The Language of Graphics

Leland Wilkinson, Daniel J. Rope, Daniel B. Carr, Matthew A. Rubin

SPSS Inc., 233 South Wacker, Chicago, IL 60606

Abstract

We describe a system, called GPL, that implements a language for quantitative graphics. The structure of this system differs from existing statistical graphics, visualization, and mapping systems. Instead of treating a graphics display as a viewer for underlying data, GPL treats data as an accessory to viewing a graph. GPL is based on the mathematical definition of the graph of a function and uses that definition to organize data linked to the graph.

To be published in *Journal of Computational and Graphical Statistics*.

GPL has been renamed **nViZn**

1 Introduction

When thinking about computers, we often associate the word "language" with text. Statisticians often compute with command languages, such as those in SAS, SPSS, S-Plus, or SYSTAT. Software developers work in FORTRAN, C, Java, and other programming languages. We may also think occasionally of visual programming languages. Less frequently do we think that a non-textual expressive setting, such as a paint program, can constitute a language. In any case, however, we can benefit from examining the linguistic rules underlying the specification of a problem on a computer. This is especially true in the world of graphics.

This paper is about the language of statistical graphics. It outlines a framework called the Graphics Production Library (GPL), which we have developed as a language for presenting graphics on the World Wide Web. Although GPL is tailored to the Internet, it reflects a number of ideas that are independent of computing environment. First, it is based on an assumption that statistical procedures serve graphics; graphics are not ancillary displays of statistical results, but are means for perceiving statistical relationships directly (Chambers et al., 1983; Cleveland, 1985; McDonald and Pedersen, 1991). Second, it assumes that graphical elements are "alive"; wherever possible, graphical features such as points, lines, bars, legends, and axes are connected to data, metadata, or statistics in a way that allows users to drill-down, link, rotate, filter, brush and zoom directly in the display (Fisherkeller et al., 1974; Becker and Cleveland, 1987; Stuetzle, 1987; Velleman, 1988; Tierney, 1991; Swayne et al., 1998). Third, it is based on a formal model of graphics (Wilkinson, 1999); individual displays are not *ad hoc* visual arrangements of data, but reflect instead a quantitative or qualitative model of the variables in the display.

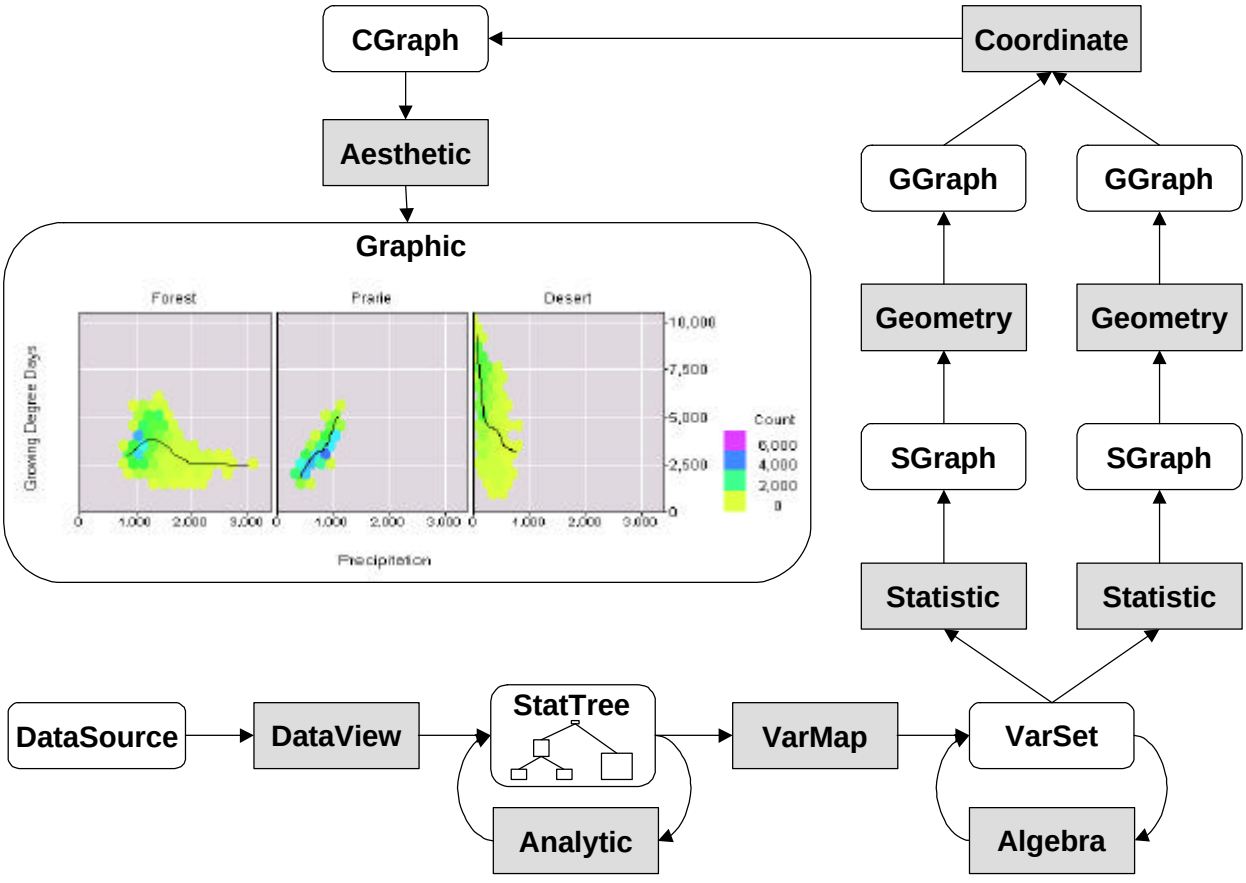
Figure 1 contains a diagram of the temporal model underlying GPL. It is one of several architectural views of GPL, and is used here to illustrate the issues involved in devising a statistical graphics language. The shaded rectangles represent functional objects in the system. The rounded rectangles represent the sets on which these functions operate. This view derives from data-flow and pipeline models devised for scientific and statistical visualization systems (Buja et al., 1988; Upson et al., 1989)

The data for the graphic in Figure 1 are described in Carr et al. (1999) as follows. Daly et al. (1994) developed an analytic method called Parameter-elevation Regressions on Independent Slopes Model (PRISM) that uses point data and a digital elevation model (DEM) to generate 2.5 minute by 2.5 minute gridded estimates of monthly and annual U.S. climatic parameters. Their data sets and further details can be found at *www.ocs.orst.edu/prism*. Carr et al. (1999, 2000) used those gridded summaries for the time period 1961-1990 to develop graphics characterizing the spatial variation of climatic parameters within ecoregions. They associated each grid cell with an Omernik level II ecoregion (Omernik 1987, 1995) using a point-in-polygon matching procedure. Figure 1 shows panels for three of the Omernik ecoregions (Ozark/Ouachita Appalachian Forests, Chihuahuan Desert, and West-Central Semi-Arid Praries). The horizontal axis of each panel represents the average yearly precipitation in millimeters over the three decades. The vertical axis represents average annual growing degree days, a measure of the number of degrees in daily average temperature above 50 degrees summed over all days with a daily average temperature above 50. There are 78,766 grid cells underlying Figure 1. (No binning adjustments have been made for the latitudinal variation in cell area; Carr, Kahn, Sahr and Olsen (1997) discuss equal area gridding alternatives.)

A graphical system like GPL needs to be able to represent these 78,766 data points in real time, in a Web browser, in a distributed data environment, with instant access to associated meta-data through pop-up annotations and other viewers. The right-most panel of the graphic, for example, shows a pop-up containing count information when a user clicks on a selected region of the graphic. Any element in the graphic, including legend items, scale values, labels, smoothers, etc., can be queried in this manner. GPL also offers real-time controllers and widgets (sliders, buttons, list boxes, etc.) for transforming, manipulating, and selecting subsets of the underlying data.

The diagram in Figure 1 summarizes how this functionality is accomplished. The remainder of this paper is a walk through Figure 1. In each of the following sections, we will step successively through each functional object to see how it operates. At the end, we should be able to see that a language for expressing the capabilities of a system like GPL differs in important respects from the languages used in statistical packages, visualization systems, databases, or data mining systems.

Figure 1. Data flow model for GPL system.



2 DataViews

The GPL data source is an abstraction. Avoiding concrete data formats and structures encourages us to define a greater variety of graphs than is customary in relational databases or statistical packages. Having the graph organize the data, rather than having the data organize the graph, frees us from having to limit graph types to the particular structures we find in our data sources. Moreover, an abstract DataView allows us to connect our graph to heterogenous and distributed sources of data. For example, we can collect the tuples defining 25 points in an XY plot from 25 different Web sites in a live feed to our DataView.

Database client-server graphics and visualization systems attempt to solve these problems with several *ad hoc* strategies. First, they can collect heterogeneous data into binary large objects (BLOBs) inside a database. With this approach, structural organization can be tailored to the needs of specific object-oriented clients rather than to a relational table or flat file model. This method places the organizational burden on the database server rather than on the client, however. Adding new functionality requires reorganization of the data on the heavy-weight server. Second, database systems can employ *bots* (software robots) to collect data from distributed sources and embed them in a single database. This method requires job scheduling to synchronize updates, however. It is not suitable for streaming data applications such as stock quotes and other dynamic time series.

In short, if we want a user to be able to explore Web pages, databases, ftp repositories, and other data sources by interacting directly with a graphic, we must maintain references all the way back to the original sources. By using abstract interfaces to data, we implement this flexible functionality.

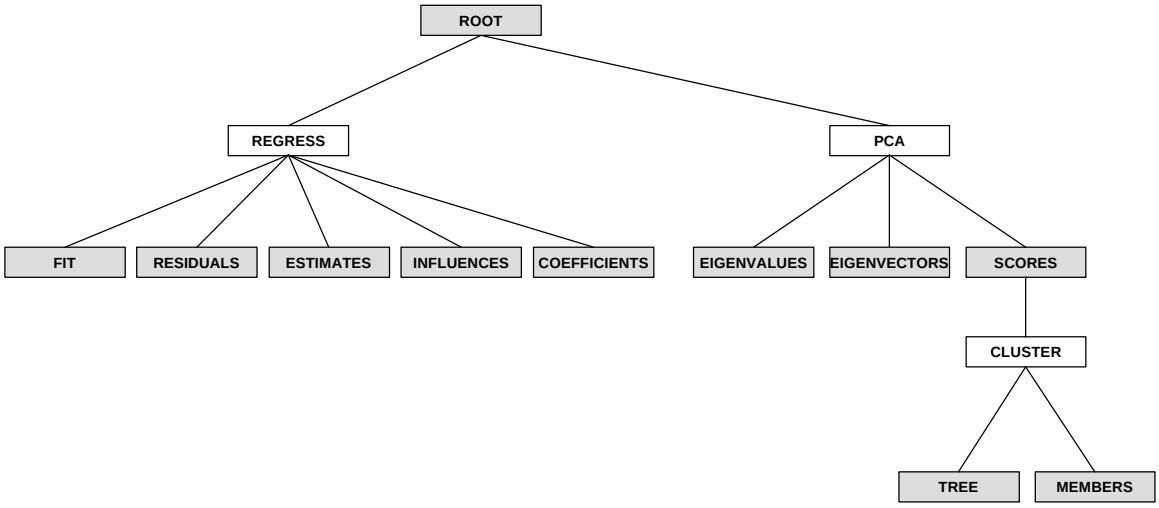
3 Analytics

Analytics involve filtering, recoding, aggregating, segmenting, modeling, or summarizing data. GPL Analytics are transformations that operate on an object called a StatTree. A StatTree contains a snapshot of a DataView plus, optionally, the results of dependent analyses. Because they are transformations, GPL Analytics can be chained. And because their domain is a StatTree, they offer a relatively high degree of flexibility in a relatively simple object.

A tree structure has many advantages. A StatTree is easily encoded in extensible markup language (XML) for use as a portable Web resource. Also, a StatTree is serializable (externalizable) with a simple interface so that it can be passed between separate components in a distributed system. These capabilities make a StatTree easy to work with in a distributed network environment containing a variety of protocols.

Figure 2 shows an example of a StatTree. A StatTree is a rooted tree whose nodes hierarchically alternate between data nodes containing data objects and dependent analysis nodes that identify analytic methods. The data nodes are represented in Figure 2 by shaded rectangles and the analytic nodes by clear rectangles. This simple structure allows us to walk a StatTree to locate a particular analysis or sequence of analyses. We can also determine both the input to an analytic method (a data object) and its output (one or more data objects).

Figure 2. Stat Tree.



Data nodes of the tree contain a data object identifiable by the label of the node. Instances of this object contain an array of numerical data, an array of associated string data, as well as optional resources such as case weights and metadata. Thus, a node can contain resources such as raw data, parameter estimates, fit statistics, confidence intervals, diagnostics, and model comparison statistics or information measures. All data in a StatTree must be derivable from the data at the ROOT node. Thus, descendent data nodes are either proper subsets of the ROOT data or are the results of sequences of analyses on the ROOT data.

A practical consequence of this architecture is that we can annotate graphics with goodness-of-fit statistics, model expressions, and other metadata from a StatTree without making additional passes through a data source to compute them. Data passes can be expensive, so it is worth collecting and persisting relatively cheap calculations even when they are not known in advance to be needed in a graphic.

Because Analytics can have StatTrees as their input and output, we may collect them in a transformation chain. Each Analytic adds one or more children to a StatTree. Thus, we can build graphics from compound analyses (e.g., cluster analyses on principal components), while maintaining case IDs, weights, and other information we need to perform brushing, linking, and sensitivity analysis.

Another benefit of transformation-chains is in handling large datasets. An abstract DataView can be used to hand Analytics chunks of data, one row or table at a time, to be aggregated by rectangular or hexagonal binning. With binning, we can process datasets with millions of cases, maintain case weights, and compute weighted statistics on the aggregates. This is how we handled the bulky ecoregion data in Figure 1. The computationally intensive LOESS smoothers in Figure 1 were computed from pre-aggregated hex-bin data.

Analytics in GPL currently include statistical methods like cluster analysis, regression, multidimensional scaling, principal components, and singular value decomposition. GPL Analytics also include organizing methods such as merging StatTree data nodes, reshaping matrices (e.g.,

triangular-to-rectangular), recoding variables, partitioning variables (e.g., subgrouping), jackknifing, bootstrapping, and simple random sampling.

4 Var Map

VarMap extracts one data object from a StatTree and outputs a table called a VarSet. A VarSet is a set of variables, a matrix whose columns are variables and whose rows are instances of values on those variables. We need VarMap to make a VarSet because Algebra operates on variables, not on raw data.

VarMap finds the source table to make a VarSet through a simple StatTree addressing mechanism: a string representing the path from root to node. For example, the path *ROOT/PCA/SCORES/CLUSTER/MEMBERS* points to the cluster members data in Figure 2. StatTree paths encapsulate what has been done to data before graphing. The StatTree path for the graphic in Figure 1 is *ROOT/AGGREGATE/AGGREGATION*. The *AGGREGATION* data object contains the coordinates and counts for the hex bins.

5 Algebra

The graph $G = \{(x, f(x)) : x \in R \text{ and } f(x) = e^{-x^2}\}$ is a subset of R^2. To display G, we choose a bounded region of R^2, $F = [x_{min}, x_{max}] \times [y_{min}, y_{max}]$, and we physically represent the set of points $P = F \cap G$ by choosing a coordinate system and making a graphic with ink or some other perceivable medium.

Graphics algebra provides a method for specifying F (which we call a frame) when we wish to construct a graphic based on some function of a set of data. Wilkinson (1999) presents three algebraic operators called *cross* (*), *nest* (/), and *blend* (+), together with the rules for their use. They are derived from the set operators *product* ($\times$), *discrete union* ($\sqcup$), and *union* ($\cup$), respectively. We use *cross* to construct a computer-generated graphic of the error function in the example at the beginning of this section. The algebraic expression for Figure 1 is *rainfall*days*region*. The frame for each panel is given by *rainfall*days* and the frame for the set of three panels is derived by crossing with *region*.

It is easy to confuse graphics algebra with command languages or scripts used to construct statistical charts in some computer packages. GPL does not simply parse graphics algebra; it symbolically evaluates it. For example, the expression $a*(b + c)$ is equivalent to $a*b + a*c$; GPL produces the same graphic when presented with either expression.

With graphics algebra, GPL can build either single graphics or tables of graphics. GPL creates tables differently from a tables-producing language (TPL), however. TPL formats multi-way tables by specifying rows, columns, layers, headings, and contents. By contrast, graphics algebra is indirectly related to the physical appearance of tables. GPL has a separate layout component to generate a specific table format from its algebraic structure. It is this separation-of-function between algebra and layout that gives GPL its extraordinary scope. In GPL, a table is not a rectangular layout of cells; it is a lattice-structured frame that may be arranged on a rectangular grid, on the circumference of a circle, on the trajectory of a spiral, or on some other geometric object.

6 Statistics

At this point in our excursion through Figure 1, we have data and a frame, but we have no graph. The Statistics component of GPL contains functions that receive a VarSet from Algebra and output a statistical graph, called an SGraph. The most familiar statistical graphs are location statistics such as means. Statistical graphs also include confidence regions, smoothers, densities, directed graphs such as trees, and other functions. The modifier "statistical" is something of a misnomer, since our requirement for this component is only that Statistics output a unique tuple or set of tuples or collection of sets of tuples for each input tuple. This more general definition of an SGraph allows GPL to produce a wide variety of graphics not limited to statistical charts.

Figure 1 indicates that a frame may include more than one SGraph. Our ecoregion example includes a hex-bin density and a LOESS smoother. Each is computed from the same VarSet. This architecture is ideally suited for a multi-threaded environment in which certain tasks can be handled simultaneously. The possibility of more than one SGraph in a frame is one of the obvious ways GPL differs from a charting program. Standard charts have only one or two hard-wired graphical elements per frame.

Computing statistical values requires a lot of housekeeping. We must not only handle sample weights and missing data, but we must also carry along the pointers necessary to link geometric components to data. If we compute a schematic (box) plot, for example, we need to maintain a list of cases in the central box, whiskers, and possible outliers so that a user can brush these objects and link them to other graphs in real-time. If we compute a tessellation, we need to maintain enough information to compute the perimeter or area of a polygon if requested. Doing this in a general and efficient way, for linear, order and other statistics, while allowing for missing data and sample weights, is nontrivial.

It is important not to confuse the hex-bin Statistics element with the hexagon binning Analytic. If we eliminated the hexagon binning Analytic from our specification, we would have to compute the hex-bin Statistic from the raw data. This would not have taken more time (since the same algorithm is involved in both computations), but it would have prevented us from computing the LOESS smoother in reasonable time. Instead, we pre-computed the bins in an Analytic and then used the bin counts and locations to compute both the hex-bin and LOESS Statistics. The same trade-offs exist for other Statistics and Analytics. Where we choose to locate the computations depends on the functionality we want in our application.

7 Geometry

We now have a graph we can draw. Since a graph is a set of points, we could represent each point by a spot of ink or pixel of light. For example, we could render each point in a 2D graph by placing a spot of ink on a piece of paper, perhaps with an ink-jet printer.

There are two problems with this approach. First, many graphs are infinite, even within the bounds of a frame. A regression line, for example, is an infinite set of points. We could not plot every point in the set. We can solve this drawing problem by sampling on a regular or irregular grid. This is how we draw lines on a computer screen or laser printer, for example. Second, statistical graphs are not always in the form we expect to see in a chart. For example, some viewers

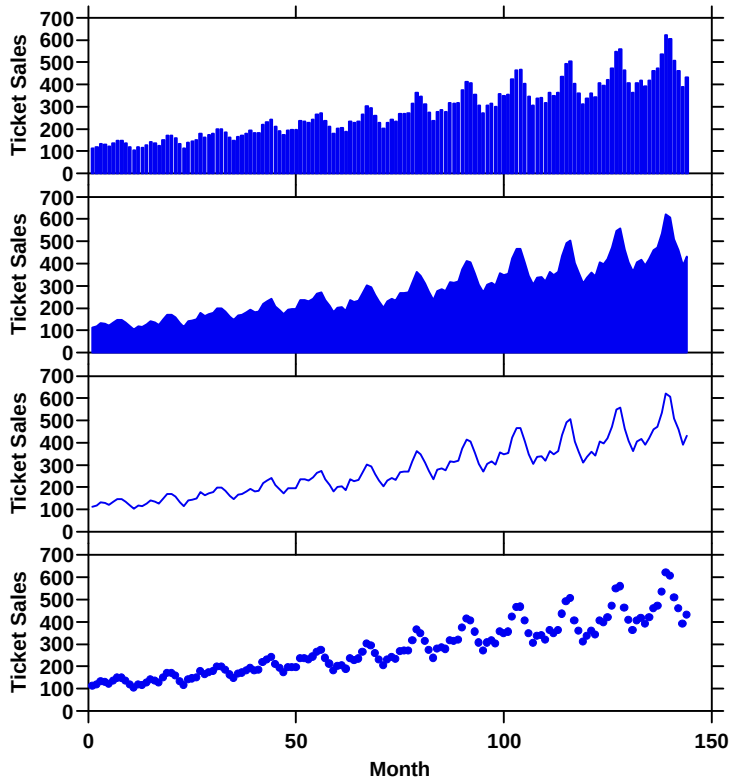
want a point estimate to be represented by a bar or spike rather than a dot. Solving this problem requires making a graph from a graph, using a composition of functions. We need another transformation object, called Geometry, in our system.

Geometry converts an SGraph (a statistical graph) to a GGraph (a geometric graph). The classes of GGraph include *point*, *line*, *area*, *bar*, *histobar*, *schema*, *tile*, *contour*, path, and *link*. With the *point* geometry object, we can represent a point estimate of a mean with a dot. With the *bar* geometry object, we can represent the same point estimate by locating one end of the bar at the coordinates of the point.

Sometimes, Geometry cannot produce a GGraph from a particular SGraph it has been given. Undefined instances (e.g., a histobar of a tree) result in null objects. These instances, some of which were exposed only when we attempted to code the complete crossing of classes, are surprisingly rare. As elsewhere in the GPL system, modularity of function and orthogonality of design increase the potential output of the system. This design strategy also encourages us to think more broadly about graphical representation.

Figure 3 shows an example. This figure displays the airline ticket sales series from Box and Jenkins (1976). Four different geometric objects are used to display the series. From bottom to top, these are *point*, *line*, *area*, and *bar*. Each highlights a different aspect of the series. The graphic in Figure 1 employs two Geometrics: *line* (for the LOESS smoother) and *tile* (for the hexbins).

Figure 3. Point, line, area, and bar geometric elements.



8 Coordinates

Most of us are used to seeing graphics in rectangular coordinates. Sometimes, as with pie charts, we are accustomed to polar coordinates. We rarely expect to see bar charts in spherical coordinates, however, or time series charts in polar. Geographers and spatial statisticians are more inclined to transform their viewing space, as a consequence of having to map the sphere to the plane.

We can generalize this capability by locating coordinate transformations in a separate object and making them work on most geometric graphs. Coordinates convert one or more geometric graphs (GGraph) to a composite graph (CGraph). A CGraph embeds one or more geometric graphs in a single frame and its associated coordinate system. The coordinate system used in Figure 1 is rectangular. It embeds the density and smoother graphics in the frame.

Figure 4. Robinson projection

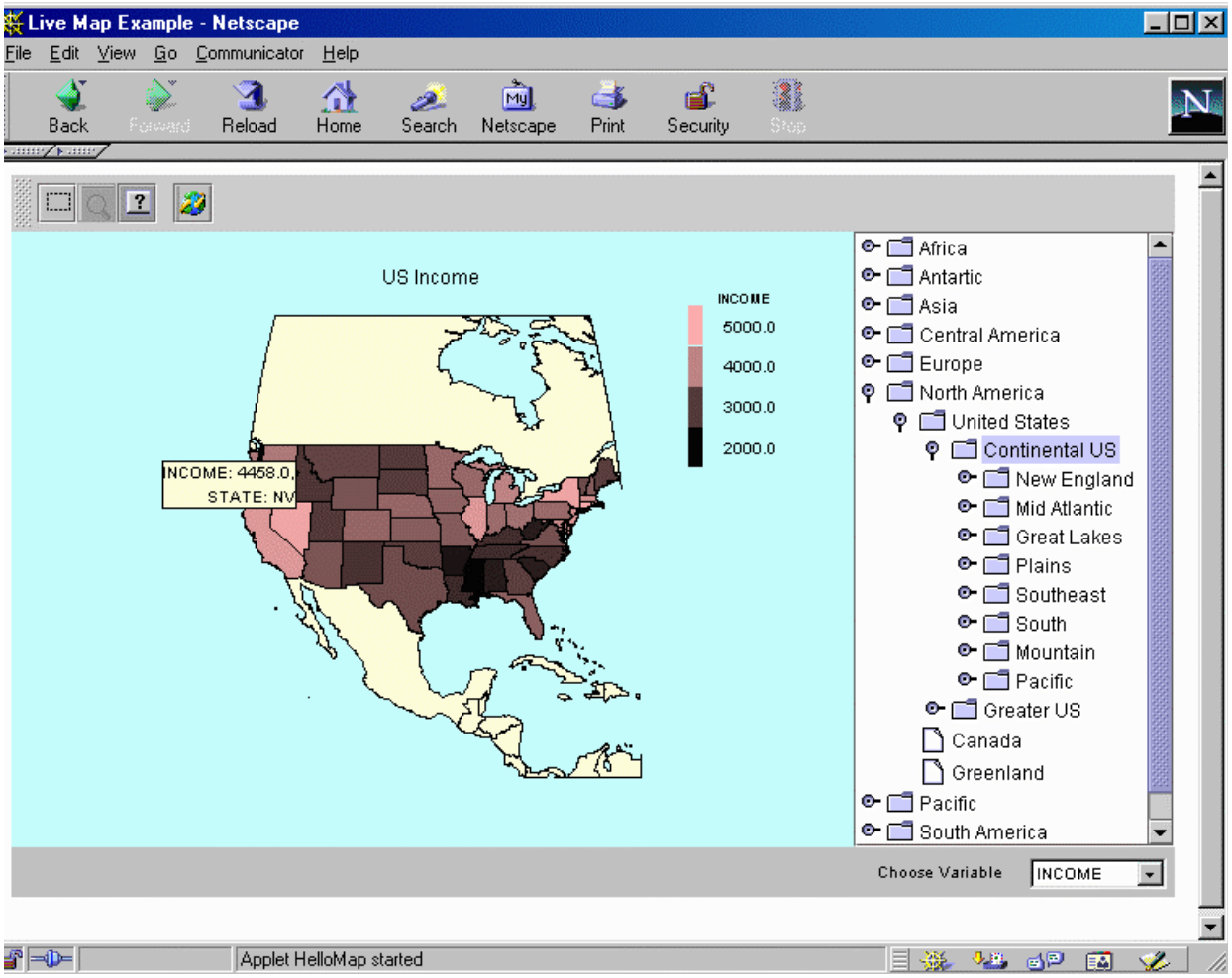


Figure 6 shows a Robinson map projection. This geographic projection is a blending of local transformations of the sphere. It was designed to give more prominence to countries in the southern hemisphere. In this example, the North American continent is clipped in a rectangular frame prior to the transformation, so the result is curved at the left and right edges and straight at the top and bottom (the Robinson projection maps latitudes to horizontal straight lines). Only objects whose coordinates are contained in the frame-bounded region are transformed by GPL. This includes axes and grid lines (not shown), but excludes legends and other annotations.

Note that the pop-up annotation has been designed to work in different coordinate systems. This is a consequence of the transformation architecture in Figure 1. Most GPL transformations are invertible, so that location messages can be passed in either direction through the pipeline.

Figure 5. Lensing transformation

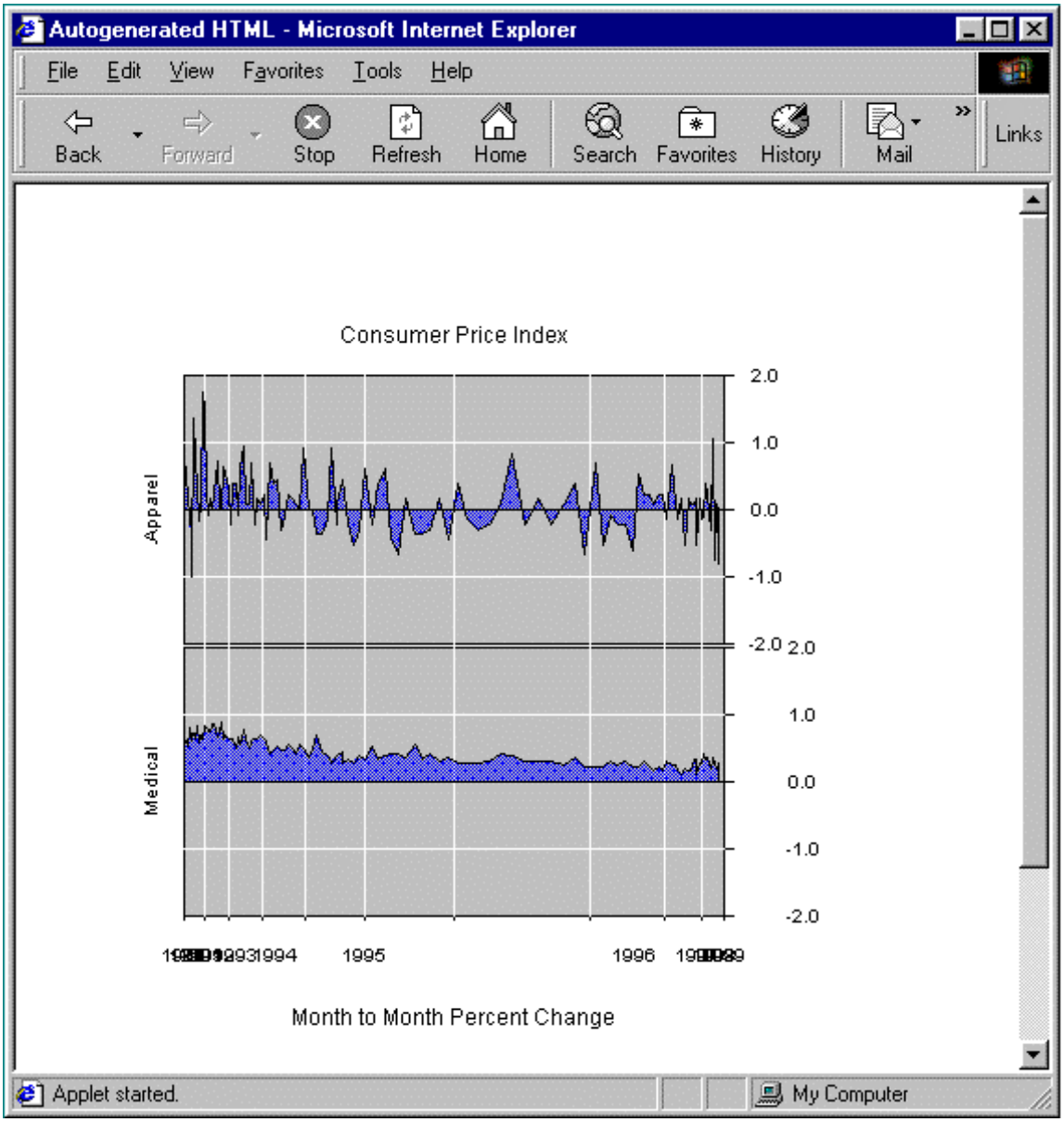


Figure 5 shows a coordinate transformation used to reveal local detail in time series. The example is taken from the Bureau of Labor Statistics' Consumer Price Index (CPI) for 1989-1999. This fisheye projection is a lensing transformation that expands the plane near the focal point. When connected to a controller and user tool (e.g., a magnifying glass), this transformation can be used in exploratory analysis. It is, in effect, a nonlinear scroller that maintains a view of the whole range of data at all lensing points. In this example, the lens is a 1D coordinate transformation applied to the horizontal dimension.

9 Aesthetics

At this point in our travel through Figure 1, we have data, a frame, and a graph, but we cannot see or otherwise perceive this graph. In order to produce the examples we have been viewing, we need to map tuples in our graph to visually perceivable attributes, called Aesthetics. Aesthetics convert a composite graph (CGraph) to a perceivable graphic. When we colloquially call a chart a graph, we are really speaking of the realization of a CGraph. A CGraph is a mathematical graph; it is not visible or perceivable. A graphic, on the other hand, is perceivable in some sense.

Aesthetics would appear to be a quaint term to use for describing the conversion of a graph to a graphic. The post-Enlightenment meaning of this term is associated with art and expression. The classical Greek meaning of the word, however, is perception -- the representation of an idea in perceivable form. GPL includes a variety of Aesthetics that extend the work of Bertin (1967). These are position, size, shape, rotation, color, texture, blur, and transparency. GPL Aesthetics also include an attribute not generally thought of as an aesthetic or visual variable: a label. A label is a text object glued to an element of a graph. In a graphic, a label functions like a color, texture, or other attribute to make a graph perceivable to a reader.

The abstraction and localization of Aesthetics in GPL yields some interesting behaviors. First, GPL can construct tables of numerals or text by using a label attached to an invisible geometric element such as a point or tile. Second, a brushing event can be attached to any attribute such as a label, color, rotation, or blur. Using a brush in one frame, for example, can cause points in another linked frame to show their labels, change their color, rotate, or blur. Third, GPL can be used to construct graphics that do not even remotely resemble XY plots. These include such images as appear in Eick (1992) and Rao et al. (1994).

10 Controllers

A Controller is an object that connects a user gesture to a function. For example, a brush is a controller that wires a user-manipulatable brushing region (usually a rectangular brush tool) to an Aesthetic through a graph-subsetting function. In subsetting a graph, we select a region that defines a subset of the values on one or more variables. When we drive this back through the pipeline, we select a subset of our VarSet. Any Frame that is dependent on the same VarSet will receive brushing messages identifying elements in the subset and these identifiers will be mapped to a selected Aesthetic.

GPL has over 30 controllers that allow a programmer to connect functions in the graphics system to user widgets such as sliders, list boxes, buttons, and modal cursor tools. These controllers extend the scope of GPL beyond visualization, making it a system for manipulating as well as

viewing data. Several of these controllers are apparent in the figures we have reviewed so far. Figure 1, for example, contains a check-box that allows an end-user to panel the display by ecoregion. Figure 4 contains a tree controller for selecting map regions. This controller can be used to drill-down a hierarchy, such as *world/continent/country/region/state/county/tract.* Figure 4 also contains several other button controllers in the upper left corner. Leftmost is a drill-down controller for selecting a subset of the map. To its right is a lensing button for zooming into a region of the map with the fisheye projection. Next is a question button for querying points on the map. This button was used to activate a pointing tool that produced the pop-up annotation shown for Nevada. A projection button on the right allows us to select a geographic projection. Finally, Figure 4 contains a list-box controller at the bottom right for selecting a variable to represent by the brightness Aesthetic. The currently selected variable is INCOME.

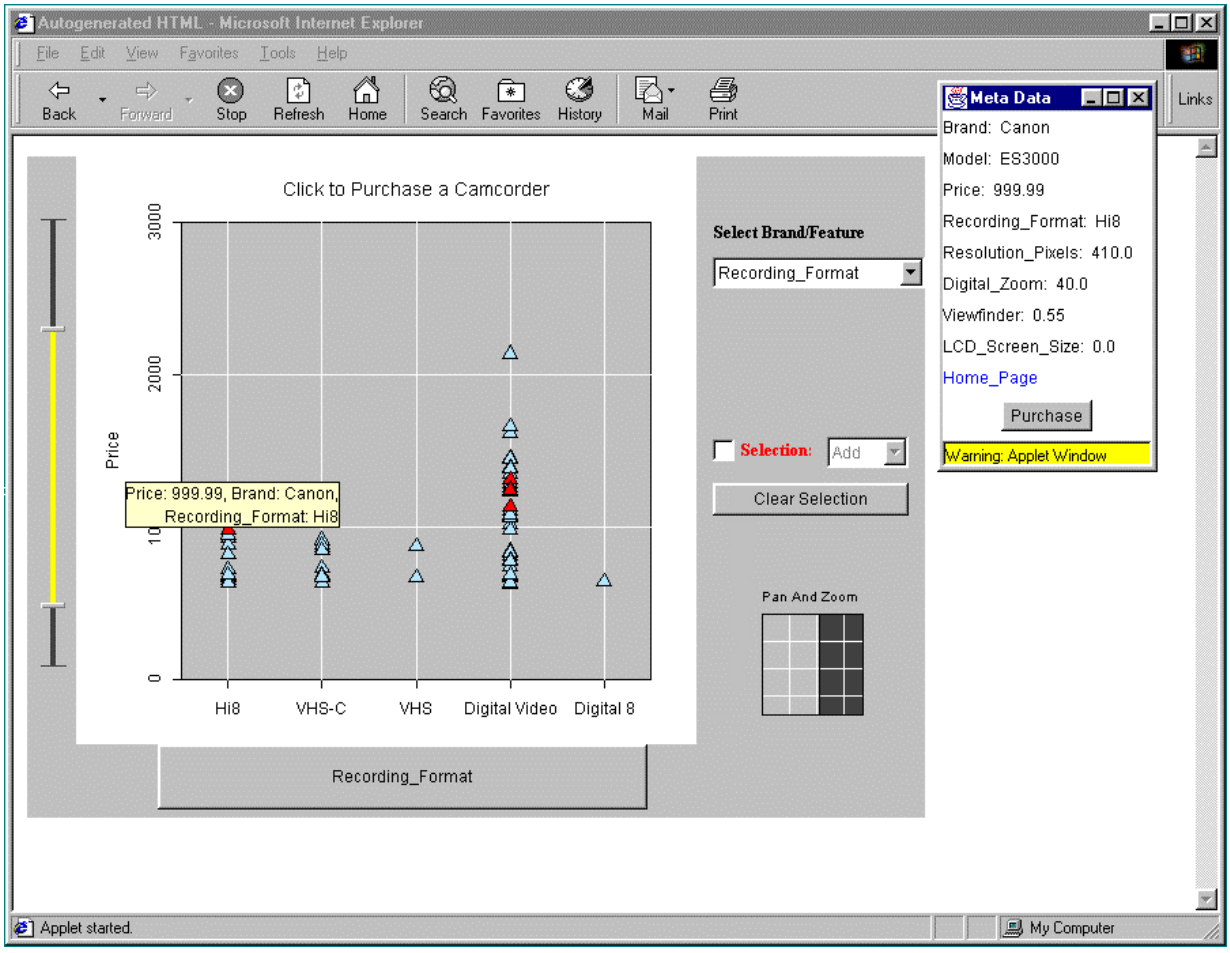
Figure 6 shows several additional controllers used to implement a Web-based ordering system for video cameras. The graphic in the figure shows a momentary state a user encountered in the ordering process. It is a plot of recording format versus price. The user produced this plot by selecting recording format in the horizontal axis list-box controller at the upper right-hand corner of the window. Prior to this point, the user had examined a plot of brand by price. We need to review what the user did in that previous state in order to understand the current plot.

While looking at the plot of brand by price, the user decided to select all the Canon cameras for highlighting in red. The user did this with the selection controller in the middle-right region of the window. Choosing the selection controller allowed the user to turn the Canon camera symbols red by clicking on them. Next, the user decided to limit the view to cameras costing more than $400 and less than $2400 by adjusting the yellow range filter controller at the left of the window. This hid all cameras outside the selected price range. At this point, the user changed the horizontal axis variable to see how Canons were distributed on recording format.

After seeing the distribution of Canons, the user decided to limit the display to only five formats. The user did this with the 2D pan-and-zoom controller at the bottom right of the window. The setting on the controller forced the frame to display only the left half of the horizontal axis, which includes the five tick marks. Next, the user found a suitable camera by touching the leftmost Canon symbol with the cursor. The popup annotation controller revealed summary details. Finally, the user decided to place an order for the camera by double-clicking this symbol. The pop-up metadata window controller appeared to the right of the display. By clicking on the Purchase button, the user was taken to the Canon Website to order the camera.

Many of the actions described here resemble what database and data mining engineers call drill-down methods. These methods are invoked when a user selects subsets of a multidimensional array of data for closer examination. There is an important functional difference between the GPL and ordinary drill-down implementations, however. Ordinary drill-down graphics are limited to pie or bar charts because the variables are defined to be categorical so that they can be indexed in the database. By contrast, GPL begins with a graphic definition of subsetting; subsets on continuous or categorical variables are selected directly in \em any \em graphic using sliders, pointers, lassos, and other tools.

Figure 6. Controllers for a Web ordering system.



11 Conclusion

The contrast between different interpretations of drill-down points to the main distinction between GPL and relational data mining systems that employ graphics. Data mining systems based on relational databases begin by defining, organizing, and modeling data. In these systems, graphics are treated as passive views into data. Even when such systems implement graphical controls for manipulating data, they are defined in terms of the data model underlying the system. In short, these systems begin by specifying a range and domain for the function mapping elements of data to a relational table or hierarchical or associational object. All graphics possible in such systems follow from these definitions.

GPL, in contrast, begins by defining, organizing, and constructing a graphic. As Wilkinson (1999) stated, "These definitions are embedded in the mathematical history that determined the evolution of statistical charts and maps." In short, we begin by considering what is the range and what is the domain of a graph underlying a graphic. From there, we recurse our definitions until we reach a specification of data underlying the graphic. For that specification, we construct an abstract DataView and link the graphic to our data.

In an early attempt to develop a statistical metadata standard, Dolby, Clark, and Rogers (1986) described a language of data. This project was aimed at developing a general language for organizing data, metadata, and images. While such high-level analyses can sometimes be fruitful, we believe that tailoring languages to more specific problem domains can sometimes yield more powerful capabilities. Computer generated graphics have for too long been regarded as views of pre-determined data structures. It is time to consider the possibility of structuring data to fit the view rather than structuring the view to fit the data. This effort requires a language of graphics.

References

Becker, R.A., and Cleveland, W.S. (1987). Brushing scatterplots. *Technometrics*, 29, 127-142.

Bertin, J. (1967). *Sémiologie Graphique.* Paris: Editions Gauthier-Villars. English translation by W.J. Berg as *Semiology of Graphics*, Madison, WI: University of Wisconsin Press, 1983.

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

Buja, A., Asimov, D., Hurley, C., and McDonald, J.A. (1988). Elements of a viewing pipeline for data analysis. In W.S. Cleveland, and M.E. McGill (Eds.), *Dynamic Graphics for Statistics*. Monterey, CA: Wadsworth, 277-308.

Carr, D.B., Kahn, R., Sahr, K., and Olsen, A.R. (1997). ISEA Discrete Global Grids. *Statistical Computing & Graphics Newsletter, 8,* 31-39.

Carr, D.B., Olsen, A.R., Pierson, S.M, and Courbois, J.P. (1999). Boxplot variations in a spatial context: An Omernik ecoregion and weather example. *Statistical Computing & Statistical Graphics Newsletter, 9,* 4-13.

Carr, D.B., Olsen, A.R., Pierson, S.M, and Courbois, J.P. (2000) Using Linked Micromap Plots To Characterize Omernik Ecoregions. *Data Mining and Knowledge Discovery, 4,* 43-67.

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

Cleveland, W.S. (1985). *The Elements of Graphing Data.* Summit, NJ: Hobart Press.

Daly, C., Neilson, R.P. and Phillips, D.L. (1994). A Statistical-topographic Model for Mapping Climatological Precipitation over Mountainous Terrain. *Journal of Applied Meteorology, 33,* 140-158.

Dolby, J.L., Clark, N., and Rogers, W.H. (1986). The language of data: A general theory of data. *Computer Science and Statistics: Proceedings of the 18th Symposium on the Interface*, 96-103.

Eick, S.G., Steffen, J.L., and Sumner, E.E. (1992). SeeSoft -- a tool for visualizing line oriented software statistics. *IEEE Transactions on Software Engineering}, 18*, 957-968.

Fisherkeller, M.A., Friedman, J.H., and Tukey, J.W. (1974). PRIM-9: An interactive multidimensional data display and analysis system. SLAC-Pub-1408. Stanford, CA: Stanford Linear Accelerator. Reprinted in W.S. Cleveland, and M.E. McGill (Eds.), *Dynamic Graphics for Statistics.* Monterey, CA: Wadsworth, 91-109.

McDonald, J.A., and Pedersen, J. (1991). Geometric abstractions for constrained optimization of layouts. In A. Buja, and P.A. Tukey (Eds.), *Computing and Graphics in Statistics*. New York: Springer-Verlag, 95-105.

Omernik, J.M. (1987). Ecoregions of the coterminous United States. *Annals of the Association of American Geographers}, 77,* 118-25.

Omernik, J.M. (1995). Ecoregions: a spatial framework for environmental management. In W.S. Davis, and T.P. Simon (Eds.), *Biological Assessment and Criteria: Tools for WaterResource Planning and Decision Making*. Boca Raton, FL: Lewis Publishers, 49-62.

Rao, R., and Card, S.K. (1994). The table lens: Merging graphical and symbolic representations in an interactive focus+context visualization for tabular information. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems, 18*, 318-322.

Stuetzle, W. (1987). Plot windows. *Journal of the American Statistical Association}, 82*, 466-475.

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.

Tierney, L. (1991). *LispStat.* New York: John Wiley & Sons.

Upson, C., Faulhaber, T., Kamins, D., Schlege, D., Laidlaw, D., Vroom, J., Gurwitz, R., and vanDam, A. (1989). The application visualization system: A computational environment for scientific visualization. *IEEE Computer Graphics and Applications}, 9*, 30-42.

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

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