Prefiltering Antialiasing for General Vector Graphics

September 2013
Juan M. Vuletich
juan@jvuletich.org
www.jvuletich.org

Abstract

The graphics engines commonly used to draw vector graphics apply the antialiasing technique known as pixel coverage. The quality of the results could be greatly improved through the application of sampling theory. This work presents a simple and practical technique for the rasterization of general vector graphics with correct *prefiltering antialiasing*: An appropriate antialiasing filter is applied to a continuous model of a shape, and the result is sampled at each pixel position, producing the rasterized output. The use of this technique can significantly enhance the visual quality of most computer applications.

Keywords

Vector Graphics, Antialiasing, Prefiltering, Sampling, Signal Processing, Nyquist-Shannon

Introduction

Vector graphics form the basis of much of what a computer can display. They are used for almost all text, portable documents, many GUI elements, drawings, etc. Their importance will only grow, as users and developers become aware of the need to adapt applications and content to computer displays of varying size and resolution.

Vector graphics are specified essentially as the path a pen is to follow over a piece of paper, to produce a drawing. The pen, the paper and the path are thought of as *continuous*. But computer displays are *discrete*, comprised of distinct elements or *pixels*. Because of this, in order to be shown on a computer, vector graphics need to be converted to pixel color values, a process called *rasterization*.

Prefiltering antialiasing also called *Signal processing approach to antialiasing* ([1], [2]) is the mathematically correct way to compute the rasterization of 2D vector graphics. It is an application of the Nyquist-Shannon sampling theorem, and therefore it requires a prefiltering stage to satisfy the hypothesis that the original signal is bandlimited.

Common 2D graphics engines, used by graphics libraries, operating systems, and video cards, almost universally employ the inferior pixel coverage antialiasing technique. Pixel coverage can be seen as an incorrect implementation of prefiltering, being limited to a step filter, with a fixed size of one pixel. A step filter is a poor choice, as it attenuates useful frequencies (producing a blurry result) while at the same time it allows too high frequencies (producing visible aliasing, in the form of “jaggies” or “staircase effects”, and Moiré effects). In addition, the computation of pixel coverage is rather expensive. A few engines offer the possibility of the expensive super sampling to enhance quality somewhat, at an even higher computational cost. The main reason for this state of affairs might be that previously published algorithms for correct prefiltering only work on straight line segments.

Disclosed is a simple and practical technique for the rasterization of general vector graphics (including straight lines, curves such as Bezier, and complex paths) applying prefiltering antialiasing.

Description

The main parts of the disclosed solution are:

a) The outline of the shape to be drawn. This is a description of how a pen should be dragged over a piece of paper, in order to draw it. It is specified as a sequence of straight lines or curves. It also includes a line width w (a real number) and a color. All points (segment endpoints, center and radius for circles, control points for Bézier curves, etc.) use real numbers too. This model of the drawing to be made is continuous and independent of any specific pixel grid.

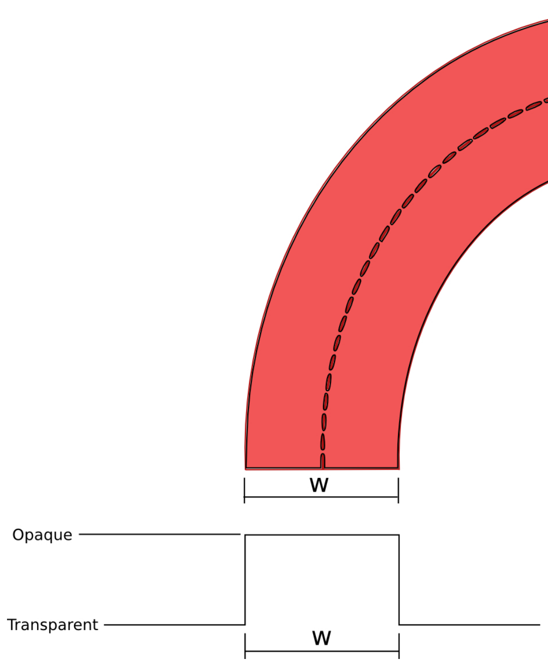

Figure 1. The outline of a sample shape, and the opacity function (i.e. the pen nib of width w) to draw it. The pen nib is completely opaque, and therefore paints over any existing background. We can say that the pen function is transparent everywhere outside the nib, as those areas won’t be affected at all. The dashed line is the path followed by the pen.

b) The destination frame buffer where the drawing is to be done. This is usually 24 bit RGB, or as appropriate for the application. The destination specifies the pixel grid that the shape must be sampled at.

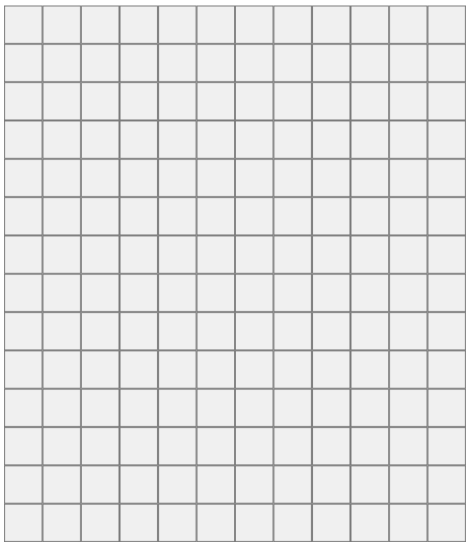

Figure 2. Destination frame buffer and pixel grid for the rasterization of our example

c) A filter size parameter r. In the following diagrams, we'll assume r = 1.5. This means that the transition from transparent to opaque will take at least 1.5 pixels. Other values are possible, and useful values were found to be between 1 (very crisp, some jaggies are visible) to 2 (very soft, absolutely no jaggies or aliasing).

d) A *prefiltering pen*. It is a pen with an opaque center and translucent borders. The area under the opacity function is the same as in (a) and equals w, meaning that the weight of the line will not change. But the transition from transparent to opaque is gradual, leading to a translucent stripe of width r at each border of the line drawn by the pen.

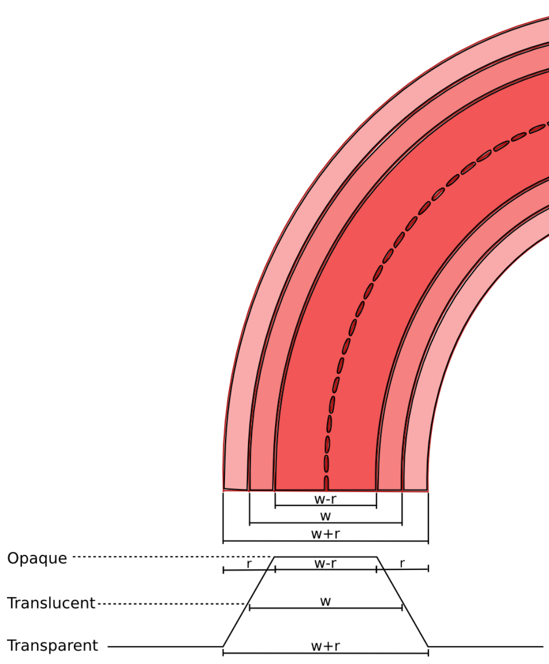

Figure 3. Opacity function of our prefiltering pen, and our example shape as drawn by it. Translucent parts of the pen will mean that the existing background is somewhat visible behind the outline.

e) An auxiliary distance buffer, with the same extent as the destination. It needs to store for each pixel in the destination, a float number, or alternately a fixed point number.

Algorithm

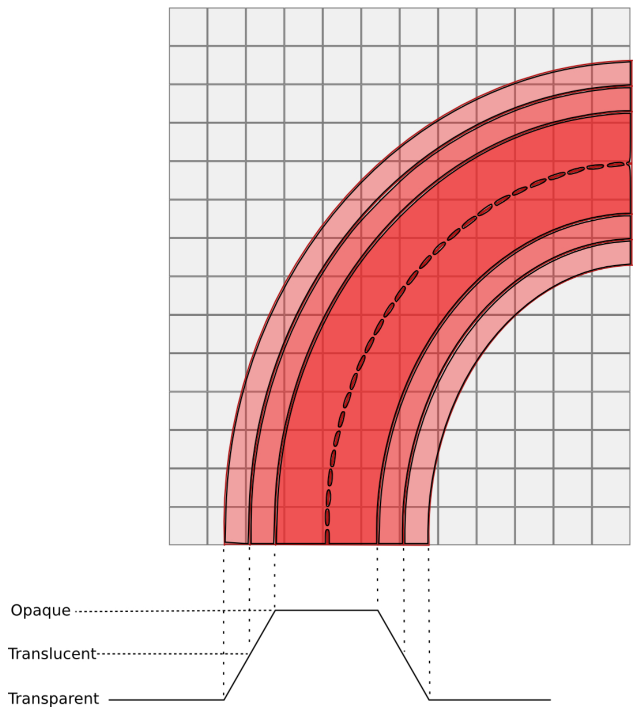

Figure 4. Our sample shape, as drawn by an ideal continuous prefiltering pen, shown over the pixel grid. The light gray background is more visible near the borders of the outline, where the pen gets progressively translucent.

The algorithm for drawing the Outline of a Shape is as follows:

1) The Destination starts filled with whatever background we need to draw on top of. The distance buffer is assumed to be initially filled with 'infinite' (in practice, any sufficiently large value will do).

2) The algorithm continues by following the trajectory specified by the outline. This is done in discrete steps or jumps. At each step, the pen is moved to a new point in the trajectory, jumping by a fraction of a pixel. A step of about 1/4 of a pixel is small enough. This is done using float (or fixed point) arithmetic.

3) For each step, the pen is at some (float) point. The possibly affected pixels are in the circle defined by the pen outer diameter (w+r). For each of these pixels, the euclidean distance from the pixel center to the pen center is computed. This distance is stored in the distance buffer if it is less that the value already there.

4) When the pen has finished following the prescribed trajectory, the distance buffer contains, for each pixel, the minimum distance d to the trajectory. The opacity value is obtained by evaluating the opacity function of the filtering pen at distance d. This opacity value is used to alpha-blend the shape color over the destination frame buffer. Afterwards, the distance buffer is filled with infinite, to be ready for the next call.

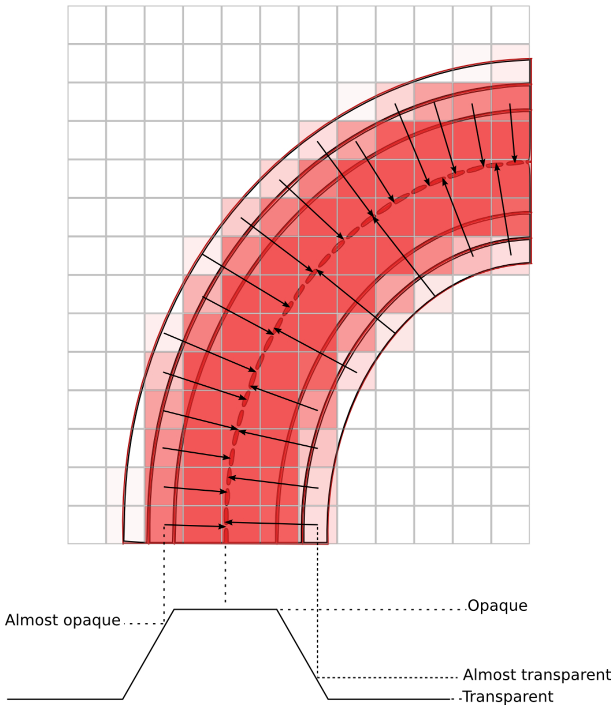

Figure 5. Each pixel is painted with the opacity obtained by taking its distance to the path, and using it as the argument to the prefiltering pen function. To illustrate, for some pixels, the small black arrows show this distance to the path. As a further example, for the two arrows closest to the bottom of the figure, the distance is projected (dashed lines) over the filtering pen function, to show how to obtain the opacity values used to paint the pixels. The same procedure is used for all pixels.

Assumptions made

For simplicity, this disclosure makes the following assumptions:

1) The outline is to be drawn with an opaque color. The extension to handle translucent strokes is straightforward, and involves the pen never being completely opaque.

2) The shape doesn't specify a fill, it only has stroke color. This means we are drawing just the outline.

3) Line thickness w can not be smaller than filter size parameter r.

4) All computations should be done in linear color space. Conversions from other color spaces (such as SVG color space), and to frame buffer color space should be done as appropriate.

Overcoming these limitations will be the focus of forthcoming disclosures.

Final remarks

The main departure from pixel coverage is that the filter support (the filter radius r) can be larger than one (pixel), and is not restricted to being an integer. In addition, the filter can now have an arbitrary shape (in pixel coverage it is restricted to being the step function of width = 1, arguably the worst possible filter). Pens can be built for any desired antialiasing filtering behavior. As a consequence, in addition to obtaining ideal visual quality, it is possible to choose a balance between crispness and resolution, perhaps to adapt to different applications, or different displays. The simpler algorithm can also enable a more efficient implementation.

At www.jvuletich.org/Morphic3/Morphic3-201006.html there are sevaral sample images generated with this technique. They can be compared side by side with versions produced by pixel coverage, to make the quality enhancement evident.

References

[1] Gupta, Satish, and Robert F. Sproull. 1981. “Filtering Edges for Gray-Scale Devices”. *Proceedings of ACM SIGGRAPH 81*, pp. 1–5.
http://www-users.mat.umk.pl/~gruby/teaching/lgim/1_gupta.pdf

[2] McNamara, Robert, Joel McCormack, and Norman P. Jouppi. 2000. “Prefiltered Antialiased Lines Using Half-Plane Distance Functions”. *Proceedings of the ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pp. 77–85.
http://people.csail.mit.edu/ericchan/bib/pdf/p77-mcnamara.pdf
http://dl.acm.org/citation.cfm?id=348226