Streaming Graphics

Andrew Norton, (SPSS, Inc), anorton@spss.com
Leland Wilkinson, (SPSS, Inc), leland@spss.com

ABSTRACT

Mining data streams requires algorithms that often differ from those used on static tables. Displaying data streams in charts and graphics requires algorithms very different from those used in conventional plotting. Issues range from fusing disparate time streams to rapid display updating.

Dancer is an object-oriented class library for processing changing tables of data and rendering them to a graphics environment. We present an architecture designed to integrate and buffer data streams of up to 10,000 events per second in a way that allows real-time analytics and display of up to 20 frames per second in a 2D or 3D environment.

WHAT IS A GRAPH?

A *graph* is a function that maps *conditions* to *representations*. This allows you to perceive conditions that would otherwise be unperceivable, such as the flow of data coming across a wire. If the condition changes, the representation also changes.

This graph function is invertible, so that the condition can be inferred from the representation. The graph typically loses information, so that the exact condition cannot be inferred. The graph may exclude certain information entirely (filtering), or only draw parts of a larger graph (clipping). The graph might summarize: We may know that the average grade at Harvard is 3.37, but not know the grades of individual students.

The mapping is defined to work within a *domain* of possible conditions. Conditions outside the domain (e.g. the thermometer melting) cannot be represented.

Applying the graph to every condition within the domain yields the *range* of possible representations. Given a condition within the domain, the graph will render a representation within the range. This allows the user to understand what would be the representation of a hypothetical condition.

The user thus interprets the graph by understanding how the range of possible representations corresponds to the domain of possible conditions. The representation of the graph (within the context of the range) allows the user to infer which condition exists.

WHAT IS DANCER?

Dancer is an SPSS project for streaming and animated graphics, written in Java.

Dancer is a framework used to define graphs that dynamically respond to data changes and specification changes.

Framework

Dancer is made up of a kernel plus a library of components.

The kernel defines the interfaces that must be satisfied by the components, including Function, Domain, Variable, and Renderer. The kernel also contains code to bind these components together, including the thread synchronization code to insure that the threads cooperate. The kernel is kept to minimal size[1] so that it can be thoroughly understood and intensively tested.

The library contains commonly used components such as an in-memory table, domains for categorical and continuous data, and renderers (based upon Java 2D[2] and Java 3D[3]). Users can supplement the library with components of their own.

Over time, most new features of Dancer will be implemented by adding to the library, without disrupting the kernel. So user extensions have the same status as Dancer features themselves.

Streaming Data

Data streams are mapped to tables. The rows represent objects, and the variables represent attributes of the objects. A cell is a crossing of a row and a variable, containing a value. Tables may have any implementation that can determine what rows exist and the current value of a variable for a given row.

Cells may change their values. For each variable, a flag determines whether a variable is constant or potentially changing. The variable may optionally fire an event when a value changes. If a variable is changing but not firing events, new values are detected through polling.

A pseudo-variable "_exists_" changes to true when a row is created, and changes to false when the row is deleted. So events can be used to determine when rows are created and deleted.

Frame

Frames establish the rules by which values are represented. These rules map from data value to aesthetic value. Frames display guides (axes and legends) explaining these rules.

Dancer provides a stable frame in which graphical elements can be displayed, and dynamically added or removed. This frame is based upon declared domains, which remain stable as data values change.

The user may define the graph frame independently of any data source, so that it remains constant as elements are added and removed and as the data values change. For example, a category for a rare event can remain on an axis regardless of whether that event is currently occurring.

The frame also defines functions to be used to convert data values to aesthetic values. So effectively the graph defines aesthetics in terms of data values.

These mappings are displayed as legends, indicating the conversion of typical data values to aesthetic values. These legends can be dynamically hidden and shown, allowing the

---

[1] The kernel is currently 483k of uncompressed Java byte code.
[2] http://java.sun.com/products/java-media/2D/index.jsp
[3] http://java.sun.com/products/java-media/3D/index.jsp

ideal use of screen real estate (since users only need to refer to legends occasionally). Legends are typically also controllers, allowing the functions to be changed on the fly.

Facets

Dancer can display multiple facets, where each facet is an independent filtered view of the data. Facet displays can be different sizes in order to best use screen real estate. Facet displays can be dynamically hidden, shown, and resized.

A typical use of facets partitions the data, such as facets for "male" and "female". But there is no need for facets to cover the data – for example, you might choose to display only a few of many mutual funds.

It is also possible to display facets that overlap with other facets. For instance, you can have "summary" facets that include all cases from the "detail" facets.

Interactivity

The user may interact with the graph. For example, the legends provided by the Dancer library display the current settings and allow the user to modify them.

The user can change dimension bounds by dragging the graph to a new position, or by zooming in and out with the mouse wheel.

Continuity

Unnecessary change distracts the user. Dancer is designed to maximize continuity so (to the extent possible) the only changes in the display are caused by data changes or user interaction.

For example, existing tick definitions continue to be used until they become unsatisfactory, rather than optimize the momentary appearance of the graph. For bounds [0, 100], the tick values (0, 20, 40, 60, 80, 100) might be used. If the bounds then change to [50, 150], Dancer would choose to use the ticks (60, 80, 100, 120, 140) to maintain continuity. If we selected ticks without regard to the previously displayed values, we might choose [50, 70, 90, 110, 130, 150] which would be distracting to the user.

Similar issues apply for assignment of symbols and colors, sort order, text formats, and page layout.

Elements

Elements represent data within the frame, using independent glyphs. In Dancer, elements in the same frame may be based upon different tables. Elements may be dynamically added and removed.

Currently supported elements include Bar, Point, Surface, Path, and Link.

IMPLEMENTATION OF DANCER

The Threads of Dancer

Dancer has three threads running simultaneously: the notification, rendering, and control threads.

The Notification Thread

The notification thread is outside of Dancer itself. When a value changes, the data source fires an event into Dancer. There may be multiple notification threads corresponding to multiple data sources.

Dancer processes events as quickly as possible to avoid disrupting the process that is doing the notification. Dancer simply marks the relevant table cell as having changed.

If a variable changes very frequently, it may be declared as continually changing, circumventing the event mechanism.

The Rendering Thread

The human eye perceives motion if frames are presented in rapid succession. For graphs (as opposed to pictures) this begins to occur around 4 frames-per-second and there is little benefit beyond 50 frames-per-second.

Dancer renders at a constant speed (if resources are available) regardless of the rate of data change. It does not waste resources by rendering faster than the specified maximum (e.g. 50 frames-per-second).

The rendering thread of Dancer updates the display incrementally by periodically reading the data cells that the Notification Thread marked as changed.

The Control Thread

The user can make changes to the graph at any time, while notification and rendering continue.

The Functional Stack

A function specifies the transformation of an input value (or values) to an input value. A variable is a function that uses a key as the input value.

Dancer uses a series of functions to transform data values to statistic summaries, and then to aesthetic values. When there is a change, notification events propagate through the functions to the renderer. When the renderer renders a new image, values are pulled through the functional stack.

All variables specify

- The current value (for a given key),
- Whether the mapping from keys to values can change over time,

- The range of possible values (taking into account the range of each input variable)

Some variables provide notification of changes via events.

Derived Variables

Derived variables obtain their values by applying a function to other variables. They may have several variables as inputs, and may have multiple derived variables depending on them.

Dancer uses chains of variables that begin with the data values and ultimately produce the aesthetic values used by the rendered glyphs.

Derived variables obtain information from their source variables:

- When the range of an input variable changes, a change event is fired from the range of each dependent variable.
- When an input value changes, a change event is fired from each dependent variable.
- When a value is read from a derived variable, each source variable is read.

Efficiency

Dancer is a resource-intensive application, so a practical implementation must conserve resources.

Incremental Updating

Dancer incrementally updates the representation as the data or mapping changes. This provides efficiency by handling changes locally, without repeating unaffected computations.

Governed Performance

Dancer allows the user to specify a maximum level of performance (e.g. frames per second). Once satisfactory performance has been achieved, resources are reallocated to other goals.

Multi-threaded Architecture

Dancer uses a multi-threaded architecture, adjusting to varying loads. If resources are tight, the rendering thread will degrade to lesser quality while the data intake thread and GUI control thread continue unimpeded.

Resource Recovery

Dancer uses garbage collection and reuse of resources so that it may continue to run indefinitely. For example, if a code (such as a color) has not been used recently, it may be reused for a different key. This allows a smaller and more distinguishable vocabulary of codes to be used.

CONCLUSION

Dancer uses a functional programming approach implemented through object-oriented techniques. This has proven to be effective at responding to live data streams and user commands, while providing a highly flexible and extensible graphics system.

AUTHORS

Andy Norton
SPSS, Inc.
233 S. Wacker Drive, 11th Floor
Chicago, IL 60606
312-651-3154
anorton@spss.com

Leland Wilkinson
SPSS, Inc.
233 S. Wacker Drive, 11th Floor
Chicago, IL 60606
(312) 651-3270
leland@spss.com