N90-23071

Faster Simulation Plots

Richard A. Fowell
Hughes S&CG
SC S12 V362
PO Box 92919
Los Angeles
CA 90009

Abstract

Most simulation plots are heavily oversampled. Ignoring unnecessary data points dramatically reduces plot time with imperceptible effect on quality. The technique is suited to most plot devices.

We tripled our department's laser printer's speed for large simulation plots by data thinning. This reduced printer delays without the expense of a faster laser printer. Surprisingly, it saved computer time as well. We now thin all plot data, including PostScript and terminal plots.

We describe the problem, solution, and conclusions. Our thinning algorithm is described and performance studies are presented. To obtain Fortran 77 or C source listings, mail a SASE to the author.

Introduction

Post-processed simulation plots have hundreds or thousands of points. Plotting time ranges from noticeable to intolerable, but faster plots are universally desirable. This paper describes how we reduce plot delays when most of the plot time is used to plot the data curves. Our approach is to thin out uninformative data points, and plot an approximate curve with fewer points. The thinned curve will be indistinguishable from the original if the approximation tolerance is set to a sub-pixel level. When the time required to plot the data is significant, and the data is transmitted to the plot device as coordinate pairs (non-raster) this approach is well worth considering. For our simulation plots, specifying a maximum approximation error of a quarter pixel typically allows elimination of two-thirds of the points. We use a fast, (>10,000 points/sec on a VAX 11/780) compact, (55 lines of FORTRAN) graphical data compression algorithm related to those used for image processing and spacecraft telemetry.

Problem Background

As a spacecraft attitude control systems department, we produce hundreds of post-processed plots each month from our dynamics simulations, as well as telemetry and frequency domain plots. The plot data is generated on our IBM 3090, AD-100 simulation processor, VAXs, Sun workstations, test equipment and spacecraft hardware. It is plotted on graphics terminals, workstation screens and laser printers. The plots are time series, parametric, linear, polar and logarithmic. We now use data thinning on almost all of these plots.

The long time required for simulation plots was originally taken for granted. It was not until we started using laser printers for plots that the problem became severe enough to receive serious attention. In 1985 our department bought a QMS Lasergraphix 800 printer. It was first used to print letter-quality text. Since it was the only plotter connected to our local computers, we soon used it for plots.

All went well at first. We got crisp, clean 300 dot per inch cut-sheet plots, just down the hall. Plotting took a few minutes, but was faster than before. The text users were unhappy. Their documents had been ready when they arrived. Now they waited for plots to finish. Large plot jobs took a quarter hour or more. For jobs with the same number of print file bytes, plots were much slower than text. Therefore, the bottleneck was in the printer's ability to process plot commands, not the communications line speed or printer mechanism.

Many responses were possible. We could have ignored the issue. We could have reprogrammed the print queue to print text as first priority, interrupting plot jobs between pages, if necessary. We could have restricted use of the laser printer as a plotter (other departments did). We could have bought another laser printer, which would have been expensive and slow to arrive. Some advantages of our solution were that it was fast, cheap, compatible with alternative solutions and reduced waiting for all types of users. Additionally, it increased computer and communications line throughput as well as printer throughput.

Why Throw out Perfectly Good Data?

We don't, actually. The thinning algorithm is invoked on a plot by plot basis, after the scaling of the data to the physical plot units has been determined. The algorithm does not alter the data points, but simply scans them and returns the indices of the points to be plotted for that particular plot.

The deviations in the resulting curve from the full curve are negligible compared to the errors introduced by the plot device quantization. One of the algorithm's inputs is an allowable approximation tolerance. Every point on the polyline defined by the vertices returned by the algorithm lies within the specified tolerance of the polyline whose vertices are the full set of data, and vice versa. Due to the pixel quantization of the plot device, the plotted vertices will be rounded by up to half a pixel in any case. If the tolerance is set at a quarter pixel, the tolerance will be dominated by the pixel quantization noise.

I can't believe my plots have lots of redundant points!

Perhaps they don't - but they probably should. Usually, plot data is sampled at a uniform rate for each variable and at the same rate for all variables. This requires that the sampling be set at a fine enough rate to capture the most active portions of the data, which is wasteful for the rest. Even for a simple sine wave, the sampling rate required near the peaks is unnecessary near the axis crossings. And sampling at the rate required for the most active variable will also be wasteful for the others.

The sampling strategy is usually set before the run. If it does not take enough data, important phenomena will not be captured. This may require that a lengthy and expensive run be repeated, or, worse yet, mislead the simulator. Since these potential consequences of undersampling are usually far worse than those for oversampling, erring on the side of oversampling is to be preferred.

Our experience is that our average uniformly-sampled simulation plot data can be compressed by 3:1 using a quarter-pixel tolerance. To evaluate our algorithm, we used a suite of 29 plots from Loren Slafer, one of our heaviest plot users. Each curve had 3929 points from a real-time satellite simulation. The curves included both quantized and non-quantized data, and most were quite irregular.

Figure 1 shows the effect of the tolerance on the resulting compression. Three points are worthy of note. First, the mean number of points required is inversely proportional to the square of the tolerance. This is largely due to the fact that the plot is a piecewise linear approximation to the curve, so the error (tolerance) is second-order. This also implies that even with very tight tolerances, considerable compression is shown. Second, although the curves were of widely different character, considerable compression was achieved for 90% of them, even for tight tolerances. Third, 3929 points was not excessive. These were time history plots rendered for a laser printer with a time axis of 9 inches at 300 dpi, for a total of 2700 pixels. A solid black plot would have required even more samples (5400).

Figure 2 is another presentation of the results. The five curves that showed the least compression were plots of noisy signals - "hash" that looked as though it would require the full 5400 point capacity of the plot to render. The missing 29th bar is the curve with the most compression - a ramp that was compressed to two points. The next three short bars correspond to curves that were pulse trains or sawtooth signals.

The Algorithm

There are many published algorithms for representing a curve by a piece-wise linear curve with a relatively small number of points. The basic trade-off is between processing speed and amount of compression. We evaluated many such algorithms and variants before choosing our fan algorithm. Eight methods were coded and run on our 29 plot benchmark suite on a VAX 11/780. Table 1 compares our fan algorithm with the fastest (strip) and most effective (minimax) alternatives considered. The reference discusses some of the alternatives in more detail, and references a number of survey papers covering such algorithms.

Table 2 shows the effect of thinning on plot and CPU time. Note that total CPU time was reduced by thinning - the time used to thin the data is far outweighed by the reduction in CPU time caused by fewer points being formatted for the plotter. The reduction in plot time is less than the reduction in points (17% as many points take 33% as long to plot). This is because the time for plot gridding, axes, annotation and paper handling is not reduced by data thinning,

Conceptually, the fan algorithm proceeds as follows. Every point on the approximate curve is to lie within a specified distance, ε, of the original curve, and vice versa. Label the original points from first to last, 1 to n. Save point 1. Find the largest index, m, such that all points less than m lie within ε of the line segment from 1 to m. Delete points 2 to m-1, and save point m. If m is less than n, relabel the points starting with the mth point, and repeat; else, exit. The new curve is constructed by connecting the saved points. As described, the number of comparisons is proportional to n^2 (abbreviated as "$O(n^2)$").

The fan algorithm obtains O(n) performance by calculating a region from points 1 through i, such that point i can be deleted if point i+1 lies in this "feasible" region. The region is a truncated cone, or fan, hence the name. Point i is kept or rejected based on this test, then

Figure 1: Fraction of Points Retained vs. Allowable Error in Pixels

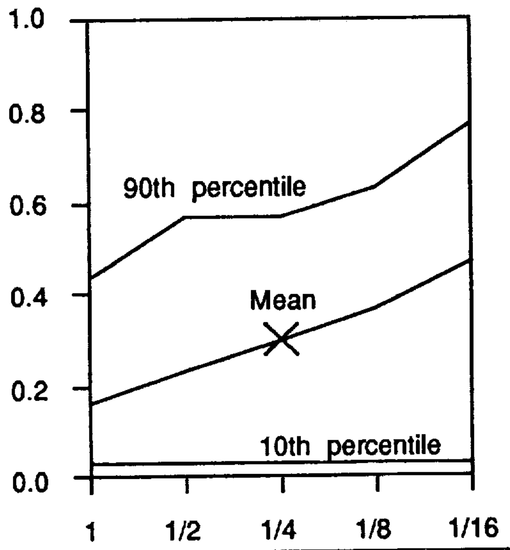

Figure 2: Fraction of Points Retained (1 Pixel Tolerance)

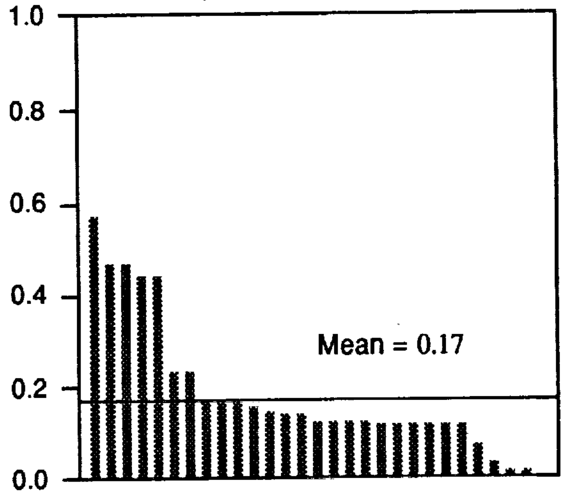

Table 1: Comparison of Methods

Method	Speed (Theory)	Speed on VAX 11/780 (Points/sec)	Effectiveness (Compression)	Complexity (Lines of FORTRAN)
Strip	O(n)	14,000	0.50	40
Fan	O(n)	10,000	0.30	55
Minimax	O(n•log n)	< 5,000 (est.)	0.25 (est.)	>150 (est.)

Table 2: Average Times (thinned and unthinned) on 29 Plots (ε = 1 pixel)

	Fan CPU time (sec/plot)	Total CPU time (sec/plot)	Plot time (sec/plot)	
Unthinned	N/A	10.5	155	(On Vax 11/780 with Apple LaserWriter)
Thinned	0.26	6.2	50	

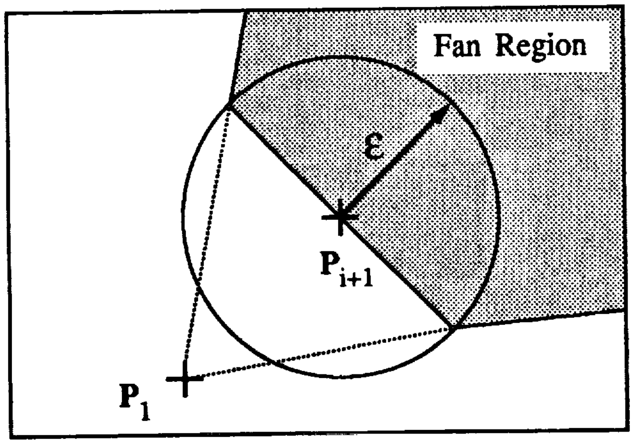

Figure 3: Fan Region Relative to Tolerance Circle

the feasible region is updated by taking the intersection between the fan determined by points 1 and i+1 (Figure 3) with the previous fan.

The reference devotes three pages to presenting the algorithm, with equations, narrative, three figures and a pseudocode listing, so the interested reader is referred there for a detailed explanation. For Fortran and C source, send the author a SASE.

The algorithm has been validated by peer review, program proof and extensive testing. The algorithm itself is essentially a proof that the omitted points can safely be omitted. The automated test suite includes test curve generators and an automated checking routine which compares every point of the input curve against the thinned curve to make sure the tolerance was not exceeded. Six test curves comprising thousands of points are used which included such boundary cases as empty and one-point curves, vertical lines, repeated points and direction reversals. The algorithm and its validation suite have been run on our Suns, VAXs and IBM 3090.

Conclusions

We are quite enthusiastic about this approach to faster plotting. It began as a practical solution to a problem with plot speed. It improved performance with existing hardware, on several computer systems and plot devices. If plot time is a concern, and the time required to plot the curves is significant, we highly recommend this method. Several commercial simulation vendors have shown interest in this algorithm, and more are being contacted.

Acknowledgments

Richard Straka proposed data thinning to speed up plots, Loren Slafer provided the benchmark plots, and Dave McNeil ran the benchmarks and coded the C version.

Reference

R.A. Fowell and D.D. McNeil, "Faster Plots by Fan Data-Compression," IEEE Computer Graphics and Applications, V9 N2, March 1989, pp. 58-66.