THE APPLICATION OF AREA ANTIALIASING ON RASTER IMAGE DISPLAYS

Yu-Tse Chen and P. David Fisher
Department of Electrical Engineering
Michigan State University, East Lansing, MI 48824
and
Michael D. Olinger
Lear Siegler, Inc./Instrument Division
Grand Rapids, MI 49508

ABSTRACT

Most drawing algorithms produce aliasing images while rendering geometric objects in computer graphics applications. The use of gray levels is a very common technique in applying antialiasing algorithms on images. Area antialiasing algorithms assign the intensity level related to the occupied area of the drawings for each pixel. These algorithms can operate at very high speeds from their given intensity level to each pixel while drawing scenes. In this paper, we develop an area antialiasing algorithm with more accurate results and much flexible operation than previous algorithms. This algorithm was designed so that it could be efficiently implemented in a parallel hardware architecture. It uses the midpoint distance to calculate the shaded area and is suitable for most curve drawing applications. The use of this antialiasing algorithm is illustrated with examples of lines with variable thickness and circle antialiasing operations. General considerations involved in using of this algorithm for curve drawings are discussed including an algorithmic complexity analysis.

1. Introduction

In computer graphics applications, aliasing is a display degradation problem resulting from inaccurate digital sampling and filtering of computer generated images. The result is known as the "jaggy" or "staircase" appearance on geometric shapes. Most antialiasing methods use filtering techniques, by calculating the filtering function with the image then assigning the intensity to each pixel, to smooth the sharp lines or edges in the display. The assignment of gray-scale, instead of only black and white, for each pixel in the frame buffer is required for antialiasing operations in raster display systems. Pixel averaging [1], i.e., window averaging, is a filtering technique used to produce a pixel in a low-resolution image by averaging several pixels in a high-resolution image. This technique can achieve very realistic antialiasing results but needs a very long processing time for operating the window to the whole image buffer.

Area antialiasing is another simple heuristic technique. It assigns an intensity to each pixel according to the area of each pixel intersected by the image. Several approaches [2-4] have been described in the literature which use the idea of area antialiasing. These methods are generally easy to apply in rendering geometric images and very easy to implement in hardware. One of them uses a table look-up method [2] to implement conical area antialiasing. A variation of this algorithm can be used to draw the lines with various line thicknesses and to smooth the edges of polygons. The algorithm uses a small number of entries in the table to reduce table searching time and memory space. But, because of interpolation, it yields less accurate results and longer operating time for drawing the lines without having the same line slopes as listed in the table. Some direct area antialiasing calculation methods combined with the Bresenham line-drawing algorithm [5] generate the shaded pixels for line or edge antialiasing [3], [4]. These algorithms can be executed at high speed. But the accuracy of the results and the flexibility of applying the algorithms to other drawing algorithms represent serious deficiencies.

This paper describes and illustrates a direct area antialiasing algorithm, which is more realistic than previous algorithms. The algorithm (known as the CFO algorithm) works with the curve-drawing algorithms for curve antialiasing. Because the processes in the algorithm can be executed in parallel, the algorithm may be efficiently implemented in a parallel hardware.

2. Background

Non-parametric curve-drawing algorithms are used to select pixels while rendering curves on the raster display. Most non-parametric curve-drawing algorithms can be classified as either the mid-point method or the two-point method. For example, the Bresenham line-drawing algorithm [5] uses the difference of the distances from the desired line to the two closest points for selecting pixel in each drawing step which is also called the two-point line-drawing algorithm. The Pitteway and Watkinson line-drawing algorithm [3] uses the relative position between the pixel mid-point and the desired line for selecting the pixel in each drawing step which is also called the mid-point line-drawing algorithm. These two methods can be applied to other curve-drawing algorithms, such as the Bresenham circle-drawing algorithm and the mid-point circle-drawing algorithm. A comparison of the mid-point and the two-point curve drawing algorithms is provided by Aken and Novak [6].

Most area antialiasing algorithms generate the shaded pixels in drawing lines or edges. Gupta and Sproull [2] use a conic filtering technique to precalculate the intensity information, and then store the data into the tables for antialiasing operations. This algorithm incrementally calculates the distance between the line center and the conic center, and then utilizes a look-up table to fetch the intensities of the pixels for each line-drawing computation. The table generated from this algorithm contains the mapping from the given distance and the line slope information to the intensity results of shaded pixels. Because the GS algorithm gives lines which appear to be bended or of non-constant line thickness. The utilization of GS algorithm should consider the trade-offs among look-up table size, operating speed and realism [7]. Pitteway and Watkinson [3] use an unweighted area antialiasing technique to incrementally select the pixels and the intensity information while drawing lines and edges. This PW algorithm has a high operating speed, especially for edge shading operations. But it needs to draw a line three times to approximately generate a unit-width antialiased line. Similarly, Fujimoto and Lwata [4] use an extra intensity control variable and an adjustable Fourier window to generate a pixel's coordinates and intensity simultaneously for drawing shaded lines. By using the addressable sub-pixel and linearized intensity calculation, this algorithm can be used in polygonal curve antialiasing operations. But the algorithm merely has variable line thickness results by using different Fourier windows.

In the algorithm, known as the CFO algorithm, presented in this paper, we refine and extend the incremental method to draw variable thickness lines and edges and to generate intensity and coordinates of shaded pixels in each drawing step. We also extend the CFO algorithm to non-parametric curve antialiasing.

3. The CFO Algorithm

The unweighted area antialiasing technique utilizes that the intensity of each pixel is proportional to the ratio of shaded area to a unit square area. Figure 1 shows the ideal case of a unit-width line on the pixel plane. Along the line, the shaded area of each pixel represents the amount of intensity of that pixel. In order to get an exact intensity for each shaded pixel, we must calculate the shaded area precisely. Figure 2 shows all possible cases of the mid-point distance parameter, ei, and the slope of the line, m, related to the shaded area. From the geometric calculations, we express the shaded results as the follows:

$$Area1 = \begin{cases} ei & \text{if } ei \geq m/2; \\ m/8 + ei/2 + ei^2/2m & \text{if } -m/2 < ei < m/2; \\ 0 & \text{if } ei \leq -m/2. \end{cases}$$

And

$$Area2 = \begin{cases} 1 & \text{if } ei \geq m/2; \\ 1 - m/8 + ei/2 + ei^2/2m & \text{if } -m/2 < ei < m/2; \\ 1 + ei & \text{if } ei \leq -m/2. \end{cases}$$

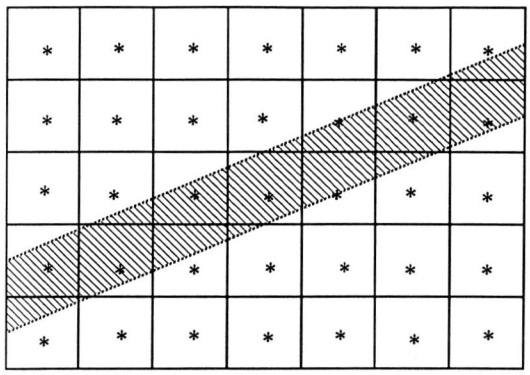
Note that

$$-1 \leq ei < 1,$$

$$0 \leq m \leq 1,$$

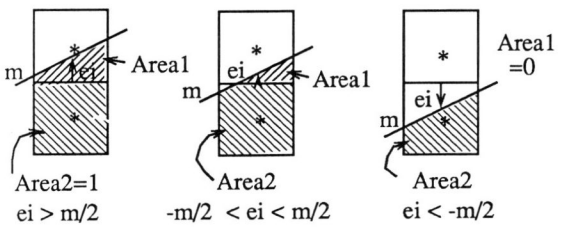
and

$$ei^2/2m \ll 1, \text{ if } -m/4 < ei < m/4.$$



Note that each "*" represents the pixel center; each unit square in the columns represents the area of each pixel.

Figure 1. An example of a shaded line shown ideally on a pixel plane.



Note that each "*" represents the pixel center; m represents the slope of the line; each unit square in the columns represents the area of each pixel.

Figure 2. The results of Area1 and Area2 with three cases of ei.

Then, by eliminating the complicate computation parts in the exact shaded area solutions, we obtain the approximated results as shown in the following expressions:

$$Area1 = \begin{cases} ei & \text{if } ei \geq m/4; \\ m/8 + ei/2 & \text{if } -m/4 < ei < m/4; \\ 0 & \text{if } ei \leq -m/4. \end{cases}$$

And

$$Area2 = \begin{cases} 1 & \text{if } ei \geq m/4; \\ 1 - m/8 + ei/2 & \text{if } -m/4 < ei < m/4; \\ 1 + ei & \text{if } ei \leq -m/4. \end{cases}$$

By calculating the difference between the approximated and the exact shaded area solutions, the maximum error of using the approximated shaded area solution is only 1/32, i.e., 3.125%.

For different input line widths, Wd, we only have to consider two mid-point distance parameters, $e1$ and $e2$, which are the the highest and the lowest distances from the mid-point to the line edges. Given the pixel selection information from the line drawing algorithm, both $e1$ and $e2$ as well as their related area results can be calculated independently. Assume d is pixel selection variable which will be combined in the the CFO algorithm later, the expressions for ei are:

$$If\ d \geq 0,\ then\ e1 = e1 - (1 - m) \text{ and } e2 = e2 + (1 - m).$$

Or

$$\text{if } d < 0,\ then\ e1 = e1 + m \text{ and } e2 = e2 - m.$$

Now, to find the number of shaded pixels and the constant intensity value in each drawing step, we define

$$At = Wd * SQRT(1 + m^2),$$

and

$$Wx = \lfloor At/2 \rfloor + 1,$$

where At is the area of the intersection of the line with each column and $2 * Wx + 1$ is the amount of pixels to be shaded each step. Therefore, the coordinates of shaded pixels can be generated in each drawing step and the expressions of initial value of $e1$ and $e2$ can be defined as

$$e1 = e2 = At/2 - \lfloor At/2 \rfloor - 1/2.$$

Summing all the definitions and formula listed above, we form the CFO line algorithm shown in Figure 3 with the case of $x2 \geq x1$ and $0 \leq m \leq 1$. The CFO line algorithm can draw the antialiasing line from $(x1, y1)$ to $(x2, y2)$ with line thickness, Wd. Note that in the algorithm, d and s are the pixel selection variables used for selecting pixels and ai is the area parameter, where

$$0 \leq ai \leq 1;$$

$$i = 1, 2, 3 \text{ and } 4.$$

Also, $a1$ and $a2$, which were generated by $e1$, are related to the intensities of upper two pixels. Whereas, $a3$ and $a4$, which were generated by $e2$, are related to the intensities of lower two pixels. The algorithm will only use upper pixels' operation, i.e., the computations of $e1$, $a1$ and $a2$, for polygon edge antialiasing. One can adjust the level of realism by controlling the line thickness, Wd, in the CFO algorithm, since Wd can be chosen to be any positive real value.

We can apply the algorithm to non-parametric curve antialiasing by modifying the line slope, m, and the constant intensity value, At. Figure 4 shows the Bresenham circle-drawing algorithm combined with the modified CFO algorithm in octant $y \geq x \geq 0$ and $Wd = 1$ *pixel* case. Note that $(x1, y1)$ is the circle center and R is the circle radius. The CFO circle algorithm uses an incremental line slope, *mstep*, and a incremental constant intensity value, *Atstep*, to maintain the circle edge width. Both are also used for replacing the complicated computations and generating the approximate circle antialiasing results. Similarly, by accompanying with the pixel section result from the non-parametric curve-drawing algorithm, the CFO algorithm can be applied to all the non-parametric curve-drawing algorithms.

Therefore, the generalization of the CFO algorithm is possible if we can specify the line slope, m, and the constant

```
LineAnt(x1, y1, x2, y2, Wd)

{m = (y2 - y1) / (x2 - x1);
 w = 1 - m;
 At = Wd * SQRT(1 + m^2);
 Wx = ⌊At/2⌋ + 1;
 d = 1/2 - w;
 e1 = At/2 - ⌊At/2⌋ - 1/2;
 e2 = e1;
 for (i=1; i<(x2-x1)+2; i++)
  { Assign_area( e1, a1, a2);
    Assign_area( e2, a3, a4);
    if (Wx > 1)
       { for (j=y1-Wx+2; j<y1+Wx-1; j++)
         display(x1, j, 1);
         display(x1, y1+Wx-1, a2);
         display(x1, y1-Wx+1, a4);
       }
    else display(x1, y1, At-a1-a3);
    display(x1, y1+Wx, a1);
    display(x1, y1-Wx, a3);
    if (d<0)
       { s = 0;   d = d + m;
         e1 = e1 + m;   e2 = e2 - m; }
    else { s = 1;   d = d - w;
         e1 = e1 - w;   e2 = e2 + w; }
    x1 = x1 + 1;
    y1 = y1 + s;
  }
}

Assign_area( e_i, a_i1, a_i2)
{ if (e_i≥m/4)
     { a_i1 = e_i;   a_i2 = 1; }
  else if (e_i≥(-m/4))
     { a_i1 = m/8 + e_i/2;   a_i2 = 1 - m/8 + e_i/2; }
  else { a_i1 = 0;   a_i2 = 1 + e_i; }
}
```

Figure 3. The CFO line algorithm with variable thicknesses.

intensity level, At, in the antialiasing algorithm. Because the main processing loops in the CFO algorithm use only simple computations, such as additions and subtractions, the algorithm can have high execution speeds. Furthermore, because most of the processes, such as the calculation of ei and ai in the algorithm can be executed in parallel, the CFO algorithm can be implemented in a parallel hardware architecture.

4. Comparisons and Demonstrations

A comparison of various line antialiasing algorithm results is given by Pitteway [8]. It contains the intensity results of the ideal calculation, the Gupta and Sproull [2] and Pitteway and Watkinson [3] algorithms. Because of the problem with non-constant line thickness in the results of the GS algorithm, we only consider the operation results from the PW algorithm. With the same condition as in [8], the

```
CircleAnt(x1, y1, R)

{y1 = y1 + R;
 xx = 0;
 yy = R;
 m = 0;
 w = 1;
 mstep = √2/R;
 At = 1;
 Atstep = (√2 − 1)*√2/R;
 d = 3 − 2 * R;
 e1 = At/2 − ⌊At/2⌋ − 1/2;
 e2 = e1;
 while (xx≤yy)
   { Assign_area( e1, a1, a2);
     Assign_area( e2, a3, a4);
     display(x1, y1, At−a1−a3);
     display(x1, y1−1, a1);
     display(x1, y1+1, a3);
     m = m + mstep;    w = w − mstep;
     At = At + Atstep;
     e1 = (At/2) − 1/2;    e4 = e1;
     if (d<0)
        { s = 0;    d = d + 4*xx +6;
          e1 = e1 + m;    e2 = e2 − m; }
     else { s = 1;    d = d + 4*(xx−yy) + 10;
          e1 = e1 − w;    e2 = e2 + w; }
     x1 = x1 + 1;    xx = xx + 1;
     y1 = y1 − s;    yy = yy − s;
   }
}
```

Figure 4. The CFO circle algorithm with $Wd = 1pixel$ in octant $y \geq x \geq 0$.

line antialiasing results of the CFO algorithm, the PW algorithm and the ideal calculation are listed in Table 1. This table shows that the area results of the CFO algorithm are much accurate than that of the PW algorithm. Figure 5 shows a general comparison graph of the shaded area results related to the distance parameters, ei, of the ideal calculation, the PW algorithm and the CFO algorithm. It shows that the maximum error of the CFO result is $m/32$ which is much less than that of the PW result, i.e., $m/8$. Figure 6 illustrates the line results of the PW algorithm, the CFO line algorithm and the aliased Bresenham line-drawing algorithm. Figure 7 shows the line results of the CFO line algorithm with several different line thicknesses. Note that all the demonstrations in this paper are using 64 intensity levels in SUN 3/160C with screen size of 1152x900 pixels.

With the comparison of algorithmic complexity for both algorithms, the CFO algorithm has almost three times complexity than that of the PW algorithm. But, for $Wd = 1$ *pixel*, the CFO algorithm can generate three shaded pixels with a more accurate result in each drawing step instead of drawing the line three times with less accurate results as with the PW algorithm. In addition, by adjusting the line width, Wd, the CFO algorithm can generate very realistic lines and edges. To reduce the algorithmic complexity, we can modify the CFO algorithm to have the same antialiasing results as that of the PW algorithm. Then, the algorithmic complexity of the modified CFO algorithm is only about one half of the original algorithmic complexity. Figure 8 shows the modified CFO algorithm, and Table 2 illustrates the line antialiasing execution speeds of these three algorithms by using the sequential software implementations. This result shows that the CFO algorithm has almost the same execution speed than that of the PW algorithm, and the modified CFO algorithm is the fastest one among all three algorithms.

Location	PW result	CFO result	Ideal result
U1	0.114	0.147	0.156
B1	1.000	0.936	0.918
L1	0.114	0.147	0.156
U2	0.000	0.004	0.024
B2	0.829	0.825	0.805
L2	0.400	0.400	0.400
U3	0.543	0.543	0.543
B3	0.686	0.686	0.685
L3	0.000	0.000	0.001
U4	0.257	0.257	0.264
B4	0.972	0.897	0.889
L4	0.000	0.075	0.076
U5	0.000	0.075	0.076
B5	0.972	0.897	0.889
L5	0.257	0.257	0.264
U6	0.000	0.000	0.001
B6	0.686	0.686	0.685
L6	0.543	0.543	0.543
U7	0.400	0.400	0.400
B7	0.829	0.825	0.805
L7	0.000	0.004	0.024
U8	0.114	0.147	0.156
B8	1.000	0.936	0.918
L8	0.114	0.147	0.156

Note that the location of the pixel and PW indicated here are the same as the definitions in [7].

Table 1. The area antialiasing results of the ideal, CFO and PW algorithms from (0,0) to (7,5) with $wd = 1$ *pixel*.

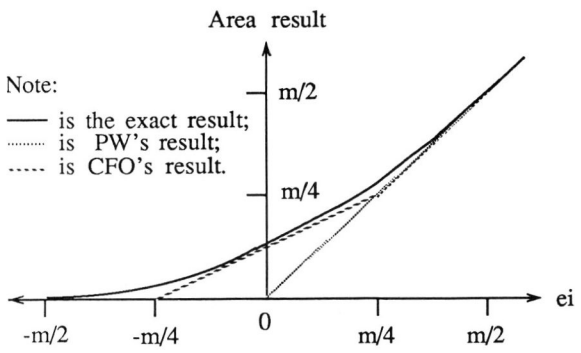


Figure 5. A comparison graph of shaded area results.

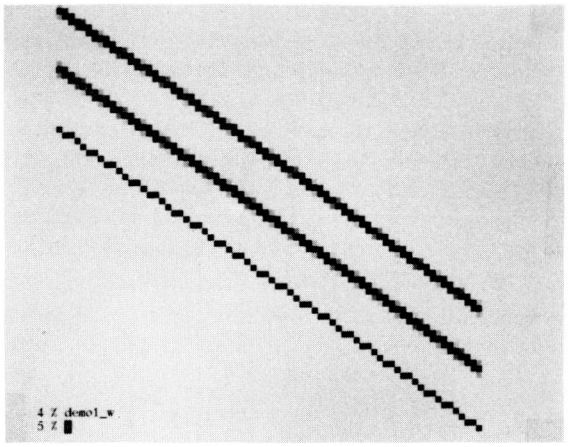


Figure 6. The results of the PW algorithm (the upper line), the CFO line algorithm (the central line) and the Bresenham line algorithm (the lower line) with *line slope* = 5/7, *Wd* = 1 *pixel* and 1 *pixel* = 10X10 *screen–pixel*^{2}.

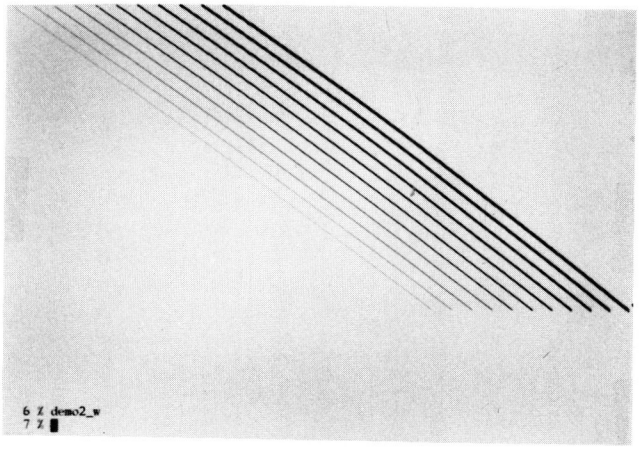


Figure 7. The results of the CFO line algorithm with *line slope* = 5/7, *Wd* = 0.3, 0.5, 0.8, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0 and 4.5 *pixels* (from left to right) and 1 *pixel* = 1X1 *screen–pixel*^{2}.

```
Assign_area( e_i, a_i1, a_i2)
{ if (e_i≥0)
     { a_i1 = e_i;    a_i2 = 1; }
  else { a_i1 = 0;    a_i2 = 1 + e_i; }
}
```

Figure 8. The modified CFO line algorithm with variable thicknesses. Note that it only shows Assign_area parts and the main parts of the algorithm is the same as in Figure 3.

Line	PW	CFO	Modified CFO
(10,10)-(80,60)	40	40	20
(0,0)-(700,500)	440	420	300
(0,0)-(1000,600)	640	600	400

Note that all timing test results are given in CPU milliseconds on SUN-3, and PW indicated here is the same as the definitions in [7].

Table 2. The execution speed of the PW, CFO and modified CFO algorithms on three different lines.

By using antialiasing line-drawing operations, most antialiasing algorithms can be applied to polygonal curve drawings. This piecewise curve-drawing operation takes a lot of time to figure out the polygonal lines of curves, and then to draw all the antialiasing lines for curve antialiasing. Also, most antialiasing algorithms can not operate with non-parametric curve-drawing algorithms, such as Bresenham circle-drawing algorithm, for curve antialiasing. The CFO algorithm can not only be used for line and edge antialiasing, but can be applied to curve antialiasing as well. With a little modification to the nominal CFO algorithm, the CFO circle algorithm, which combines with the Bresenham circle-drawing algorithm, can generate antialiasing circle results incrementally. Figure 9 compares the results of the CFO circle algorithm with the Bresenham circle algorithm. The results show that the CFO circle algorithm will smooth the small jaggy edges but does not have a significant improvement on the large jaggy edges. This problem can be solved by applying the accurate line slope, *m*, and distant parameter, *ei*, in each drawing step. Figure 10 illustrates the results of this improved circle antialiasing operation, the CFO circle algorithm and the aliased Bresenham circle algorithm. Although it shows a very realistic circle result, this improved circle antialiasing operation will involve more complicated computations than that of the nominal CFO circle algorithm.

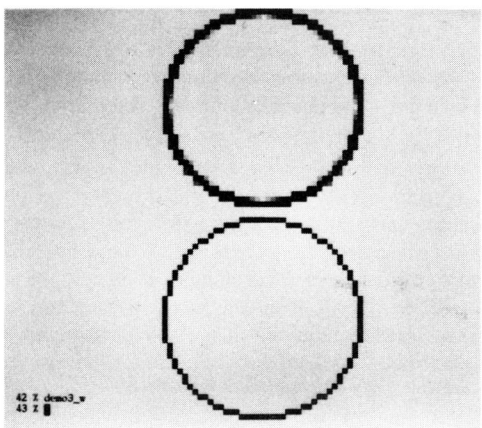


Figure 9. The results of the CFO circle algorithm (the upper circle) and the Bresenham circle algorithm (the lower circle) with *R* = 19 *pixels*, *Wd* = 1 *pixel* and 1 *pixel* = 10X10 *screen–pixel*^{2}.

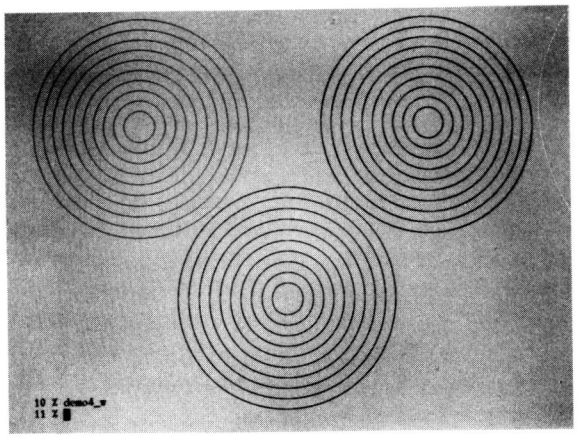


Figure 10. The results of the improved CFO circle algorithm (the lower circles), the CFO circle algorithm (the upper-right circles) and the Bresenham circle algorithm (the upper-left circles) with R = 30 *to* 310 *step* 20 *pixels*, Wd = 1 *pixel* and 1 *pixel* = 1X1 *screen–pixel*^{2}.

5. Conclusions

Area antialiasing is one of the useful antialiasing techniques. Most area antialiasing algorithms operate in a high operating speed and have acceptable line antialiasing results. When required to have a high degree of realism, we should have accurate pixel intensity information and the adjustable line width in the antialiasing algorithms. This paper provides a more accurate antialiasing algorithm than previous algorithms, and the algorithm can draw lines with various line thicknesses and can generate all shaded pixels in each drawing step. Since the CFO algorithm can generate the special line endpoints which edges are parallel to x or y axes, one can use other related papers for squared or rounded line ends. The operating speed of the CFO algorithm can be very fast when implemented in a parallel hardware architecture. Furthermore, the CFO algorithm also can be applied to all non-parametric curve-drawing algorithms, which can generate pixel selection signal.

Thus, the results of this paper provide an improved antialiasing algorithm with enhanced performance coupled with a versatile operational environment for the users. Because the intensity decision processes in the CFO algorithm are very regular and incremental, it is suitable to implement the CFO algorithm into a VLSI architecture and work with other drawing facilities in the advanced computer graphics system.

REFERENCES

1. Rogers, D. F., "Procedural Elements for Computer Graphics," *McGraw-Hill Inc.*, 1985.
2. Gupta, S. and Sproull, R. F., "Filtering Edges for Gray-Scale Displays," *Computer Graphics*, Vol. 15, No. 3 (August 1981).
3. Pitteway, M. L. V. and Watkinson, D, "Bresenham's Algorithm with Grey Scale," *Communications of the ACM*, Vol. 23, No. 11 (November 1980).
4. Fujimoto, A. and Lwata, K., "Jag-Free Image on Raster Displays," *IEEE Computer Graphics and Applications*, Vol. 3, No. 9 (December 1983).
5. Foley, J. D. and Van Dam, A., "Fundamentals of Interactive Computer Graphics," *Addison-Wesley*, 1982.
6. Aken, J. V. and Novak, M., "Curve-Drawing Algorithms for Raster Displays," *ACM Transactions on Graphics*, Vol. 4, No. 2 (April 1985).
7. Earnshaw, R. A., "Fundamental Algorithms for Computer Graphics," *NATO ASI Series*, Vol. F17, Spring-Verlag Berlin Heidelberg (1985).
8. Pitteway, M. L. V., "On Filtering Edges for Grey-Scale Displays," *Computer Graphics*, Vol. 15, No. 4 (December 1981).