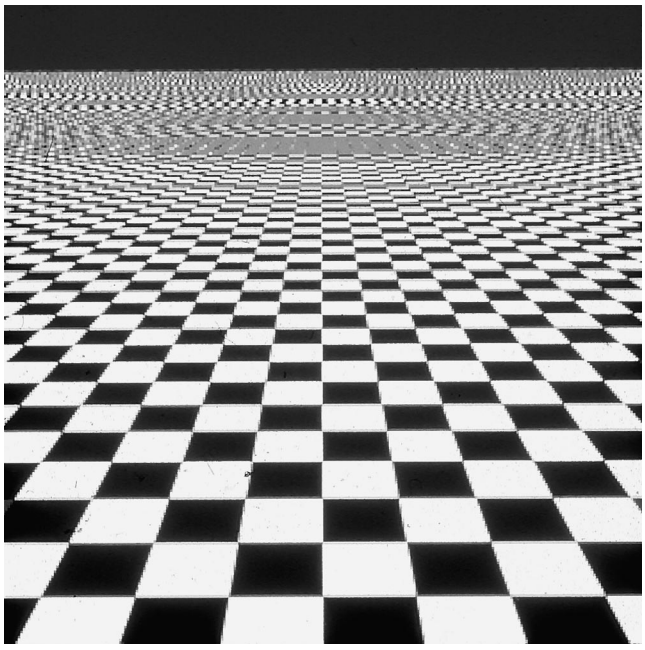

Plate 12. Checkboard image. Used in timing comparisons of prefiltering and jittered supersampling. This checkerboard image has fewer rows than the checkerboard image of Plates 8 through 10.

CONCLUSION

Quadrature prefiltering is an efficient, robust, and relatively simple new prefiltering algorithm. It allows arbitrary filter kernels but only requires a two-dimensional lookup table, and is only slightly more complicated to implement than box prefiltering. Quadrature prefiltering with a good filter kernel dramatically reduces aliasing artifacts. Very high supersampling rates are necessary to match the performance of quadrature prefiltering. Computation times for quadrature prefiltering are significantly faster than for supersampling at rates high enough to give comparable aliasing artifacts.

APPENDIX: CONSTANTS FOR GAUSSIAN QUADRATURE

Gaussian quadrature of degree n approximates the solution of a definite integral of any smooth function $r(v)$ on the interval $(-1, 1)$ as the weighted sum of the values of $r(v)$ at n sample points v_j using weights w_j. These sample points and weights are constants, and the approximation is exact if

Table III. Gaussian quadrature points and weights

Degree n	Points v_j	Weights w_j
2	0.5773502692	1.0
	−0.5773502692	1.0
3	0.7745966692	0.5555555556
	0.0	0.8888888889
	−0.7745966692	0.5555555556
4	0.8611363116	0.3478548451
	0.3399810436	0.6521451549
	−0.3399810436	0.6521451549
	−0.8611363116	0.3478548451
5	0.9061798459	0.2369268850
	0.5384693101	0.4786286705
	0.0	0.5688888889
	−0.5384693101	0.4786286705
	−0.9061798459	0.2369268850
Degree n	Points v_j	Weights w_j
10	0.97939065285	0.0666713443
	0.8650633666	0.1494513491
	0.6794095682	0.2190863625
	0.4339539410	0.2692667193
	0.1488743389	0.2955242247
	−0.1488743389	0.2955242247
	−0.4339539410	0.2692667193
	−0.6794095682	0.2190863625
	−0.8650633666	0.1494513491
	−0.97939065285	0.0666713443

$r(v)$ is a polynomial of degree $2n - 1$ or less:

$$\int_{-1}^{1} r(v)\mathrm{d}v \cong \sum_{j=1}^{n} w_j r(v_j). \tag{12}$$

The n sample points v_j are the roots of the nth Legendre polynomial $P_n(v)$, which is generated recursively:

$$P_0(v) = 0; \quad P_1(v) = v;$$

$$\cdots P_{n+2}(v) = \frac{2n+3}{n+2} vP_{n+1}(v) - \frac{n+1}{n+2} P_n(v); \quad \cdots, n \geq 0. \tag{13}$$

The n weights w_j are also found from these roots v_j by solving

$$w_j = \int_{-1}^{1} \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{(v - v_k)}{(v_j - v_k)} \mathrm{d}v. \tag{14}$$

These values are extensively tabulated and published; the values used to render our images are listed in Table III. The odd symmetry of v_j values and even symmetry of w_j values are helpful in implementation; also note that w_j values sum to 2.0. See Burden and Faires [1989] for a detailed tutorial, and see Stroud and Secrest [1966] for more extensive tables.

REFERENCES

ABRAM, G., WESTOVER, L., AND WHITTED, T. 1985. Efficient alias-free rendering using bit-masks and look-up tables. *ACM Comput. Graph. 19*, 3; *SIGGRAPH 85 Conference Proceedings* (July), 53–60.

BURDEN, R. L. AND FAIRES, J. D. 1989. *Numerical Analysis*, 4th. ed., PWS-Kent Publishing, Boston, MA, 200–203.

CASSELS, J. W. S. 1971. *An Introduction to the Geometry of Numbers*. Springer-Verlag, New York.

CATMULL, E. 1984. An analytic visible surface algorithm for independent pixel processing. *ACM Comput. Graph. 18*, 3; *SIGGRAPH 1984 Conference Proceedings* (July), 109–116.

DIPPE, M. A. Z. AND ERLING, H. W. 1992. Stochastic sampling: Theory and application. In *Progress in Computer Graphics, Vol. 1*. Ablex Publishing, Norwood, NJ.

ESSOCK, E. A. 1982. Anisotropies of perceived contrast and detection speed. *Vision Res. 22*, 1185–1191.

FEIBUSH, E., LEVOY, M., AND COOK, R. 1980. Synthetic texturing using digital filtering. *ACM Comput. Graph. 14*, 3; *SIGGRAPH 1980 Conference Proceedings* (July), 294–301.

MCCOOL, M.D. 1995. Analytic antialiasing with prism splines. In *ACM Comput. Graph; SIGGRAPH 1995 Conference Proceedings* 429–436.

MERSEREAU, R. M. AND DUDGEON, D. E. 1984. *Multidimensional Digital Signal Processing*. Prentice Hall, Englewood Cliffs, NJ.

MITCHELL, D. P. AND NETRAVALI, A. N. 1988. Reconstruction filters in computer graphics. *ACM Comput. Graph. 22*, 4 (Aug.); *SIGGRAPH 88 Conference Proceedings*, 221–228.

OPPENHEIM, A. AND SCHAFER, R. 1989. *Discrete Time Signal Processing*. Prentice Hall, Englewood Cliffs, NJ, 82–91.

PARKER, A., KENYON, R., AND TROXEL, D. 1983. Comparison of interpolating methods for image resampling. *IEEE Trans. Medical Imaging MI-2*, 1 (March), 31–39.

STROUD, A. H. AND SECREST, D. 1966. *Gaussian Quadrature Formulas*. Prentice-Hall, Englewood Cliffs, NJ.

TURKOWSKI, K. 1982. Antialiasing through the use of coordinate transformations. *ACM Trans. Graph. 1*, 3 (July).

Received July 1993; revised March 1996; accepted April 1996