Streaming Graphics

Andrew A. Norton, Leland Wilkinson *

December, 2000

Abstract

Streaming graphics, a cross between streaming video and dynamic graphics, is a new approach to visualizing data. The goal is to integrate and synchronize multiple streams of data in real time and display them at rates of up to 20 frames per second. This article describes the design of a system, called *Dancer*, that implements these ideas in Java.

1 Introduction

The term *streaming graphics* evokes the terms *streaming media* (e.g., Feamster, 2001) , and *dynamic graphics* (e.g., Cleveland and McGill, 1988). While similar to both in its outword appearance, streaming graphics is basically different from both. Streaming media systems are generally concerned with delivering sound and video information in real time. Dynamic graphics are concerned with using motion to reveal structure in static data. Streaming graphics systems are concerned with displaying analyses (summaries) of streaming data in real time. Applications of streaming graphics involve many different environments, including real-time monitoring of manufacturing processes, health indicators, financial statistics, and Web data.

2 Data

A streaming data source is fundamental to streaming graphics. In the simplest streaming data model, data arrive in a fixed-length buffer (window) at equally spaced time points. Upon the arrival of a new data packet, we lose one of the packets already in the buffer (usually the last). When a new data packet arrives, we refresh our display. We can also have a fixed-length buffer that receives data packets at irregular time intervals. In this environment, the arrival of new data can trigger a display refresh. There are many other prevalent streaming data environments, however, particularly ones involving multiple, asynchronous data events occurring simultaneously within multiple processing threads.

*Andrew A. Norton is Senior Manager, SPSS, Inc., 233 So. Wacker Drive, Chicago, IL 60606; e-mail: anorton@spss.com. Leland Wilkinson is Sr. VP, SPSS, Inc., 233 So. Wacker Drive, Chicago, IL 60606; e-mail: leland@spss.com.

2.1 Streaming vs. Static Data

A popular metaphor for the difference between streaming data sources and static data warehouses is a *stream* versus a *pool*. In the massive data-mining environment, we often hear this metaphor upgraded to a *river* versus an *ocean*. The significance of this distinction, whatever the scale, is that our graphical and statistical algorithms for mining a stream (to stretch the metaphor) must adapt to its temporal nature. We must pay attention to speed of calculation, lest we be swamped with new data packets before we can compute a displayable result. We must allow calculations to persist wherever possible so that we do not waste time redoing subsets of previous calculations. We must learn to sample our data stream, or know how to bail-out of calculations when the results might be out-of-date. We must synchronize our results, lest we mistakenly display summaries of disparate events in the same time frame.

Many of these concerns do not apply to the traditional, static data mining environment. The premier consideration in the static environment is scalability with regard to the *size* of a dataset. We seek algorithms that scale comparatively well, in the sense of computing time being a constant, linear, or logarithmic function of the size of a dataset. In the streaming data environment, we need to worry about scalability with regard to the *momentum* of a stream. The *momentum* of a stream is its *mass* (the average size of data packets arriving in a fixed interval) times *velocity* (the average number of packets arriving in a fixed interval). This Newtonian physics metaphor is an over-simplification, but it helps us to understand that attending to scalability with regard to the total bulk of the data we encounter will not be sufficient for building a streaming display system. Even a constant cpu-time calculation may still be too slow to handle the real-time data stream we must display.

2.2 Multiple vs. Single Data Source

In the static data environment, we merge different data sources into a single table before computing graphical and statistical summaries. We cannot employ this simplification in the streaming environment. Instead, *Dancer* is designed to handle multiple datasources. For example, we might want to attach a single display to a live feed from a stock exchange and another live feed from a commodities exchange. There is no time to index and merge these two feeds into a temporal database.

Multiple feeds imply a multi-threaded computing environment with event-notification through broadcasters and listeners. The data feeds run simultaneously, and each broadcasts the arrival of new data it receives. The statistical and graphical listeners attend to those messages and react as they choose. In our financial example, the result is a display that shows a moving time-series of a particular stock superimposed on a moving time-series of a related commodity.

2.3 Continuous vs. Discrete Time Scales

A consequence of this functional architecture is that *Dancer* operates on a continuous rather than discrete time scale. In theory, *Dancer* displays data at any instant. Its time scale moves (at least perceptually) continuously. This is in contrast to some time series displays, such as seen in traditional ARIMA (Box and Jenkins, 1976) packages, that contain measurements indexed on equally spaced time points.

Dancer has two time-scale modes. In the static-frame mode, geometric representations of data (points, lines, etc.) move through a motionless frame (the data area). We may clip the elements at the frame boundaries or let them pass beyond.

In the dynamic-frame mode, geometric elements are continuously repositioned within a frame that is continuously scrolling forward or backward in time. This mode requires careful programming to allow tick marks, grid lines, and scale labels to scroll smoothly and continuously, in 2D or 3D as required.

3 Geometry

Dancer is based on a geometric model of statistical graphics described in Wilkinson (1999). That book contrasts the geometric model with the "chart-centric" model of statistical, business, and scientific graphics packages. In the geometric model, elements such as points, lines, and intervals are embedded in a frame defined by an algebra with three operators. These geometric elements are realized (made viewable, hearable, etc.) through a set of aesthetics such as size, shape, texture, and color. The *nViZn* system for rendering graphs on the Web is based on the same model (Wilkinson, et al., 2000).

Unlike *nViZn*, *Dancer* can bind each geometric element to a different data source. The behavior of every element in the frame is synchronized through the functional data model. Special rendering technology is required to enable each element to move up to 20 times a second based on incoming data.

3.1 Scene Tree vs. Geometric Primitives

A *scene tree* is a data structure that contains the information in a geometric scene. Scene trees are frequently used in 3D modeling to organize the arrangement and rendering of a collection of geometric objects. For example, if the root of a tree is a body, then the root's children could be torso, head, arms, and legs. The children of arms could be upper arms, lower arms, and hands. And the children of hands could be fingers and palms. The advantages of this organization are several. First of all, children inherit attributes from parents, including aesthetics (e.g., color, texture) and localized coordinates. Second, adding objects to a scene (another body, for example) requires only appending a new subtree to an appropriate node. Third, a tree is a simple object that can be explored through depth-first or breadth-first search. This makes rendering

rapid and efficient. And it makes interactions with elements (editing, brushing, linking) a simple matter of walking the tree to locate a selected element.

Wilkinson (1994) employed a *graph tree* model for constructing and rendering statistical graphics in the SYSTAT package. The hierarchical nodes of the tree were Window, Pane, Frame, Graph, and Element. With the Fisher-Anderson Iris data, for example, SYSTAT plots three scatterplot matrices (three species, four variables) in a window by assigning one Pane node to Window and three Frame nodes to Pane. Each Frame contains sixteen Graph nodes. Each Graph node contains two Element nodes (one for a cloud of points and one for a smoother). The most important consequence of this architecture is that SYSTAT has no scatterplot matrix routine. Instead, SYSTAT has a method for assembling graphical elements in a structure of one or more graphs and frames. Anything it can do inside a single scatterplot it can do simultaneously inside multiple scatterplot matrices. For the same reasons, SYSTAT has no routine for drawing a Trellis (Becker et al., 1996). Instead, SYSTAT constructs a Trellis structure by assembling graphs in a hierarchy determined by values on categorizing variables. Details of look-and-feel can be handled in display modules that render this structure in different styles.

Scene trees and graph trees organize geometric primitives. They are essential for creating efficient real-time streaming graphics systems for a variety of reasons, some of which are evident in the SYSTAT architecture. Most important, perhaps, scene tree architecture enables rapid rendering.

3.2 Supervised vs. Immediate Rendering

When we render a scene that is rapidly changing, we may re-render the entire scene every time a change is detected or render only the parts that change. If our goal is to render complex scenes that may change up to 20 times a second, we choose the latter strategy.

In a graphics foundation like Java3D, the re-rendering housekeeping is taken care of by the Java scene tree. When the scene tree is updated, only parts of the screen that need updated are re-rendered. An added benefit is that foundations like Java3D are designed to accommodate 3D hardware accelerators.

4 Statistics

Statistical graphics depend on the calculation of statistics – point estimates, interval estimates, smoothers, summaries. In a static data environment we can precompute these statistics before rendering a graph. The most prevalent example of this strategy is in OLAP (On Line Analytic Processing) systems based on ETL (Extracting, Transforming, Loading) technology. These systems aggregate data from various sources, store them in a warehouse, and display graphics such as pie and bar charts on the aggregates. The Temple MVV visualization system of Milhalisin et al. (1995) works similarly.

4.1 Update/Downdate

Streaming graphics require a different type of statistical algorithm. In the simplest case, such as accumulation of sums and sums of squares and cross-products, we use an update/downdate strategy. When new data arrive in our computational buffer, we discard the oldest data item, downdate its contribution to the accumulated statistics, and update the statistics from the new contribution. This process can accumulate rounding error, so it is essential to recalculate occasionally all values from the data in a window. This recalculation can be scheduled to occur at convenient times, similar to the way garbage collection is handled in an interpretive system.

Iterative calculations are not as simple. We can update and downdate Hessians and Jacobians, but handling convergence in a real-time environment is problematic. Other statistical algorithms present different problems. Guha et al. (2000), Datar et al. (2002), and others are concentrating on these issues.

5 Visualization

The simplest display for streaming data reflects the current state of the stream. Geometric elements such as points and lines move as the stream progresses. Real-time is only one aspect of what Dancer does with streaming data, however.

5.1 Instant replay vs. animation

Whatever the time interval (seconds, minutes, hours, ...) we cannot expect a viewer to observe a display continuously. A real-time system needs to incorporate alerts for out-of-bounds conditions. Visual and audio alerts are common in control applications in military, health care, and manufacturing.

When an alert occurs, it is not always clear what antecedent conditions led to its occurrence. Consequently, Dancer provides an instant replay feature to allow a viewer to rewind a scenario and replay it as an animation. This is an aspect of a more general Dancer feature allowing us to animate any data series. Although designed for real-time applications, Dancer's functional time model can be applied to any indexed (ordered) variable so as to allow animation of other variables. In this respect, Dancer can behave like exploratory systems such as xGobi (Swayne et al., 1998) or DataDesk (Velleman, 1988) that animate over a variable.

This type of *archival* animation needs to be distinguished from Dancer's normal real-time model, however. Archival animation offers the opportunity to pre-process data that we do not have in real-time. In the extreme, animation systems such as *Flash* (www.macromedia.com) can prepare bitmap frames that are played in a media player. These animations are fundamentally different from the architecture in Dancer.

5.2 Transformations of time

When we animate a sequence by buffering and replaying captured data, we may transform time in a variety of ways. Reversing time helps us to untangle sequential dependencies. Differencing time helps us to recognize rates. Double-differencing time reveals acceleration/deceleration. Polarizing time (a cyclical transformation) helps us to compare cycles. Logging time helps us to view order-of-magnitude effects. Geologists and cosmologists often log time scales and, as Graham Wills has pointed out (personal communication), ordinary people often use a similar transformation when they look into the future on a day ... week ... month ... year time-scape.

6 Conclusion

Streaming data require streaming algorithms. Much of the technology for processing these data is relatively new and much is yet to be done. Because statistical graphics is more involved with processing data than with drawing pictures, there is considerable technology transfer than can occur from related fields in computer science.

An indication of the challenges involved in processing real-time data can be seen in the contrasts between *Dancer* and its sibling *nViZn* . Both are based on the graphics grammar model and both are programmed in Java to take advantage of a Web environment. The two programs share not a line of code.

References

[1] Becker, R.A., and Cleveland, W.S., and Shyu, M-J. (1996). The Design and control of Trellis display. *Journal of Computational and Statistical Graphics, 5,* 123-155.

[2] Box, G.E.P., and Jenkins, G.M. (1976). *Time Series Analysis: Forecasting and Control*. San Francisco: Holden-Day.

[3] Chambers, J.M., Cleveland, W.S., Kleiner, B., and Tukey, P.A. (1983). *Graphical Methods for Data Analysis*. Monterey, CA: Wadsworth.

[4] Cleveland, W.S. and McGill, M.E., Eds. (1988). *Dynamic Graphics for Statistics*. Belmont, CA: Wadsworth Advanced Books.

[5] Datar, M., Gionis, A., Indyk, P., and Motwani, R. (2002) Maintaining stream statistics over sliding windows. Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco.

[6] Feamster, N.G. (2001). Adaptive delivery of real-time streaming video. Master's thesis, Department of Electrical Engineering and Computer Science, MIT. Online at http://nms.lcs.mit.edu/papers/feamster-thesis.pdf

[7] Guha, S., Mishra, N., Motwani, R., and O'Callaghan, L. (2000). Clustering data streams. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS 2000)*.

[8] Mihalisin, T., Timlin, J., Gawlinski, E., and Mihalisin, J. (1995). Visual analysis of very large multivariate databases. *ASA Proceedings of the Section on Statistical Graphics*, 18-27.

[9] Swayne, D.F., Cook, D., and Buja, A. (1998). XGobi: Interactive Dynamic Data Visualization in the X Window System. *Journal of Computational and Graphical Statistics*, *7*, 113-130.

[10] Velleman, P.F. (1988). *Data Desk*. Ithaca, NY: Data Description Inc.

[11] Wilkinson, L. (1994). *SYSTAT, Version 6*. Evanston: SYSTAT, Inc.

[12] Wilkinson, L. (1999). *The Grammar of Graphics*. New York: Springer Verlag.

[13] Wilkinson, L., Rope, D.J., Carr, D.B., and Rubin, M.A. (2000) The language of graphics. *Journal of Computational and Graphical Statistics, 9,*530-543.