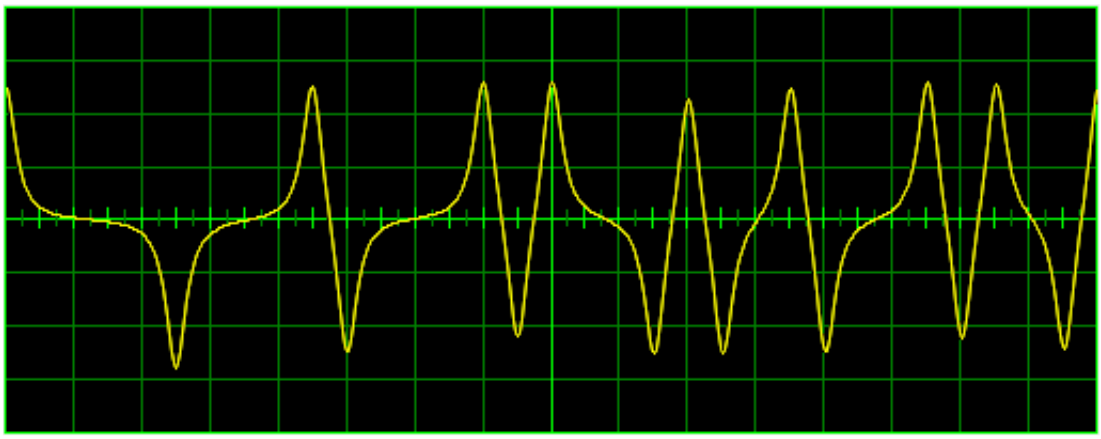
Antialiased Lines and Curves with Improved Diagonal Stepping and Color Blending

by C. Bond
www.crbond.com



Contents

List of Figures

1 Overview

Ever since the first digital displays were built, there has been an interest in developing display algorithms which provided the best possible overall appearance. With low resolution displays, the discrete points, called *pixels*[1], have limited the ability to produce smooth images of arbitrary shape or color. Photographic images and typographic text are good examples of rendering challenges.

The problem is inherent in grid-based display platforms, whether raster scanned or individually pixelated. Of course, the trend toward higher and higher resolutions has improved image quality continually and it is likely that in the future the resolutions will be sufficiently high that no individual pixels will be discernible at ordinary viewing distances.

Until then, however, it is important to be able to do the best we can with existing hardware and software.

The subject of this paper is the reduction of distortion in a displayed image caused by limiting the display space to discrete point. Here, the distortion consists of differences between the ideal image and the displayed one and is referred to as *aliasing.*[2]

Note that relevant fields of application include signal processing as well as computer graphics and may cover one or more dimensions.

2 Antialiasing

Minimizing the distortion of aliasing artifacts is called *antialiasing* and may be accomplished by using filters, transformations, smoothing algorithms, re-sampling, etc.

We restrict our treatment to the display of lines and curves on a graphic terminal or computer monitor. In this context, the problem is reduced to its simplest form.

[1] IBM coined the abbreviated term *pels* as a pixel descriptor.

[2] Another use of the term focuses on the degree of matching between the two images.

2.1 Examples of Aliasing

To illustrate the problems of aliasing and hint at some strategies for artifact reduction, we will use examples of simple graphic primitives.

In Fig. (1), we show a line segment as it might appear on a low resolution display. The first important observation is that the line is not a smooth

Figure 1: Line Segment on Low Resolution Display

straight line. It suffers from what are often called *jaggies*, which refers to the noticeable steps as the elevation of the line increases. This is caused by the inability of the display to move upward smoothly. It must move from horizontal scan line to scan line in discrete shifts.

In Fig. (2) we revisit this line as it would appear if the underlying grid or lattice were visible.

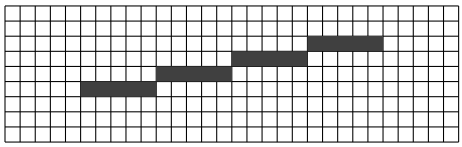


Figure 2: Line Segment With Grid

Fig. (3illustrates the difference between displays with resolutions ranging from eight (lowest line) to one (highest line). It is obvious that increasing the resolution of the hardware will improve the appearance of lines and other graphics primitives.

Of course, at higher resolutions the line is thinner and would carry a different weight in the display. This can be corrected by equalizing the linewidth as shown in Fig. (4). Clearly the higher resolution display renders lines with improved appearance.

We will continue to illustrate line drawing problems and proposed solutions

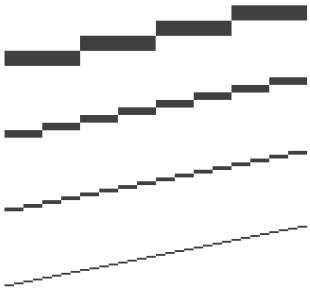


Figure 3: Similar Lines at Different Resolutions

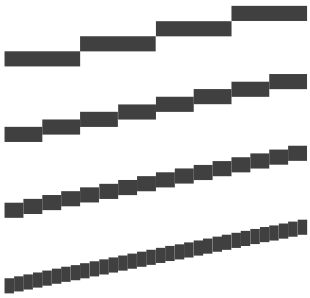


Figure 4: Similar Lines with Equal Weights

using the lower resolution display. It is easier to visualize problems and solutions when we have the benefit of a magnified view.

It is useful at this point to present an actual screen shot of a curved line.[3] Fig. (5) will be used as a basis for comparing antialiasing techniques.

There is another way of representing a grid-limited display which leads to suggestions for mitigating the problem with jaggies. In Fig. (6) we show the pixel centers as small dots, and an ideally smooth, thin line at an angle. A close look at this figure reveals that motion along the line will either land on a pixel center or will pass between two 'nearest neighbor' pixels. For the line shown, these neighbors are above and below the desired line. With typical line draw routines, the nearest of these two neighbors becomes the selected pixel for plotting. This causes the displayed line to periodically jump a full

[3]This unedited graphic is a representation of a magnetic recording signal sampled at about 16 samples per bit.

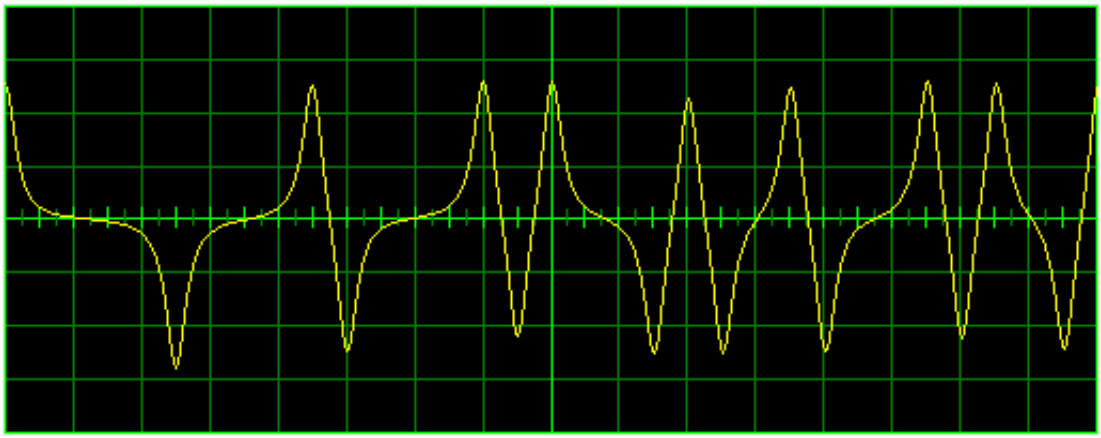


Figure 5: Curved Lines with Jaggies

pixel distance.

As the line moves away from one pixel elevation to another, it would seem that brightness of the lower neighbors should be reduced and that of the upper ones increased. In other words, since the geometry of the display is fixed, we consider controlling the intensity to improve the appearance of the displayed line.

In Fig. (7) we have identified the nearest neighbors (*green*) and an exact match of the line and pixel (*red*).

2.2 Wu's Algorithm

One method for modifying the pixel brightness results in *Wu's algorithm*.[4]

He suggested splitting the illumination between nearest neighbors in such a way that the total illumination is constant, but the contribution of each pixel is determined by its relative distance from the line.

[4]Wu, Xiaolin,"An Efficient Antialiasing Technique",Compute Graphics 25(4),pp:143-152.

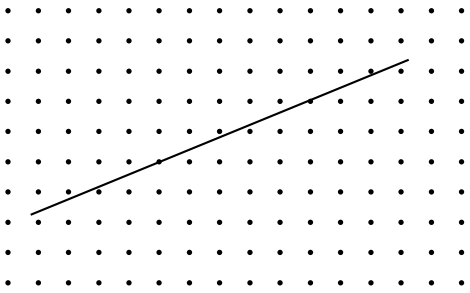


Figure 6: Ideal Line on Pixel Grid

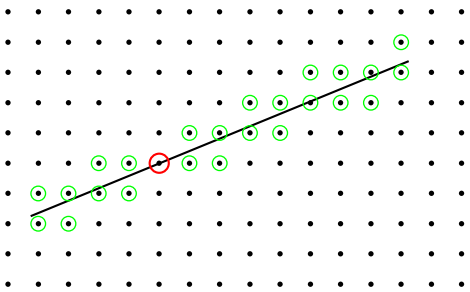


Figure 7: Line With Nearest Neighbor Pixels

A magnified view of the relation betwen the desired line and its neighbor pixels is shown in Fig. (8). According to Wu, if we scale the distance between pixels to unity, the brightness of the upper pixel should be proportional to the distance shown in blue, and the lower pixel should be proportional to the distance shown in red. For the geometry shown the lower pixel would be illuminated at 25% brightness and the upper pixel at 75%. If we imagine moving the line up or down in parallel the brightness of a pixel at crossing would be 100% and as the line moves away until it crosses the next pixel the brightness of this pixel would reduce linearly to 0%. In Fig. (9) we have shown the line segment implemented using Wu's algorithm. The original, desired line is drawn in yellow. If the result is not very convincing, part of

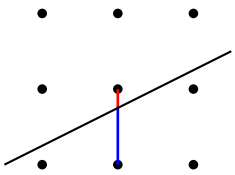


Figure 8: Magnified View

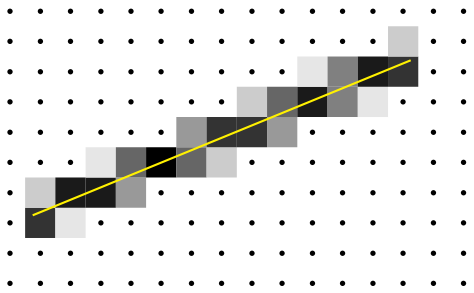


Figure 9: Line With Nearest Neighbor Pixels

the reason is that the magnified view brings all the warts to the surface.

In Fig.(10) we have applied Wu's algorithm to the previously shown screen shot. This graphic is also unedited.

Wu's algorithm is very popular, largely because it is an efficient method for reducing the negative effects of jaggies. It has an intuitive appeal, but harbors weaknesses discussed in the next section.

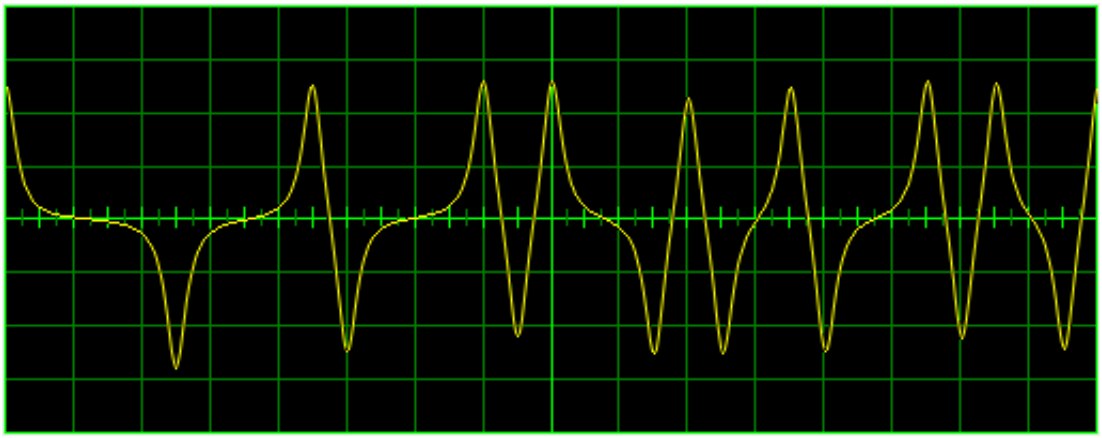


Figure 10: Wu Antialiasing

3 Variable Line or Color Density

An ideal line or curve plotted on a display should have uniform weight or density at every point. Wu's antialiasing algorithm does not exactly produce this result, so lines or curves drawn by his method will appear to have differing color densities at different points. I have seen this described as giving the line a 'ropey' appearance.

There are several reasons for this, and we will address them in the following paragraphs.

3.1 Diagonal Lines

With typical line drawing algorithms, including Wu's, diagonal lines are treated the same as horizontal or vertical lines. That is, the brightness of each pixel is the same.

But this does not result in uniform color density. The reason is that a horizontal or vertical line of, say, 100 pixels covers a specific reference distance, but a 100 pixel diagonal line covers a greater linear distance on the display. In fact, a diagonal line covers about 40% more distance. The visual effect is

that diagonal lines will be fainter by about 30%..

This difference is not quite as dramatic as it sounds, because the response of the eye to illumination is approximately logarithmic. Nevertheless, the reduction in brightness can be seen by examining Fig. (11). This shows a low-resolution diagonal line with horizontal and vertical reference lines.

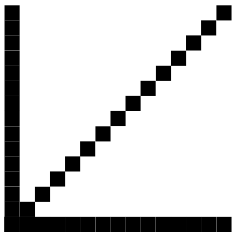


Figure 11: Horizontal, Vertical and Diagonal Lines

3.1.1 Diagonal Steps

Note that the reduction in color density applies to any diagonal step, and not just line segments. For example, if the distance between adjacent horizontal or vertical pixel centers is d, the distance between diagonal pixels is $\sqrt{2}\,d$. Hence, even a single diagonal step will exhibit this anomaly.

3.2 Uneven Color Spreading

With Wu's algorithm the line color is of full intensity on a matching pixel, but is divided between two pixels otherwise. For low slopes the two pixels are above and below the desired line. For high slopes, they are to the left and right.

Linear interpolation is used to determine the appropriate brightness for each pixel. This is not quite right, as can be seen by examining the case where the line is halfway between the two adjacent pixels. In this case, each of the pixels will be give 50% of the full brightness. But this illumination will be

spread over a two pixel space. Now, the eye will interpret the two pixel space with half-bright pixels as if were part of a thicker, but weakly illuminated line.

The effect is noticeable, but it is still better than a jagged line. Again, the logarithmic response of the eye mitigates the effect somewhat.

There are a number of alternatives to linear interpolation in the literature[5] with various arguments supporting their use.

4 Correction Strategies

4.1 Diagonal Lines

Several possible ways of improving the appearance of diagonal lines are possible. Recall that the color density of diagonal lines is reduced by about 30%, due to stretching the pixels over a greater distance.

In Fig. (12) we show a diagonal line with its adjacent neighors filled at reduced intensity. This method has a downside in that it makes the line

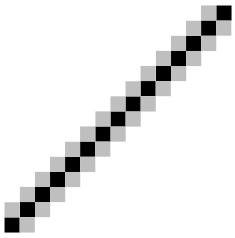


Figure 12: Diagonal Lines With Adjacent Neighbors Partially Colored

appear thicker than lines with other orientations.

In the next figure, the extra pixels are restricted to one side of the line. This has a less dramatic effect on the line width but it has a downside as

[5]The Gupta-Sproull algorithm, for example.

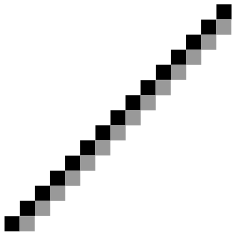


Figure 13: Diagonal Lines With Lower Neighbors Partially Colored

well. The original line will appear to be slightly shifted downward or to the right. Shifting the line by moving the 'center of gravity' with adjacent pixels may not be problematic if the display is a purely visual medium. If measurements are to be made a small error will be present.

Another strategy which does not effectively shift the line is shown in Fig. (14).

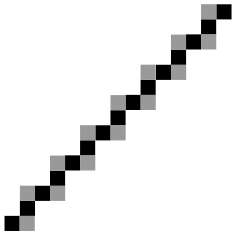


Figure 14: Diagonal Lines With Alternating Neighbors Partially Colored

With Wu's algorithm, the appearance of diagonal steps can be improved by simply increasing the color saturation of the adjacent pixels by about 20% each. The intent is simply to replace the color which was lost by stretching the distance between centers.

For an individual developer interested in antialiasing techniques, it would be a good idea to create a graphics environment which supports the adjustment

of color saturation with experimental lines and curves.

4.2 Color Blending

It is fairly well accepted that linear interpolation is inferior to other weighting methods. Better weights can be found by treating points along an ideal line as cone-shaped regions or Gaussian distributions. Resampling filters which give greater-than-linear weights to closer pixels can produce lines with better appearance.

A common method for efficient rendering uses a lookup table for determining the weight to be given to each pixel plotted. However, we can retain the simplicity of Wu's algorithm by using a special palette. Note that the color palette in a computer graphic system is already a lookup table, and can be modified to support arbitrary weighting functions.

Fig. (15) is an example of an antialised line drawn using Wu's algorithm, but with a palette computed so that the linear distance range between $d = 0$ and $d = 1$ corresponds to a brightness of $1 - d^2$.

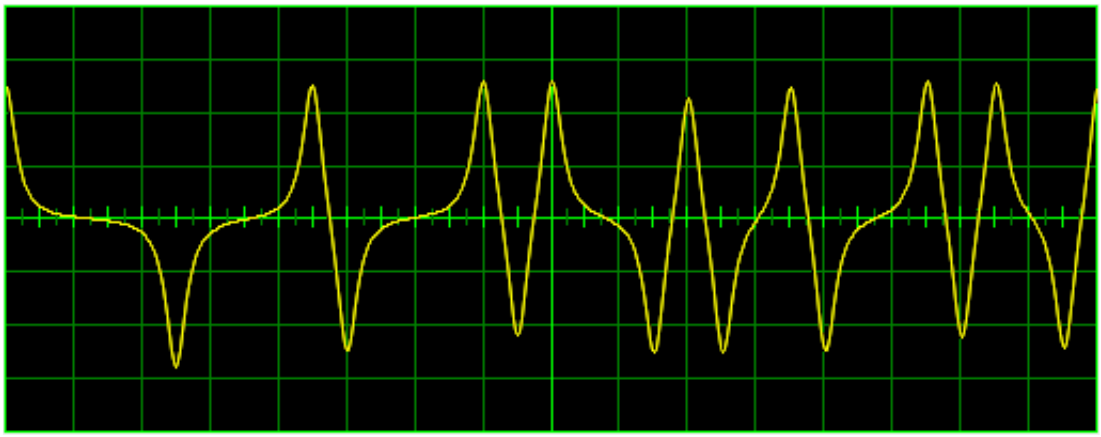


Figure 15: Improved Antialiasing

This is a noticeable improvement over the linear antialiasing.

The structure of the palette is discussed in the Appendix.

The author uses the following strategy to draw lines and curves with improved antialiasing:

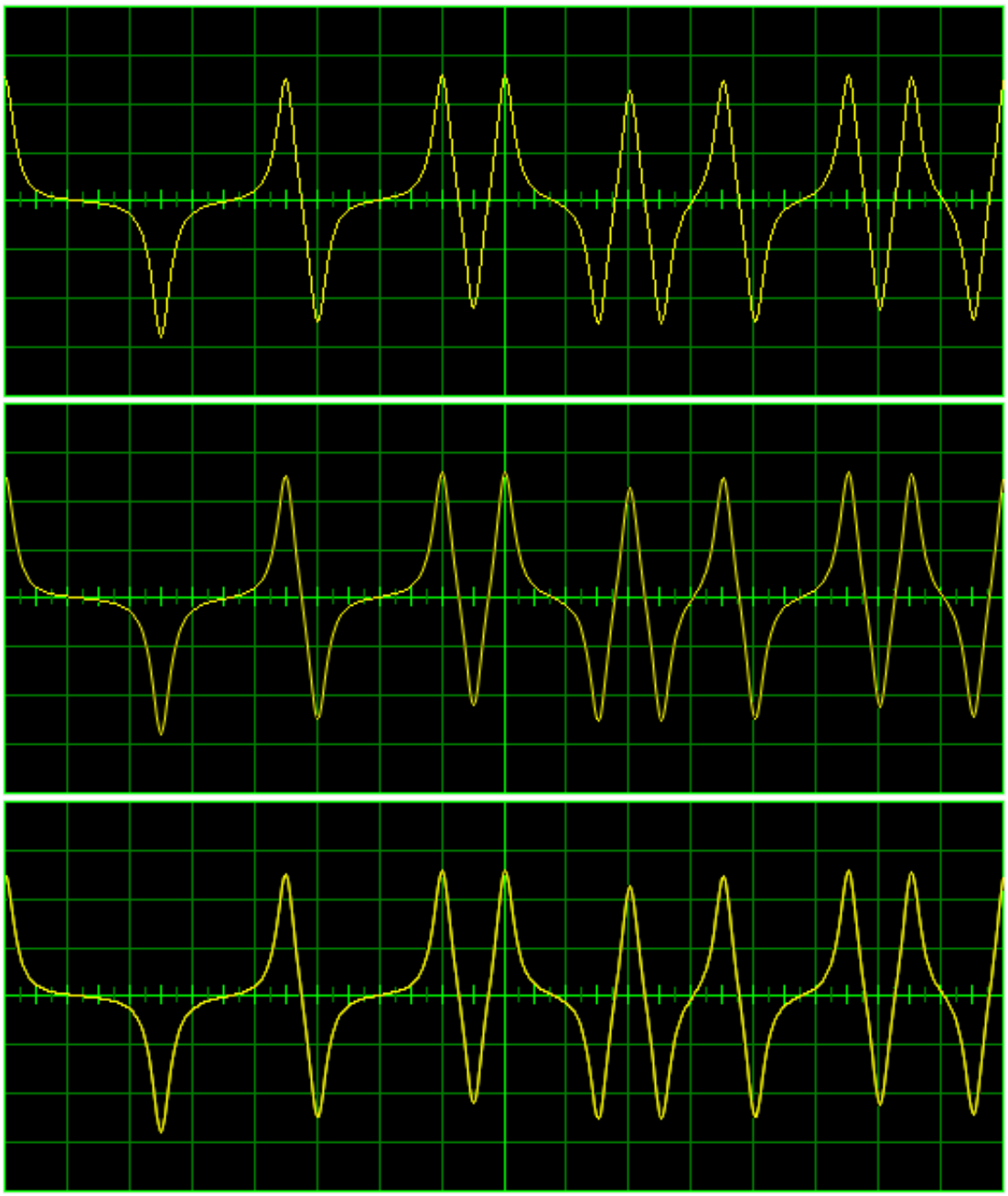
- Determine the foreground and background colors,
- Modify the portion of the palette devoted to antialiasing,
- Draw the line or curve using Wu's linear algorithm.

Note that this technique applies to drawing on a uniform background. It can be extended to bring antialiasing to filled plane figures, but is not suitable for antialiasing over varying backgrounds.

Additional details are given in the appendix.

Appendix

Raw signal, Wu's method, Modified Palette



Palette Architecture

For the screen shots used in this paper, the author used an 8-bit, 256 color bitmap. The first 16 colors represent a system palette consisting of the standard (ancient) 16-color palette. The upper 128 positions contain the antialiasing palette which blends the foreground, at position 128, to the background, at position 255. This upper portion of the palette is recomputed whenever the foreground or background colors change.

Each palette entry is found by calculating the R, G and B elements individually.

Indexing is done by using the upper 7 bits of the control variable driving the line draw routine.[6] For a line such as the one used for our examples, the index refers to the lower pixel and the complementary index, found by an 'XOR' with 0x7F is for the upper one.

[6] See the paper on line drawing on my website.