Anti-aliasing with Stratified B-spline Filters of Arbitrary Order

MANUEL N. GAMITO and STEVE C. MADDOCK
Department of Computer Science
The University of Sheffield

A simple and elegant method is presented to perform anti-aliasing in computer-generated images. The method uses stratified sampling to reduce the occurrence of artifacts in the image and features a B-spline filter, of some desired order, to compute the final luminous intensity at each pixel. The method is scalable through the specification of the filter order. A B-spline filter of order one amounts to a simple anti-aliasing scheme with box filtering. Increasing the order of the B-spline generates progressively smoother filters. Computation of the filter values is done in a recursive way, as part of a sequence of Newton-Raphson iterations, to obtain the optimal sample positions in screen space. The method is currently used to anti-alias images generated by ray casting of implicit procedural surfaces.

Categories and Subject Descriptors: G.1.5 [**Numerical Analysis**]: Roots of Nonlinear Equations—*Iterative Methods*; G.2.1 [**Numerical Analysis**]: Combinatorics—*Recurrences and Difference Equations*; G.3 [**Probability and Statistics**]: Probabilistic Algorithms and Random number generation; I.3.3 [**Computer Graphics**]: Picture/Image Generation—*Antialiasing*

General Terms: Algorithms, Design

Additional Key Words and Phrases: Anti-aliasing, B-spline filter, ray tracing, stratified sampling

1. INTRODUCTION

Anti-aliasing is an important component in the rendering of computer-generated images and is used to eliminate high frequencies that would otherwise show up as objectionable image artifacts. The most common approach to anti-aliasing relies on the concept of *super-sampling*, where several image samples are rendered per pixel. The final pixel colour is averaged down from the computed luminous intensities of all the samples [Glassner 1995].

Super-sampling, by itself, pushes coherent artifacts into higher frequencies and makes them less noticeable but cannot eliminate aliasing completely. An improvement was introduced by Cook with the concept of *stratified sampling* [Cook 1989]. In stratified sampling, the position of each sample is slightly perturbed from its original position on the node of a regular sampling grid. This has the advantage that those high frequencies still remaining in the image now become disguised as noise, which, as the authors argue, is much less objectionable to the human visual system than coherent aliasing artifacts.

Anti-aliasing can be further enhanced with a low-pass filter. Instead of performing a simple arithmetic averaging to compute the pixel colour from the colour of

Author's emails: M.Gamito@dcs.shef.ac.uk, S.Maddock@dcs.shef.ac.uk.
Author's address: Department of Computer Science, Regent Court 211 Portobello Street, Sheffield, S1 4DP, United Kingdom.

all neighbouring samples, a weighted average is taken, where the weight for each sample is given by the filter and usually depends on the distance, in screen space, between the sample and the pixel position.

This article will present a method for performing stratified anti-aliasing on computer generated images with a low-pass filter chosen from a family of B-spline basis functions. A similar approach, has been presented by Stark et al. [2005]. We consider, however, our approach to be more elegant, simpler to implement, and also more general, since Stark et al. present the solution for B-spline filters of up to order four (cubic B-splines) whereas our method can work with any filter order. By choosing the order of the B-spline filter, one can have different filter behaviours for anti-aliasing, starting with the box filter and going through the tent and cubic filters.

Section 2 presents a more detailed formulation of the anti-aliasing problem. Section 3 introduces some properties of B-spline basis functions that are necessary for this work. Section 4 presents our anti-aliasing method. Section 5 presents some results, with Section 6 presenting conclusions.

2. STRATIFIED MONTE CARLO ANTI-ALIASING

We seek to compute the luminous intensity $I(\mathbf{x})$, for each point $\mathbf{x}$ in screen space, in such a way that high frequencies are removed and do not cause aliasing when $I(\mathbf{x})$ is regularly sampled onto the discrete pixel positions. Removal of high frequencies from a signal is possible by convolving it with some appropriate low-pass filter $h(\mathbf{x})$. The anti-aliased intensity $I'(\mathbf{x})$ becomes:

$$I'(\mathbf{x}) = \int I(\mathbf{u})h(\mathbf{x}-\mathbf{u})\,d\mathbf{u} \tag{1}$$

Anti-aliasing algorithms for computer graphics always try to provide some numerical approximation to this integral that is both accurate and not too expensive to compute. The simplest approximation is to replace the integral by a summation over several image positions:

$$I'(\mathbf{x}) \approx \sum_i I(\mathbf{u}_i)h(\mathbf{x}-\mathbf{u}_i) \tag{2}$$

The samples $\mathbf{u}_i$ are regularly placed around point $\mathbf{x}$, with N samples along each of the horizontal and vertical directions, for a total of N^2 intensity computations necessary to obtain a single pixel intensity. This amounts to performing a weighted average of all the computed luminous intensities, where the weights are obtained from the filter kernel.

The approach, expressed by (2), is not particularly efficient because it does not take into account the influence of the filter when placing the samples. The samples $\mathbf{u}_i$ are regularly spaced around $\mathbf{x}$, regardless of which values the filter will take there. It can happen, and usually does, that many samples will be placed in regions farthest from $\mathbf{x}$ where the values of $h(\mathbf{x}-\mathbf{u}_i)$ will be small[1] These farthest

[1]Any reasonable low-pass filter will have a kernel function with a maximum value at the origin that decreases monotonically with distance.

samples will have a negligible impact on the anti-aliased intensity $I'(\mathbf{x})$, due to the small value of their weights during averaging. This can be seen as a major source of inefficiency when one remembers that computation of the luminous intensities $I(\mathbf{u}_i)$, with ray tracing or some other rendering algorithm, is always an expensive procedure.

A better approach than (2) is to numerically compute the anti-aliasing convolution (1) with Monte Carlo integration [Shirley and Morley 2003]. In Monte Carlo integration, a simple arithmetic average is used to compute the filtered intensity value:

$$I'(\mathbf{x}) \approx \frac{1}{N^2} \sum_i I(\mathbf{u}_i) \tag{3}$$

The trick with Monte Carlo is that the positions $\mathbf{u}_i$ are taken to be the outcome of a random variable, whose probability density function (PDF) is the low-pass filter itself. There is now a very low probability that samples will be placed in regions where $h(\mathbf{x}-\mathbf{u}_i)$ is small. By treating the filter as the PDF of some random process, we are assured that samples will be placed in regions where they are more likely to contribute to the anti-aliased intensity. We must now deal with the issue of how to randomly place samples in screen space, according to some arbitrary probability density $h(\mathbf{x})$.

First we consider $h(x)$ itself. It must obey certain constraints if it is to be a valid PDF. The most obvious constraint is that $h(\mathbf{x})$ must be strictly positive, since it does not make sense to talk about negative probabilities. Secondly, the integral $\int_\infty h(\mathbf{x})d\mathbf{x}$ must be unity. This amounts to saying that there is a probability of one (a certainty) that a sample will be located somewhere in screen space. If this later constraint does not hold but the filter still has a finite positive integral, it is possible to obtain a normalized PDF by using $h(\mathbf{x})/\int_\infty h(\mathbf{x})d\mathbf{x}$ as the filter kernel.

With a valid PDF, we can now obtain another important measure for a random process, the cumulative density function (CDF). If $h(\mathbf{x})$ is a PDF, then its CDF is given by:

$$H(\mathbf{x}) = \int_{-\infty}^{\mathbf{x}} h(\mathbf{t})\, d\mathbf{t} \tag{4}$$

The CDF $H(\mathbf{x})$ gives us the probability that the random variable will take values below $\mathbf{x}$. In the case of two-dimensional random variables, as is the case in this work, it gives us the probability that a screen sample will be inside the rectangle that has a lower left vertex at minus infinity, in both horizontal and vertical coordinates, and an upper right vertex at point $\mathbf{x}$. If $h(\mathbf{x})$ is a proper PDF, then we have again $H(+\infty) = 1$, meaning the sample is bound to be somewhere on the screen.

To numerically compute a sample from a random process, knowing its CDF, we use the method of *function inversion* [Press et al. 1992]. We begin by generating a uniform random variable $\mathbf{y}$ that takes values in the unit rectangle $[0,1[\times[0,1[$. The sample having our desired CDF, $H(\mathbf{x})$, and PDF, $h(\mathbf{x})$, is now obtained by finding the solution to the equation:

$$\mathbf{y} = H(\mathbf{x}) \tag{5}$$

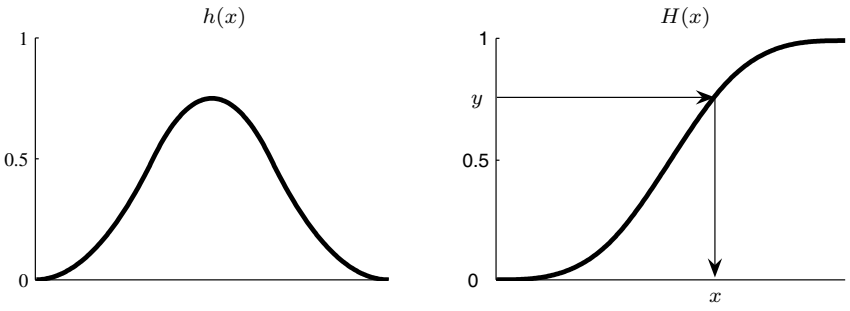

Fig. 1. A probability density function (left) and its corresponding cumulative density function (right). To obtain a sample x with this PDF, sample uniformly with y and obtain x by inverting the CDF.

Figure 1 exemplifies this process for a typical low-pass filter in one dimension. It is possible to see that $H(\mathbf{x})$ is a monotonically increasing function from 0 to 1. This is true of any CDF and it is a property that will be used to advantage in Section 4.

To distribute sample points around the point $\mathbf{x}$ for Monte Carlo integration we create a regular grid of samples, not in screen space, but in the unit square space of the $\mathbf{y}$ variable. These samples become stratified by the addition of a random component $\boldsymbol{\xi}$ that breaks up the regularity of the grid:

$$\mathbf{v}_i = \Delta\mathbf{v}\, i + \boldsymbol{\xi} \tag{6}$$

If we use N samples, along both the horizontal and vertical coordinates, then the vector $\Delta\mathbf{v}$ has equal components $1/N$ and the random vector $\boldsymbol{\xi}$ has components taking uniform random values in the interval $[0, 1/N[$. We thus obtain a stratified distribution of samples $\mathbf{u}_i$ to use in the Monte Carlo approximation (3) to the anti-aliasing integral. The samples obey a PDF equal to the filter $h(\mathbf{x})$, by having them be the solution of $\mathbf{v}_i = H(\mathbf{u}_i)$ for all i.

3. A FAMILY OF CARDINAL B-SPLINES

A family of cardinal B-spline bases $n_m(x)$, with knots placed at integer positions, can be obtained by performing consecutive convolutions with the characteristic function on the interval $[0, 1[$. We start off with the B-spline of order one, which is the aforementioned characteristic function itself:

$$n_1(x) = \begin{cases} 1 & \text{if } x \in [0, 1[, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

The convolution of a B-spline of order $m-1$ with the characteristic function gives the next B-spline in the sequence:

$$n_m(x) \;=\; (n_{m-1} * n_1)(x) \;=\; \int_0^1 n_{m-1}(x-t)\, dt, \quad \text{for } m > 1. \tag{8}$$

Because of the consecutive convolutions, the B-spline curves become progressively smoother as m increases.

We state, in this section, properties of the family of B-spline functions that make them particularly attractive for use as low-pass filters in anti-aliasing. Most of this material is taken from Chui [1992] without demonstration and the reader is invited to consult that work for a thorough treatment of the subject. The properties of B-spline bases that are relevant for anti-aliasing are as follows:

$$n_m(x) > 0, \quad \text{for } 0 < x < m. \tag{9a}$$

$$\int_{-\infty}^{+\infty} n_m(x)\, dx \;=\; 1, \quad \text{for all } m. \tag{9b}$$

$$\operatorname{supp} n_m \;=\; [0, m]. \tag{9c}$$

$$n_m(x) \;=\; \frac{x}{m-1} n_{m-1}(x) \;+\; \frac{m-x}{m-1} n_{m-1}(x-1), \quad \text{for } m > 1. \tag{9d}$$

Properties (9a) and (9b) together tell us that any $n_m(x)$ is a valid probability density function for Monte Carlo anti-aliasing. The support of a B-spline $n_m(x)$, when defined according to (8), is the interval $[0, m]$. For anti-aliasing, however, we require that random samples be centered around some desired pixel position. This can be achieved for B-spline filters by generating the samples, as explained in the previous section, and performing a simple offset of $-m/2$ along the horizontal and vertical coordinates. Property (9d) is the most important. It gives us an algebraic relation between the B-spline of order m and the B-spline of order $m-1$. With this knowledge and with knowledge of the shape of $n_1(x)$ (7) we can compute $n_m(x)$ recursively, at any point x and for any order m.

We now know that B-splines can be used as filters for anti-aliasing and that a simple recursive procedure exists to evaluate them. To study the behaviour of B-splines as low-pass filters, we must also study their spectra $\hat{n}_m(f)$, as given by the application of the Fourier transform to $n_m(x)$:

$$\hat{n}_m(f) \;=\; \mathcal{F}\{n_m(x)\} \;=\; \left(\frac{\sin \pi f}{\pi f}\right)^m e^{-i\pi f m/2} \tag{10}$$

If we admit a unit distance between pixels on the screen, we then have a sampling frequency of 1 Hz, along the horizontal and vertical directions. The Nyquist Sampling Theorem tells us that we must filter out all frequencies above 0.5 Hz if no aliasing is to occur. Ideally, we would like to have a perfect low-pass filter with a sharp frequency cut-off at 0.5 Hz. Such an ideal filter, however, would have an infinite support and would be intractable under any of the approximations to the anti-aliasing integral. The family of B-spline basis functions provides a sequence of approximations to this ideal low-pass filter.

Figure 2 shows the shape of four B-splines with orders of 1, 2, 4 and 20, on the left, together with the modulus of their respective spectra, on the right. The spectrum of the ideal low-pass filter for anti-aliasing has been superimposed as a dashed rectangle. The basis $n_1(x)$ originates the well-known box filter for anti-aliasing. It is quite simple to implement but it allows too many high frequencies to pass through, as evidenced by the significant lobes that fall outside of the spectrum for the perfect filter.

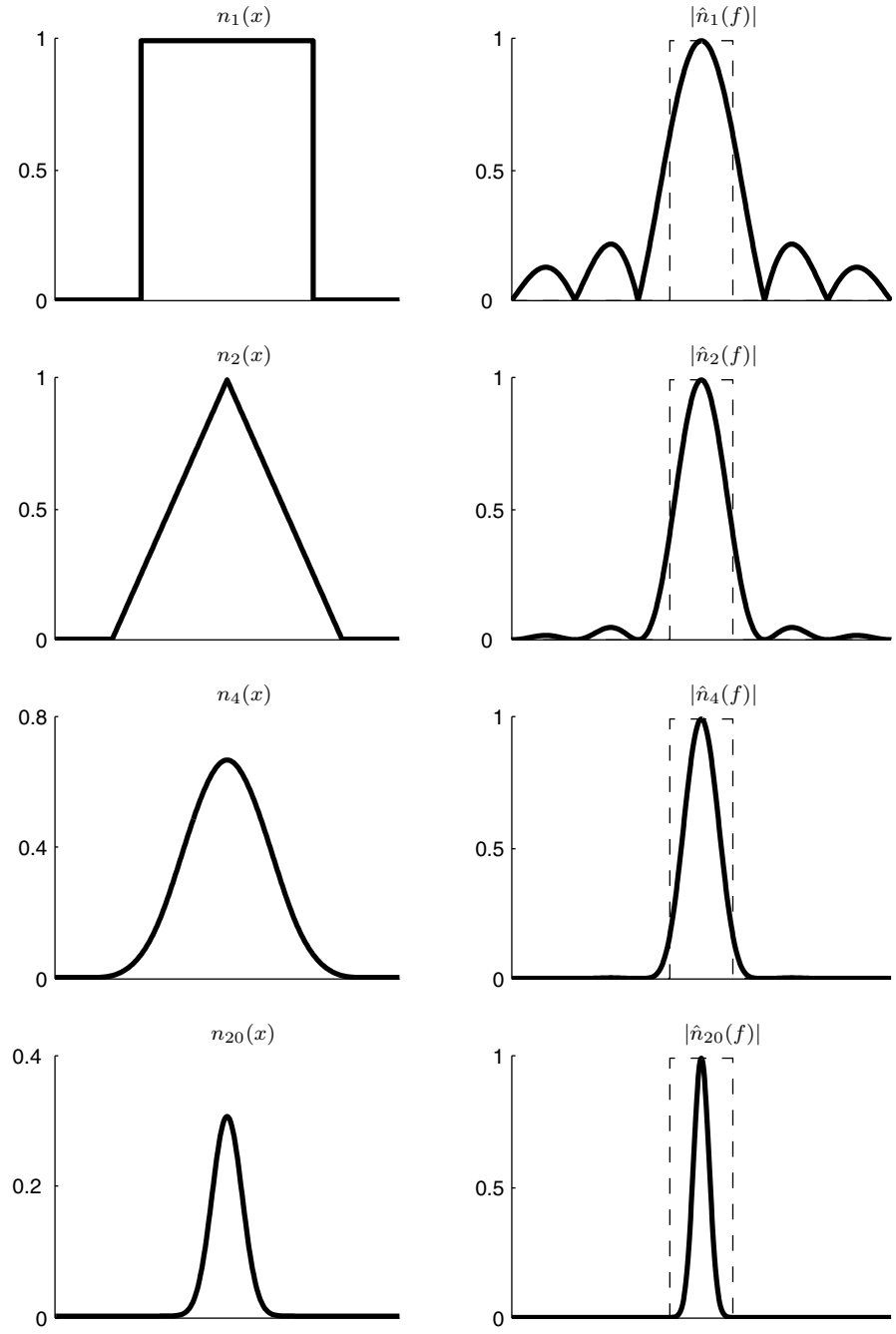

Fig. 2. B-spline basis functions (on the left), for m equal to 1, 2, 4 and 20, and their respective spectra (on the right). The spectrum for the ideal anti-aliasing filter is superimposed, as a dashed rectangle, on the later.

The basis $n_2(x)$ is also known as the tent or triangle filter because of its shape. It has a better spectral behaviour than the box filter, since the lateral spectral lobes are now more attenuated. However, the trend of increasing m in order to block more of the high frequencies cannot continue indefinitely. For too large a value of m, the low frequencies also become excessively attenuated. This is exemplified by the B-spline filter of order 20 in Figure 2. Visually, this distortion in the low frequencies translates into a blurry image, where the amount of blurring is far greater than necessary to eliminate aliasing. It is generally considered that a good compromise between blocking the high frequencies and not distorting the low frequencies is achieved with the cubic B-spline filter $n_4(x)$, also shown in Figure 2.

Once a particular order for the B-spline is chosen, a two-dimensional anti-aliasing filter is made from the cartesian product of the $n_m(x)$ kernel with itself:

$$h(\mathbf{x}) \;=\; h(x_1, x_2) \;=\; n_m(x_1) n_m(x_2) \tag{11}$$

It would be equally as simple to have different orders for the horizontal and vertical kernels but there does not seem to be, however, any significant advantage in doing so.

4. STRATIFIED ANTI-ALIASING WITH B-SPLINES

Stratified Monte Carlo anti-aliasing with B-spline low-pass filters requires random screen samples to be computed with a PDF given by $n_m(x)$. This, in turn, requires the following CDF to be known:

$$N_m(x) = \int_0^x n_m(t)\, dt \tag{12}$$

We present here four properties about the integral $N_m(x)$ of a B-spline basis function that will be important when writing an implementation of anti-aliasing. Unlike (9), these properties were not taken from Chui [1992], although they can be derived from results in Chui with little effort.

$$\operatorname{supp} N_m(x) = [0, +\infty[\tag{13a}$$

$$N_m(x) \text{ increases monotonically from } N_m(0) = 0 \text{ to } \lim_{x \to +\infty} N_m(x) = 1. \tag{13b}$$

$$n_m(x) \;=\; N_{m-1}(x) - N_{m-1}(x-1), \quad \text{for } m > 1. \tag{13c}$$

$$N_m(x) \;=\; \frac{x}{m} N_{m-1}(x) \;+\; \left(1 - \frac{x}{m}\right) N_{m-1}(x-1), \quad \text{for } m > 1. \tag{13d}$$

Properties (13a) and (13b) are a direct consequence of (12), together with (9a), (9b) and (9c). These two properties tell us that $N_m(x)$ is a valid CDF. Properties (13c) and (13d) are derived in the Appendix. Property (13c) gives an algebraic relationship between the spline function and its integral at the next lower order. Property (13d) presents a numerical recipe for computing any value of $N_m(x)$ in a recursive way. At the end of the recursion lies the $N_1(x)$ function, whose shape has a trival expression, as given by (14).

$$N_1(x) = \begin{cases} 0 & \text{if } x < 0, \\ x & \text{if } x \in [0, 1[, \\ 1 & \text{if } x \geq 1. \end{cases} \tag{14}$$

The generation of random samples for Monte Carlo integration, with a B-spline acting as the PDF, now requires solving the following equation for a sample x in either horizontal or vertical screen coordinates, where y is a stratified uniform random variable in the $[0, 1[$ interval:

$$y = N_m(x) \tag{15}$$

The solution is immediate when $m = 1$ but, unfortunately, becomes rather involved for higher orders. Rather than try to solve (15) analytically, the best approach is to use a numerical iterative method like Newton-Raphson to find the zero of the auxiliary function $f(x) = N_m(x) - y$ [Press et al. 1992].

Newton-Raphson is a powerful root finder since it has quadratic convergence but some care must usually be taken before its application. The root must first be bracketed inside a suitable interval. Then, $f(x)$ must be monotonic inside that interval with a non-vanishing derivative everywhere. All these conditions are naturally met in the case of a B-spline CDF. Clearly, there is one and only one root x inside the interval between $y = 0$ and $y = 1$. The function is monotonically increasing inside that interval, as assured by property (13b), and the only points where the derivative vanishes are the two interval extremes. If we first clear the boundary case $y = 0 \Rightarrow x = 0$ then a series of Newton-Raphson iterations can be started, with full confidence that it will converge to the solution[2]:

$$x_{i+1} \;=\; x_i - \frac{f(x_i)}{f'(x_i)} \;=\; x_i - \frac{N_m(x_i) - y}{N'_m(x_i)} \;=\; x_i - \frac{N_m(x_i) - y}{n_m(x_i)} \tag{16}$$

The cost of performing (16) is $2^m - 1$ recursive calls of $N_m(x_i)$ and $2^m - 1$ recursive calls of $n_m(x_i)$, giving a total complexity of $O(2^{m+1})$ per Newton-Raphson iteration. It is possible to do better by replacing (13c) and (13d) in (16):

$$x_{i+1} \;=\; x_i - \frac{\left(x_i N_{m-1}(x_i) \;+\; (m - x_i)\, N_{m-1}(x_i - 1)\right)/m - y}{N_{m-1}(x_i) - N_{m-1}(x_i - 1)}, \quad \text{for } m > 1. \tag{17}$$

The cost is now $2^{m-1} - 1$ recursive calls to compute $N_{m-1}(x_i)$ and similarly for $N_{m-1}(x_i - 1)$. The complexity is $O(2^m)$, half of what it was before.

The Newton-Raphson iterations are started off with $x_0 = m/2$, which is at the center of the interval (9c) where the random variable x is bound to lie.

5. RESULTS

Figure 3 shows results of rendering an implicit surface by ray casting [Gamito and Maddock 2005]. The upper image does not feature any anti-aliasing, with a single

[2]The boundary case $y = 1$ need not be considered because y is a uniform random variable in $[0, 1[$.

ray being cast exactly from the center of each pixel. The bottom image shows the same surface rendered with our anti-aliasing method. The differences between the two images are more easily discernible at the internal edges between different surface features.

Figure 4 shows the effect of changing the number of samples per pixel for a small section of the implicit surface from Figure 3. From left to right, top to bottom, the sequence shows no anti-aliasing, and anti-aliasing with a cubic B-spline filter with N^2 equal to 4, 9 and 100 samples per pixel. Performing anti-aliasing by taking only four random samples (two samples along each coordinate direction) is not very effective but it does show that coherent aliasing artifacts are converted into noise by the stratification of sample points. When N^2 increases to 9 and then to 100 results become progressively better.

The number of Newton-Raphson iterations, whilst rendering the images of Figure 4, was from 1 to a maximum of 8, depending on the value of y when inverting (15). If $y = 0.5$ then $x = x_0 = m/2$ and the algorithm converges after exactly one iteration. If, on the other hand, y moves away towards one of the extremities of the $[0, 1[$ interval, a larger number of iterations will be required for x_i to converge to the solution. Nevertheless, the number of iterations always remains small due to the quadratic convergence properties of Newton-Raphson root finding.

6. CONCLUSIONS AND FUTURE WORK

B-splines are useful filters for anti-aliasing. They include the widely used box and tent filters as the first two members of their family. Cubic B-splines, however, have better filtering behaviour. Our method can generate any of these filters, and any B-spline filter in general, through a simple and elegant recursive procedure. This recursive procedure keeps us from having to deal with the actual piecewise polynomials, which describe the shape of the B-splines. Explicitly working with the polynomial representation for some B-spline $n_m(x)$ can become quite an involved procedure, even for moderate values of m. The cost of recursively evaluating $n_m(x)$ grows geometrically with m. This is not, however, a serious constraint for our method since filters of too high an order introduce excessive blurring and should be avoided.

Generation of random samples for Monte Carlo anti-aliasing, having $n_m(x)$ as their probability density function, requires inversion of $y = N_m(x)$ where $N_m(x)$ is the cumulative density function associated with the B-spline filter. This can be accomplished numerically with Newton-Raphson root finding in a way that is always guaranteed to converge. Experiments have shown that the number of required iterations remains within single digits.

Performing anti-aliasing on computer generated images is an expensive proposition. By using N^2 samples per pixel we are doing the same amount of work as though we were rendering a virtual image with a resolution N^2 higher than the actual image. The present method does not take into account local image information when performing anti-aliasing. In the example presented in the previous section, it is quite wasteful to be casting N^2 rays for a pixel located on the inside of the implicit surface and far from any edge. A single ray would have sufficed, as the non-antialiased image in Figure 4 can attest to. In this image, aliasing is only evident at the edges, while the internal surface features remain smooth. Techniques for

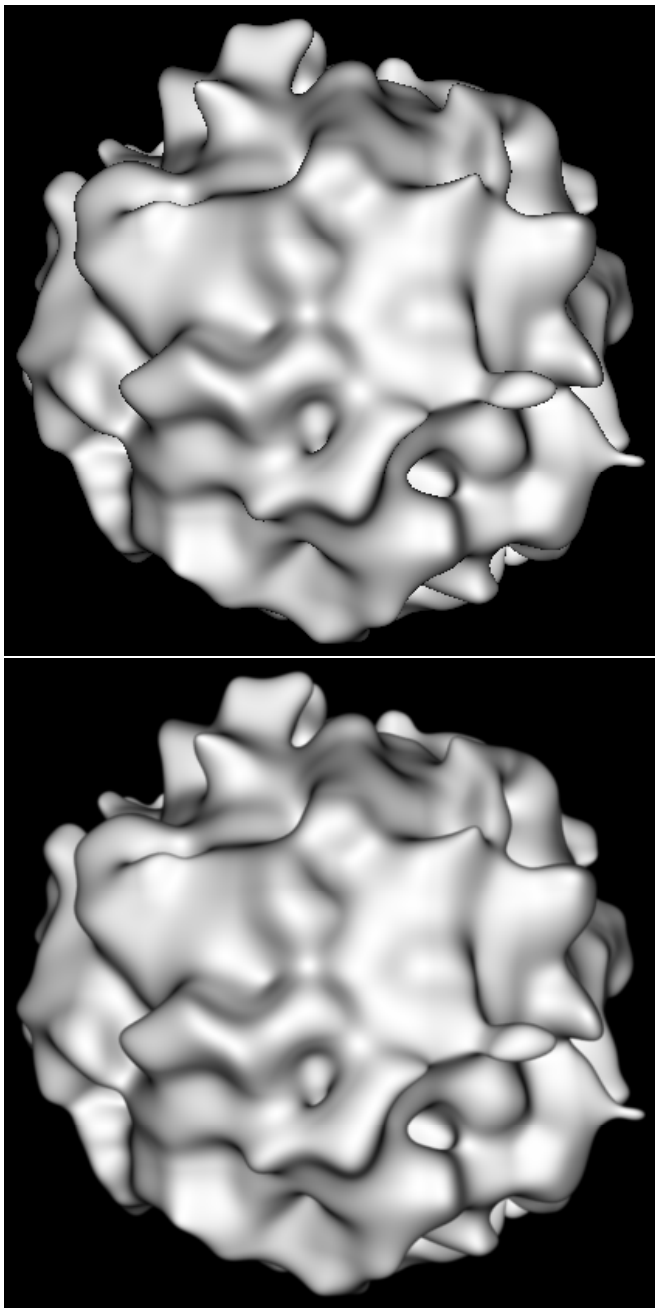

Fig. 3. Comparison between no anti-aliasing (above) and anti-aliasing (below) for a computer rendering of a procedural implicit surface.

turning the present anti-aliasing method with B-splines into an adaptive procedure, similar to the work by Painter and Sloan [1989], should be further investigated.

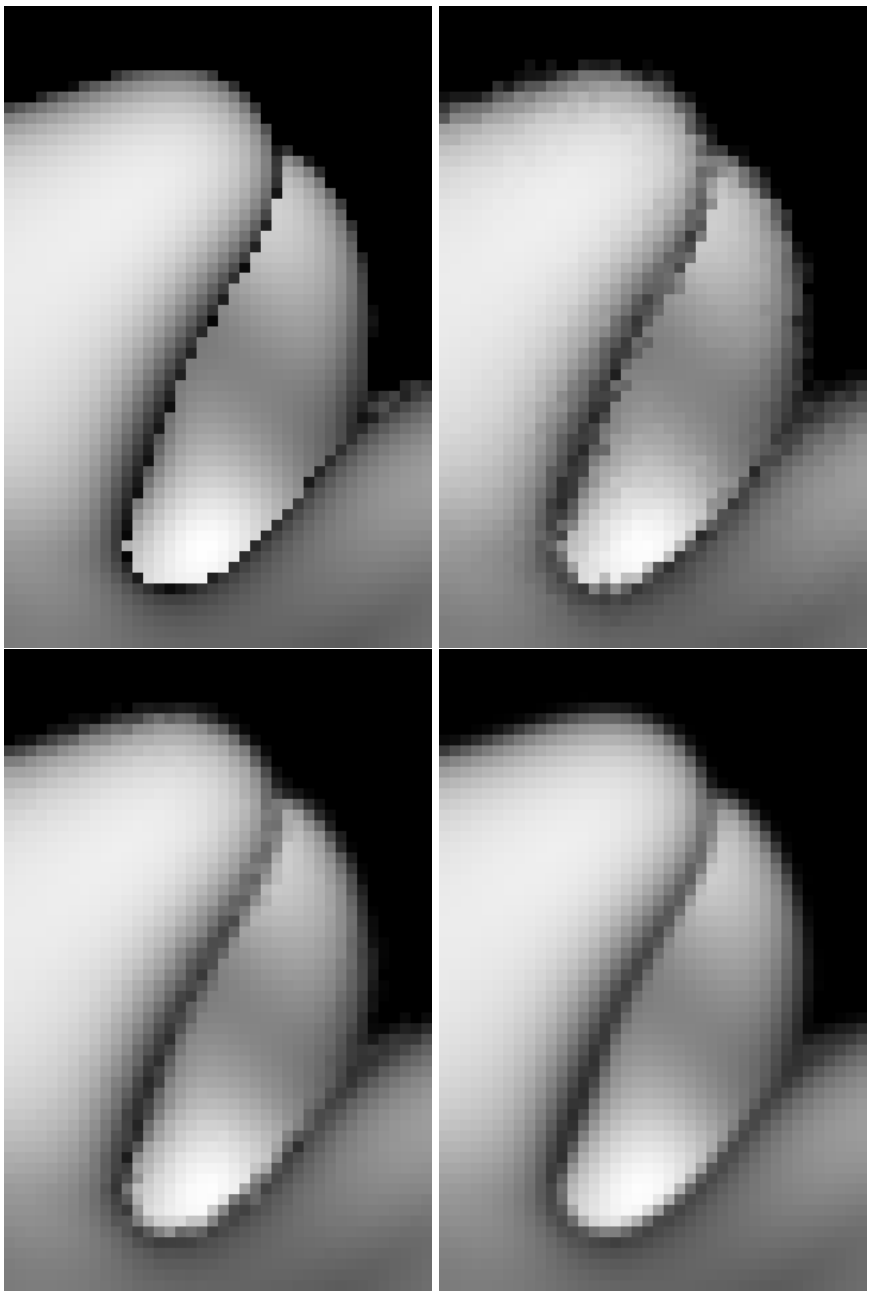

Fig. 4. From left to right, top to bottom: no anti-aliasing, anti-aliasing with a cubic B-spline filter with N equal to 2, 3 and 10 samples along each coordinate direction.

APPENDIX

We begin by deriving the relation (13c) between a B-spline of order m and the integral of B-splines of order $m-1$. To do so, we need to express $n_m(x)$ as an mth-order finite difference of powers of x. The reader is invited to consult Chui [1992] for more details. Take the notation $x_+ = \max(x, 0)$ to represent the restriction of x to positive values only. Similarly, $x_+^m = (x_+)^m$. Consider also the finite differences of order m for some function $f(x)$. These are defined with the following recurrence relation:

$$\begin{aligned} (\Delta f)(x) &= f(x) - f(x-1) && \text{(A.1)} \\ (\Delta^m f)(x) &= \left(\Delta^{m-1}(\Delta f)\right)(x), \quad \text{for } m > 1. && \text{(A.2)} \end{aligned}$$

Armed with this notation, we can write a B-spline $n_m(x)$ as:

$$n_m(x) = \frac{1}{(m-1)!}\,\Delta^m x_+^{m-1}, \quad \text{for } m > 1. \tag{A.3}$$

Taking the integral of (A.3), we arrive at a similar equation for $N_m(x)$:

$$\begin{aligned} N_m(x) &= \int_0^x n_m(t)\,dt = \frac{1}{(m-1)!}\int_0^x \Delta^m t_+^{m-1}\,dt \\ &= \frac{1}{(m-1)!}\int_0^x \Delta^{m-1}\left\{t_+^{m-1} - (t-1)_+^{m-1}\right\}\,dt \\ &= \frac{1}{(m-1)!}\,\Delta^{m-1}\left\{\int_0^x t_+^{m-1}\,dt - \int_0^x (t-1)_+^{m-1}\,dt\right\} \\ &= \frac{1}{(m-1)!}\,\Delta^{m-1}\left\{\int_0^x t_+^{m-1}\,dt - \int_{-1}^{x-1} t_+^{m-1}\,dt\right\} \\ &= \frac{1}{(m-1)!}\,\Delta^{m-1}\left\{\int_0^x t_+^{m-1}\,dt - \int_0^{x-1} t_+^{m-1}\,dt\right\} \\ &= \frac{1}{(m-1)!}\,\Delta^m \int_0^x t_+^{m-1}\,dt = \frac{1}{m!}\,\Delta^m x_+^m, \quad \text{for } m > 1. \end{aligned} \tag{A.4}$$

Using (A.4) twice, we have:

$$\begin{aligned} N_{m-1}(x) - N_{m-1}(x-1) &= \\ &= \frac{1}{(m-1)!}\,\Delta^{m-1}x_+^{m-1} - \frac{1}{(m-1)!}\,\Delta^{m-1}(x-1)_+^{m-1} \\ &= \frac{1}{(m-1)!}\,\Delta^{m-1}\left\{x_+^{m-1} - (x-1)_+^{m-1}\right\} \\ &= \frac{1}{(m-1)!}\,\Delta^m x_+^{m-1} = n_m(x), \quad \text{for } m > 1. \end{aligned} \tag{A.5}$$

This completes the derivation of (13c).

We now take the recurrence relation that exists for $n_m(x)$ and arrive at a similar relation for $N_m(x)$, expressed in (13d). We place an integral on both sides of (9d) and perform integration by parts:

$$\begin{aligned} N_m(x) &= \int_0^x n_m(t)\,dt \\ &= \int_0^x \frac{t}{m-1} n_{m-1}(t)\,dt + \int_0^x \frac{m-t}{m-1} n_{m-1}(t-1)\,dt \\ &= \frac{t}{m-1} N_{m-1}(t)\Big|_0^x + \frac{m-t}{m-1} N_{m-1}(t-1)\Big|_0^x \\ &\quad - \frac{1}{m-1}\int_0^x N_{m-1}(t)\,dt + \frac{1}{m-1}\int_0^x N_{m-1}(t-1)\,dt \\ &= \frac{x}{m-1} N_{m-1}(x) + \frac{m-x}{m-1} N_{m-1}(x-1) \\ &\quad - \frac{1}{m-1}\int_0^x \{N_{m-1}(t) - N_{m-1}(t-1)\}\,dt, \quad \text{for } m > 1. \end{aligned} \tag{A.6}$$

Replacing (13c) in (A.6), we get:

$$\begin{aligned} N_m(x) &= \frac{x}{m-1} N_{m-1}(x) + \frac{m-x}{m-1} N_{m-1}(x-1) - \frac{1}{m-1}\int_0^x n_m(t)\,dt \\ &= \frac{x}{m-1} N_{m-1}(x) + \frac{m-x}{m-1} N_{m-1}(x-1) - \frac{1}{m-1} N_m(x), \quad \text{for } m > 1. \end{aligned} \tag{A.7}$$

Sending the $N_m(x)$ term on the right to the left hand side of the equation and rearranging terms, we finally arrive at:

$$N_m(x) = \frac{x}{m} N_{m-1}(x) + \left(1 - \frac{x}{m}\right) N_{m-1}(x-1), \quad \text{for } m > 1. \tag{A.8}$$

This completes the derivation of (13d).

REFERENCES

Chui, C. K. 1992. *An Introduction to Wavelets*. Wavelet Analysis and its Applications, vol. 1. Academic Press, Chapter 4, 81–117.

Cook, R. L. 1989. Stochastic sampling and distributed ray tracing. In *An Introduction to Ray Tracing*, A. S. Glassner, Ed. Academic Press, Chapter 5, 161–199.

Gamito, M. N. and Maddock, S. C. 2005. Ray casting implicit procedural noises with reduced affine arithmetic. to be published.

Glassner, A. S. 1995. *Principles of Digital Image Synthesis*. Morgan Kaufmann Publishers Inc.

Painter, J. and Sloan, K. 1989. Antialiased ray tracing by adaptive progressive refinement. In *Computer Graphics (SIGGRAPH '89 Proceedings)*. Vol. 23. 281–288.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. 1992. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press.

Shirley, P. and Morley, R. K. 2003. *Realistic Ray Tracing*, Second ed. A K Peters Ltd.

Stark, M., Shirley, P., and Ashikhmin, M. 2005. Generation of stratified samples for B-spline pixel filtering. to appear in Journal of Graphics Tools.