II.10

RENDERING ANTI-ALIASED LINES

Kelvin Thompson
Nth Graphics, Ltd.
Austin, Texas

Problem

Render an anti-aliased line segment.

Solution I

Model the line segment as having a finite thickness and set each pixel's intensity according to how much it overlaps the line. We accomplish this with an extension to the traditional Bresenham line algorithm (Bresenham,1965). With each iteration, the usual algorithm moves by one pixel along a major axis and by zero or one pixel along a minor axis (for example, if the line's slope is in the range [–1, 1], then the major axis is *X* and the minor is *Y*). To expand the algorithm we add two loops—called *orthogonal loops*—in sequence inside the traditional loop. Immediately after the traditional algorithm chooses the central pixel of the line, the first orthogonal loop examines adjacent pixels in the positive direction along the minor axis, then the second orthogonal loop examines adjacent pixels in the negative direction.

At each pixel (including the central pixel) the algorithm updates a variable that contains the distance between the center of the pixel and the middle of the thick line; this *distance variable* can be used to calculate (usually via a look-up table) how much the pixel overlaps the thick line.

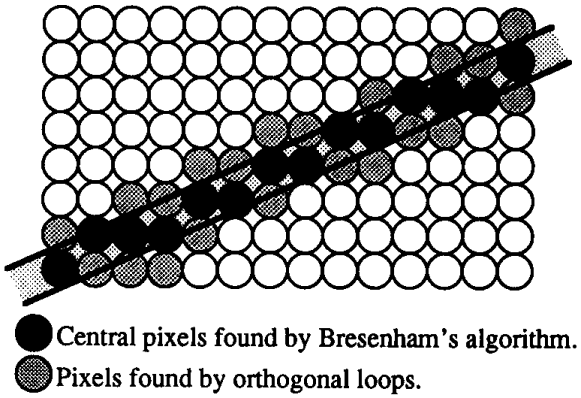

Figure 1.

Also see Gupta and Sproull (1981) for a more detailed description of the algorithm; "Vertical Distance from a Point to a Line" (in this volume) for the mapping between the "vertical" and true distances between a point at a line; "Area of Intersection: Circle and a Thick Line" (in this volume) for the overlap calculation; and subroutine *Anti_Line* for example code.

Solution 2

Render several slightly offset lines using the traditional Bresenham line algorithm, but use alpha blending with progressively smaller coverage values (for example, 1, $\frac{1}{2}$, $\frac{1}{3}$, $\frac{1}{4}$, , . . .; see "Alpha Blending" in this volume). The lines should all be parallel with slightly different starting positions. You can change the subpixel starting position in Bresenham's line algorithm by adding values in the range [0, $2 \cdot dx$] to the initial decision variable.

See Appendix 2 for C Implementation (690)