Orca: A Visualization Toolkit for High-Dimensional Data

Peter SUTHERLAND, Anthony ROSSINI, Thomas LUMLEY, Nicholas LEWIN-KOH, Julie DICKERSON, Zach COX, and Dianne COOK

This article describes constructing interactive and dynamic linked data views using the Java programming language. The data views are designed for data that have a multivariate component. The approach to displaying data comes from earlier research on building statistical graphics based on data pipelines, in which different aspects of data processing and graphical rendering are organized conceptually into segments of a pipeline. The software design takes advantage of the object-oriented nature of the Java language to open up the data pipeline, allowing developers to have greater control over their visualization applications. Importantly, new types of data views coded to adhere to a few simple design requirements can easily be integrated with existing pipe sections. This allows access to sophisticated linking and dynamic interaction across all (new and existing) view types. Pipe segments can be accessed from data analysis packages such as Omegahat or R, providing a tight coupling of visual and numerical methods.

Key Words: Brushing; Compositional data; Data projections; Dynamic graphics; Interactive graphics; Java; Motion graphics; Multiple linked views; Multivariate space-time data; Object-oriented software; Plot matrices.

1. INTRODUCTION

Since the late 1960s, research in statistical graphics has exploited the technological advances provided by the computer and electronics industry. Hardware such as specialist graphics terminals facilitated a major advance in using graphics to assist in data analysis. For the first time moving pictures of data were displayed and a user was able to interact with a plot in real time. For example, Chang (1970) explored rotations in 5-D to detect
2-D structure; Kruskal (1964) watched a multidimensional scaling algorithm converge to a stable configuration; and Fowlkes (1969) explored interactive probability plotting. A seminal software development, called PRIM-9 (Fisherkeller, Friedman, and Tukey 1974), was the first dynamic multivariate data visualization system. PRIM is an acronym for Picturing, Rotation, Isolation, and Masking. It had tools for drawing plots, rotating variables into the plot, and conditionally masking points according to variable values. For the reader interested in technical history, PRIM-9 was implemented on an IDIIOIM vector scope driven by a Varian 620 minicomputer connected to an IBM 360/91 mainframe. It so monopolized the computing power of the mainframe that all other computing jobs came to a standstill while it was running!

Specialist graphics terminals located in remote sites required that the user make an effort to move physically, and share the resources with other researchers, to do graphics on their data. Hence, there was a high degree of inconvenience inherent when working in visualization, not unlike the inconvenience of using high-end virtual reality equipment today. The type of display device has changed from being vector-based, essentially line art, to raster-based; that is, pointillist, which changed the nature of graphics. The access to sophisticated raster displays is now universal. With workstations, quality graphics can be done conveniently on one’s desk rather than on remotely located specialist equipment.

Along with the hardware advancements have come developments in software. The first data graphics required considerable low-level programming to make the graphics terminals draw. An example, as verbally communicated by Andreas Buja, is that John McDonald “programmed Orion [McDonald 1982] in Pascal on a Sun board, and in Mortran on the IBM-360 emulator. The Sun board had no real operating system, but it had some Pascal routines that acted as a raster display device driver. The code was cross-compiled on a large IBM mainframe and the binary was downloaded over a data line.” What a nightmare! (And yes, it was Mortran, not Fortran.) Thankfully, operating systems have developed considerably, and the common way for humans to interact with the computer is through a window system on the display, such as MacOS, X11 for Unix workstations, and Microsoft Windows. Similarly, programming languages have developed from formula-based, such as Fortran, to object-oriented schemes for organizing complex systems, such as C++.

Along with the technological advances has come a dilemma for researchers working in dynamic graphics: how to adequately describe their work on the printed page. Of course, it is near to impossible. Some researchers have used film and now video technology to record their work for posterity. Many videos are carefully archived in the American Statistical Association Statistical Graphics video lending library (found by following links from www.amstat.org).

It is fun to periodically revisit history to put our current research environment in context, and contemplate where we go from here. For statistical graphics an exciting new technology is the Java programming language, an object-oriented, cross-platform, computing environment from the folds of Sun MicroSystems Inc. Java unifies the hardware world facilitating a “write one, run anywhere” approach. This article describes the development and structure of a new visual toolkit for statistical data visualization that exploits the software innovations made available by Java. The toolkit evolves from ideas that underly the software XGobi (Swayne, Cook, and Buja 1998). When confronted with
a new technology, it is important not to simply carry through old habits and customs, but to understand the complexities and richness offered by the new tools, to foster novel thinking and creativity in future work. In this spirit, we begin by discussing the foundations of statistical graphics thinking, and develop these into the software design principles for data visualization systems using Java. Two applications are described that use the toolkit to provide new types of dynamic graphics for exploring compositional data and multivariate time-dependent data.

2. FOUNDATIONS OF DATA VISUALIZATION

Data visualization is the science of picturing data, where "data" is defined as information that exists in some schematic form such as a table or list. Data are often but not always quantitative, and some translation of unstructured information is often required to derive the data. Data always include some attributes or variables, such as the number of atoms in a molecule, or the value of the Australian dollar relative to the U.S. dollar, or the weight of a crab.

Data visualization differs from information visualization, scientific visualization, and cartographic visualization. Information visualization is broader than data visualization. It seeks to visualize more generally unstructured information—for example, visualizing lines of code in software (Eick 1994). Scientific visualization is primarily concerned with visualizing 3-D, or 3-D + time phenomena, such as for medical purposes, displaying molecular structure of drugs, or in construction projects, displaying architectural prototypes. It involves more physical realism. Cartographic visualization concerns visualizing maps, geography, and spatial domains. But these types of visualizations are not mutually exclusive, and indeed it is common that data arise in conjunction with a geographic component, or from restructuring lines of code into counts of particular expressions, or from databases of chemical properties of molecules. So it is common that data visualization needs to be done simultaneously with the other types of visualizations.

A characteristic of data visualization is the concern with abstract relationships among such variables: for example, the degree to which income increases with education. In data, the number of variables is arbitrary: 5–10 are common, and 50 and even hundreds of variables often arise in very real contexts. If we think of each variable as plotted on an axis, with values of that variable mapped along the axis, and that each axis represents a dimension, then the challenge for data visualization is how to picture more than two or three variable dimensions simultaneously on a 2-D paper surface. The solution is available with computers.

2.1 REPRESENTING HIGH-DIMENSIONAL DATA

The approach we advocate for drawing plots of data is called the "multiple views" paradigm. Multiple plots, corresponding to different views of the data from multiple aspects, are shown to the user simultaneously. Interaction tools facilitate linking information between plots. The methods adhere to very specific guidelines for graphics: use simple, easy-to-read plots, with a healthy collection of interaction tools, such as refo-
At the root of graphical methods in statistics is a division of data visualization into three areas:

1. Rendering, or what to show in a plot.
2. Manipulation, or what to do within plots.
3. Linking, or what information to share between plots.

The first area, rendering of data, comprises all decisions that go into the production of a static image. Rendering is concerned with the appropriate representation of information in data variables: a scatterplot, a density plot, a time series plot, or a parallel coordinate plot are examples of renderings. Wegman and Carr (1993) gave an excellent introduction to the wide array of rendering methodology in statistical data visualization.

The second area, the manipulation of plot elements, refers to how we operate on individual plots and how we organize multiple plots. The purpose of these manipulations is to support the search for structure in data.

The third area, linking, refers to the connection of elements from one rendering to another rendering. We most often think of linked brushing, where points are given the same appearance (color, glyph) between renderings. More generally, linking can include matching scales, matching axes, linking a point to a record in a database, or index of chemical compounds. In the practice of data visualization, there usually exists a larger context of open-ended problem solving. In such contexts, data visualization systems are most useful if they provide plot manipulation tools that support extensive searching and linking of information.

Behind the process of rendering is the concept of a data pipeline, described by Buja, Asimov, Hurley, and McDonald (1988). [See also Weihs and Schmidli (1990) and Wilkinson, Rope, Carr, and Rubin (2000, p. 530 of this issue) for related work.] It is...
assumed here that the data come in the form of a matrix, with \( n \) rows corresponding to cases, and \( p \) columns corresponding to the number of variables. The data pipeline is the conveyor belt which takes the raw data through a series of transformations to go from \( p \)-D data to a \( d \)-D rendering, where \( d < p \). (Most commonly \( d = 2 \).) Figure 1 illustrates the data pipeline for multivariate data. The steps that we describe, transform the data from a raw form, as described above, to a unit-free visualization world, and finally, projected to a 2-D planar viewing frame (possibly enhanced to 3-D by virtual reality technologies). The concept of a data pipeline parallels ideas in computer graphics to render a physical scene realistically (Foley, van Dam, Feiner, and Hughes 1990).

Typically the first step (between raw and world) in a data pipeline includes some type of variable standardization. Examples of standardization are standardizing each variable to mean 0, variance 1, or transforming to fit within a limited range such as \([0, 1]\).

The second step (between world and planar) primarily involves dimension reduction. Dimension reduction may be done using variable selection, or through projection methods such as principal components or discriminant coordinates. Motion graphics, such as tours (Asimov 1985; Cook, Buja, Cabrera, and Hurley 1995; Buja, Cook, Asimov, and Hurley 1997), can be applied to address the dimension reduction problem, also. In a tour, although the number of variables may remain the same, the tour algorithm projects the data into a continuous sequence of low-dimensional planes. The second step can also be considered to be the stage where a graphical object is built: dimension reduction and construction of components of a specific rendering of the data, including axes. It is also the appropriate place for considering what manipulation modes are to be implemented with rendering; for example, a slider for changing bin widths in a histogram view.

The third step essentially scales the data into the viewing window. The screen coordinates are also the location for translating mouse actions on a window into manipulation actions on the data.

When the data come with a much broader mixture of variables, such as time, space, or sampling weight, an expansion of the pipeline is required, in terms of the operations at each step or in adding more steps. Figure 2 illustrates such a pipeline. At each step, variables that describe the sampling structure rather than actual measurements need to be handled differently.

The first step may now include casting variables into statistically relevant types for visualization. This would include tagging them as time variables, space variables, or to be used for weighting measurements. Time variables may need to be passed on in time order, and it is common to construct lags of variables. A weight variable may be separated out to define point appearance in a later plot.

The second step needs to take the type of data into consideration when choosing a dimension reduction method. For example, in a canonical correlation problem, where there are multiple responses and multiple explanatory variables, it is appropriate to do dimension reduction separately on each group. The second step can be considered a restructuring stage where lag variables are constructed for time-dependent data. It may also involve constructing a structure to maintain linking information in complex cases—for example, linking lag plots to other types of plots. The construction of views of the data require tradeoffs between comprehensive representation of high-dimensional data distributions due to reduced plot space used for time or space variables. For example, a
Figure 2. Data pipeline representing multivariate data, in combination with other variable types such as time and space.

2-D time series view that preserves the time structure leaves only one dimension to show projections of the multiple variables in the data. It is useful to augment this view with a second view that suppresses the time axis and shows 2-D projections. The tradeoffs are more severe for data measured over space, where effective display of even one dimension of attributes is challenging.

We emphasize that we are not constrained to just four steps in the pipeline. It may be natural to think of more steps that have singularly and specifically defined functionality.

Now turning from pipelines to multiple views. The multiple views method specifies that the data pipeline is like a river delta, piping the data out into multiple renderings. The difficulty is to define appropriate mechanisms to link information from one plot to another. In the simplest case of linking, where one view has a scatterplot of variable 1 versus variable 2, and another view has a scatterplot of variable 3 versus variable 4, the points in each plot are linked one-to-one, the correspondence of a point in one plot is to a point in the other plot (Figure 3). More complex types of linking arise often, for example, if there is a time or spatial component to the data. In the situation where there is a spatial component we need to explore the spatial dependence between sample points. So we may have a point, in one view as an element of a variogram cloud, corresponding to a pair of locations in another view, the map. In the case of time-dependent data it is often desirable to link a point in one view to a time series in another view. An example might be a multivariate longitudinal study, where there are both demographic variables for each patient, and multiple measurements over a follow-up period. We could think of this as one-to-many, or indeed many-to-one. Other types of plot element linking are common as well—for example, axis scale and projection coefficients. In general, linking can be as complex as one’s imagination can conceive, but in real-time implementations we also need to consider speed of interaction. One-to-one linking is very commonly implemented: it is conceptually simple and it can be done extremely quickly. More complex schemes
may need an intermediate broker, which can slow down response time. Figure 4 illustrates linking of information in association with the data pipeline, where the appearance of a line is associated with the appearance of a point, through a row-wise appearance variable defined with the raw data.

When plot elements are linked, it ensures that manipulation of elements in one plot directly affects the representation of the data in the other plots. The taxonomy of manipulations is described in Buja et al. (1996). Here we provide a short summary:

- Focusing views: By focusing we mean any operation that is an extension of manipulating a camera, such as deciding from which side to look at the object and in which magnification and detail. Focusing views includes choosing the variables or (more generally) the projections for viewing, but also choosing aspect ratio,
magnification (zoom), and location in the data space (pan).

- Posing queries: In graphical data analysis it is natural to pose queries graphically. For example, with the familiar brushing techniques, coloring or otherwise highlighting a subset of the data means issuing a query about this subset. It is then equally natural that the response to the query be given graphically. This is achieved by showing information about the highlighted subset in other views. It is therefore desirable that the view where the query is posed and the views that present the response are linked. Ideally, responses to queries are instantaneous.

- Arranging many views: One informal technique is to arrange large numbers of related plots for simultaneous comparison. Useful arrangements are matrix-like, such as in scatterplot matrices of pairwise variable plots, but other arrangements such as conditional plots (co-plots) are also useful. The most common known arranging views approach is trellis graphics (Becker, Cleveland, and Shyu 1996).

In statistical terms, we can further categorize some types of manipulations. Linked brushing can be considered to be exploring conditional distributions of variables, where the brush is a conditioning tool. On the other hand, statistically, motion graphics such as the tour facilitate exploring the joint distributions, because the motion facilitates perception of the “shape” of the data from the sequence of marginal views. If we know the distribution of all low-dimensional projections of the data, then we also know the joint multivariate distribution, following a result of Cramér-Wold (Mardia, Kent, and Bibby 1979).
2.2 Restructuring Data

Clever restructuring of variables is perhaps one of the most hidden, yet valuable, tools for data visualization. Particular types of data lend themselves to obvious approaches to restructuring: data with a time or space component, modular variables such as wind direction, or compositional data where variables contain a constraint. When there is a time or space component it is important to explore the time or space dependency using lag plots or variogram cloud plots. With time, it is also likely that there are different scales of time (daily/weekly/yearly) to explore. Regrouping these different resolutions into different variables facilitate exploring trend (yearly or weekly). A variable such as wind direction can be better handled in the interactive setting by using sine and cosine values. This allows the user to brush around the compass points to explore the relationship with other variables. Compositional data is best approached by preprojecting the data into a subspace orthogonal to the variable constraint, a \((p - 1)\)-D simplex, called a ternary diagram in 2-D. In situations where modeling is a part of the analysis, components associated with the model are useful to append to the dataset, and appending samples or quantiles from standard distributions facilitates inference. When the goal of modeling is classification, exploring a dendrogram in association with the variables may illuminate clustering in the data. With any type of prediction, appending the predictions, residuals, or diagnostics to the data can help improve the visualization of the model.

3. ORCA DESIGN

The main design principles behind Orca have been to provide researchers with a graphics framework that allows easy development and incorporation of new types of graphics with existing Orca implementations and other statistical languages. Ideally, integration with a graphics framework should be simple, yet allow access to sophisticated linking and dynamic interaction across all Orca view types. Additionally, basic infrastructure for file parsing and standard data interaction should be provided by the system, so the developer need only concentrate on the specific aspects of rendering and implementing graphics that interest them. The amount to which they adhere to the few simple principles will reflect the level of system integration that their graphics exhibit.

Orca has evolved as a programming framework that enables new graphics to be quickly developed and linked to existing modules. While it is possible to create an application that provides run time access to all the features that Orca provides, there are no plans to implement such an environment. Several simple command-line and GUI-capable environments already exist that provide the ability to link and interact with pipeline segments in real time (BeanShell, www.beanshell.org; JPython, www.jpython.org; OmegaHat www.omegahat.org; R, www.r-project.org).

3.1 ORCA VISUALIZATION FRAMEWORK OVERVIEW

The Orca framework separates different aspects of data processing and rendering into segments of a pipeline. A complete graphics pipeline links some or all of the following
segments: Source pipe, Preprocess pipe, Transformation pipe, Tour pipe, Render pipe, Window pipe. Each segment of the pipeline performs its function independently of any other pipes that are connected to it, and communicates to the adjacent pipes using a limited set of functions described by the OrcaPipe interface. This allows for the design of novel pipe segments and in linking together previously created pipes, but does sacrifice some error checking. For example, the order of the pipes can be important but is not enforced by the software.

In a system of OrcaPipes, a pipe segment will link backward to only one pipe but may be linked forward to any number of other pipes. By branching multiple pipe segments in this way, one can easily create multiple views of the same data source, and by choosing the stage at which the branches diverge one can control what linking and viewing information is shared between these branches. For example, two different windows will provide 1-D and 2-D views of the same tour if their pipelines branch downstream of a tour pipe, or they will have independent tours but common linking if the branch occurs upstream of the tour pipe. Java Interfaces are used to define the structure of this graphics pipeline metaphor and they are the key element that allow the flexibility required in coupling pipe objects.

3.2 Java Interfaces and Object-Oriented Design

In object-oriented programming, polymorphism provides the ability to refer to an object by any of the classes or interfaces that it extends. With careful use of polymorphism through interface classes, rich systems of objects can be built. This richness stems from making an object's references to other objects at the most abstract level. In Orca, we make substantial use of Java Interfaces to provide these abstract reference points. Java Interfaces allow a programmer to specify certain methods that an object must implement but leaves the specifics to the designer of each instance of that interface. Objects that implement interface methods can then be referred to via that interface type. This simple contractual agreement allows for looser coupling between objects without some of the complications that can arise from traditional object inheritance as found in C++ and Java inheritance classes.

Objects that are only interested in the functionality that is specified through an interface will only need to refer to objects that display such functionality through that particular interface type (see, e.g., Section 3.4 on Observables). Referencing objects at this abstract level allows an object to interact with any number of different classes that implement the same interface, without needing to know their specific class. This is how pipes are able to couple with other pipes regardless of what specific functionality they provide. The coupling mechanism only sees that the object upstream or downstream extends the OrcaPipe interface. It does not need to know anything more specific.

Interfaces approach polymorphism with the idea that the contract methods provided will allow objects to implement similar but unique functionality. For a mathematical analogy, the real numbers can be thought of as implementing the vector space, field, algebra, and metric space interfaces, independent of whether they are constructed from Cauchy sequences or Dirichlet cuts, or exist purely by assumption. We choose the inter-
face that guarantees the desired properties, and do not worry about the implementation. Throughout the core Orca objects, we make object references as abstract as we can and in doing so allow them to interact with any objects providing the core functionality required by the interface.

3.3 DETAILS OF THE FRAMEWORK

Seven Orca interfaces make up the foundation: OrcaPipe, OrcaData, OrcaAppearance, OrcaNavigation, OrcaControl, OrcaEvent, and OrcaCommand.

The basic pipe section functionality is specified by the OrcaPipe interface. Here contract methods, to assure pipe section linking and proper data flow, are defined. An example of multiple pipes linked up to create a complete pipeline is illustrated in Figure 5.

Each of the seven interface types serves a distinct function in the Orca framework. OrcaPipe implementations provide the basic infrastructure for linking other pipeline segments and passing data. OrcaData, OrcaAppearance, OrcaNavigation, and OrcaControl can all be thought of as types of data. OrcaPipe segments are responsible for passing these four types of data through the system. Each data type is passed through a separate channel. OrcaEvents and OrcaCommands provide ways for pipe segments to communicate among one another, to update or augment the dataset.

OrcaPipe objects handle aspects of linking pipe segments and of propagating data and events along the pipeline. The structural properties involving linking segments are common to all pipe objects and therefore implemented in an abstract base class that all other OrcaPipe objects can inherit from. These linking methods include functions to maintain connections between pipes and to branch pipes. Methods for event propagation are also supplied by this abstract class.

Individual methods in the OrcaPipe interface are responsible for implementing the
four major data flows through each section. These flows (core data, navigation data, appearance data, and control methods) will at a minimum allow data from the previous pipe to flow through to the downstream pipes as needed. Often a section of pipe may only augment one or two of the types of data in some way as it passes through. For the data channels that leave the data untouched, methods should simply pass the object through to the next segment of pipe.

The segments of the pipeline communicate with four data channels:

1. OrcaData. The OrcaData interface specifies five contract methods that a pipe must implement: `getValue(int x, int y)`, `getNumRow()`, `getNumCol()`, `getAttribute(String name)`, and `setAttribute(String name, Object attribute)`.

   With the exception of setting attribute information the object is immutable (it is not possible to change it), and will provide only information about the size of the data and access to the data values. To operate on the data once it is in the OrcaData format the OrcaData object must be wrapped with another OrcaData object. By allowing OrcaData objects to “wrap” or adapt other existing data objects a programmer can add functionality to an object without needing to make his object aware of all other existing functionality.

   The basic OrcaData object maintains a local array of data or a link to another OrcaData object that it can use as a data source. This allows several data objects to chain together and delegate operations to data objects farther down the chain if it is queried for information it does not have locally.

2. OrcaAppearance. For each root OrcaData object there is a corresponding OrcaAppearance object. The handle to this object can be requested by any section of the pipeline through the pipeline’s `getAppearance` method. The client object that requests the OrcaAppearance object must register itself to allow for method callbacks. Both objects will then have handles to one another, allowing the OrcaAppearance object to contact the client object when changes have occurred to the appearance data and also allowing the client object to notify the OrcaAppearance object of any changes it has made to the appearance data.

   The OrcaAppearance object allows views to register as observers and then request different appearance types. An appearance type is nothing more than a name representing the type of appearance and an integer array that represents an appearance state of each row of the OrcaData object. It is the responsibility of the client object to maintain a naming convention that is consistent with the other Orca graphics. Additionally, there are no set types of appearance; if an appearance is requested that does not yet exist the OrcaAppearance object will create and initialize one by that name. This means that new appearance attributes can be added at any time, without needing to update pre-existing code to handle or ignore them.

   Information that is associated with the appearance object can be thought of as information associated with the rows of the data object. The most immediately visible row appearance is the color used to draw each point or line.

3. OrcaNavigation. The functions of the OrcaNavigation objects are not currently as well resolved as the other types. We see navigation data as representing information regarding variables, or columns of the data. Information about axes, limits of scale,
ORCA: A VISUALIZATION TOOLKIT

and variable selection can be managed through the OrcaNavigation object. To date only information about axes is routed through the navigation channel.

Navigation views will augment the data views. These subordinate graphics may also provide additional control functionality to an aspect of the pipeline. They may provide any type of information for placing the data in the context of the variable space, which might include geographic mapping, scale axes, or directional axes. An example of a navigation view is the tour axes: how each variable in a tour contributes to the linear combinations that are in the data view. This allows the user to see which variables are important when an interesting feature is visible.

4. OrcaControl. The OrcaControl channel passes a collection of control widgets and other GUI elements that can be used to manage different segments of the pipeline. As the control panel object is passed forward through the pipeline, it allows pipe sections to add control panels to a single container of controls that it manages. This allows a collection of controls to be available to the window segment of pipe from any or all of the other segments. Control panels make extensive use of interfaces for interacting with pipe segments as it is anticipated that many of the control panels may be reused. Examples of OrcaControls include the speed control for a tour and the color chooser for brushing.

The OrcaPipe interface includes methods for propagation of events and commands throughout the system, as well as the four main data channels. OrcaEvents provide a method for communication between pipes; OrcaCommands provide a way of managing simple operations on the OrcaData objects. By making these operations objects it becomes straightforward to control when and how they execute their operations. As objects they can delay their execution until they have all the parameters they need. They can also be more easily queued and threaded for more efficient batch processing.

An OrcaEvent is propagated through the pipeline and can check the type of each segment to find an appropriate segment to operate on. Once it has located the segment it can either operate directly on the pipe segment, execute an OrcaCommand on the data at that segment, or create a new OrcaData type and add it to the data chain. For example, loading and system notification of a new data set into an existing pipeline is currently implemented by an OrcaEvent.

OrcaCommands in some ways duplicate the ability of OrcaData objects to operate on the data they manage. OrcaCommands should be considered for lighter weight operations. Modifying or extending the OrcaData interface is a relatively major task, and as noted previously, long chains of OrcaData objects incur overhead in processing. Operations for adding attributes to a data object would be a good candidate for an OrcaCommand object. For example, an OrcaCommand is used to specify that a particular variable is a space or time index rather than an observed measurement.

3.4 CALLBACKS AND THE OBSERVER PATTERN

Linking between views is perhaps the most important functionality that Orca provides. Views are considered peer objects with one another. There is no master-slave relationship where a single view is responsible for linking or notifying other views of
state change. Instead all views link directly to an OrcaAppearance object that acts as a broker between them. Individual views do not need to know about the other views that share the same dataset, they need only to concern themselves with the state of the appearance data and act accordingly when it is modified. To achieve this, each view must register itself with the OrcaAppearance object so that both objects, the view and the appearance, can actively contact each other. This type of object coupling is referred to as a callback.

The basic idea underlying a callback is that one object registers an interest in the state of another object and is notified when state changes are made. In a real-world analogy this is equivalent to leaving your phone number rather than continuously redialing or holding the line (Warren and Bishop 1999). In Java the ability to organize these callbacks is implemented with the Observer interface and the Observable object. The OrcaAppearance objects are based on the built-in Observable class, which provides methods to maintain a list of Observers that are interested in the object. When the object changes, the Observers are notified and can update themselves appropriately.

This Observable object then becomes a type of broker between all objects interested in the same properties. They do not have to know about each other, only the single broker that they communicate to each other through. This is also a good example of the strength that Java Interfaces provide. Here the Observable can interact with any type of object that implements the Observer interface. The Observable is only interested in the one contract method that the Observer interface specifies (update()) and does not need to know anything else about the object.

3.5 Adaptor Pattern and Operations on the Data Form

Several complications with data management arise while the data objects are propagated through the pipeline. Some operations on the data are of the kind that should only be visible to the downstream side of the pipe, while there are other operations that should be global in scope. Adaptor objects are commonly used in Java; they are a way of providing additional functionality to existing objects by wrapping them. Adaptor objects do this by implementing the same methods (often under an interface) and act as an interpreter for the values that they return and the object requesting the method call.

To create a system that can both localize some of the data and also access other global attributes, the OrcaData objects are designed to allow chaining together of multiple data objects. Each OrcaData object in a chain of objects will act as an adaptor for the previous object. It will respond to any requests that it recognizes by interpreting the data from the OrcaData object that it wraps. For the requests that do not pertain to its function it will delegate to the next object down the chain and return these results untouched.

This scheme of data management presents important tradeoffs. By chaining objects each can be designed with only its main function in mind. However, by keeping the functionality of each object restricted to a small domain, it might mean that a chain of data objects could become quite long to incorporate all the operations needed on a dataset. Longer chains will increase the overhead of operations by increasing the number of method calls that are needed to access the data.
3.6 VIEW TYPES

The seven interfaces creating the main framework for the Orca system do not provide a complete system by themselves. In fact none of them address the real issue at the heart of the Orca framework, the graphics renderings. Most renderings to date have been written to follow very simple applet style programming techniques to try and keep the framework accessible to as many developers as possible. Although Orca does not enforce any particular graphics doctrine and should be able to support more sophisticated rendering systems, it is important that at least some core implementations remain quite simple.

To integrate a rendering within the Orca framework an object should have access to an OrcaData object and the OrcaAppearance object that is associated with that object. Through proper use of the five or six methods that these two objects provide for data and appearance access, a graphics rendering can easily be integrated into the system.

Although object-oriented programming does provide a way to think about and organize graphics programming, there are costs associated with this sophistication. Many researchers cannot afford to devote the resources to learn advanced object-oriented techniques and the students who may actually be doing much of the coding may not have the experience to approach design issues properly. Our hope is that Orca can scale conceptually; that is, that it can be picked up quickly by a novice Java programmer or a graduate student with only a few applets under his or her belt, and still appeal to a seasoned, object-oriented programmer.

4. EXAMPLE APPLICATIONS

This section describes piecing together Orca pipe segments for two unusual data visualization applications. The first describes working with compositional data, where the data is constrained by a sum across all variables. It is common when studying populations or soils. The unique aspect of this example is inserting a dimension-reduction step into the data pipeline, reducing the viewing space to be the simplex containing the data. The second example describes an application containing multivariate space and time measurements. Here, multiple viewers are wired together facilitating exploration of the multivariate structure with regard to time and, to some extent, space.

4.1 COMPOSITIONAL DATA

When data, $X = (x_1 \, x_2 \ldots x_p)'$, is constrained by the relationship $x_1 + x_2 + \cdots + x_p = 1$ it poses special problems for graphics. Typically, a ternary diagram (alternatively a reference triangle, or barycentric coordinate space) is used, which works well when there are only three components in the composition. It has been vexing to naturally extend the ternary diagram to arbitrary dimensions. Various solutions and work-arounds are used: conditional ternary diagrams to explore subcompositions, or plots of the percentages as the ratio of the $x_i$ ($i = 2, \ldots p$) to $x_1$.

Compositional data effectively lie in a $(p - 1)$-D simplex in $p$-space. If we naively
examine the raw data space using a grand tour, we will see the data "collapse," when the viewing dimension contains the empty 1-D subspace. This is undesirable for examining the distribution of the data in the simplex. So our approach is to first project the data into the \((p - 1)\)-D space where the data exist, and build barycentric axes (the simplex shell) around the data. This is done using the SimplexPipe in Orca. The SimplexPipe projects the data into the \((p - 1)\)-D subspace, and adds barycentric axes to the data.

The data for the example come from a designed experiment, conducted by Bill Fagan, University of Washington, to evaluate the effect of increased omnivorous predators on arthropod community stability. The data are counts of bugs in one of five functional groups: apical herbivores, basal herbivores, chewing herbivores, generalist predators, and specialist predators. The five compositional location parameters describe the relative abundance of each group in each of six different experimental treatments. The six different treatments are:

1. increased omnivory-vegetation disturbance (OV)
2. increased specialist-vegetation disturbance (SV)
3. control predator-vegetation disturbance (CV)
4. increased omnivory-control vegetation (OC)
5. increased specialist-control vegetation (SC)
6. control predator-control vegetation (CC)

The data are samples from the posterior distributions of each treatment for the compositional location parameters. If omnivory has a stabilizing influence, then the composition for treatments 1 (OV), 4 (OC) and 6 (CC) should be similar. Also, treatments 2 (SV), 3 (CV) and 5 (SC) should be similar but different from 1, 4, and 6.

Two viewers are used—a scatterplot matrix and a tour (Figure 6). (Figure 5 contains the Orca pipe diagram for this example.) The tour view shows the projected data in a tour over the 4-D subspace in which they lie, effectively rotating the simplex in front of the viewer. It can be observed that there are differences in the location between groups, but also that the shape of each group (the variance-covariance) is similar, which is not a surprise because the variances and covariances were constrained to be the same in the simulation. We can also observe, especially if we examine the treatment means alone, that treatments 2 (SV), 3 (CV), and 5 (SC) are very similar, to the point of being almost indistinguishable. Treatment 1 (OV) is the most different, but treatments 4 (OC) and 6 (CC) are different from each other and 1, as well as from 2, 3, 5. These observations differ slightly from what was expected.

### 4.2 Multivariate Time Data

The Tropical Atmosphere Ocean (TAO) project consists of 70 moored buoys in the equatorial Pacific, measuring air temperature, relative humidity, surface winds, and sea surface temperatures. The TAO buoy data are collected to monitor the El Niño phenomenon, which is the effect of the Pacific Ocean oscillating in its basin on the weather patterns observed in North America. We use a subset of the TAO buoy data dating from March 7, 1980 to May 3, 1998, giving 178,080 observations. Ongoing
Figure 6. Scatterplot matrix and tour viewers for the insect population data. Barycentric axes around the point cloud represent the species proportion.

measurements are available from www.pmel.noaa.gov/toga-tao/home.html.

We reduce the dataset further to monthly averages of meridian winds, zonal winds and sea surface temperatures from buoys moored at $-2^\circ$ lat, and $(-110^\circ, -140^\circ)$ long. (Humidity has many missing values so we ignored it. Air temperature is so strongly correlated to sea surface temperature that it was dropped, too.)

The example has three viewers. Figure 7 illustrates the pipe structure of the application. Figure 8 displays a snapshot of the three viewers.

From the time-constrained tour we can see: (1) There is a variance difference between the two longitudes. (2) There are some projections where the two series are completely shifted from each other. That is, the measurements of the projections at the two buoys are distinctly different. (3) There are some global trends that are different for each variable. (4) The seasonal trends (peaks and valleys) mostly match, although at some projections there are strong lag dependencies (peaks and valleys slightly offset), which can differ from year to year. Looking at the variable axes allows the viewer to determine which combinations of variables are present when the lag relationships exist; the very last part of the series, corresponding to 1998, shows a different, or more extreme pattern than all
From the scatterplot matrix we can see the following: The two locations have location differences visible in the pairwise plots. The scatterplots overlap considerably, though; brushing on the outliers in the plot of Meridian Winds and SST, and linking the scatterplot to the time plot shows that these points all correspond to 1998.

From the multivariate tour we can see the following. The difference in location is even more pronounced in the three-dimensional space, so that there is almost a boundary between the two buoys’ measurements on the three variables. This was reflected occasionally in the time tour, when the two time curves are virtually distinct in some projections; the outliers noted in the pairwise plots are even more outlying in these plots. The plot shows that 1998 was a very strange year. The outlying values are extreme even in the multivariate space, so they are different from all previous years in this dataset (1985–1997).

5. CURRENT AND FUTURE DEVELOPMENTS

Three current applications that are actively being developed include clustering, mapping, and graph layout. In the clustering application, two issues are being explored: first, dynamic updating of the appearance of points, based on a running clustering algorithm; second, the display of uncertainty related to the classification of points into clusters. In the mapping application, the issues being explored are: displaying maps, constructing spatial dependence viewers, and projection pursuit methods for spatially constrained data.
Figure 8. Scatterplot matrix, multivariate tour, and time-constrained tour viewers for the TAO buoy data. The two different shades of grey represent buoys at two different longitudes. The scatterplot matrix view displays pairwise plots of the three measured variables, meridian and zonal winds, and sea surface temperature. The multivariate tour also uses these three variables. The time-constrained tour uses the horizontal axis to tour the three multivariate measurements. Linked brushing reveals that the outlying group of points correspond to a contiguous time period corresponding to early 1998.

The graph layout application is being used for examining statistical models for social network analysis, and telecommunications networks.

Structurally in Orca there are two compelling areas of work: building graphical objects that can draw themselves, and providing base classes to more easily perform interactive tasks. Graphical objects are at the core of any statistical graphics application, discussed by Wilks (1996) and Murrell (1998). In the language of multivariate data visualization, graphical objects can be considered to be coordinate-free renderings of the data. For interactive tasks, a small collection of mouse objects needs to be developed that will be commonly used throughout most of the graphics to interface querying and focusing.

Beyond the core of Orca, several tools and utilities are being developed. A visual programming interface to the current Orca pipes is available, which allows the user to lay out pipes and wire them together, using icons on the screen. Connections with R and Omegahat are currently possible, and are actively being developed further. Developing closer connections with numerical algorithms is of high priority, since many graphical methods require complex optimization procedures.

In conclusion, we hope that this work inspires others to think about data visualization, pick up and delve into the code, and implement new ideas.
ACKNOWLEDGMENTS

This work was initiated by funding from the National Research Center for Statistics and the Environment (NRCSE) and has been subsequently supported by NRCSE, the Center for AIDS Research, University of Washington, Seattle, WA, and a Spring Research Initiation Grant from the College of Liberal Arts and Sciences, Iowa State University. The authors thank Peter Guttorp, Deborah Swayne, Andreas Buja, Duncan Temple Lang, Werner Stuetzle, and Graham Wills for helpful discussions in relation to the work discussed in this article. In addition, we thank Dean Billheimer for providing us with the insect population data. This article was started while the last author was visiting BiometricsSA, Adelaide, Australia, and the Software Production Research Group, Lucent Technologies, Indian Hill, IL.

APPENDIX

The Web page for the project is www.pyrite.cfas.washington.edu/orca/.

The Web page contains a brief overview of the project, and a picture gallery of applications developed. There is a mailing list for the project, where inquiries can be made: orca-devel@pyrite.cfas.washington.edu. One can find subscription information on the Web site, and help with getting started coding, and implementing new features.

Access to the code—It is available through anonymous CVS from the Web site. The code is released under the Lesser General Public License (LGPL), which basically permits the use of Orca in closed-source and commercial projects, as long as Orca and any changes to Orca are made available. Please mail the authors for directions on how to download the code. The Web site runs a CVS to WWW gateway, for viewing the code.

[Received June 2000. Revised July 2000.]

REFERENCES


