2

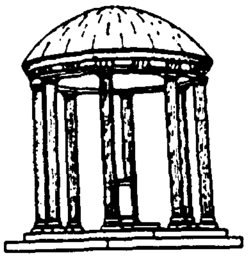
Efficient Supersampling Antialiasing for High-Performance Architectures

TR91-023

April, 1991

Steven Molnar

The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175



This work was supported by DARPA/ISTO Order No. 6090, NSF Grant No. DCI-8601152 and IBM.
UNC is an Equal Opportunity/Affirmative Action Institution.

EFFICIENT SUPERSAMPLING ANTIALIASING FOR HIGH-PERFORMANCE ARCHITECTURES

Steven Molnar
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175

Abstract

Techniques are presented for increasing the efficiency of supersampling antialiasing in high-performance graphics architectures. The traditional approach is to sample each pixel with multiple, regularly spaced or jittered samples, and to blend the sample values into a final value using a weighted average [FUCH85][DEER88][MAMM89][HAEB90]. This paper describes a new type of antialiasing kernel that is optimized for the constraints of hardware systems and produces higher quality images with fewer sample points than traditional methods. The central idea is to compute a Poisson-disk distribution of sample points for a small region of the screen (typically pixel-sized, or the size of a few pixels). Sample points are then assigned to pixels so that the *density* of samples points (rather than weights) for each pixel approximates a Gaussian (or other) reconstruction filter as closely as possible. The result is a supersampling kernel that implements importance sampling with Poisson-disk-distributed samples. The method incurs no additional run-time expense over standard weighted-average supersampling methods, supports successive-refinement, and can be implemented on any high-performance system that point samples accurately and has sufficient frame-buffer storage for two color buffers.

1 Introduction

An important requirement for any graphics system, real-time, or otherwise, is that it be able to display images without noticeable aliasing. Aliasing results when high frequency components of an underlying continuous image map onto low frequency components of the sampled digital image. A digital image can only represent spatial frequencies up to its Nyquist frequency—half the frequency of its pixel grid [OPPE75]. An image containing lines and polygons has components of infinitely high frequency. During sampling, these alias to produce the familiar stairstepping (jaggies), moving artifacts (crawlies), and scintillation that have plagued computer graphics.

Aliasing can be mitigated in a number of ways; the central theme is to bandlimit the image before sampling at pixel rates. A full description of the various methods is beyond the scope of this paper. [FOLE90] provides a good introduction to the area and pointers to the literature.

Most hardware and software systems antialias using some form of supersampling. This involves sampling the image at higher-than pixel resolution and filtering the samples down to a single sample per pixel. Supersampling can be done adaptively, based on the local scene complexity of an image, but its simplest and most regular form involves oversampling the image uniformly. We will use this definition of supersampling throughout this paper.

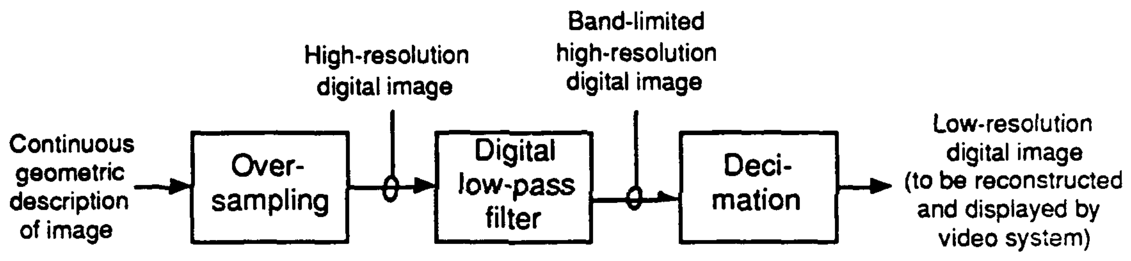

Continuous geometric description of image
High-resolution digital image
Band-limited high-resolution digital image
Over-sampling
Digital low-pass filter
Deci-mation
Low-resolution digital image (to be reconstructed and displayed by video system)

Figure 1: Conceptual pipeline of operations performed during supersampling.

Although supersampling appears simple, it really is a combined implementation of the three conceptual steps depicted in Figure 1. The image is first sampled at several times the pixel resolution. This high-resolution digital image can represent higher frequency components than can a single-sample-per-pixel image. The digital low-pass filter (sometimes called a reconstruction filter) removes frequency components that cannot be represented in the low-resolution image. Decimation resamples the band-limited image with a single sample per pixel. The basic idea behind supersampling is to capture and remove frequency components that cannot be represented in the final image. These three operations are combined into the single sampling and weighted-average operation that gives supersampling its simplicity.

Supersampling has become the preferred method for antialiasing on most high-performance graphics architectures [FOLE90]. It can be used to antialias images containing any type of primitive and shaded in any way, and can incorporate effects such as depth-of-field and motion blur [HAEB90] and transparency. Unfortunately, supersampling an image with k samples per pixel requires rendering the image at k times the resolution or rendering it k times with slightly different coordinate offsets—a k-fold increase in computational cost over the corresponding unantialiased image. For good quality images, k is typically chosen to be 16 or greater. This is a burdensome computational expense for any system; it is prohibitive for most interactive systems.

A great deal of research has been aimed at finding sampling patterns and low-pass filters that make supersampling as effective as possible. Gaussian and other filters have been shown to produce images superior to those produced by a simple box filter [MITC88]. Randomly jittered sample points have been shown to convert residual aliasing into less objectionable random noise [DIPP85, COOK86]. Sophisticated stochastic and adaptive sampling techniques have been developed for ray tracing [COOK86, KAJI86, MITC87, SHIR90]. These approaches have focused on the needs of software systems, as opposed to hardware systems, however.

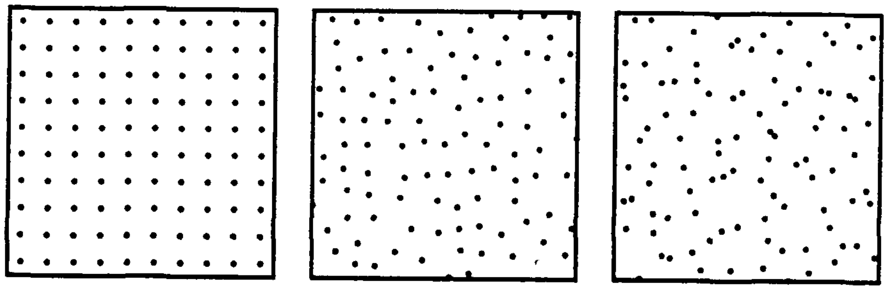


Figure 2: a) Regular sampling, b) Poisson-disk sampling, and c) jittered sampling.

A software renderer allows one to choose different sample points for every pixel of an image. If samples are placed non-uniformly, this minimizes the correlation in the noise pattern produced by residual high-frequency aliasing. It also places constraints on the algorithm used to generate sample points; since it must be invoked for every pixel of every image, it must be efficient. The best sampling distributions that are known, such as Poisson-disk sampling (random points separated by a minimum distance—see Fig. 2b) are too expensive to compute for each pixel, and as a consequence are seldom employed. The most popular technique is jittering (Fig. 2c), in which a pixel is subdivided into uniform subregions, and a sample point is chosen at a random location within each subregion.

Jittering only requires calculating two uniformly-distributed random values for each sample point.

Software algorithms also can increase sampling efficiency by sharing samples between neighboring pixels. For example, a sample halfway between two pixels can contribute equally to each pixel. This capability is seldom found in hardware systems (at least in real-time), because neighboring pixels are often handled on different processors.

Hardware rendering systems have a different set of constraints. Since most high-performance architectures use pixel parallelism and forward differences to interpolate parameter values, they are restricted to sampling on a regular grid. By repeatedly rendering the image with small x and y displacements, pixels can be supersampled with arbitrarily placed samples, but with two important restrictions:

- Sample locations must be identical for every pixel in the image.
- Samples cannot be shared between pixels.

These characteristics prevent using the best antialiasing techniques employed in software renderers. However, they also create opportunities: since the same sample locations are used in every pixel of every image, a great deal of effort can and *should* be spent choosing these sample locations. No one has looked at this issue comprehensively to date. This paper seeks to do so.

We first examine the tradeoffs between different sampling patterns, sample weights, and low-pass filters. We then develop two methods for supersampling efficiently on different types of parallel architectures. The first approach achieves good results and is applicable to all commercially available hardware graphics systems that implement point sampling. The second achieves near ideal results (comparable to supersampling with subpixel jittered samples) and can be implemented with little cost on many systems that use interleaved frame buffer memories. Both allow the image to be displayed after one or a few samples have been computed, and successively refined as more samples are computed.

2 Efficient Supersampling: Producing the Highest Quality Image with the Fewest Samples

The efficiency of a supersampling algorithm can be defined as the quality of the images it produces (in some appropriate units) divided by the cost of creating them (generally, the number of samples per pixel). If samples are located and weighted appropriately, the quality of a supersampled image can be enhanced arbitrarily by increasing the number of samples. The objective is to produce an image of acceptable quality with a minimum number of samples. This section establishes criteria for an efficient supersampling antialiasing algorithm.

Four characteristics contribute to image quality in a supersampling algorithm:

- Flat field response
- Correct low-pass filter
- Sample density
- Gamma correction

Flat field response. Ideally, a given image feature of constant size should contribute the same intensity to the image regardless of where it lies. In practice, this means the sum of the weights of the sample points in each small area of the image should be constant throughout the image.

Poor flat field response on a small scale produces graininess in regions of constant intensity and scintillation when small features move with respect to pixel boundaries from frame to frame. Poor flat field response on a larger scale means that large objects with constant color appear to have shadows or bright spots. A corollary to flat field response is that an image of constant intensity should be reconstructed perfectly after antialiasing.

Sample distributions like jittering, which clump sample points, adversely affect flat field response: a feature that is hit by a clump of samples contributes more strongly than an identical feature that is hit by only a few samples. More homogeneous distributions, such as samples on a regular grid or Poisson-disk distributions (see Fig. 2), have better flat field response.

A second cause of poor flat field response is a low-pass filter whose weights do not add to a constant value when adjacent filter kernels are superimposed. Unfortunately, the best reconstruction filters from a signal-processing standpoint do not have this property.

Low-pass filter. The low-pass filter attenuates frequency components in the oversampled image that would alias in the low-resolution image. The ideal low-pass filter from a signal processing standpoint is a sinc function ($\sin(x)/x$). Unfortunately, a sinc filter has a slow falloff (asymptotically $1/x$) and is prone to ringing, which make it a poor choice for most graphics applications. Narrower filters, such as a Gaussian or various empirically derived filters, have given better results for graphics applications [SCHR85, MITC88]. In the remainder of this paper we will assume a normalized 2D Gaussian filter, which has the formula:

$$\frac{1}{2\pi\sigma^2_{\text{low-pass}}} e^{\frac{-(x^2+y^2)}{2\sigma^2_{\text{low-pass}}}}$$

The width of the filter, $\sigma_{\text{low-pass}}$ must be a compromise as well: it should attenuate frequencies above the Nyquist frequency of the output image and should minimize the variation in sampling density when Gaussian filters at neighboring pixels are superposed. To attenuate frequencies above the Nyquist frequency, $\sigma_{\text{low-pass}}$ should approximately equal the pixel-spacing. To minimize the variation in sampling density, $\sigma_{\text{low-pass}}$ should be 0.43 times the pixel spacing (this reduces the variation to approximately 1 percent by volume) [PAVI90]. In practice, $\sigma_{\text{low-pass}}$ is generally chosen around 0.5 because larger values result in blurring. The price for this image sharpness is aliasing of frequency components between the Nyquist frequency and the cutoff frequency of the low-pass filter. The only solution to this problem is to use a higher resolution monitor!

Number of samples. The number of samples per pixel determines the Nyquist frequency for the high-resolution, oversampled image. This is a sharply defined value if samples are spaced evenly, but is a useful notion even if non-uniform samples are used. The Nyquist frequency is then defined in terms of the *average* sampling rate, and corresponds to a blurry transition rather than a sharp cutoff. Some frequency components above the average Nyquist frequency are captured, and some below it are lost., in contrast to the sharp frequency cutoff that occurs with regular sampling.

In either case, frequency components above the Nyquist frequency alias in the high-resolution image and cannot be removed by the low-pass filter. Increasing the sampling rate allows the low-pass filter to attenuate higher and higher frequency components, thereby reducing the amount of aliasing in the final image. No amount of sampling can remove all aliasing, however, since geometric images contain components of infinitely high frequency.

Gamma correction. For supersampling to work correctly, there must be a linear relationship between color values and perceived image intensities on the display screen. Unfortunately, CRT monitors and photographic film are notoriously nonlinear (the intensity of light output from a CRT monitor is a function of the input voltage raised to the γ power, where γ is a constant typically lying between 1 and 3).

This and other nonlinearities can be avoided by setting color map values that compensate for this, a process called *gamma correction* [FOLE90]. Gamma correction must be employed for supersampling to work effectively on most monitors.

The four characteristics above enable a supersampling algorithm to converge to a high quality image if enough samples are taken. Two techniques allow one to minimize the number of samples for a given quality image:

Uncorrelated sample points. Dippé and Cook have shown that residual aliasing can be made much less objectionable by choosing uncorrelated samples. This transforms the energy of the aliased high-frequency components into uncorrelated random noise, which is largely ignored by the human visual system [DIPP85, COOK86].

Virtually any random sampling pattern has this effect. The frequency of the noise is determined by the power spectrum of the sampling pattern. Sampling patterns with low frequency components lead to low frequency noise; sample patterns with high frequency components lead to high frequency noise. The latter case is preferable, since the human visual system is less sensitive to noise of high frequencies. Yellott showed that a Poisson-disk distribution has a power spectrum with the desired characteristics. He also noted that light receptors in monkeys' eyes are distributed in this fashion [YELL83].

Poisson-disk distributions have been a favorite topic in theoretical papers on antialiasing, but have found their way into few implementations, largely because they are expensive to compute. Ideally, the sample distribution should cover the entire image. A smaller sample distribution with periodic boundary conditions can be repeated to tile the screen, though this introduces correlation into the sample pattern.

Equal weights for each sample. The shape of the low-pass filter is determined by the positions and weights of the samples. A desired filter shape can be achieved in one of *sampling* (Fig. 3a), or samples can be distributed according to the filter shape and given equal weights—*importance sampling* (Fig. 3b).

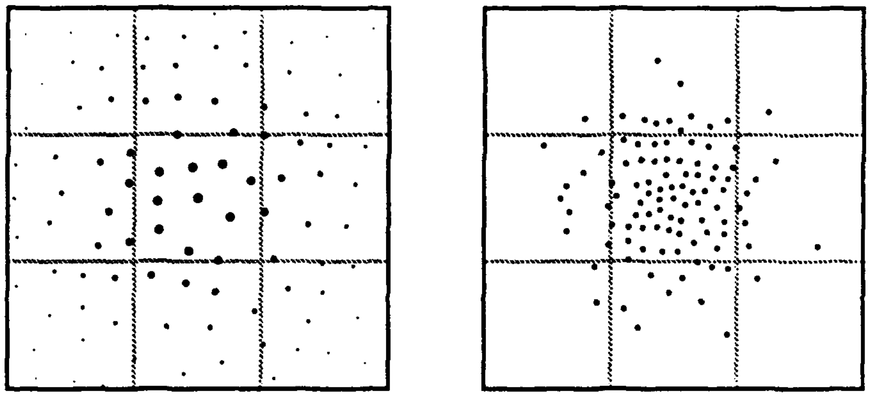


Figure 3: a) Uniform sampling with weighted samples and b) Importance sampling with unweighted samples.

Uniform sampling has been almost universally used in software and hardware graphics systems. In software systems, the image is sampled with uniform density (but not necessarily on a regular grid). Samples in the neighborhood of each pixel are assigned weights based on the distance to the pixel center and summed together. A single sample, therefore, contributes to several pixels, and each sample contributes approximately equally to the final image. This is an efficient method for sampling if samples can be rapidly communicated to multiple pixels, but this is seldom the case in hardware systems.

Importance sampling chooses sample locations by their importance to the final image: samples of equal weight are concentrated in areas of high filter weight, and comparatively few are placed in areas of low filter weight. If each sample costs the same to compute, this maximizes the efficiency of the antialiasing algorithm, since it maximizes the average information content of the samples.

Importance sampling is the method of choice if samples cannot be shared between pixels. It has the added advantage that sample values can simply be summed, rather than needing to be differentially scaled. This can simplify rendering hardware in certain applications. A word of caution is in order: importance sampling can lead to a poor approximation of a low-pass filter if very few samples are used.

Given these criteria, how, then does one synthesize an antialiasing kernel that is as close to optimal as possible? A preliminary observation is that no solution satisfies all criteria perfectly. For example, ideal flat field response requires uniformly spaced samples, but uniform samples add coherence that makes aliasing more noticeable. Several but uniform samples add coherence that makes aliasing more noticeable. Several researchers have developed antialiasing kernels appropriate to ray tracers. The following section develops a kernel tailored to the characteristics of high-performance graphics systems.

3 The Poisson-Disk-Gaussian Kernel

Based on the criteria above, we wish to develop an antialiasing kernel with: 1) Poisson-disk-distributed samples over the entire screen, 2) Gaussian distribution of equally-weighted sample points within a pixel, and 3) As little clumping as possible for samples within a pixel. We considered three ways to build such kernels:

1) Compute a sample distribution for the entire screen and assign *k* samples to each pixel.
2) Compute a *k*-sample kernel for each pixel, superimpose them, and adjust sample positions to reduce clumping.
3) Compute samples that satisfy both criteria at once.

We investigated all three alternatives. Although all were outwardly appealing, the second two led to difficulties: Approach 2 resulted either in a poor overall sampling distribution or distortions to the sample pattern for individual pixels. Approach 3 led to regularities in the sample pattern when we used attractive and repulsive forces to position samples, and poor sampling patterns and approximations to the low-pass filter when we attempted to generalize the Poisson-disk sample generation algorithm.

Approach 1 led to good results, however. By assigning samples to pixels appropriately, we were able to achieve an excellent approximation to the desired Gaussian filter while reduce clumping between samples in a single pixel to an acceptable level. This section describes this approach in furrther detail. There are two main steps: generating a Poisson-disk distribution of sample points with equal weights in a pixel-sized region, and exchanging sample points between adjacent pixels to produce a sampling pattern that approximates the desired reconstruction filter.

Throughout this section, we will be concerned with antialiasing kernels with *k* equally weighted samples to be used in each pixel of the image. Later, we will describe how to generate kernels for each pixel in an *m* x *n* pixel tile of the image and kernels for successive refinement antialiasing.

3.1 Computing a Poisson-Disk Distribution

The first step is to compute a Poisson-disk distribution of k samples within a rectangle of pixel height and pixel width with periodic boundary conditions (i.e. the distribution wraps around from right to left edges and from top to bottom).

Any of several methods can be used to compute the Poisson-disk distribution [RIPL77]. The simplest to implement is probably the dart throwing technique described in [COOK86]. In this approach, uniformly distributed candidate points are generated in the sample region. Each candidate point is checked against all of the existing points. If it is closer than a predetermined distance d to any of them, it is disgarded. Otherwise, it is added to the distribution. This process is continued until no further points can be added (in practice, a large number of attempts are made and if all of them fail, the process terminates). As has been reported, this algorithm takes a great deal of CPU time to compute distributions with many samples. This is not a limitation for our purposes, however, since we are interested in distributions with relatively few samples and only calculate the distribution once.

In generating a *Poisson-disk distribution,* one controls the number of samples; indirectly through the choice of d. A binary search algorithm can be used to home in on the correct radius for a given number of samples.

3.2 Permuting Samples to Approximate the Low-Pass Filter

The second step is to permute sample points between pixels so that the density of samples for each pixel approximates the desired reconstruction filter as closely as possible. If Δx is the horizontal spacing between pixel centers and Δy is the vertical spacing, one can translate a sample point by $\pm\Delta x$ in x or $\pm\Delta y$ in y and still obtain the same overall sampling pattern when samples from all pixels are superimposed. Translating sample points in this manner alters the shape of the low-pass filter. We want to find the arrangement that most closely approximates the filter we want.

The best method we have found is a relaxation algorithm that translates sample points in pairs in all possible combinations and optimizes the distribution using a merit function. If any changes are made during a pass through the sample-point pairs, another pass is made. This process continues until a pass is made with no changes.

Translating sample points in pairs is a heuristic to avoid an exponential-time search for the best sampling pattern. If k samples are taken per pixel, the k samples can be arranged in 9^k possible patterns (each sample point can be translated by $[-1, 0, 1]\Delta x$ in x and by $[-1, 0, 1]\Delta y$ in y). For k larger than 6 or so, an exhaustive search becomes prohibitively expensive. Samples are translated in pairs because this allows samples to exchange places, a heuristic that allows the sample pattern to approach the optimum. It requires evaluating $9 \cdot 9k(k-1)/2 = 40.5k^2 - 40.5k$ sample patterns for an antialiasing kernel with k samples—still a large number, but with quadratic, rather than exponential time complexity.

3.3 The Merit Function

The merit function is used to compare a candidate arrangement of sample points with the desired reconstruction filter. The sample density of the actual sampling pattern is measured at typically 30 x 30 grid points spread over a 3-pixel x 3-pixel rectangle. At each grid point, the density of the actual sample pattern is calculated and subtracted from the density of the ideal filter at the same grid point. The difference is squared and summed over all grid points to get an estimate of the overall error of the sampling pattern.

A troublesome detail is how to obtain an accurate estimate of the local density of sample points for each grid point. Clearly, nearby sample points should contribute most heavily, but large numbers of more distant samples should contribute as well. We obtained the best results by convolving the "image" of sample points with a 2D Gaussian low-pass filter with $\sigma = \sigma_{\text{low-pass}}/2$. The convolution integral only needs to be evaluated at grid points, and the only contributions to the integral occur at sample points. It degenerates to the following sum for grid point (x_g, y_g) and sample points (x_i, y_i), where $0 \le i \le k$:

$$\sum_{\text{samples } i} \frac{1}{2\pi\sigma^2} e^{\frac{-((x_g - x_i)^2 + (y_g - y_i)^2)}{2\sigma^2}}$$

Samples further than 2.5 σ from (x_g, y_g) can be ignored, since the Gaussian drops to less than 5 percent of its peak value at that radius. Calculating the sum from scratch each time can be avoided by storing the density at each grid point, and subtracting and adding the contribution of individual sample points as they are moved.

For the merit function to work correctly, the sum of the measured densities must match the "volume" of the reconstruction filter (also measured and summed over the same grid points). To do this normalization, we first place a single sample at the pixel center. We then compute the local density of this trivial sample pattern at each grid point and sum to obtain v_1, the measured volume for a single sample point. We then sum the weights of the reconstruction filter at each grid point to obtain v_{ideal}, the volume of the reconstruction filter. When evaluating the merit function, we scale the measured local density at each grid point by $v_{\text{ideal}}/(k \cdot v_1)$. The resulting pattern of sample points determines where equal-weighted samples should be taken for each pixel when rendering the final image.

3.4 The Antialiasing Kernel

The antialiasing kernel resulting from the optimal arrangement of sample points implements a Gaussian-low-pass filter with equally weighted samples and Poisson-disk sampling over the entire image—the ideal we were striving for. Fig. 4 shows kernels containing 5, 9, 16, and 25 sample points. Since samples are chosen randomly, each run of the program produces a different antialiasing kernel. The variation between kernels is insignificant when k is large, but when k is small (less than 16 samples or so), they noticeably affect the quality of the kernel.

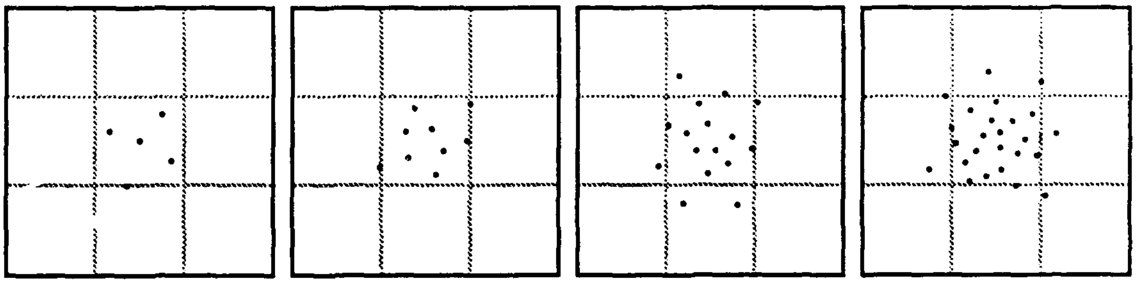


Figure 4: Kernels containing 5, 9, 16, and 25 samples.

To find a good kernel for small k, we run the program many times using different seed values for the random number generator. During each run we log the value of the merit function for the final sampling pattern, together with the seed value used to produce the pattern. We choose the pattern with the least error (merit function result) out of all the runs.

We have used these algorithms to calculate kernels with a Gaussian reconstruction filter and from 4 to 49 samples per pixel. The resulting images have fewer artifacts than others we have seen rendered with the same number of samples per pixel and the same sampling pattern in each pixel. Section 6 contains test images that were antialiased in this way and compares them with the results of other algorithms.

4 Successive Refinement Supersampling

Successive refinement is a technique for dynamically trading image quality for rendering speed in interactive systems [BERG86]. The image is first rendered crudely, but presented to the user as rapidly as possible to maximize interactive response. If no input from the user is received, the image is successively refined over a period of time until a maximum quality level is reached. Successive refinement has been applied to supersampling by constructing a series of kernels with increasing numbers of samples, each of which contains all of the samples of the previous kernels [FUCH85]. The techniques we have described can be extended to construct kernels in this fashion. For a successive refinement kernel with r levels, let k_0, k_1, ... k_r be the number of samples in each level.

First, a Poisson-disk distribution with k_0 samples per pixel is constructed and stored. These k_0 samples are then considered fixed and k_1-k_0 samples are added to the distribution using the dart-throwing algorithm combined with a binary search as described in Section 3.1. The new distribution with k_1 samples is then stored. The process is repeated for k_2, k_3, ...k_r sample points until a distribution with k_r samples is obtained.

Even though distributions with $k_1 - k_r$ samples are not pure Poisson-disk distributions, we have found them to be very close approximations in terms of their appearance, their Fourier transforms, and the final images they produce. Fig. 5a and 5b show a Poisson-disk distribution with 25 samples computed from scratch and its Fourier transform. Fig. 5c and 5d show a distribution with 25 samples built successively from distributions with 1, 5, 9, 16, and 25 samples together with its Fourier transform.

The next step is to permute samples, as described in Section 3.2, for each kernel in turn, keeping the samples from previous kernels fixed. Fig. 6 shows successive refinement kernels with 5, 9, 16, and 25 samples built from the distributions in Fig. 5c.

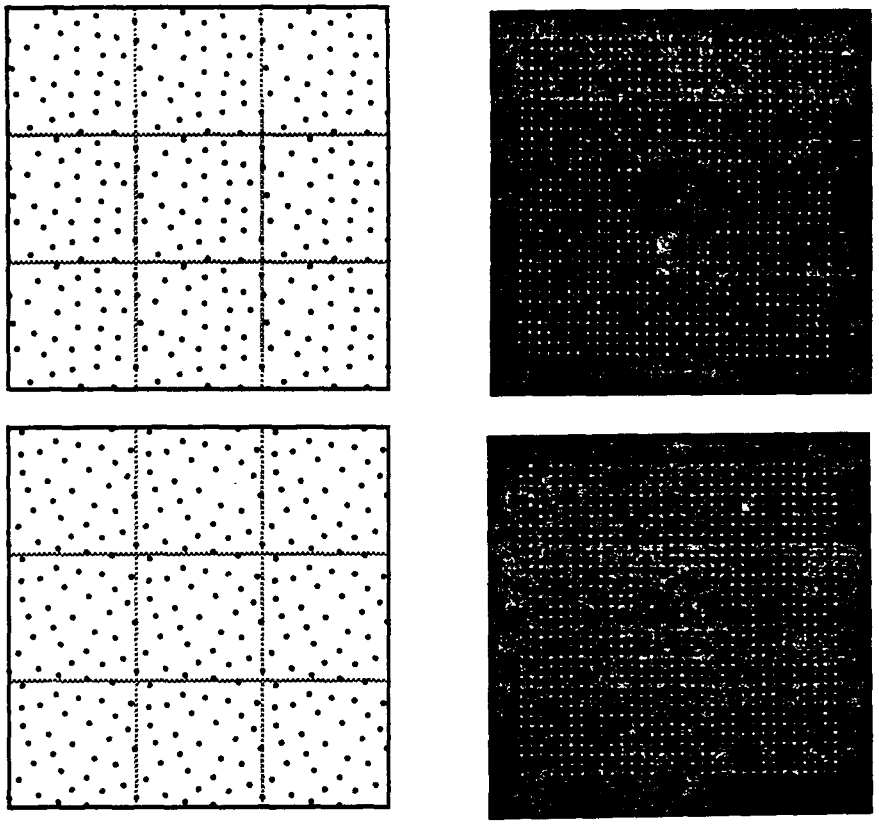


Figure 5: a) Poisson-disk distribution with 25 sample points and b) its Fourier transform. c) Successive-refinement distribution with 25 sample points built from distributions with 1, 5, 9, and 16 samples and d) its Fourier transform.

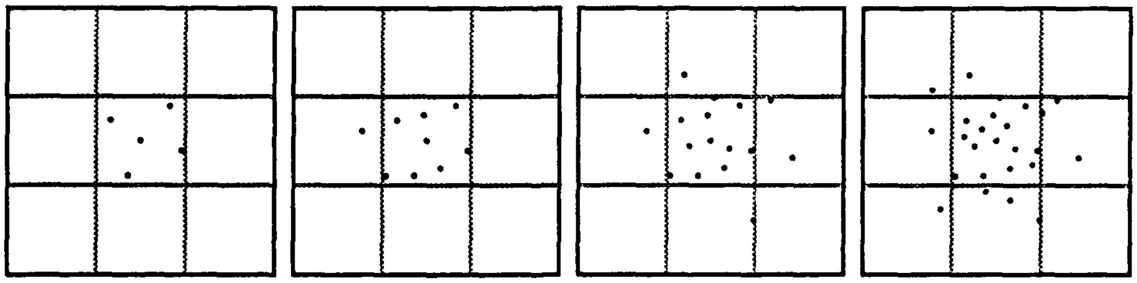


Figure 6: Successive refinement kernel with 5, 9, 16, and 25 samples.

An additional advantage of successive refinement kernels is that they are faster to compute. Since the relaxation algorithm is $O(k^2)$, the work in computing all of several smaller steps is less than the work in computing a single large step. This can be used to speed the calculation of kernels with large numbers of samples.

5 Interleaved Sample Patterns

Although most hardware systems require the same sample pattern to be used in each pixel of the screen, several popular graphics architectures, including AT&T's Pixel Machine and Silicon Graphics' VGX, have somewhat looser requirements: sample patterns can be different for each pixel in any m x n tile of the screen. These systems use an interleaved array of processors, each with a portion of frame-buffer memory. Each processor is responsible for every nth pixel of every mth scanline. Different processors can be provided with different sample patterns without affecting the inner loop of the rendering algorithm. This brings us one step closer to the ideal of different samples for each pixel of the screen. This section describes extensions of the above techniques to compute different kernels for each pixel in an m x n pixel tile.

The first step is to compute a Poisson-disk distribution with $k \bullet m \bullet n$ samples in an $m \bullet \Delta y$ x $n \bullet \Delta x$ region with periodic boundary conditions. Each of the m x n pixels is initialized by assigned k samples to it arbitrarily.

Next, we modify the relaxation algorithm to exchange samples *between* pixels, rather than translating samples as before. We wish to find the assignment of samples to pixels that makes the sampling pattern in each pixel approximate the reconstruction filter (now moved to the pixel center) as closely as possible. To do this we modify the merit function to compute the error between a distribution of samples and the ideal reconstruction filter for any pixel in the m x n array. The merit function also must handle the periodic boundary conditions of the array properly as well.

Samples are then considered in pairs, this time each from a distinct pixel. The merit functions for both pixels are evaluated and summed. The two samples are then temporarily swapped and the merit functions are evaluated and summed again. If the sum of the errors (merit functions) is less in the new configuration, the swap is made permanent. Then the next pair of samples is considered.

This process continues until all pairs of samples from different pixels have been considered. If any samples were swapped, the entire process is repeated until an entire pass has been made with no changes. Note that an exhaustive search of all sample permutations is truly impossible here: an *m* x *n* footprint with *k* samples per pixel has

$$\binom{mnk}{k}\binom{(mn-1)k}{k}\binom{(mn-2)k}{k}\cdots\binom{k}{k}$$

possible permutations—an astronomical number even for small footprints with few samples per pixel (a 2 x 2 pixel tile with 16 samples per pixel has $4*10^{145}$ permutations!). The pair-wise comparisons can be made more efficient by only considering swaps in which both samples remain within a $2.5\sigma_{\text{low-pass}}$ radius of their respective pixel centers. With this optimization, computing a kernel with 32 samples on a 5x4 pixel footprint took 23 hours on a DECStation 3100. This is a long time, but is tolerable, since a sample pattern only needs to be calculated once.

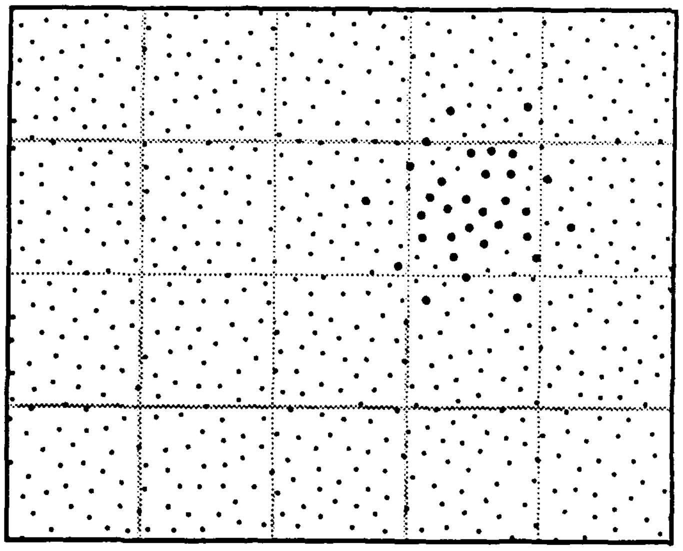


Figure 7: Samples for pixels in a 5x4-pixel tile. Samples corresponding to pixel (3,2) are emphasized.

Fig. 7 shows the sample pattern for a 5x4 pixel footprint with 16 samples per pixel. Samples corresponding to pixel (3,2) are emphasized. Images rendered with these kernels look very good. They are slightly noisier than images rendered with the same samples in each pixel, but worst-case artifacts like horizontal and vertical lines look much better rendered in this way. Figure 8 shows test images rendered with this approach.

6 Results

We incorporated the one-pixel kernels described here into an interactive display program on Pixel-Planes 5 [FUCH89] and into a prototype scan-line renderer. Multiple-pixel kernels were tested on the scan-line renderer. The single-pixel kernels were a drop-in change to our existing supersampling code.

Figure 8 shows a test pattern of 40 white triangles that converge to a single point rendered using various antialiasing methods. The converging triangles produce distinct Moiré patterns and are very difficult to antialias. As expected, increasing the number of samples increases the image quality for every antialiasing method. Also, as expected, random sampling produces results superior to uniform sampling, both with our method and with jittered, weighted samples. Although each kernel has its own artifacts due to residual aliasing, the Poisson-disk Gaussian kernels show a noticeable improvement over the earlier approaches.

Figure 9 shows a 512x512 image of a proposed space station and space shuttle dataset rendered with a single-pixel 25-sample kernel. The dataset contains 3784 polygons, many of which are small and thin and have very high frequency components. We found that all of the random sampling approaches produced good results on this and other datasets we tried (these databases don't have the correlation of the test pattern, so the improvements were less dramatic). Differences became more apparent, however, when we animated the display.

We created film loops of the space station and shuttle rotating slowly and antialiased them with different methods to test for scintillation and motion artifacts. The space station solar panels contain a sub-pixel-width center line that slowly changes from positive slope to negative slope. This line exhibited noticeable crawlies when rendered with all of the single-pixel methods, but the crawlies disappeared with the 5x4 kernel. This was the only instance we found in which the 5x4 kernel was noticeably better on a real image than the single-pixel kernels. This was a considerable surprise to us and offers encouragement that hardware architectures of the future may be able to remain simple and still produce high-quality images.

7 Conclusion

We have presented algorithms for constructing supersampling antialiasing kernels tailored to the capabilities of parallel hardware systems. These methods allow one to produce high quality images more efficiently than previous algorithms. Although these kernels are expensive to compute, they only need to be computed once and can be incorporated with no run-time cost into most programs that use supersampling antialiasing, and since samples receive equal weights, they can be summed together directly, rather than requiring a multiply before the addition, as is the case with weighted-sample kernels. This can result in time savings during rendering, and can simplify hardware architectures that antialias by supersampling.

In addition to the final algorithms we developed, we experimented with many others that didn't work so well. One negative result is worth mentioning: using "repulsion" between samples is a very poor way to generate random sample patterns. Whether linear or inverse-square repulsion forces are used, this method tends to force samples into a regular, closest-packed lattice. Certain values of k lead to regularly spaced samples (these can be seen in Figure 4 of [HAEB90]). Even prime numbers of samples lead to local regularity. Randomly generated sample patterns have proven to be far superior to patterns generated by sample repulsion.

A surprising result for us is how good the single-pixel kernel images appear relative to the multiple-pixel kernels. One can, of course, construct images with features spaced at exact multiples of the pixel spacing, and these will alias terribly. This does not happen as often as one might expect. Image features seem to be largely uncorrelated with the pixel grid for most images, so single-pixel kernels do a reasonably good job. This is confirmed by our groups' several years of experience with supersampling with single-pixel kernels: such artifacts have been noticed only rarely.

Acknowledgements

I wish to thank Turner Whitted, Don Mitchell, and Paul Haeberli for reading and critiquing drafts of this paper. Trey Greer, Greg Turk, Brice Tebbs, Henry Fuchs, John Eyles, and John Poulton all contributed to this paper through numerous suggestions and fruitful discussions. The space station database was provided by Don E. Eyles of the Charles Stark Draper Laboratories, Cambridge, MA. This work was supported by DARPA/ISTO Order No. 6090, NSF Grant No. DCI-8601152 and IBM.

References

BERG86 Bergman, L., H. Fuchs, E. Grant, and S. Spach, "Image Rendering by Adaptive Refinement," *Computer Graphics* (Proceedings of SIGGRAPH '86), Vol. 20, No. 4, pp. 29–145.

COOK86 Cook, R.L., "Stochastic Sampling in Computer Graphics," *ACM Transactions on Graphics*, Vol. 5, No. 1, January 1986, pp. 51–72.

DEER88 Deering, M., S. Winner, B. Schediwy, C. Duffy, and N. Hunt, "The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics," *Computer Graphics* (Proceedings of SIGGRAPH '88), Vol. 22, No. 4, pp. 21–30.

DIPP85 Dippé, M.A.Z. and E.H. Wold, "Antialiasing Through Stochastic Sampling," *Computer Graphics* (Proceedings of SIGGRAPH '85), Vol. 19, No. 3, pp. 69–78.

FOLE90 Foley, J., A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics: Principles and Practice*, Second Edition. Addison-Wesley, Reading, MA 1990.

FUCH85 Fuchs, H., J. Goldfeather, J. Hultquist, S. Spach, J. Austin, F. Brooks, J. Eyles, and J. Poulton, "Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-planes," *Computer Graphics* (Proceedings of Siggraph '85), Vol. 19, No. 3, pp. 111–120.

FUCH89 Fuchs, H., J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, and L. Israel, "A Heterogeneous Multiprocessor for Graphics System using Processor-Enhanced Memories," *Computer Graphics* (Proceedings of Siggraph '89), Vol. 23, No. 3, pp. 79–88.

HAEB90 Haeberli, P. and K. Akeley, "The Accumulation Buffer: Hardware Support for High-Quality Rendering," *Computer Graphics* (Proceedings of SIGGRAPH '90), Vol. 24, No. 4, pp. 309–318.

KAJI86 Kakiya, J.T., "The Rendering Equation," *Computer Graphics* (Proceedings of SIGGRAPH '86), Vol. 20, No. 4, pp. 143–150.

LEE85 Lee, M., R.A. Redner, and S.P. Uselton, "Statistically Optimized Sampling for Distributed Ray Tracing," *Computer Graphics* (Proceedings of SIGGRAPH '85), Vol. 19, No. 3, pp. 61–67.

MAMM89 Mammen, A., "Transparency and Antialiasing Algorithms Implemented with the Virtual Pixel Maps Technique," *IEEE Computer Graphics and Applications*, Vol. 9, No. 4, July 1989, pp. 43–55.

MITC87 Mitchell, Don P., "Generating Antialiased Images at Low Sampling Densities," *Computer Graphics* (Proceedings of SIGGRAPH '87), Vol. 21, No. 4, pp. 65–72.

MITC88 Mitchell, Don P., and Arun. N. Netravali, "Reconstruction Filters in Computer Graphics," *Computer Graphics* (Proceedings of SIGGRAPH '88), Vol. 22, No. 4, pp. 221–228.

MITC90 Mitchell, Don P., "The Antialiasing Problem in Ray Tracing," Coursenotes from Advanced Topics in Ray Tracing, Siggraph 1990.

OPPE75 Oppenheim, A.V. and R.W. Schafer, *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1975.

PAVI90 Pavicic, Mark. J., "Convenient Anti-Aliasing Filters that Minimize 'Bumpy' Sampling," in *Graphics Gems*, Andrew Glassner, ed., Academic Press, San Diego, 1990, pp. 144–146.

RIPL77 Ripley, B. D., "Modelling Spatial Patterns," *Journal of the Royal Statistical Society*, Series B, Vol. 39, No. 2, 1977, pp. 172–212.

SCHR85 Schreiber, William F. and Donald E. Troxel, "Transformation Between Continuous and Discrete Representations of Images: A Perceptual Approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-7, No. 2, March 1985, pp. 178–186.

SHIR90 Shirley, P.S., *Physically Based Lighting Calculations for Computer Graphics*, Ph.D. dissertation, University of Illinois at Urbana-Champaign, 1990.

YELL83 Yellott, J.I.Jr., "Spectral Consequences of Photoreceptor Sampling in the Rhesus Retina," *Science* 221, July 22, 1983, pp. 382–385.

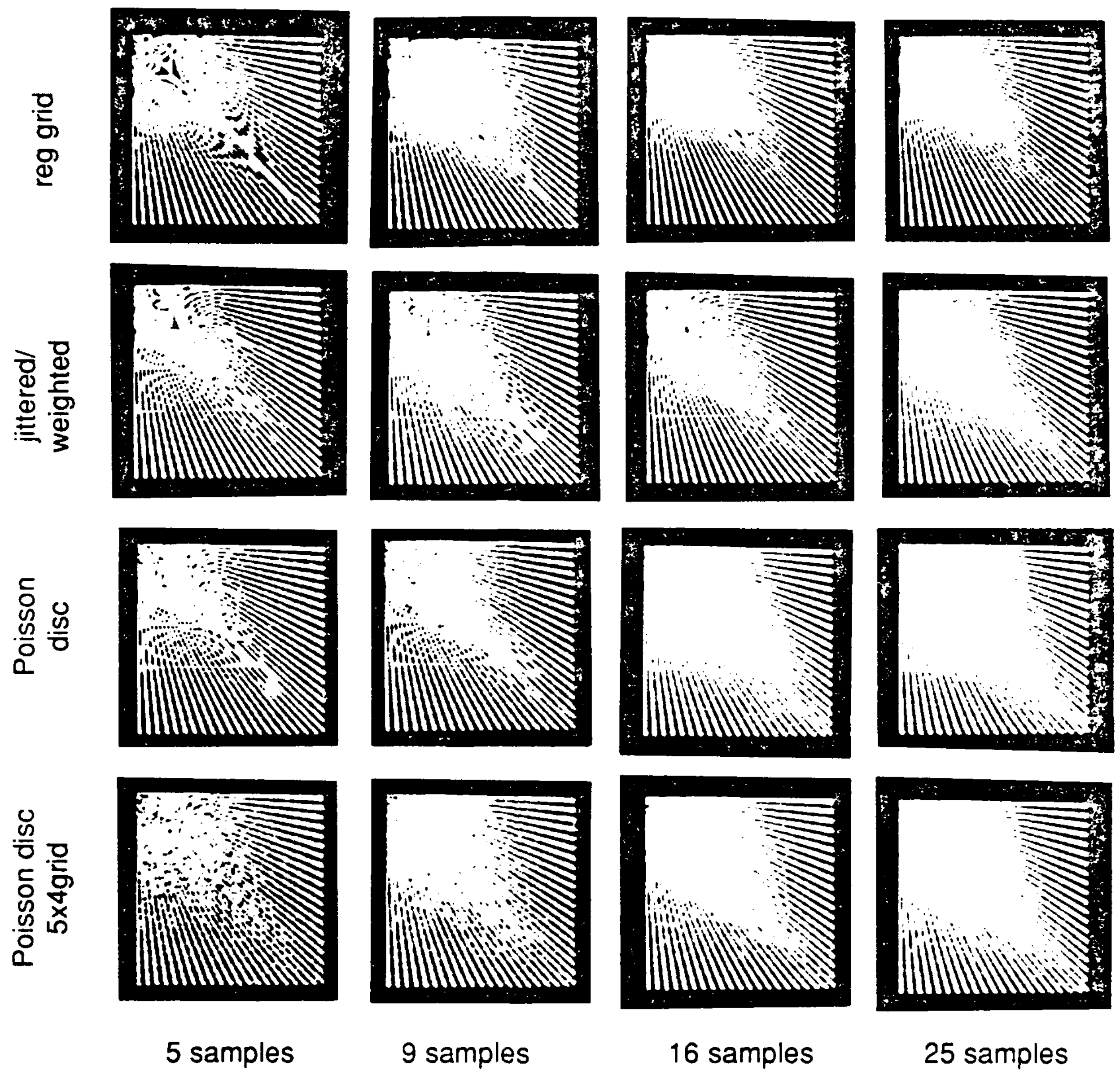


Figure 8: Test pattern containing 40 triangles rendered with four different antialiasing methods and varying numbers of samples per pixel.

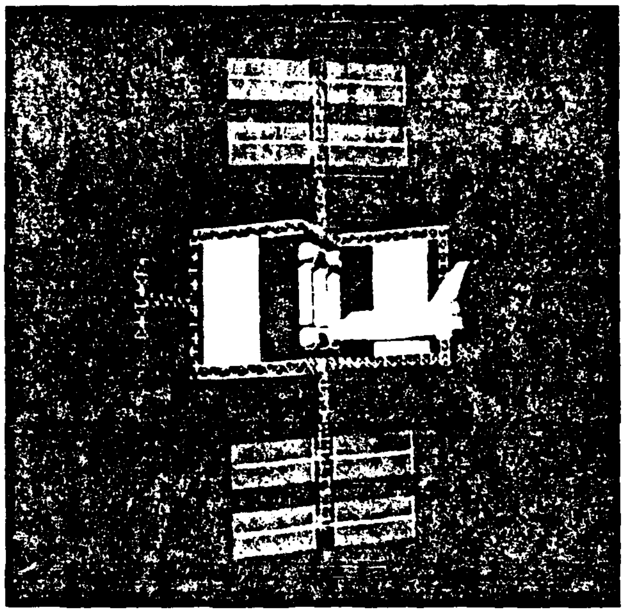


Figure 9: Space station and space shuttle containing 3784 polygons. Image was rendered at 512x512 resolution with a 25-sample Poisson-disk Gaussian kernel.