Antialiasing with Line Samples

Thouis R. Jones, Ronald N. Perry
MERL[1]

Abstract. Antialiasing is a necessary component of any high quality renderer. An antialiased image is produced by convolving the scene with an *antialiasing filter* and sampling the result, or equivalently by solving the *antialiasing integral* at each pixel. Though methods for analytically computing this integral exist, they require the continuous two-dimensional result of visible-surface computations. Because these computations are expensive, most renderers use *supersampling*, a discontinuous approximation to the integral. We present a new algorithm, *line sampling*, combining a continuous approximation to the integral with a simple visible-surface algorithm. Line sampling provides high quality antialiasing at significantly lower cost than analytic methods while avoiding the visual artifacts caused by supersampling's discontinuous nature.

A line sample is a line segment in the image plane, centered at a pixel and spanning the footprint of the antialiasing filter. The segment is intersected with scene polygons, visible subsegments are determined, and the antialiasing integral is computed with those subsegments and a one-dimensional reparameterization of the integral.

On simple scenes where edge directions can be precomputed, one correctly oriented line sample per pixel suffices for antialiasing. Complex scenes can be antialiased by combining multiple line samples weighted according to the orientation of the edges they intersect.

1 Introduction

Aliasing, the masquerading of a signal's high frequencies as low frequencies when the signal is sampled at too low a rate, has plagued computer graphics since its inception. Images (two-dimensional signals) generated from scenes comprised of polygons are particularly prone to aliasing since polygon edges contain infinitely high frequencies which alias as the well-known "jaggies."

The sampling theorem states that a signal can be correctly reconstructed from a regular sampling only if the signal's maximum frequency component is below the Nyquist limit, defined as half the sampling rate. Frequencies above the Nyquist limit appear as low-frequency aliases in the reconstructed signal. The naïve method to reduce aliasing is to represent the signal with more samples (e.g., by adding more pixels to the display), but this is usually unacceptable. The preferred method is to *bandlimit* the signal or image prior to sampling by convolving it with a filter that attenuates the high frequencies. Convolving an image I with a bandlimiting filter F and then sampling at location P can be expressed as the integral

$$S(P_x, P_y) = \iint I(x, y)\, F(x - P_x, y - P_y)\, dx\, dy$$

where S is the resulting sampled image, made up of pixels at the sample points $\{P\}$. We refer to F as the *antialiasing filter*[2] and the equation above as the *antialiasing integral*.

[1]MERL- Mitsubishi Electric Research Laboratory, {rjones,perry}@merl.com

[2]Also known as the *prefilter* or *presampling filter*.

The perfect bandlimiting filter, $sinc$, completely removes frequencies above the Nyquist limit, but has infinite extent and causes "ringing" in images, making it unsuitable for antialiasing in most cases. A smooth, positive filter with finite extent (e.g., a truncated Gaussian) is preferable for efficiency and aesthetics.

There are two common methods for computing the antialiasing integral. *Analytic* methods compute the integral directly from a continuous representation of I. Point sampling methods, or *supersampling*, approximate the integral by sampling I at multiple points around P and computing $S(P)$ as a weighted sum according to F.

For three-dimensional scenes, I is the two-dimensional solution to the visible surface problem. Computing a continuous representation of visible surfaces is non-trivial, which has limited the use of analytic approaches despite efficient methods for computing the integral once visible surfaces are known.

Supersampling results in much simpler visible surface calculations, since visibility need only be determined at independent points. However, supersampling has some negative traits. Since the sampling points are discrete, it is a discontinuous approximation to the antialiasing integral. A small movement of an edge can cause a sample point to change from inside to outside a polygon, resulting in a jump in pixel value. Also, small features can be missed entirely by the sampling. Finally, regularly spaced samples result in patterned aliases, which are easily detected by the human eye. Adaptive and stochastic sampling can alleviate discontinuous behavior and patterned aliases, respectively, but lack a regular structure making them less amenable to efficient implementation.

In this paper we introduce *line sampling*, a new antialiasing method that combines a continuous approximation to the antialiasing integral with a simple visible-surface algorithm. Line sampling computes the antialiasing integral from the intersection of I and a line segment centered at P and spanning the footprint of the filter F, giving a one-dimensional signal that is antialiased analytically according to F. The underlying assumption is that the one-dimensional signal along the line sample is representative of the two-dimensional image I near P. This assumption is valid if the line sample is oriented perpendicularly to nearby edges in I. Unfortunately, determining such a direction is impossible in some cases, and infeasible in general.

In practice, sampling slightly off the optimal direction gives acceptable results, which leads us to taking multiple line samples per pixel and combining their results based on local image features. With two perpendicular line samples, this approach produces high quality antialiased images, comparable to those from analytic methods.

Line sampling's analytic nature with regards to one-dimensional features gives it smooth behavior when those features change slightly, as in animations. This yields much more aesthetically pleasing results, preventing the "twinkling" effects that are visible in animations generated with supersampling (unless a large number of samples are used).

1.1 Related Work

A tutorial on antialiasing is given by Foley in [12], and Joy [16] includes an excellent review of antialiasing methods [1, 4, 5, 6, 7, 8, 9, 11, 17, 23, 26, 27]. More recent work on analytic methods includes [10, 14, 19], all of which assume a precomputed solution to the visible surface problem. Recent work on supersampling includes [20, 21, 22].

An interesting variation of particular relevance is given by Max in [18], which presents a hybrid technique using data from a scanline renderer to antialias scenes, providing a method similar to analytic antialiasing in one direction and regular supersampling in the other. This is roughly equivalent to our method with a single line sample per pixel and each line sample having the same orientation. He extends his method in the

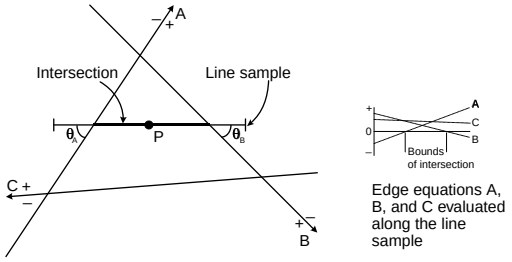

Fig. 1. The intersection of a triangle defined by the edge equations {A, B, C} and a line sample centered at pixel P. The intersection can be computed as the segment along the line sample where the edge equations are all positive, as shown in the graph. θ_A and θ_B are the angles of intersection between the line sample and the edges A and B of the triangle.

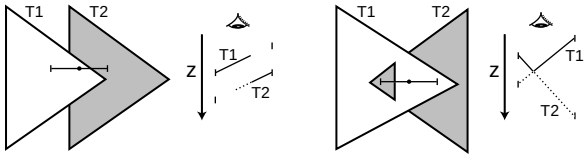

Fig. 2. Visible segment calculation. On the left, triangle T1 partially obscures triangle T2. On the right, T1 and T2 intersect. The depth information is shown in the graphs.

same paper to use data from adjacent scanlines and the slopes of the edges intersected by scanlines to infer a more accurate approximation of the image prior to filtering. An extension of this method is described by Tanaka in [25] in which every polygon's exact coverage is determined by inserting a vertical scanline at each position where an active edge is created, destroyed, or intersects a horizontal scanline. These vertical scanlines are bounded by the two neighboring horizontal scanlines to reduce computational cost. Even so, their method is 4-8 times slower than traditional scanline rendering. Guenter [14] also uses one-dimensional samples, but takes them from a precomputed visible surface solution and uses them to approximate the antialiasing integral via Gaussian quadrature.

We extend Max's work in [18] by using multiple sampling directions simultaneously, by considering independent sampling directions for each pixel, and by combining multiple samples per pixel. We do not use edge slopes in the approximation of I, though our method could be extended to do so. As will be shown (§3.2), our method produces high quality results with a cost of only about twice that of a scanline render.

2 Line Sampling

Computing an approximation to the antialiasing integral at a pixel with line sampling involves three steps: intersecting the line sample (a line segment in the image plane) with polygons in the scene, calculating visible segments along the line sample, and computing the filtered result. These calculations take place after the polygons in the scene have been converted to triangles and transformed to screen space. A line sample can be oriented in any direction; we assume a horizontal sample for the following discussion.

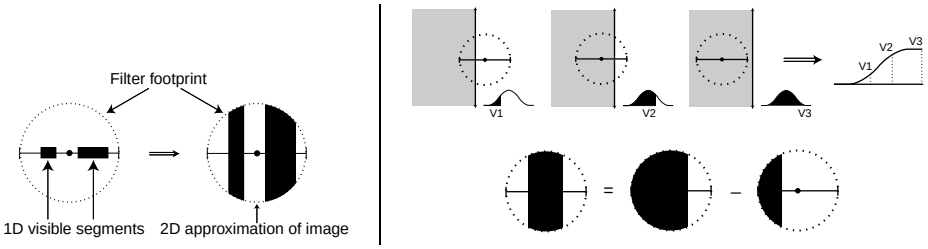

Fig. 3. Left: Extension of visible segments to two dimensions. The visible segments are "stretched" perpendicularly to the line sample within the filter footprint. Right: Computing and applying the one-dimensional summed-area table. The table stores the fraction of the filter covered as a perpendicular edge moves along the line sample. The filtered contribution of any segment can be calculated as the difference of two values from the table.

The intersection of a line sample and a triangle can be calculated by finding the segment of the line sample along which the signed distances to each edge of the triangle are all positive; these distances can be computed via edge equations as in Pineda [24], as shown in Figure 1. Since the edge equations are linear, they need only be evaluated at the endpoints of the line sample[3]. Intersections are calculated for each triangle and buffered for the visibility calculation.

Given the multiple intersections for a line sample, the visible segments along the line sample must be determined. Since depth is a linear function (see Blinn [2]), depth at segment endpoints is sufficient to determine visibility. Segments on a line sample can obscure or intersect each other as in Figure 2, and it is possible that a segment will be split into separate segments by the visibility calculation.

After the visible segments along a line sample are known, the value for the corresponding pixel P is computed as the sum of the filtered contribution of each visible segment. Since the antialiasing filter F is defined in two dimensions but the visible segments are one dimensional, the segments must be extended to two dimensions before filtering. In effect, the line sample acts as a very thin "window" onto the image I from which we must guess the rest of I within the footprint of F. Limited to one-dimensional information, we must assume that I changes only in that dimension, and is static in the other dimension. This is equivalent to "stretching" the visible segments perpendicularly to the line sample as shown in Figure 3.

Rather than computing a two-dimensional approximation of I and then filtering, it is possible to compute the filtered result directly from the visible segments. As shown in Figure 3, the contribution of each visible segment can be calculated using a one-dimensional summed-area table computed from a reparameterization of the antialiasing integral along the line sample.

3 Multiple Line Samples

To properly antialias an image with a single line sample per pixel requires that each line sample be oriented perpendicularly to any nearby edges. This is impossible near corners, and difficult even in simpler cases. Within these limitations, one possible antialiasing method is to render the image without antialiasing and postprocess it with an edge detector to yield approximate edge orientations, similar to the technique used by Bloomenthal [3]. Given these edge orientations, an antialiased image can then be

[3] Analogous to [24], a tiebreaker rule must be used when a line sample is collinear with a triangle edge.

rendered with line sampling. However, aliases in the non-antialiased pass can cause the wrong orientation to be chosen for the line samples, leading to aliasing in the final image. Another method to determine edge orientations, given by Fujimoto in [13], is to project all edges regardless of visibility to the screen. The projected edges could then be used to orient line samples for rendering. However, this can result in obscured edges affecting the orientation of line samples, again leading to aliasing.

Rather than attempting to antialias using a single line sample, we extend line sampling to use multiple line samples at each pixel. To determine the number of line samples necessary per pixel, we consider how close to optimal the orientation of a line sample needs to be to still give acceptable results. We have found that near an edge, line sample orientations within 45° of optimal are adequate (§4.1). Therefore, with two perpendicular line samples per pixel, one of the line samples will produce a value suitable for an antialiased image. Thus, we use two line samples, oriented horizontally and vertically; this simplifies implementation and improves efficiency (§3.2).

3.1 Combining Multiple Line Samples

Given two perpendicular line samples at a pixel, some method must be devised to combine their values. Simple averaging of the two values is unacceptable, equivalent in the worst case to averaging an antialiased image with a non-antialiased one. Some method is needed for weighting the two line samples based on how close one or the other's orientation is to optimal. We compute these weights from the edges that the line samples intersect.

For a line sample that intersects an edge, the accuracy of the antialiasing by that sample depends on the relative orientation of the line sample and the edge, with greater accuracy achieved the closer they are to perpendicular. We use $\sin^2\theta$ as a weight that follows this behavior, with θ the angle between the line sample and the edge, as shown in Figure 1. The relative weights of the two line samples at a pixel are computed as the sum of $\sin^2\theta$ for every intersection of an edge with either line sample at that pixel. Using $\sin^2\theta$ rather than $\sin\theta$ normalizes the total weight from a single intersection.

Once we have relative weights for the two line samples, a blending function is used to combine their values. Early experiments showed that simply choosing the value with the higher weight gave good results, but in some cases such a discontinuous blend causes temporal aliasing. Instead, we perform cubic blending[4] of the values from the line samples according to their relative weights, which gives a smooth approximation of the step function.

Obscured, Created, and Shared Edges. Properly weighting line samples based on edges present in the image requires the determination of whether an edge from the scene data is visible in the image or not. As in Figure 2, some edges might be obscured, or two objects that intersect might create a new edge. Finally, some edges might be shared between adjacent triangles in an object, and should contribute to the weights according to the change in color across the shared edge.

Obscured edges and edges created by intersections are handled as follows. For each triangle that a line sample intersects, the orientation of edges crossed by the line sample (e.g., A and B in Figure 1) and the triangle's surface normal are stored. During the visibility calculation, obscured edges are culled by depth comparison. When segments intersect, the orientation of the created edge is calculated from the cross product of the surface normals of the corresponding triangles.

[4]The cubic blend of values (v_1, v_2) according to the weights (w_1, w_2) is defined as $v = v_1 + (v_2 - v_1)(w^2(3-2w))$ with $w = \frac{w_2}{w_1+w_2}$.

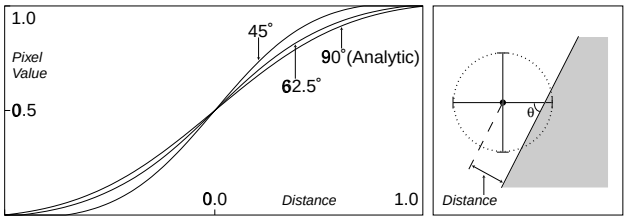

Fig. 4. Plots of line sampling of single edges, oriented at $90°$, $62.5°$, and $45°$ relative to one of the line samples. Plots are of pixel value versus distance from the edge to the pixel center, as shown on the right. Two line samples, oriented horizontally and vertically, were used to generate the data for the plots. Because of symmetry, the plots do not depend on which line sample the orientations are measured from.

Detecting shared edges is trivially accomplished by comparing segment endpoints. Determining how much a shared edge should contribute to the weighting calculation requires a heuristic based on the change in color across the edge. We do not explore such a heuristic in this paper, noting that other antialiasing methods such as proper texture filtering are more relevant in these cases.

3.2 Computing Visible Segments Along Scanlines

When using two line samples per pixel, oriented horizontally and vertically, the horizontal line samples are all subsegments of the horizontal scanlines running through those pixels. Likewise, the vertical line samples are subsegments of "vertical scanlines." It is possible to compute visible segments along entire scanlines at a time, as in a standard scanline renderer, and then extract the visible segments for individual pixels.

Such an implementation of line sampling is equivalent to generating two images via scanline techniques, with the weights of the two line samples being used to perform a per-pixel blend of the two images. Each of the two images is antialiased well in one direction and poorly in the other, and the weights indicate which image is better antialiased at a particular pixel.

With such an implementation, the computational cost of line sampling is not much more than twice the cost of a scanline renderer, and the storage required is proportional to the size of the image being rendered (not including short-term storage for the scanline visibility calculation). After taking into account transformations, shading, and other calculations independent of the antialiasing method, line sampling becomes a viable alternative for achieving high quality antialiasing without a large increase in cost.

4 Results

4.1 Error Analysis for Single Edges

Figure 4 shows the behavior of line sampling applied to a single edge for different edge orientations. Each plot shows the pixel value computed for varying distances from the edge to the pixel center. Two line samples, oriented horizontally and vertically, were used to generate the plots, with a Gaussian filter e^{-2r^2} (truncated at $r = 1$ and normalized), and cubic blending used to combine the values of the two line samples.

For edges oriented at $90°$, line sampling is equivalent to analytic antialiasing. For $45°$ and $62.5°$ edges, the maximum errors compared to the analytic result are 0.09 and 0.03, respectively. The $45°$ orientation is the worst case for a single edge.

When antialiasing a $45°$ edge, the result is equivalent to the analytic solution for a filter with $r = \sin 45° (\approx 0.71)$[5]. In the $62.5°$ case, the computed value is a blend between two filters, one with $r = \sin 62.5° (\approx 0.89)$ and the other with $r = \sin 27.5°$ (≈ 0.46). Cubic blending ensures that the wider filter dominates.

It is also important to notice the smooth behavior of line sampling in all cases as the distance to the edge changes. This prevents temporal artifacts as edges move slightly in animations.

4.2 Images

Figures 5 and 6 compare different antialiasing methods. Figure 5 is a scene comprised of thin triangles scaled and rotated around a single point; Figure 6 is a triangle comb, where each triangle is 1.01 pixels wide at the base and 100 pixels high. Each figure shows four images, two from stochastic sampling (256 or 16 point samples per pixel, shared during filtering), one from line sampling, and one without antialiasing (regular sampling, one sample per pixel). Supersampling with 256 samples is indistinguishable from the analytic solution for these images.

The stochastic sampling and line sampling images were generated with a Gaussian filter (truncated at $r = 1$ and normalized). Two line samples, oriented horizontally and vertically, were used to render the line sampling images, with cubic blending used to combine the values of the two line samples.

The 256-supersampled and line sampling images compare quite favorably. The Moiré artifacts in the line sampling image in Figure 5 are slightly more pronounced near edges oriented at about $45°$; this is due to the effectively narrower filter in those areas (§4.1).

Stochastic noise is quite noticeable in the 16-supersampled images in areas with high frequencies. When animated, these areas "twinkle" distractingly, due to the high frequency content of the closely-spaced parallel edges. Line sampling performs almost as well as analytic antialiasing in those areas, and the same animations generated with line sampling do not suffer from such temporal artifacts.

4.3 Failure Cases

There are some cases where line sampling can fail, primarily because it is unable to properly antialias areas with high-frequency content in multiple near-orthogonal directions. For example, a $90°$ axis-aligned corner can protrude into a pixel without intersecting either line sample. A slight change in its position can cause it to cover both line samples halfway, resulting in a jump in pixel value. Other cases such as sub-pixel polygons and endpoints on thin rectangular polygons are not properly antialiased by line sampling.

Line sampling can also fail due to interactions of patterned scenes with the pattern of line samples. For example, a geometric (rather than textured) checkerboard tilted at $45°$ in screen space could appear all white if the line samples and checkerboard squares were to line up perfectly.

5 Conclusions and Future Work

Line sampling provides high quality antialiasing of polygonal scenes without a large increase in the computational cost of rendering. It provides near-analytic antialiasing

[5] A single line sample interacts with an edge as if the line sample were perpendicular to the edge, but scaled by $\sin\theta$, where θ is the angle between the edge and the line sample.

on one-dimensional features, and prevents most of the distracting "twinkling" artifacts that occur in animations generated with stochastic sampling. Because one-dimensional features are visually important, line sampling's emphasis on properly antialiasing such features is justified.

In the future, we plan to explore using more line samples at each pixel to improve line sampling's robustness. Some of the failure cases might also be prevented by stochastic line sampling, in which the orientations of line samples are chosen randomly for each pixel. We would also like to investigate the natural extension of line sampling to motion blur. Finally, we note that line sampling's per-pixel memory requirements could be unbounded, similar to Carpenter's A-buffer [4]. Recent work by Jouppi [15] on a simplified A-buffer with fixed per-pixel memory leads us to believe that analogous methods might be applied to line sampling.

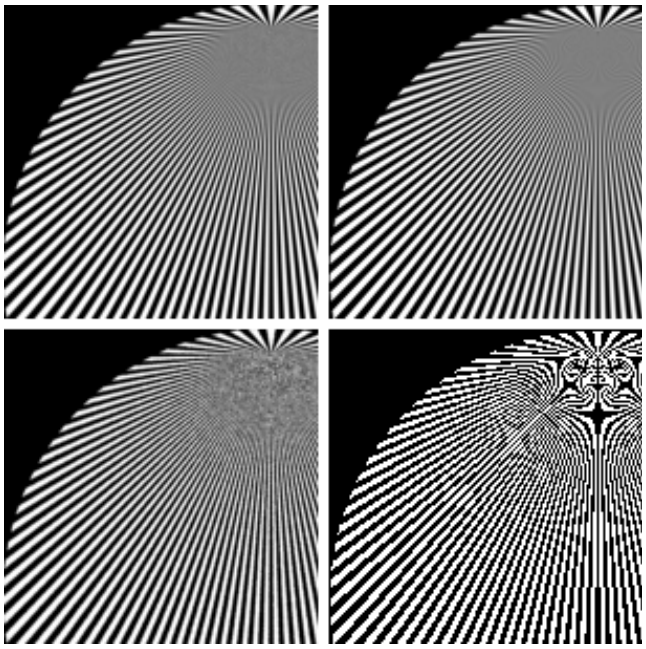

Fig. 5. Clockwise from upper left: stochastic sampling (256 samples/pixel), line sampling, regular sampling (1 sample/pixel), stochastic sampling (16 samples/pixel).

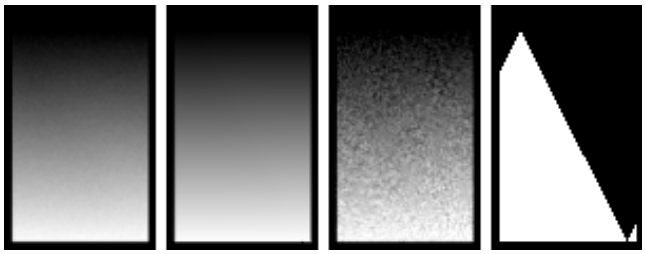

Fig. 6. Left to right: stochastic sampling (256 samples/pixel), line sampling, stochastic sampling (16 samples/pixel), regular sampling (1 sample/pixel). In the rightmost image, the thin vertical triangles of the comb break up completely and alias as large triangles.

References

1. Greg Abram, Lee Westover, and Turner Whitted. Efficient Alias-free Rendering Using Bit-Masks and Look-up Tables. *Proceedings of SIGGRAPH 85*, pages 53–59, July 1985.
2. James F. Blinn. Jim Blinn's Corner: Hyperbolic Interpolation. *IEEE Computer Graphics & Applications*, 12(4), July 1992.
3. J. Bloomenthal. Edge Inference with Applications to Antialiasing. *Proceedings of SIGGRAPH 83*, pages 157–162, July 1983.
4. Loren Carpenter. The A-buffer, an Antialiased Hidden Surface Method. *Proceedings of SIGGRAPH 84*, pages 103–108, July 1984.
5. E. Catmull. A hidden-surface algorithm with anti-aliasing. *Proceedings of SIGGRAPH 78*, pages 6–11, August 1978.
6. Edwin Catmull. An Analytic Visible Surface Algorithm for Independent Pixel Processing. *Proceedings of SIGGRAPH 84*, pages 109–115, July 1984.
7. Robert L. Cook. Stochastic Sampling in Computer Graphics. *ACM Transactions on Graphics*, 5(1):51–72, January 1986.
8. F. C. Crow. A Comparison of Antialiasing Techniques. *IEEE Computer Graphics & Applications*, 1(1):40–48, January 1981.
9. Mark A. Z. Dippé and Erling Henry Wold. Antialiasing Through Stochastic Sampling. *Proceedings of SIGGRAPH 85*, pages 69–78, 1985.
10. T. Duff. Polygon Scan Conversion by Exact Convolution. In *Raster Imaging and Digital Typography*, pages 154–168. 1989. ISBN 0521374901.
11. E. A. Feibush, Marc Levoy, and Robert L. Cook. Synthetic Texturing Using Digital Filters. *Proceedings of SIGGRAPH 80*, pages 294–301, July 1980.
12. James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics, Principles and Practice, Second Edition*. Addison-Wesley, 1990.
13. A. Fujimoto, C. Perrot, and K. Iwata. A 3D Graphics Display System with Depth Buffer and Pipeline Processor. *IEEE Computer Graphics & Applications*, 4(6):11–23, June 1984.
14. Brian Guenter and Jack Tumblin. Quadrature Prefiltering for High Quality Antialiasing. *ACM Transactions on Graphics*, 15(4):332–353, October 1996.
15. Norman P. Jouppi and Chun-Fa Chang. Z3: An Economical Hardware Technique for High-Quality Antialiasing and Transparency. In *Proceedings of the 1999 Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 85–93. 1999.
16. Kenneth I. Joy, Charles W. Grant, Nelson L. Max, and Lansing Hatfield, editors. *Tutorial: Computer Graphics: Image Synthesis*. Computer Society Press, 1988.
17. Mark E. Lee, Richard A. Redner, and Samuel P. Uselton. Statistically Optimized Sampling for Distributed Ray Tracing. *Proceedings of SIGGRAPH 85*, pages 61–67, July 1985.
18. Nelson L. Max. Antialiasing Scan-Line Data. *IEEE Computer Graphics & Applications*, 10(1):18–30, January 1990.
19. Michael D. McCool. Analytic Antialiasing With Prism Splines. *Proceedings of SIGGRAPH 95*, pages 429–436, August 1995.
20. Don P. Mitchell. Consequences of Stratified Sampling in Graphics. *Proceedings of SIGGRAPH 96*, pages 277–280, August 1996.
21. Don P. Mitchell. Generating Antialiased Images at Low Sampling Densities. *Proceedings of SIGGRAPH 87*, pages 65–72, July 1987.
22. Steven Molnar. Efficient Supersampling Antialiasing for High-performance Architectures. Technical Report 91-023, University of North Carolina, 1991.
23. Alan V. Oppenheim and Ronald W. Schafer. *Digital Signal Processing*. Prentice-Hall, 1975. ISBN 0132146355.
24. Juan Pineda. A Parallel Algorithm for Polygon Rasterization. *Proceedings of SIGGRAPH 88*, pages 17–20, August 1988.
25. Toshimitsu Tanaka and Tokiichiro Takahashi. Cross Scanline Algorithm. *Proceedings of EUROGRAPHICS '90*, pages 63–74, 1990.
26. K. Turkowski. Anti-Aliasing through the Use of Coordinate Tansformations. *ACM Transactions on Graphics*, 1(3):215–234, July 1982.
27. Turner Whitted. An Improved Illumination Model for Shaded Display. *Communications of the ACM*, 23(6):343–349, June 1980.