Graphics and Image Processing
J. Foley Editor

The Aliasing Problem in Computer-Generated Shaded Images

Franklin C. Crow
The University of Texas at Austin

Certain defects, such as jagged edges and disappearing detail, have long been an annoyance in digitally generated shaded images. Although increasing the resolution or defocusing the display can attenuate them, an understanding of these defects leads to more effective methods. This paper explains the observed defects in terms of the aliasing phenomenon inherent in sampled signals and discusses prefiltering as a recognized cure. A method for evaluating filters is presented, the application of prefiltering to hidden-surface algorithms is discussed, and an implementation of a filtering tiler is shown accompanied by examples of its effectiveness.

Key Words and Phrases: aliasing, computer graphics, convolutional filtering, hidden-surface removal, sampling

CR Categories: 8.2

The work reported in this paper took place at the University of Utah and was supported in part by the Advanced Research Projects Agency of the Department of Defense under Contracts DAHC15-73-C-0363 and F30602-80-C-0300. Author's address: Department of Computer Sciences, Painter Hall 3.28, The University of Texas at Austin, Austin, TX 78712.

Introduction

Shaded computer-synthesized images of opaque objects with only visible surfaces displayed have become relatively common in recent years. The primary commercial use of such images has been visual simulators, which require the most realistic possible image obtainable at real-time rates. To create realistic images, relatively complicated scenes must be depicted, and defects due to the quantization necessary for computer generation must be minimized.

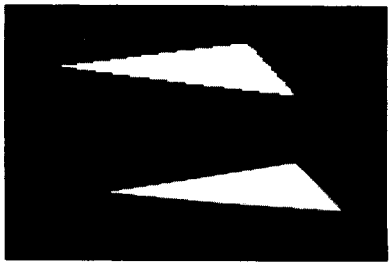
A close look at virtually any shaded synthetic image reveals that major problems exist in the rendition of detail (Figures 1 and 4). These problems characteristically occur in three specific situations: (1) along edges on the silhouette of an object or a crease in a surface, (2) in very small objects, and (3) in areas of complicated detail. The most obvious problems occur on object silhouettes, where edges often have an annoyingly jagged appearance. If computer-synthesized images are to achieve a greater degree of realism, it will be necessary to generate images of arbitrarily complicated scenes which contain many potentially jagged edges, small objects, and details.

Small objects pose a problem because they can disappear between the dots. This occurs because each dot in the image represents a sample point in the scene, an infinitely small spot on some surface being depicted. If an object is small enough it is possible that no part of it will coincide with a sample point. Therefore a very small object may disappear entirely; a long thin object may appear in some places and not in others, giving the appearance of a string of beads; and a highly detailed object such as a human face may lose some of its features.

In animated sequences of images these problems become very obvious. Armies of ants appear to run along edges as their slopes change; small objects and details flash on and off distractingly; slightly larger objects appear to change shape and size without reason; even a simple horizontal edge which looks fine in a still picture can be seen to jump from one raster line to another as it moves vertically in the display.

There are essentially three techniques for improving the rendition of detail. The first is to increase the resolution, causing sample points to occur more frequently. This allows representation of finer details and

Fig. 1. Jagged edges can be attenuated by convolutional filtering. The horizontal resolution in this image is approximately 128 samples.



diminishes the obtrusion of jagged edges. However, it is impractical to increase the resolution sufficiently to accommodate small, bright objects owing to the increased cost of image production. The expense of the most commonly used hidden-surface algorithms is proportional to the resolution, and the number of dots which must be produced grows as the square of the resolution.

The second technique is to process the output image by blurring it or applying contour-smoothing algorithms such as those suggested by Freeman [3]. Although this approach can lessen the impact of jagged edges, it can do nothing to restore objects or details which have been lost. Furthermore, the image loses sharpness which may be retained by other methods.

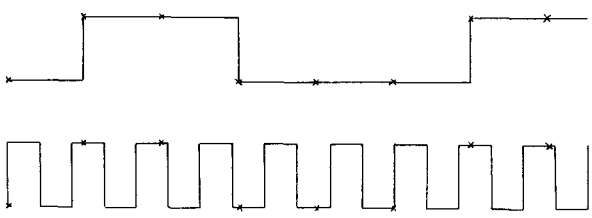
The third and most attractive technique is to make each sample point represent a finite area in the scene rather than an infinitesimal spot. Thus a very small object would occupy a part of such a small area, causing the intensity of the corresponding dot in the output image to be computed as a weighted average of the colors of the small object and its local background. This approach corresponds closely to what actually takes place in television and screen printing processes [5, 10]. While the first two techniques offer somewhat ad hoc approaches to improving rendition of detail, it will be seen that the third technique is based on sound principles.

Making each sample represent a finite area has the effect of applying a convolutional filter before the scene is sampled. It is well known that a signal may be faithfully reproduced from digital samples only if the highest frequency in the signal does not exceed one-half the sampling frequency [7]. Convolutional filtering may be used to satisfy this condition closely enough to greatly improve the output image.

The consequence of failing to filter the signal properly before sampling is known as "aliasing." Aliasing occurs when a lower frequency signal appears as an "alias" of a high frequency signal after sampling (Figure 2). Therefore highly periodic images of scenes involving, for example, picket fences or venetian blinds may appear, when sampled, to be made up of a few broad strips rather than many fine lines.

Reproducing the signal involves representing each sample in such a way that the reproduced signal has no frequencies higher than the original signal. This can be accomplished by representing each sample as a rectangular pulse and then low-pass filtering the resulting signal. In the two-dimensional case, the result of failing to filter the signal properly during reconstruction is known as "rastering." Rastering is an artifact of the structure of the displayed image. If the beam in a television monitor is incorrectly focused, the resulting effects are due to rastering.

Fig. 2. Aliasing. x's represent a sampling rate of 10 samples per unit on 12 cycle and 2 cycle signals. Samples are the same in both cases.



Filtering Shaded Synthetic Images

To produce an image by computer, the scene is first modeled by approximating all surfaces with easily handled entities (e.g. line segments, polygons, or bicubic patches). These entities are then stored in memory at a precision determined by the available word size. If N bits of precision are available, the scene is defined to a resolution of 2^N elements (R_s). To produce an image, the scene definition is sampled at the image resolution (R_i).

To ensure that the high frequencies in the scene do not exceed one-half the sampling rate, the scene must be convolved with a two-dimensional filter. Fast convolution methods [8] involving the two-dimensional fast Fourier transform (FFT) are impractical since taking the FFT would require producing an image of resolution R_s, not to mention the $2R_s^2 \log R_s$ operations each FFT would require. Direct convolution can be much more easily applied since the convolution need only be evaluated at the sample points. This requires R_i^2 times R_f^2 operations, where R_f is the resolution of the filter. If R_f is chosen to be $2R_s/R_i$ (empirically found by us to be adequate), then the number of operations needed to compute the direct convolution is $4R_s^2$, much less than a single FFT. Nevertheless, this is still an excessive amount of computation. In the following discussion, an algorithm which simplifies the computation by approximating direct convolution is presented.

This algorithm assumes a filter which is nonzero over a square region two sample intervals wide and separable into functions in x and y. The scene is abstracted so that all features within the compass of a single superposition of the filter are modeled as rectangular areas of constant intensity. This allows the intensity of an image element to be calculated as a weighted average of the contributions of the rectangular areas. The weighting is, of course, determined by the filter function.

Two-dimensional discrete convolution can be expressed as follows:

$$G(i,j) = \sum_{k=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} F(k,m)H(i-k, j-m). \quad (1)$$

Since H must be separable, it can be expressed as

$$H(i,j) = H_i(i)H_j(j). \quad (2)$$

With this restriction, the discrete convolution becomes

$$G(i,j) = \sum_{k=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} F(k,m)H_i(i-k)H_j(j-m) \tag{3}$$

where G represents the filtered scene produced by convolving the scene F with the filter H.

Furthermore, if the function F is approximated by rectangular blocks, then the function over any such block becomes constant. This greatly simplifies evaluation of the filtered function. The summation can now be rearranged so that the contribution of each rectangular block is independent. The function G then becomes

$$G(i,j) \simeq \sum_{k=p_1}^{q_1} \sum_{m=r_1}^{s_1} C_1 H_i(i-k)H_j(j-m) + \cdots + \sum_{k=p_n}^{q_n} \sum_{m=r_n}^{s_n} C_n H_i(i-k)H_j(j-m) \tag{4}$$

where $[p_n, q_n]$ and $[r_n, s_n]$ represent the bounds of a given rectangular block of intensity C_n and n such blocks give the approximation to the filtered scene at (i,j).

To allow H_i and H_j to be considered separately for a given rectangular block, the summation can be rearranged as follows:

$$\sum_{k=p_n}^{q_n} \sum_{m=r_n}^{s_n} C_n H_i(i-k)H_j(j-m) = C_n \sum_{k=p_n}^{q_n} H_i(i-k) \sum_{m=r_n}^{s_n} H_j(j-m). \tag{5}$$

The implementation of an algorithm for discrete convolution over the scene description becomes relatively easy as a result of this rearrangement. Note that by making the rectangular blocks arbitrarily small an arbitrarily good approximation to G can be obtained.

Implementing the algorithm involves building lookup tables for summations over the functions H_i and H_j. Since H is always of limited nonzero extent, two finite tables can be built, one for H_i and the other for H_j (in practice these have usually been the same). Each table will consist of entries which represent partial sums across the function from the lower nonzero bound to each point below the upper nonzero bound. To obtain the sum over the function between any two nonzero points, it is sufficient to find the difference between the table entries for these two points. With the help of lookup tables, any of the independent summations (see eq. (5)) giving the approximation to G for a given i and j can be found with four lookups, two subtractions, and two multiplications.

To evaluate the filter functions used with the convolution algorithm, a test pattern which emphasizes the defects due to aliasing may be used. A test pattern has been invented which generates moire patterns in response to improperly represented edges and detail.

The pattern is produced by generating almost parallel sections of parabolas by using second-order differences (Figure 3). The curvatures of the parabolas decrease linearly from a maximum on the left to zero on the right. In addition, the distance between any two adjacent parabolas decreases linearly from left to right across the pattern, causing jaggedness along edges to be repeated with slight variation from curve to curve. The effects along groups of curves form elliptical patterns which are much easier to detect than jaggedness along a single edge. Furthermore, toward the right side of the pattern where the detail is too fine to be resolved by the display, similar patterns are caused by improper summing of the details represented in a sample (Figure 4).

Fig. 3. Test pattern consisting of closely spaced parabolic arcs (moire patterns in this figure and some of those in Figures 5–8 are caused by the half-tone printing process).

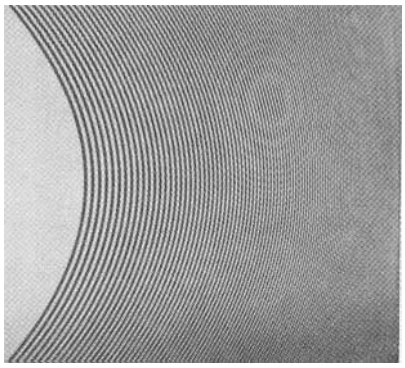
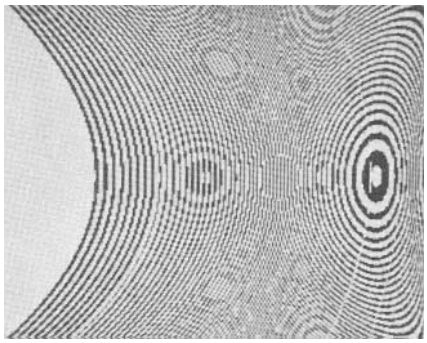


Fig. 4. Test pattern synthesized at a resolution of 256 samples by using techniques similar to those of conventional hidden-surface algorithms.



A program has been developed to display the pattern convolved with various filters. An interactive filter design routine allows quick design and modification of a filter. The pattern can then be regenerated in a few minutes to allow visual evaluation. Equipment calibration routines are also included; the test pattern sensitivity is great enough to make consistent calibration an absolute necessity.

Figures 5–8 illustrate the effectiveness of various filters. In each figure, the curve at the lower left represents the presampling filter while the upper left curve represents the calibration function.

Having developed a method for applying convolutional filters and having found effective filters, we must find methods to restrict filtering to those parts of the image where it is necessary. In other words, the filtering process must be made adaptive.

Fig. 5. Pattern convolved with a filter consisting of nine equally weighted discrete points (equivalent to tripling the resolution).

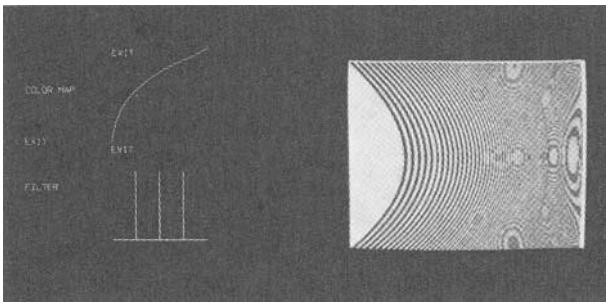


Fig. 6. Pattern convolved with a filter consisting of 25 unequally weighted discrete points.

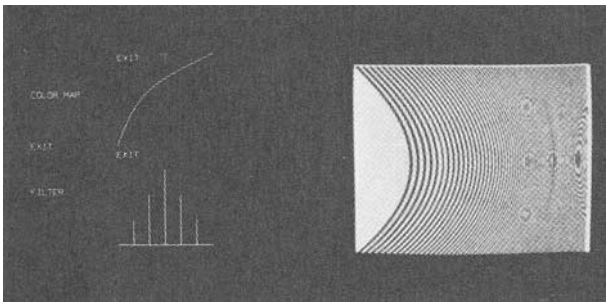


Fig. 7. Pattern convolved with a roughly triangular filter having a base width of one sample interval.

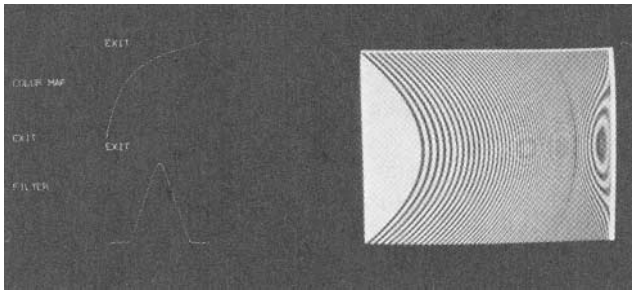
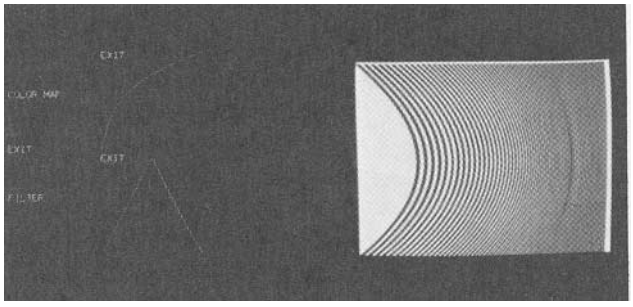


Fig. 8. Pattern convolved with a roughly triangular filter having a base width of two sample intervals.



Improved Efficiency Through Selective Filtering

To isolate most of the conditions which contribute to aliasing before the hidden-surface computation is begun, certain parts of the data can be tagged for special treatment. Tagging the data allows the hidden-surface routine to operate normally over most of the image, applying the more expensive convolution techniques only where necessary.

Nearly all the difficulties in shaded images appear where abrupt changes in intensity and thus high spatial frequencies occur. If the elements of the scene description which cause these occurrences can be tagged, the difficulties can be localized. As noted above, the abrupt intensity changes typically occur in the following three cases: (1) along the silhouette of an object, (2) along creases, corners, or other sharp changes in the direction of a surface, and (3) at the edges of colored patches on a surface.

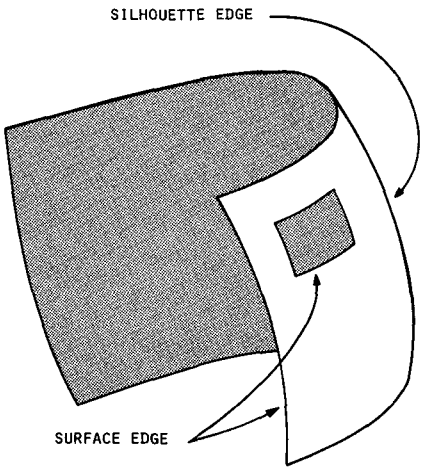
If polyhedral objects are represented, every polygon edge is a potential source of aliasing problems. On the other hand, if curved surfaces are represented by a polygonal approximation, shading techniques may be used to conceal the polygon boundaries over smooth areas [1, 4]. A curved surface approximated by polygons can be made to look smooth by calculating intensities based on the orientation of the surface at the vertices of the polygons and then using interpolation to find the intensities for the rest of the surface. The data structure for describing the polygons is usually arranged so that adjacent polygons can share data where they have common vertices. If there is a sharp change of surface orientation or color across a polygon border, there can be two sets of vertices defining the edge which joins the two polygons. Therefore creases and color changes define two different edges over the same position, and this property can be used to isolate such edges.

Since the most noticeable jaggedness occurs on the silhouettes of objects, it is clearly necessary to find those edges which lie on the silhouettes. Any edge which lies on the silhouette must join a polygon facing the viewer to one facing away from the viewer. Of course an edge associated with only one polygon may also lie on the silhouette of an object. In this case the edge must be a surface edge as opposed to a silhouette edge. A surface edge occurs wherever the surface halts, for example, at the edge of a sheet of paper or a hole in a surface (Figure 9).

To save space, polygons facing away from the viewer (backfacing polygons) are often discarded before the hidden-surface computations are done, in which case silhouette edges become surface edges (and therefore belong to only one polygon). Note that this step cannot be taken until all vertex coordinates have been transformed into the perspective space in which the image will be computed. After backfacing polygons have been discarded, creases, color changes, and silhouette edges occur at edges belonging to only one polygon, a characteristic which can be used to find and tag all such edges.

Although an exhaustive search could be used to find all the edges associated with a single polygon, a far more attractive alternative is to add an adjacent polygons list to the data for each object, providing a pointer to the adjacent polygon for each polygon edge.

Fig. 9. Silhouette edges and surface edges.



All neighboring polygons are then immediately accessible. With this arrangement, a null pointer immediately indicates an edge associated with a single polygon. Without the adjacent polygons list, tagging edges which are adjacent to polygons facing away from the viewer is a difficult task. With the list, a graph in which all adjacent nodes are bidirectionally linked is provided and tagging silhouette edges by nullifying appropriate adjacent polygon pointers is straightforward.

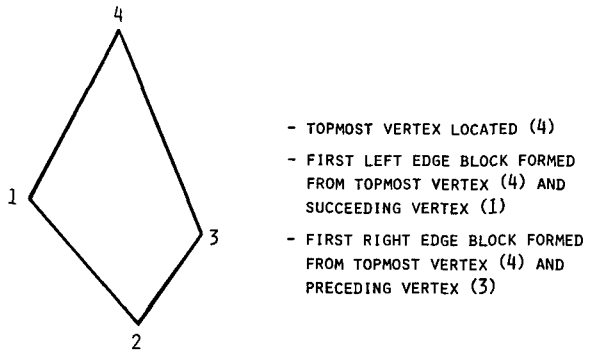
It is then necessary to consider the problem of small objects. It is quite possible to encounter a sharp change in intensity which is not caught by edge tagging. Consider the case of a cube with rounded corners defined by three or four polygons running the length of each edge. When the image of such a cube is large enough that each edge polygon spans several dots in the output image, no problems occur; the rounded edges appear rounded. However, when the cube is viewed from a considerable distance, the total span of the edge polygons may be considerably less than a single dot. In this case, the edge will appear as jagged as it would if the cube were made from the usual six square polygons. Therefore, in addition to tagging edges, it would be wise to tag small or thin polygons.

Having developed a method for efficient application of adaptive convolutional filtering, we must now integrate this method into ordinary hidden-surface algorithms. The following discussion outlines this integration with respect to different classes of hidden-surface algorithms.

Application of Filtering to Hidden-Surface Algorithms

Hidden-surface algorithms for shaded images can be reduced, for this discussion, to three classes: scanning algorithms, in which the image is generated scan line by scan line; depth-priority algorithms, in which the image is generated from the rear forward, without regard to vertical or horizontal order; and depth-buffer algorithms, in which the order of generation is immaterial [9].

Fig. 10. Tiling a convex polygon.



In order to properly compute the intensity at a sample point, all visible surfaces which lie under nonzero areas of the superposed filter must be taken into account. Of the three classes of hidden-surface algorithms, only the scanning algorithms make all necessary information simultaneously available. Both the depth-priority algorithms and depth-buffer algorithms deliver the necessary information for a given sample at intervals while accumulating the image in a frame buffer, and there is no way of knowing whether two surfaces involved in the same sample lie next to each other or overlap.

By using pointers to neighboring polygons, some of these problems can be resolved since neighboring polygons must lie next to each other. This allows creases to be handled correctly. However, where silhouette and surface edges are involved, a correct intensity cannot be guaranteed. If surfaces are rendered in strictly back-to-front order as in some depth-priority algorithms [6], an acceptable edge can be obtained under most conditions. However, the depth-buffer algorithms, which render surfaces in any order, clearly violate this constraint. To be able to calculate the proper intensities where a surface appears behind a previously rendered edge, the intensity of the edge and the extent of its contribution to the sample must be recorded. A more complete discussion of these problems can be found in [2].

It should be noted that where scanning algorithms are used with a frame buffer to achieve greater image complexity by separating foreground and background objects, all the problems of the depth-priority algorithms can be expected. Therefore, if a correct intensity must be guaranteed at every sample point, a single-pass scanning algorithm is required. However, if an occasional error may be accommodated or sufficient memory space and processor time may be devoted to maintaining records on all filtered samples, the frame-buffer-based algorithms can be used.

In the interests of simplicity, the results obtainable with convolutional filtering are demonstrated by using a filtering tiler (a tiler is a procedure which generates

the individual dots, or "tiles," from the description of a polygon). The tiler is simple enough to be described in this space yet demonstrates generally applicable techniques for displaying surfaces.

Implementation of a Filtering Tiler

Only convex polygons are considered in this implementation; thus each scan line intersects a polygon in a single segment. Therefore it is sufficient to establish the position and intensity of the end points for each scan segment and pass them to a shader-interpolator routine which generates the intensity for each sample point. Assuming the tiler proceeds from top to bottom, the polygon is first searched for its highest vertex. The vertices of the polygon are known to be stored in a given order (usually clockwise or counterclockwise). If the vertices are stored clockwise the vertex preceding the top vertex defines an edge which lies to the left of an edge defined by the topmost vertex and the succeeding vertex (Figure 10).

Thus a left-edge block is formed which stores attributes for the left edge; similarly a block is formed for the right edge. The attributes for each block include present position, increments yielding the position at the next scan line, and shading attributes associated with their increments. Each block also includes a count of the number of scan lines remaining until the bottom of the line segment is reached. The increments are used to update the edge blocks after each invocation of the shader-interpolator routine. When a vertex is reached, the appropriate edge block must be recalculated to reflect the attributes of the edge below. The algorithm terminates when the next vertex for a block lies above the current one, when both edge blocks reach the same vertex, or when the lowest extent of the polygon, determined by the bottom-most vertex, is reached.

Figure 11 shows a flowchart for the tiler just described, and Figure 12 shows the tiler extended to include a presampling filter. One important difference between the filtering and nonfiltering tilers is that the filtering tiler may not "ignore" edges with a vertical range of less than one scan line. In particular, an edge block which lies along the top or bottom of a polygon, and thus may have considerable horizontal extent, must be properly filtered.

The filtering and nonfiltering tilers also differ in their treatment of edge blocks. In the nonfiltering tiler, edge blocks which may be "ignored" are immediately marked "done" and a new edge block made. The filtering tiler, on the other hand, must keep track of as many edge blocks as may affect intensities on a given line. Therefore a queue of edge blocks must be provided for both the left and right sides. In practice, the length of these queues rarely exceeds two edge blocks, and images can usually be made by using queues restricted to that length.

Fig. 11. Conventional tiler for convex polygons.

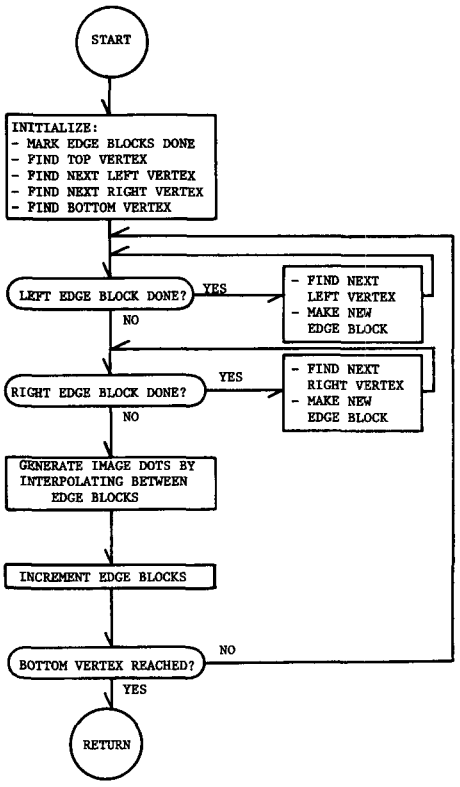


Fig. 12. Filtering tiler for convex polygons.

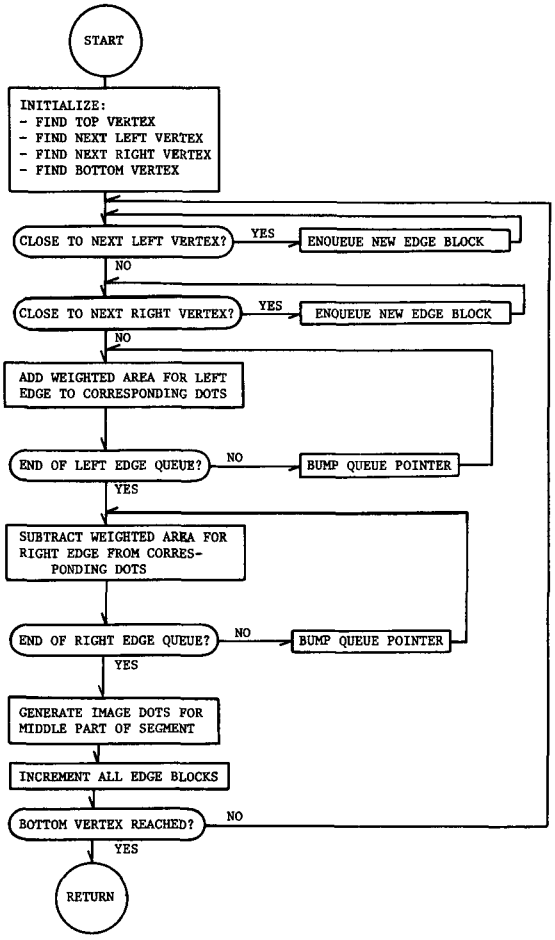


Fig. 13. Calculating the weighted area of a small polygon.

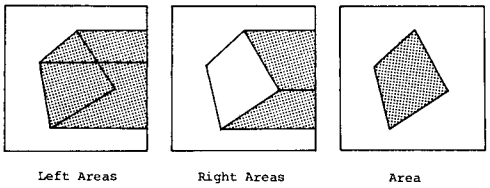


Fig. 14. Three images each from the filtering tiler (left side), the conventional tiler (middle, top to bottom), and a doubled-resolution tiler (lower right) displayed together.

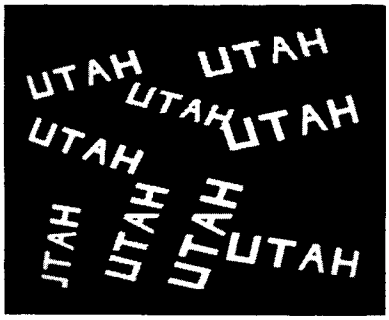
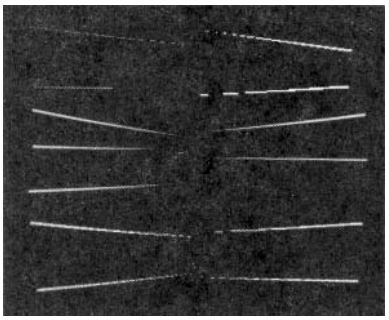


Fig. 15. A particularly difficult object, a slender, nearly horizontal triangle, rendered by the conventional tiler (top four), the filter tiler (middle five) and the doubled-resolution tiler (bottom four).



The process of filtering the left and right edges proceeds as described earlier. The approximate area lying to the right of each left edge in each intersecting filtered area is weighted and then added to the intensity for the corresponding image dot. Conversely, the areas lying to the right of right-hand edges are subtracted from the intensity for the affected image dots. Note that since the final sum is the weighted area covered by the polygon, very small polygons are treated correctly (Figure 13). Figures 14 and 15 compare the filtering tiler with conventional tilers.

A detailed evaluation of the performance of the filtering tiler in comparison with conventional tilers awaits further research. However, observed execution times were two to five times longer for the filtering tiler than for a conventional tiler working at the same resolution. Such figures can be expected to range widely, varying inversely with the size of the polygons displayed and the amount of computing overhead included.

Conclusion

The intention here has been to provide a discussion of the aliasing problem and to offer a solution based on the theory of sampling and reproduction of two-dimensional signals. The examples shown here have been chosen to illustrate aliasing at its worst. Although such cases can sometimes be avoided in still pictures, animated sequences virtually always exhibit obvious defects due to aliasing. It follows that genuinely realistic images will require more than token efforts at resolving the problems discussed above. A general approach to a solution has been offered here. Further ideas and more specific suggestions are offered in [2].

Acknowledgments. This paper has been greatly improved by the helpful comments of the referees. They deserve praise for their careful reading of an earlier version. It should also be noted that the presence of excellent research groups in both computer graphics and signal processing at the University of Utah made this work possible.

References

1. Bui Tuong Phong. Illumination for computer-generated images. UTEC-CSc-73-129, Dept. Comptr. Sci., U. of Utah, Salt Lake City, Utah, July 1973. Abridged in *Comm. ACM 18*, 6 (June 1975), 311-317.

2. Crow, F.C. The aliasing problem in computer-synthesized shaded images. UTEC-CSc-76-015, Dept. Comptr. Sci., U. of Utah, Salt Lake City, Utah, March 1976.

3. Freeman, H. Computer processing of line-drawing images. *Computing Surveys 6*, 1 (March 1974), 57-97.

4. Gouraud, H. Computer display of curved surfaces. UTEC-CSc-71-113, Comptr. Sci., U. of Utah, June 1971. Abridged in *IEEE Trans. Comptrs. C-20* (June 1971).

5. Hunt, R.W.G., *The Reproduction of Colour in Photography, Printing and Television*. Fountain Press, England, 3rd Ed., 1975.

6. Newell, M.G., Newell, R.G., and Sancha, T.L. A solution to the hidden-surface problem. Proc. ACM 1972 Annual Conf., Boston, Mass., Vol. I, pp. 443-450.

7. Oppenheim, A.V., and Schafer, R.W. *Digital Signal Processing*. Prentice-Hall, Englewood Cliffs, N.J., 1975.

8. Stockham, T.G. Jr., High-speed convolution and correlation. Proc. AFIPS 1966 SJCC, Vol. 28, AFIPS Press, Montvale, N.J., pp. 229-233.

9. Sutherland, I.E., Sproull, R.F., and Schumaker, R.G. A characterization of ten hidden-surface algorithms. *Computing Surveys 6*, 1 (March 1974), 1-55.

10. Zworykin, V.K., and Morton, G.A. *Television*. Wiley, New York, 2nd Ed., 1954.