Anti-Aliased Lines Using Run-Masks

Nicholas A. Diakopoulos and Peter D. Stephenson

1GVU Center, Georgia Institute of Technology, Atlanta, GA 30332-0760, USA
2imedia, 400 Westminster St., Providence, RI 02903, USA

Abstract

In recent work, a set of line digitization algorithms based on the hierarchy of runs in the digital line has unified and generalized the iterative line-drawing algorithms used in computer graphics. In this paper, the additional structural information generated by these algorithms is leveraged to describe a run-based approach to draw anti-aliased line segments in which anti-aliased run-masks are substituted for the individual run lengths as the line is being drawn. The run-masks are precomputed using a prefiltering technique such that one or more run-masks are defined for each of the one or two possible run lengths that occur in the line. The run-masks can be defined for any order or level of the hierarchy of runs in the digital line and the technique is illustrated using runs of pixels. Comparing the use of run-masks to applying the prefiltering technique for each pixel in the line, a line of similar visual quality can be produced more efficiently. We place no restrictions on the placement of the end points of the line, which may reside anywhere on the two-dimensional plane.

Keywords: Line Drawing, Anti-Aliasing.


1. Introduction

The resolution of common display devices remains insufficient to hide aliasing effects from the human visual system. Aliasing artifacts such as jagged edges, scintillation of small moving primitives and the nonuniform intensity of edges across rotation occur when a function cannot be sampled at a rate at least twice that of its highest component frequency as described by the Nyquist sampling theorem. Therefore, to reduce the visual impact of aliasing artifacts, the theoretically correct approach is to prefilter the image using a low-pass filter to band limit the original function to half of the sampling rate before sampling commences. With the original function band limited, no aliasing artifacts can be introduced during sampling.

The ideal filter is a convolution of the continuous intensity function with the Sinc function but the Sinc function has an infinite extent and negative lobs, hence the filter is impossible to implement. For real-world applications, the Sinc function can be defined within a finite extent by combining it with a window function such as the Hann, Hamming, Blackman, Kaiser or Lanczos windows. These different window functions each represent tradeoffs in the frequency response of the final filter and thus the quality of the resulting anti-aliased image. For an application such as line drawing, a more common approach is to employ a simpler filter such as the box filter that can be easily incorporated into a pixel-based line-drawing algorithm and implemented in hardware. The box filter is based on area sampling and determines the fraction of a pixel covered by the line but has far from optimal frequency response. A good compromise between the Sinc and the box filter can be achieved by convolving with the cone filter.

For many applications speed of the digitization process is an important benchmark, therefore to further accelerate the drawing of an anti-aliased line, Gupta and Sproull used a reduced number of pixel intensities. Regardless of the filter used to generate the pixel intensities, if the filter is circularly...
symmetric, using a look-up table can reduce the amount of computation required. In Gupta and Sproull’s original paper, they precomputed the values of a cone filter of radius one and stored them in a table indexed by the perpendicular distance from a pixel center to the center of the ideal line. For each major axis coordinate, three pixels were intensified as only extreme cases would more than three pixels be needed to anti-alias a line of unit thickness. The idea also extends to lines of nonunit thickness. Barkans [1] used the Sinc function windowed with a Hamming function to also intensify three pixels per major axis step for a unit width line.

In this paper, we extend and generalize the idea of using precomputed pixel intensities for drawing anti-aliased lines from pixel-based to run-based algorithms. A run is typically defined as a contiguous collection of pixels; however, we include the hierarchical extension of the definition in terms of the digital line [14]. Iterative and parallel run-based algorithms offer a number of advantages over pixel-based algorithms for a number of line digitization applications, which we will discuss further.

2. Anti-Aliased Run-Based Line Drawing

In this article, we will concentrate on the application of using the coherence of runs and the efficiency of run-based line digitization techniques to accelerate the process of anti-aliasing the line. Our goal is to maximize the speed of anti-aliasing the line while minimizing the loss of quality in the resultant image. To achieve this, we define and precompute a number of run-masks, which approximate the anti-aliased pixel intensities for the one or two run lengths possible in the line, and we apply these run-masks to create an anti-aliased line using a run-based line-drawing algorithm. The run-masks can be generated using any prefiltering technique (e.g. box, cone, windowed-Sinc, etc.). The run-based algorithm can be based on any type of arithmetic, namely integer, fixed point or floating point. Alternative arithmetics yield comparable results, but by using floating- or fixed-point arithmetic the end points of the lines are not bound to pixel positions.

2.1. Run-Based Line Digitization

A set of iterative and parallel algorithms was recently defined for line digitization based on the hierarchy of runs in the digital line [14–16]. This collection of algorithms unifies and generalizes the iterative algorithms used for line drawing including the digital difference analyser (DDA) algorithm, Bresenham’s pixel-based algorithm [17], Reggiori’s run-based algorithm [18] and Bresenham et al.’s run length slice algorithm [19]. The hierarchy is constructed by pixels at the lowest level, runs of pixels at the first level, runs of runs of pixels at the second and so forth [14].

Each of the line digitization algorithms from the hierarchy share an identical iterative structure in which each step of the algorithm involves choosing a tile of which there are only one or two possible. For a pixel-based algorithm, the choice is between the pixels \((x + 1, y)\) and \((x + 1, y + 1)\) when lines between a slope of 0 and 1 are considered. All other lines can be derived from lines of this octant.

For run-based algorithms, the choice of the next tile is between the one or two possible run lengths that can exist in the line, \(r_s = \lceil 1/\alpha \rceil\) and \(r_l = \lfloor 1/\alpha \rfloor\), where \(\alpha\) is the slope of the line. The two run lengths we refer to as short and long.

Each run of any level or order in the hierarchy exists in the digital line in at most two run lengths, which can be decided by an iterative decision mechanism. The hierarchy is infinite if the slope of the line is an irrational number and finite, having only a single run length at the maximum order, if the slope of the line is a rational number.

The length of a run when measured in pixels increases exponentially with the order of the run, countered only by a linear increase in the initialization costs. Therefore, using higher-order runs can lead to significant improvements in the cost of scan converting the line except when the line does not consist of enough runs to recoup the initialization costs [16].

As well as improving the efficiency of scan converting the line for many applications, using a run-based algorithm offers information about the structure of the line that can be used to accelerate secondary processes in a given application. In line drawing, for example, the process of writing the line in memory is the central bottleneck. Runs, however, offer a closer coherence to the layout of memory that could be exploited to accelerate memory operations.

2.2. Run-Masks

We will institute the use of run-masks with an iterative algorithm to draw the line based on runs of pixels and implemented using floating-point arithmetic [15]. The technique can also be applied to higher-order run-based algorithms, which in fact will yield more accurate results as we shall discuss.

Various run-masks can be used to apply the anti-aliased pixels. We will examine the Height-3 (H3) mask and the Height-1 mask (H1) as described in Figure 1. Each is constructed by extending the run length calculated by the scan conversion algorithm to cover the neighboring pixels affected by a line of unit width. H1 masks are extended horizontally and H3 masks vertically.

The H3 mask is based on the observation of Gupta and Sproull that no more than three pixels per column have to be intensified per major axis step in order to anti-alias a line of unit thickness. As described in Figure 1(c), the H3 mask is defined to cover both the run of pixels calculated using the run-based line-drawing algorithm and the runs immediately above and below this run thereby covering the pixels that should be intensified using the three pixel per column idea.
2.3. Computing the Run-Masks

To construct the run-masks, any prefiltering algorithm [1,13,12] can be adapted to compute the intensities of the composite pixels. Run-masks can be constructed for any level of the hierarchy of runs in the digital line. However, we will concentrate on run-masks defined for runs of pixels. The ideas we describe extend easily to higher-order runs.

To calculate the length of a run, Stephenson and Litow [14–16] use the fractional component of the intercept value of the line and the run, \( \beta_j \), as defined in Figure 2(a). The decision of the length of the run is based on a critical value of the intercept \( v = 1 - \alpha r_s \) defined in Figure 2(b), where \( \alpha \) is the slope of the line defined by \( \alpha = \Delta y / \Delta x \), \( \Delta x = x_1 - x_0 \), \( \Delta y = y_1 - y_0 \); \((x_0, y_0)\) and \((x_1, y_1)\) are the end points of the line segment; and \( r_s \) is the length of a short run.

For each run in the digital line, if the continuous line passes below this critical value, \( 0 < \beta_j < v \), the run is long because pixel A is included in the line. If the line passes on or above this point, \( v < \beta_j < \alpha \), the run is short as pixel B is a part of the line.

To generate the run-masks, we define representative short and long runs by placing the line such that the intercept value of the line and the run lies within these two ranges. In the simplest case where we have only one representative run per run length, we define a representative long run by setting \( \beta_j = v/2 \) and a short run by \( \beta_j = (\alpha + v)/2 \).

To calculate the run-mask for each representative run, we use a pixel-based approach to anti-alias each run. If this approach is based on a look-up table the intercept value can also be used to index the look-up table of pixel intensities when precalculating the run-mask.

The starting and ending points of the line segment form a boundary on the run structure of the line that can cause truncation of the first and/or last runs of the line. The separate calculation of additional first and last run-masks can be avoided by observing that truncated run-masks can be constructed directly from either the short or long run-masks [14–16].

3. Efficiency Results

To discuss the improvement in the efficiency of drawing anti-aliased lines by using run-masks, we base our...
For an anti-aliased line, if we consider that the complexity of calculating a run-mask is \( O(\tau) \) and assume that the run-masks are precalculated for each line, the complexity of the run-mask algorithm is \( O(\tau + n/\tau) \). Therefore, as the slope of the line approaches 1 or 0, the algorithm tends to \( O(n) \).

In practice, however, this does not necessarily mean that the algorithm is slower than a pixel-based algorithm since the pixel algorithm must process three pixels per iteration. This effect can be seen in Figure 3, which describes the results of a wall-clock experiment that timed the scan conversion of lines of length 512 pixels between a slope of 0 and 1. Each line was rendered 500 times. Wall clock experiments are never conclusive but they are included here to support the theoretical analysis. The higher costs of using run-masks for lines of slope near 0 and the dependency of the algorithm on the order of runs to use for a line-drawing algorithm dictate that for very short lines using run-masks makes little sense. The cost of initialization, however, is quickly recouped as the length of the line increases.

4. Accuracy Results

The quality of anti-aliasing in any image is often subjective and difficult to quantify. Following the example of others, we have included output of the lines produced by using H1 and H3 run-masks in Figure 5(c) and (d) generated using the Gupta–Sproull filter and in Figure 6(c) and (d) generated using quantized values of an analytic filter. Gamma correction has been applied to the final images [20]. These results can be compared to the minimal 8-connected line with and without analytical anti-aliasing applied. The analytical method used is a box filter aligned with the slope of the line (i.e., not area sampling) similar to Fabris and Forrest [21]. Thus, the convolved ideal signal has the cross-section of a triangle (or trapezoid if the line is thicker). The ideal signal is then point sampled at the center of each pixel to get the gray scale value.

In order to measure the error introduced to the anti-aliased line representation by using run-masks, the mean pixel-by-pixel error (MPE) was measured with respect to the analytically anti-aliased line. The MPE was calculated for every possible line with slope in \((0,1]\) defined within a \(512^2\) lattice. Both run-masks perform similarly. The H1 mask had a MPE of just 6.1% given a standard deviation from the mean error of 2.2%. The range of errors was from 2.5% to 13.6%. The H3 mask performs comparatively with a MPE of 5.9% given a standard deviation from the mean error of 2.2%. The range of errors was from 2.4% to 10.8%.

To compare efficiency and accuracy of the run-based technique we used two different anti-aliasing trials. We compared the efficiency of the run-based technique against the Gupta–Sproull algorithm; however, we compare the accuracy of the technique against the analytic technique of Fabris and Forrest. When comparing the Gupta–Sproull or run-based technique against the analytic technique, errors will be introduced by the fact that the lines generated cover only three pixels per column and the pixel values have been quantized to fit in a look-up table. Therefore, the run-based techniques compare more favorably to the Gupta–Sproull technique where the
Figure 5: The line output from (a) an aliased line-drawing algorithm, (b) the Gupta–Sproull pixel-based algorithm, (c) $H1$ run-based algorithm using GS prefiltering, (d) $H3$ run-based algorithm using GS prefiltering. The exponentially brightened difference image between the (e) Gupta–Sproull and $H1$ algorithm, (f) Gupta–Sproull and $H3$ algorithm.
Figure 6: The line output from (a) an aliased line-drawing algorithm, (b) the analytic solution, (c) H1 run-based algorithm using analytic prefiltering, (d) H3 run-based algorithm using analytic prefiltering. The exponentially brightened difference image between the (e) analytic solution and H1 algorithm, (f) analytic solution and H3 algorithm.
MPE is cut to approximately 4.1% with a standard deviation of around 2.4%.

To show the distribution of errors over the image, the difference images of the H1 and H3 image and the analytically anti-aliased line fan are included in Figure 6(e) and (f). The brightness of each difference images is exponentially increased to display the distribution of errors.

The majority of the visual errors introduced by using run-masks occur at the boundary between runs. Therefore, the more runs in the line such as when the slope of the line is close to 1, the less visually appealing the image as can be seen in Figure 6(e) and (f). To reduce the disparity generated by using run-masks and the prefiltering technique itself, more than one run-mask can be defined for each run length. Another approach to improve visual quality is to apply run-masks based on runs of runs [14], run lengths slices [19] or higher-order runs. While the length of a run increases exponentially with its level in the hierarchy the choice of which order to use, however, must be made individually for each application as the costs of initializing each new level of the algorithm dictate that for short lines, lower-order algorithms are more efficient.

5. Conclusions

We have described a technique that exploits the structural information produced by run-based algorithms to draw anti-aliased line segments with end points that can be placed anywhere on the plane. For the two run lengths apparent in the line, anti-aliased run-masks are precomputed using a prefiltering technique, which are then substituted for the individual run lengths as the line is being drawn. Comparing the use of run-masks to applying the prefiltering technique for each pixel in the line, a line of similar visual quality can be produced more efficiently.

Animation introduces additional aliasing possibilities. Based on initial trials, the run-mask algorithm exhibits a similar image quality to the Gupta–Sproull algorithm when both use the same prefiltering technique.

References


