Filtering Edges for Gray-Scale Displays

Satish Gupta
Robert F. Sproull
Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

While simple line-drawing techniques produce "jagged" lines on raster images, more complex anti-aliasing, or filtering, techniques use gray-scale to give the appearance of smooth lines and edges. Unfortunately, these techniques are not frequently used because filtering is thought to require considerable computation. This paper presents a simple algorithm that can be used to draw filtered lines; the inner loop is a variant of the Bresenham point-plotting algorithm. The algorithm uses table lookup to reduce the computation required for filtering. Simple variations of the algorithm can be used to draw lines with different thicknesses and to smooth edges of polygons.

Keywords: Computer Graphics, line-drawing, anti-aliasing, gray-scale, raster displays

CR Categories: 8.2, 5.25

This research is supported by the Defense Advanced Research Projects Agency, Department of Defense, ARPA Order 3597, monitored by the Air Force Avionics Laboratory under contract F33615-78-C-1551. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

1. Introduction

Computer-generated images that are not properly filtered and sampled display annoying visual effects known as *aliasing*. The most renowned of these effects is the "jaggy" or "staircase" appearance of lines and edges that arises from sampling errors. On a display device that can present gray-scale images, these effects can be avoided by displaying lines and edges using gray values lying between white and black. To determine these intermediate intensities, the image must be filtered using an appropriate low-pass filter before it is sampled [2]. This task can be computationally expensive.

In this paper we present an efficient algorithm for producing images with smooth edges of lines and polygons. We will describe the algorithm used to draw straight lines with unit thickness, and then discuss its variations for lines of different thicknesses and for edges of polygons. The algorithm is a variation of the point-plotting algorithm developed by Bresenham [1], which requires one integer comparison and one integer addition in the inner loop. The inner loop of our algorithm requires slightly more arithmetic precision and a table-lookup step. Unlike Bresenham's algorithm, line endpoints must be treated as a special case, and are the subject of Section 4.

This paper does not present a new filtering technique, but rather shows how existing filtering techniques can be implemented efficiently. Doubtless algorithms of the sort we present are already in use; our objective is to promote the use of filtering by publicizing a simple method.

2. Filtering

Filtering an image with a low-pass filter is an averaging process: the intensity of a pixel is determined by the image brightnesses within a small distance of the pixel, not by the brightness at a single point, such as the pixel's center. Filtering is controlled by a *filter function*, which describes the spatial distribution of light emitted by a pixel on the display. The filter function in effect supplies a weighting function for the averaging process. Filter functions used to prepare images need not match the light emission properties of the display exactly; in practice, a filter function is selected for its mathematical tractability and is altered until the image appears acceptable [7].

A popular approximate filter is a conical function: the function has its maximum value at the center of a pixel and decreases linearly to zero at a distance r from the pixel center. The radius of the filter is r, measured using the convention that a unit distance is the distance between two adjacent pixel centers. The filter function is normalized so that the enclosing volume is 1. Figure 1 shows a filter function of radius 1, which we shall use for the remainder of this paper.[1]

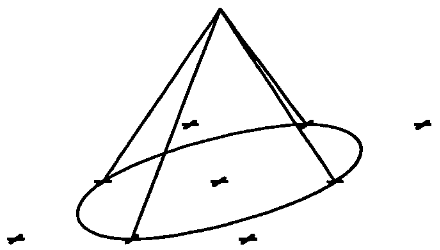

Figure 1: Conical pixel filter. Crosses mark pixel centers.

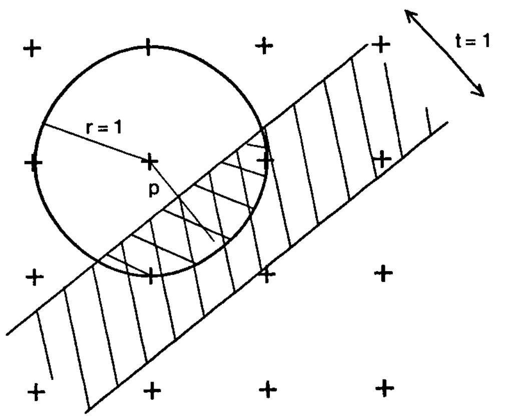

Figure 2: Line intersecting a pixel.

$\|p\|$	$F(p,1)$	$\|p\|$	$F(p,1)$
0/16	0.780	13/16	0.228
1/16	0.775	14/16	0.184
2/16	0.760	15/16	0.145
3/16	0.736	16/16	0.110
4/16	0.703	17/16	0.080
5/16	0.662	18/16	0.056
6/16	0.613	19/16	0.036
7/16	0.558	20/16	0.021
8/16	0.500	21/16	0.010
9/16	0.441	22/16	0.004
10/16	0.383	23/16	0.001
11/16	0.328	$\geq$24/16	0.000
12/16	0.276		

Table 1: Values of pixel intensity given distance to line.

When a line passes through a pixel, the pixel's intensity should be proportional to the volume of the cone intersected by the line (see Figure 2). Because of the circular symmetry of the filter, only two parameters are needed to determine the volume intersected: the thickness t of the line and the perpendicular distance p from the pixel center to the line center. Thus we may write $I = F(p,t)$, where F is determined solely by the choice of filter function. Table 1 illustrates values of F for lines of thickness 1, assuming a conical filter of radius 1. Note that $F(p,t) = 0$ for $p \geq r + t/2$. The table is obtained by convolving the filter function and the line intensity, i.e., by numerically integrating the filter function over the region covered by the line. Similar tables may be built for other values of t and r or for other filter functions.

3. The algorithm

The algorithm to draw a line must compute the distance p between each pixel center and the line. To reduce computation, this calculation is performed incrementally, as in Bresenham's algorithm [1]. Our discussion is restricted to lines of unit thickness in the first octant (i.e., $0 \leq y \leq x$); extensions to other regions are obvious. Such lines intensify two or three pixels in each column of pixels[2] (see Figure 3). The algorithm will keep track of the location of the center pixel and the perpendicular distance to the line's center from the pixel center.

Suppose a line is to be drawn from (x_1, y_1) to (x_2, y_2). We shall assume that line endpoints lie at pixel centers, and hence that x_1, y_1, x_2, and y_2 are integers. If $dx = x_2 - x_1$ and $dy = y_2 - y_1$, then from our first octant assumption, the slope $m = dy/dx$ has a value between 0 and 1. In algorithm A1 below, x and y track the central pixel in each column through which the line passes, and v is the vertical distance from that pixel to the line. This vertical distance v is a signed value; a positive value indicates that the center of the line is above the center of the pixel, and a negative value indicates the opposite. The variable s is a threshold distance used to decide whether the central pixel in the next column lies diagonally or horizontally across from the central pixel in the current column.

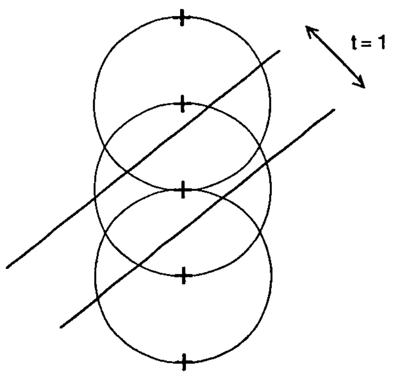

Figure 3: Three pixels are shaded in each column.

[1]Selecting a filter radius is a compromise between a desire for very smooth edges (choose a large r) and preserving high spatial frequencies (choose a small r). A filter of radius 1, on most displays, will produce edges that still appear somewhat wavy. A filter of radius 1.5 will produce quite smooth edges.

[2]The algorithms and tables presented in this paper are designed to produce images using 4-bit intensity values. A diagonal line at almost 45 degrees will actually intersect five rather than three pixels, but the top and bottom pixels are intensified at less than 0.2% of the maximum. Our algorithm ignores these pixels because a 4-bit intensity value will record zero for such an intensity. If a wider range of intensities is available, the algorithm may be modified in an obvious way to illuminate more pixels in each column; the tables must also provide more precision.

```
Algorithm A1:

PROCEDURE PlotLine(x1,y1,x2,y2 : INTEGER);
    VAR x,y : INTEGER; v,m,s : REAL;
    m := (y2-y1)/(x2-x1);
    v := 0; s := 0.5-m;
    y := y1;
    FOR x := x1 TO x2 DO
    BEGIN
      Shade pixels at (x,y-1),(x,y),(x,y+1);
      IF (v ≥ s) THEN
        BEGIN
          y := y+1;
          v := v+m-1;
        END
      ELSE
          v := v+m;
    END;
```

The pixel at (*x*, *y*) is located at a vertical distance *v* from the line, and the pixels at (*x*, *y*−1) and (*x*, *y*+1) are at distances *v*−1 and *v*+1 respectively.

In order to determine the shade of the pixels, we need to compute the perpendicular distance *p* from the pixel to the line. The vertical distances are related to the perdendicular distances by a factor of $c = dx/sqrt(dx^2+dy^2))$, such that $p = cv$. Algorithm A2 shows the modifications to compute perpendicular distances.

```
Algorithm A2:

PROCEDURE PlotLine(x1,y1,x2,y2 : INTEGER);
    VAR x,y : INTEGER; p,m,c,s : REAL;
    m := (y2-y1)/(x2-x1);
    c := 1/sqrt(m*m+1);
    p := 0; s := (0.5-m)*c;
    y := y1;
    FOR x := x1 TO x2 DO
    BEGIN
      Shade pixels at (x,y-1),(x,y),(x,y+1);
      IF (p ≥ s) THEN
        BEGIN
          y := y+1;
          p := p+(m-1)*c;
        END
      ELSE
          p := p+m*c;
    END;
```

The two expressions $(m-1)*c$ and $m*c$ can be precomputed and do not have to be computed repeatedly in the inner loop. To compute the pixel shades, the absolute values of *p*, *p*−*c*, and *p*+*c* are used as indices into Table 1 to determine intensities at (*x*, *y*), (*x*, *y*−1) and (*x*, *y*+1) respectively.

4. Line endpoints

The algorithm presented in the previous section does not compute the sampled values of pixels lying on or near the endpoints of the line. The situation is illustrated in Figure 4, which shows an endpoint of a line. The pixels shown with their surrounding filters are not intensified properly by algorithm A2; in fact, some of them are not intensified at all. There are several methods available to compute such pixel intensities.

As for other pixels near the line, the intensity of one of the pixels is obtained by convolving the filter function and the line, i.e., by integrating the filter function over the region covered by the line. This computation may be performed exactly using geometric operations [3], or may be approximated by sampling the image at points much more closely spaced than pixel centers. Both of these approaches are computationally expensive.

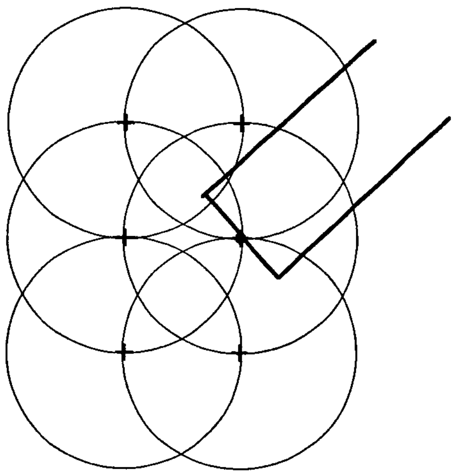

Figure 4: Endpoint of a line, marked by a large dot, showing the six pixels that may be intensified.

Slope = 0/16		Slope = 1/16		Slope = 2/16	
0.000	0.056	0.000	0.063	0.000	0.072
0.000	0.393	0.000	0.390	0.000	0.390
0.000	0.056	0.000	0.048	0.000	0.042
Slope = 3/16		Slope = 4/16		Slope = 5/16	
0.000	0.082	0.000	0.094	0.000	0.107
0.000	0.390	0.000	0.390	0.000	0.390
0.000	0.037	0.000	0.032	0.000	0.028
Slope = 6/16		Slope = 7/16		Slope = 8/16	
0.000	0.121	0.000	0.136	0.000	0.152
0.001	0.390	0.001	0.390	0.002	0.391
0.000	0.025	0.000	0.022	0.000	0.019
Slope = 9/16		Slope = 10/16		Slope = 11/16	
0.000	0.169	0.000	0.187	0.000	0.206
0.002	0.390	0.003	0.390	0.003	0.390
0.000	0.017	0.000	0.015	0.000	0.013
Slope = 12/16		Slope = 13/16		Slope = 14/16	
0.000	0.225	0.000	0.245	0.000	0.264
0.004	0.390	0.005	0.390	0.006	0.390
0.000	0.012	0.000	0.010	0.000	0.009
Slope = 15/16		Slope = 16/16			
0.001	0.284	0.001	0.304		
0.007	0.390	0.007	0.391		
0.000	0.008	0.000	0.007		

Table 2: Endpoints for lines with different slopes.

The pixel intensities can be precomputed and stored in a table, as we did for lines in the preceding section. To display the endpoint of a line, the intensities of the six pixels in the vicinity are determined from the table. For the accuracy we desire, it is sufficient to compute a set of endpoints for lines with slopes between 0 and 1 at intervals of 1/16 (see Table 2). The entries in the table are in the same configuration as the six pixels shown in Figure 4. Using a combination of mirroring and transposition transformations, these endpoint intensities can be used for lines in every octant.

5. Variations

Simple variations of the algorithm presented above can handle lines of different thicknesses and can produce smooth edges for polygons.

Lines of different thickness can be produced by preparing different tables for various line thicknesses and using the thickness of the line to select an appropriate table. In addition, if the thickness is greater than one, the algorithm must be changed to illuminate more than three pixels in each column. The endpoint table must be modified, because more than six pixels may be illuminated near wide lines.

Alternative geometries for line endpoints (see Figure 5) can be accommodated by building separate endpoint tables. The tables are built by numerically integrating the filter function in the region covered by the line.

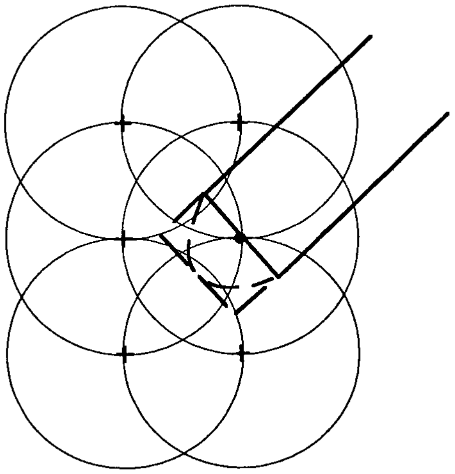
Figure 5: Alternative endpoint geometries for a line.

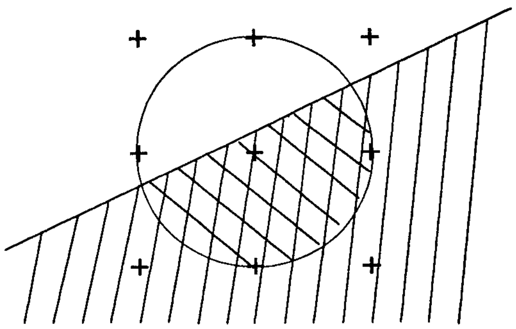
Figure 6: Polygon edge intersecting a pixel.

Polygon edges can be produced in a similar manner by having a special table which contains the intensity values when the pixel is covered by the edge of a polygon (see Figure 6). Table 3 is used to look up the intensity of the pixel based on the perpendicular distance p from the edge of the polygon to the center of the pixel; p is negative if the pixel center lies outside the edge. Note that if $p \leq -r$, $F(p) = 0$; if $p \geq r$, $F(p) = 1$. The tables required to draw lines can be derived from Table 3 because the part of the pixel covered by a line is the difference between the parts of the pixel covered by the two edges of the line. However, because line drawing is a frequent operation, it is advantageous to compile separate tables.

p	$F(p)$	p	$F(p)$
≤-16/16	0.000	1/16	0.559
-15/16	0.001	2/16	0.617
-14/16	0.004	3/16	0.672
-13/16	0.010	4/16	0.724
-12/16	0.021	5/16	0.772
-11/16	0.036	6/16	0.816
-10/16	0.056	7/16	0.855
-9/16	0.080	8/16	0.890
-8/16	0.110	9/16	0.920
-7/16	0.145	10/16	0.944
-6/16	0.184	11/16	0.964
-5/16	0.228	12/16	0.979
-4/16	0.276	13/16	0.990
-3/16	0.328	14/16	0.996
-2/16	0.383	15/16	0.999
-1/16	0.441	≥16/16	1.000
0/16	0.500		

Table 3: Pixel intensities given distance to edge of polygon.

Another desirable variation is the ability to experiment with different filters, because the aesthetic appearance of the output depends upon the filter used for anti-aliasing. The optimal filter will be different for different output devices. In this paper we have used a circularly symmetric filter which has the property that only the distance to the line is needed to compute the intensity of pixels. The use of an asymmetric filter will require knowledge of the slope of the line to compute intensities. This does not affect the algorithm significantly because we can use a few bits of the slope (2 or 3 depending upon the filter) to select a table that can then be used by our algorithm.

The algorithm may be easily adapted to display lines of any shade on backgrounds of any shade by mixing intensities. The pixel intensity is $I = I_L F(p,t) + I_B(1-F(p,t))$, where I_L is the line shade, and I_B is the background shade. On color displays, the red, green and blue components can be mixed independently.

The algorithm can be adapted to display lines and edges drawn between endpoints that are not integers; that is, that do not lie on the pixel grid. Such a scheme is necessary to avoid additional aliasing, for example the jitter in a moving image caused by quantizing endpoints to lie on pixel centers. To accommodate non-integer endpoints, two modifications must be made. First, the initialization of algorithm A2 must be changed to compute the (x, y) coordinate of the first pixel to be illuminated and to compute the initial value for p at this point. Algorithm A3 shows these modifications.

Algorithm A3:

```
PROCEDURE PlotLine(x1,y1,x2,y2 : INTEGER);
    VAR x,y : INTEGER; p,m,c,s : REAL;
    m := (y2-y1)/(x2-x1);
    c := 1/sqrt(m*m+1);
    y1 := y1+m*(round(x1)-x1);
    y := round(y1);
    p := (y1-y)*c ;
    s := (0.5-m)*c;
    FOR x := round(x1) TO round(x2) DO
    BEGIN
      Shade pixels at (x,y-1),(x,y),(x,y+1);
      IF (p ≥ s) THEN
        BEGIN
          y := y+1;
          p := p+(m-1)*c;
        END
      ELSE
          p := p+m*c;
    END;
```

The second change is that samples near endpoints need to take account of the exact location of the line endpoint. We either have to spend a lot of processing to filter these pixels or use a much larger table to look up the filtered values. If the table contains endpoints at intervals of 1/16 for each of the coordinates and an interval of 1/16 for the slope, then the size of the table at six pixels per endpoint would be 29478 entries!

6. Precision

One of the major accomplishments of Bresenham's algorithm is to perform line drawing computations using only integer additions and comparisons. Although we use floating-point numbers in the exposition above, the algorithms do not require the large range provided by a floating-point representation. To understand the precision required, let us examine the variable p in algorithm A2. Since the intensity table is spaced at intervals of 2^{-4}, we need to know p accurately to within 2^{-4} to find the correct intensity value. Assume that the value of p is computed incrementally using at most 2^{10} additions for a display that is 1024 pixels wide. Consequently, the expressions $m*c$ and $(m-1)*c$ must be known to an accuracy of 2^{-15} (i.e., if the error in representing one of these numbers is as much as 2^{-15}, 2^{10} additions will accumulate an error of 2^{-5}, which is not enough to introduce an error in the intensity selected). Thus p can be replaced in the algorithm by an integer $q = 2^{15}p$; since p lies between -0.5 and +0.5, q will lie between -2^{14} and $+2^{14}$. We observe too that $p = (q/2^{11})/16$, and therefore that $q/2^{11}$ can be used as an index into Table 1.[3] The precision required in the incremental computation of p is a direct consequence of the size of the display (the number of incremental additions) and the intensity precision we are trying to achieve. If the function $F(p, t)$ were linear, in order to achieve n bits of gray-scale precision, n bits of distance precision would be required, and therefore the lookup tables would record entries spaced a distance 2^{n} apart. Although F is not precisely linear, the deviations are sufficiently small that the distance precision need be no greater than the intensity precision. The choice of gray-scale precision is an aesthetic one, which depends upon factors such as type of output device, viewing distance, etc. [4]. Thus the choice of intensity precision will determine the number of entries needed in the various tables of the algorithms.

[3]The precision problem is actually somewhat more intricate than this paragraph implies. The problem is that even the tiniest error in the value of p may change the outcome of the test in the inner loop. However, the shading will not be affected by this change, because the distance p is still sufficiently accurate. When the Bresenham algorithm is used on binary devices, this error would result in a non-optimal line being displayed [6].

7. Conclusion

Our objective in presenting a fast algorithm for producing properly filtered images of lines and edges on gray-scale displays is to demonstrate that filtering need not be expensive. The algorithm achieves its speed by using table lookup to avoid complex filtering computations. The computations are so simple that it is not reasonable to exploit gray-scale and not filter properly (e.g., in [5]).

The algorithm leaves open the important problem of sampling two or more interacting lines or edges properly. The algorithm presented by Feibush *et. al.* [3] treats this problem properly, but at considerable computational expense. Approximations often produce usable results, e.g., taking the maximum intensity or multiplying the intensities of the two objects that lie in a single pixel.

References

[1] Bresenham, J.E.
Algorithm for computer control of a digital plotter.
IBM Systems Journal 4(1):25-30, July, 1965.

[2] Crow, F.C.
The aliasing problem in computer-generated shaded images.
Comm. ACM 20:799-805, Nov., 1977.

[3] Feibush, Eliot A., Levoy, Mark, and Cook, Robert L.
Synthetic Texturing Using Digital Filters.
Computer Graphics 14(3):294-301, July, 1980.

[4] Leler, William J.
Human Vision, Anti-aliasing, and the Cheap 4000 Line Display.
Computer Graphics 14(3):308-313, July, 1980.

[5] Pitteway, M.L.V. and Watkinson, D.J.
Bresenham's Algorithm with Grey Scale.
CACM 23(11):625-626, November, 1980.

[6] Sproull Robert F.
Using Program Transformations to Derive Line-Drawing Algorithms.
Technical Report, Carnegie-Mellon University, Computer Science Department, 1981.

[7] Warnock John E.
The Display of Characters Using Gray Level Sample Arrays.
Computer Graphics 14(3):302-307, July, 1980.