ORCA: A Visualization Toolkit for High-Dimensional Data

Peter Sutherland Anthony Rossini Thomas Lumley
Nicholas Lewin-Koh Dianne Cook Zach Cox

NRCSE

Technical Report Series

NRCSE-TRS No. 046

May 18, 2000

The **NRCSE** was established in 1996 through a cooperative agreement with the United States Environmental Protection Agency which provides the Center's primary funding.

ORCA: A Visualization Toolkit for High-Dimensional Data

Peter Sutherland[*], Anthony Rossini[†], Thomas Lumley[‡], Nicholas Lewin-Koh[§], Dianne Cook[¶], Zach Cox[||]

Abstract

This paper describes the background and design of the software, Orca. Orca is a flexible and extensible toolkit for constructing interactive and dynamic linked data viewers. It is specifically designed for data having a multivariate component. A main goal of the Orca project is to make interactive and dynamic graphics programming accessible to researchers from diverse fields and backgrounds. The approach to displaying data comes from earlier research on building statistical graphics based on data pipelines. Different aspects of data processing and graphical rendering are organized conceptually into segments of a pipeline. The software design of Orca takes advantage of the object-oriented nature of Java(tm) to open up the data pipeline, which allows developers greater flexibility and control over their visualization applications. Importantly, new types of data views coded to adhere to a few simple Orca design requirements can easily be integrated with existing pipe sections. This allows access to sophisticated linking and dynamic interaction across all (new and existing) Orca view types. Orca pipe segments can be accessed from modern data analysis packages such as Omegahat or R, providing a tight-coupling of visual and numerical methods.

Keywords: Multivariate space-time data, dynamic graphics, interactive graphics, object-oriented software, java, motion graphics, brushing, multiple linked views, compositional data, data projections, plot matrices.

[*]Research begun while at NRCSE, University of Washington, Seattle, WA. Currently at NeoMorphic, Berkeley, CA

[†]Biostatistics, University of Washington, Seattle, WA

[‡]Biostatistics, University of Washington, Seattle, WA

[§]Statistics and EEB, Iowa State University, Ames, IA

[¶]Statistics, Iowa State University, Ames, IA

[||]Electrical Engineering, Iowa State University, Ames, IA

1 Introduction

Since the late 1960s research in statistical graphics has exploited the technological advances provided by the computer and electronics industry. Graphics terminals facilitated a major advance in using graphics to assist in data analysis. For the first time moving pictures could be displayed and a user was able to interact with a plot in real-time. Chang (1970) explored rotations in 5D, to detect 2-D structure, and Kruskal (1970) watched a multidimensional scaling algorithm converge to a stable configuration. The seminal piece of work, called PRIM-9 (Fisherkeller, Friedman & Tukey 1974) was the first general purpose interactive statistical graphics 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. PRIM-9 was implemented on an IDIIOM 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! The earliest graphics terminals were vector-based displays.

In the mid-80s (McDonald 1982) developed Orion II, a set of statistical graphics methods using an object-oriented programming language on a single-user workstation with a raster display. (Actually this workstation was the kernel that developed into Sun workstations). By the late 80s, single user UNIX workstations running the X Window System prevailed in the academic community. The software XGobi (Swayne, Cook & Buja 1998), programmed in C, has its home in this environment, as does XmdvTool (Ward 1994). In a similar time frame, silicon graphics workstations became a standard for high-end graphics applications. ExplorN (Carr, Wegman & Luo 1996) was developed in this environment.

Diverging here with a side comment, along with the technological advances, came a dilemma for researchers working in these areas: how to adequately describe their work in 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 (`www.bell-labs.com/topic/-societies/asagraphics/`).

Now by the late 90s (and into the new millennium) raster graphics displays have prevailed, window interfaces are the most prevalent manner for users to interact with the computer, and computer languages have evolved considerably. Among the new innovations is the Java programming language, an object-oriented, cross-platform, computing environment from the folds of Sun MicroSystems. Java unifies the hardware world facilitating a "develop here, use everywhere" approach. The Java language offers bold new technology for use with statistical computing and graphics.

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. This paper describes the development and structure of a new visual toolkit for statistical data visualization which exploits the software innovations made available by 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 to be information that exists in some schematic form such as a table or list. Data is often but not always quantitative, and some translation of unstructured information is often required to derive the data. Data always includes some attributes or variables, such as the number of atoms in a molecule, or the value of the Australian dollar relative to the US dollar, or the weight of a crab.

Data visualization differs from information visualization, scientific visualization and cartographic visualization. Information visualization is broader and really encapsulates 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 phenomenon, 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 arises 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 axis, with values of that variable mapped along the axis, and that each axis represents a dimension, then the dilemma for data visualization is how to picture more than 2 or 3 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 to drawing plots of data is called the "multiple views" paradigm (Buja, Cook & Swayne 1996). 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 refocusing, linking between plots, easy rearrangement of plots, and automated sequences of views. We describe the approach in depth in this section.

At the root of graphical methods in statistics is a division of data visualization into three areas:

- **Rendering**, or what to show in a plot;
- **Manipulation**, or what to do within plots;
- **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 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 & Carr (1993) give 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, first described by Buja, Asimov, Hurley & McDonald (1988). It is assumed here that the data comes 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-dimensional data to a d-dimensional rendering, where $d < p$. (Most commonly $d = 2$.) Figure 1 illustrates the data pipeline for multivariate data. The steps which we will 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).

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; transforming to fit within a limited range such as $[0, 1]$; and ordering time variables. It can also be the place where row information, such as case labels, links to external annotation sources, appearance of points, and column information, such as variable names are are formed.

The second step (between world and planar) primarily involves dimension reduction. This step may also include methods for reducing the number of cases, by binning for example, in the situation of extremely large numbers of variables. The 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 & Hurley 1995, Buja, Cook, Asimov & Hurley 1997), can be applied to address the dimension reduction problem, also. Although the number of variables may remain the same, the tour algorithm provides a continuous sequence of low-dimensional projections to produce a "movie" which shows the data "from all sides". A tour can also be thought of as display of multiple views across time rather than space.

The second step, then, can be considered under the scope of building a graphical object: dimension reduction, and construction of components of a specific rendering of the data, including axes. It could also be the site of the definition of appropriate manipulation modes for the type of rendering, for example, a mechanism for changing bin widths. This essentially scales the data into the viewing window. The screen coordinates is also the location for translating mouse actions on a window into manipulation actions on the data.

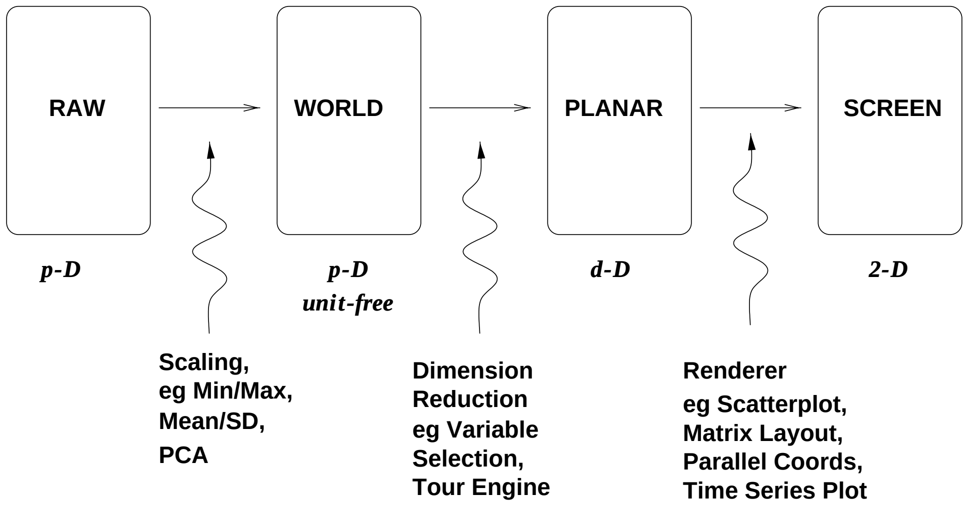
RAW
WORLD
PLANAR
SCREEN
p-D
p-D
unit-free
d-D
2-D
Scaling, eg Min/Max, Mean/SD, PCA
Dimension Reduction eg Variable Selection, Tour Engine
Renderer eg Scatterplot, Matrix Layout, Parallel Coords, Time Series Plot

Figure 1: Data pipeline representing multivariate data.

When the data comes with a much broader mixture of variables, such as time, space, or sampling weight, more complex methods are required. 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 to define point appearance in a later plot.

The second step needs to preserve the structural variables rather than reducing them to low-dimensional projections. The structure of the data may also affect the type of dimension reduction to be considered. In a multivariate regression setting, it may be more appropriate to construct separate low-dimensional summaries of the predictor variables and response variables, rather than mixing them together. This is analogous to the distinction between principal component analysis and canonical correlation in formal multivariate analysis. We see that the second step can be considered a re-structuring stage as well as a standardizing to coordinate-free stage. 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 third step involves the construction of views of the data. There are tradeoffs between accurate representation of high-dimensional data distributions and representations of the sampling structure. For example, a two-dimensional time series view that preserves the time structure leaves only one dimension to show projections of the data. It is useful to augment this view with a second view that suppresses the time axis and shows two-dimensional projections. These tradeoffs are even more extreme for data measured over space, or both space and time, where effective display of even one dimension of the data is challenging.

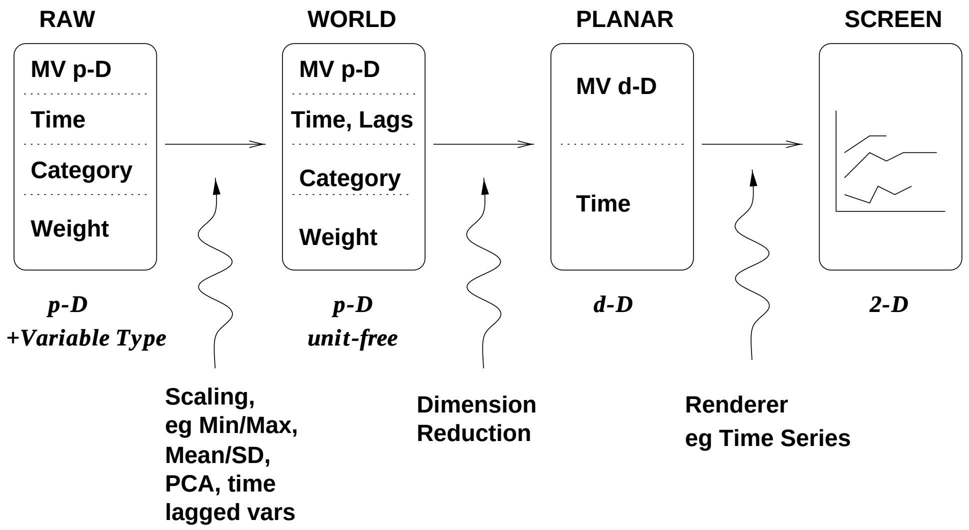

Figure 2: Data pipeline representing multivariate data, in combination with other variable types such as time and space.

In general, the multiple views approach specifies that the data pipeline is rather more like a river delta, piping the data out into multiple renderings. The difficulty with this approach is to define appropriate mechanisms to link information from one plot to another. In the simplest case, where one view has a scatterplot of variable 1 vs variable 2, and another view has a scatterplot of variable 3 vs 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. Figure 4 illustrates linking of information in association with the data pipeline.

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 powerful 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 & 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,

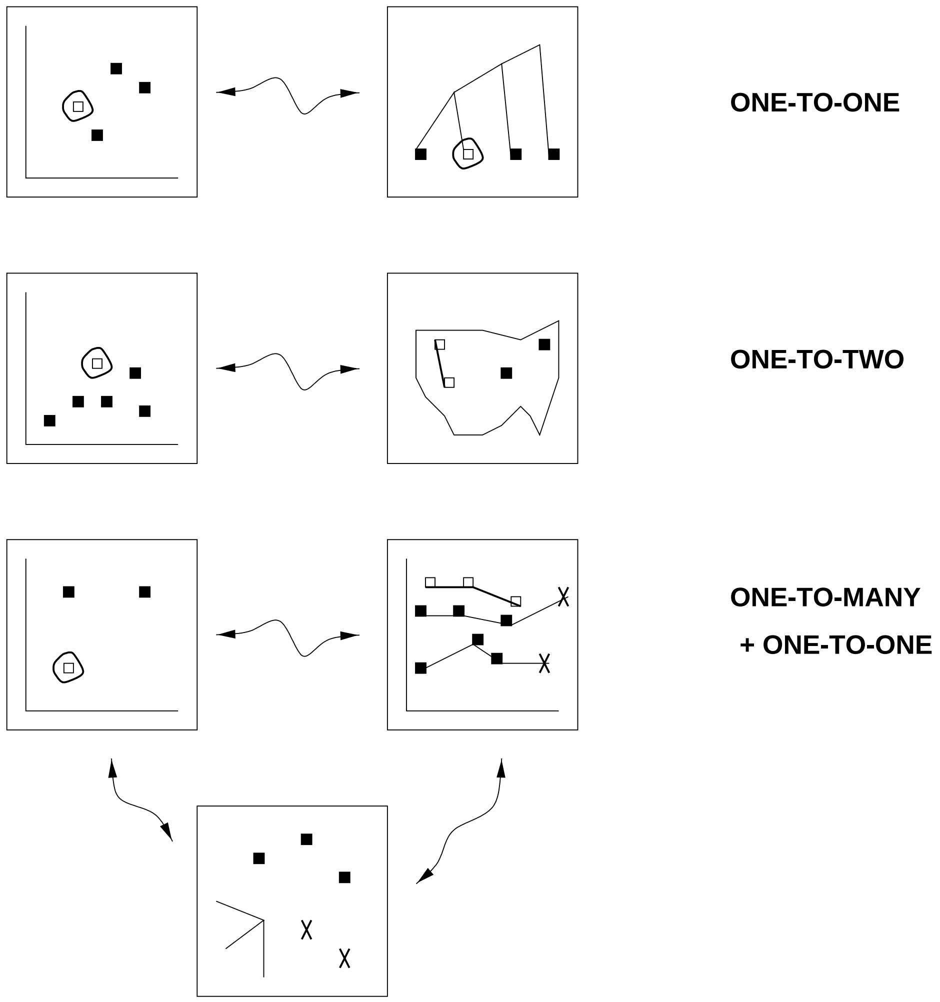

Figure 3: Common types of linking.

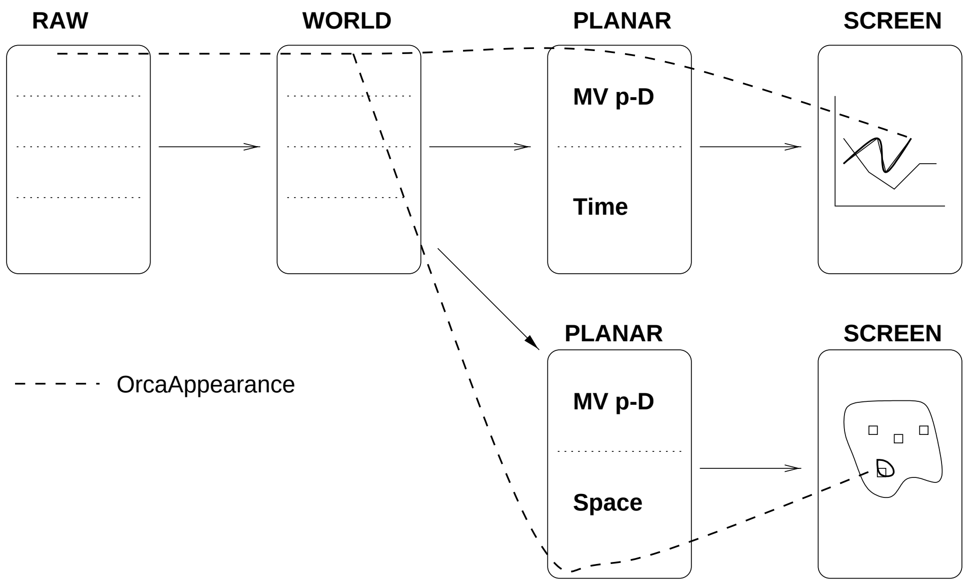

Figure 4: Linking is associated with the data pipeline, only in that a wire needs to be laid connecting the relevant data attribute to the final screen rendering.

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 & 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 re-structuring: 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 pre-projecting 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 data set, 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

One of the main design principles behind Orca is to provide researchers with a graphics framework that will allow easy development and integration of new types of graphics with existing Orca implementations and other statistical languages. Integration with the framework allows access to sophisticated linking and dynamic interaction across all Orca view types. Basic features for file parsing and standard data interaction are built into the system, so the developer need only concentrate on the specific aspects of rendering and implementing the graphics that interest them. The amount which they adhere to few simple principles will reflect the level of system integration that their graphics exhibit.

Orca is an API 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, `lib.stat.cmu.edu/R/CRAN/`).

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 independent of what pipes are connected to it, and communicates to the adjacent pipes using a limited set of functions described by the OrcaPipe interface. This allows flexibility in the design of new pipes 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.

A pipe segment links 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. Two different windows could provide one- and two-dimensional views of the same tour if their pipelines branched downstream of the Tour pipe, or have independent tours but common linking if they branched upstream of the Tour Pipe.

Java *interfaces* are used throughout the API design to create the graphics pipeline metaphor that underlies the main structure of the software. Java interfaces provide a flexible way to specify just enough of the behavior of an object to allow other objects to communicate with it. The OrcaPipe interface, for example, specifies that the object can accept new brushing information from pipes downstream of it and suitably update its output. This may be done by calculations in the object itself or merely by passing the new information back upstream. Since the upstream segment itself implements the OrcaPipe interface it guarantees that the upstream pipe will do something sensible with this information.

The different Orca object types have been designed with the goal of being simple to grasp, and to provide relevant conceptual landmarks. These design features should allow the majority of the developer's effort to be focused on more the more important issues of developing interesting views of data pertinent to their field of interest.

To date, resources for this project have focused on refining the API design, improving data structures, and implementing the basic pipe segment functionality. This has provided a foundation for experimentation and evaluation of new methodology through the addition of new pipes to the API.

3.2 Java Interfaces and Object Oriented Design

The Java programming language provides a powerful way to approach multiple inheritance (extending roles and responsibilities of existing classes) while avoiding some of its complications. 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 do implement interface methods can then be referred to via the interface type. This simple contractual agreement allows for far greater flexibility and looser coupling between objects than allowed for by traditional object inheritance as found in C++ and Java inheritance classes.

In object oriented programming, polymorphism provides the ability to refer to an object by any of the classes that it inherits from or any of the interfaces that it implements. Through careful use of inheritance and interfaces, very flexible systems of objects can be built. This flexibility results from only making an object's references to other objects at the most abstract level needed only.

Objects that are only interested in the functionality of other objects specified through an interface will only need to refer to that object as that particular interface type (see the section on Observables for an example). Referencing objects at this level allows the main object to interact with any number of different classes that implement the same interface, without needing to know their specific class.

Inheritance is often not the right way to get at the strengths that polymorphism provides. Extending an object through inheritance means that one must then also override the methods it provides to achieve any distinct functionality. There is then not really any savings in code reuse and one of the basic assumptions of inheritance has been disregarded.

Interfaces approach polymorphism with the idea that the contract methods provided will allow objects to implement similar but unique functionality, which is really what is desired for creating multiple objects. 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 interface that guarantees the desired properties, and do not worry about the implementation.

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.

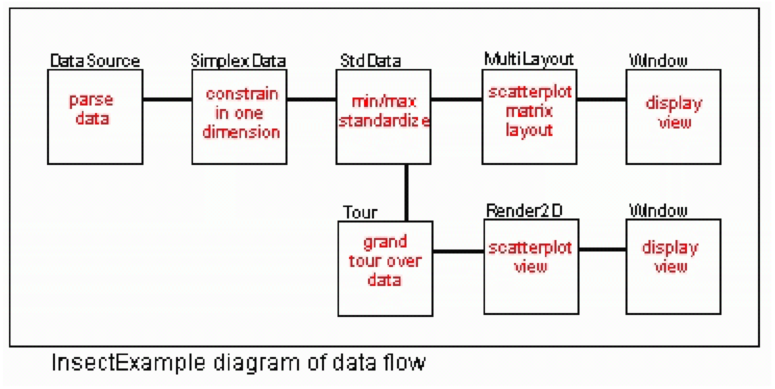

Figure 5: Pipe segment diagram for the insect example, illustrating how two views of one data source are created.

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 data set.

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. Additional common methods allow for event propagation through the pipeline are also implemented in is abstract class.

Individual sections of pipe 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 handle the data that is untouched, methods should simply pass the object through to the next segment of pipe.

The segments of the pipeline communicate with four data channels:

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.

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 pipelines 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 preexisting code to handle or ignore them.

Information that is associated with the appearance object can in a way 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.

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 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.

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 maybe reused. Examples of a 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 methods 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 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. With this type of linking, views are considered peer objects with one another. That is, 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. Each view need not know how many other views are active on the same data set, they only need 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 can actively contact each other.

The need for this type of callback is common in all types of programming. The basic idea involves registering an interest in the state of another object and being notified when state changes are made, rather than continually polling an object to look for changes. 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 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 common 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 flexible 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 data set. 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 total view of the system. In fact none of them address the real issue at the heart of the Orca framework, the graphics renderings. While object oriented programming does provide an exciting new 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.

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.

4 Example Applications

This section describes piecing together Orca pipe segments for two fairly unique 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 in a dimension reduction step into the data pipeline, to reduce the viewing space to to be the space of the simplex containing the data. The second example describes an application containing multivariate space and time measurements. Here, multiple viewers are wired together facilitating exploring the multivariate structure with regard to time, and to some extent space.

4.1 Compositional Data

When data, $\boldsymbol{x} = (x_1\ x_2\ \ldots x_p)'$, is constrained by the relationship $x_1+x_2+\ldots+x_p = 1$ 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 3 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 sub-compositions, or plots of the percentages as the ratio of the x_i $(i = 2, \ldots\ p)$ to x_1.

Compositional data effectively lies 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 exists, 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 Dr 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 5 functional groups: apical herbivores, basal herbivores, chewing herbivores, generalist predators, and specialist predators. The 5 compositional location parameters describe the relative abundance of each group in each of 6 different experimental treatments. The 6 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 (SV)
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 (SV) 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 5-dimensional 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 surprise because the variances and covariances were constrained to be the same in the simulation.

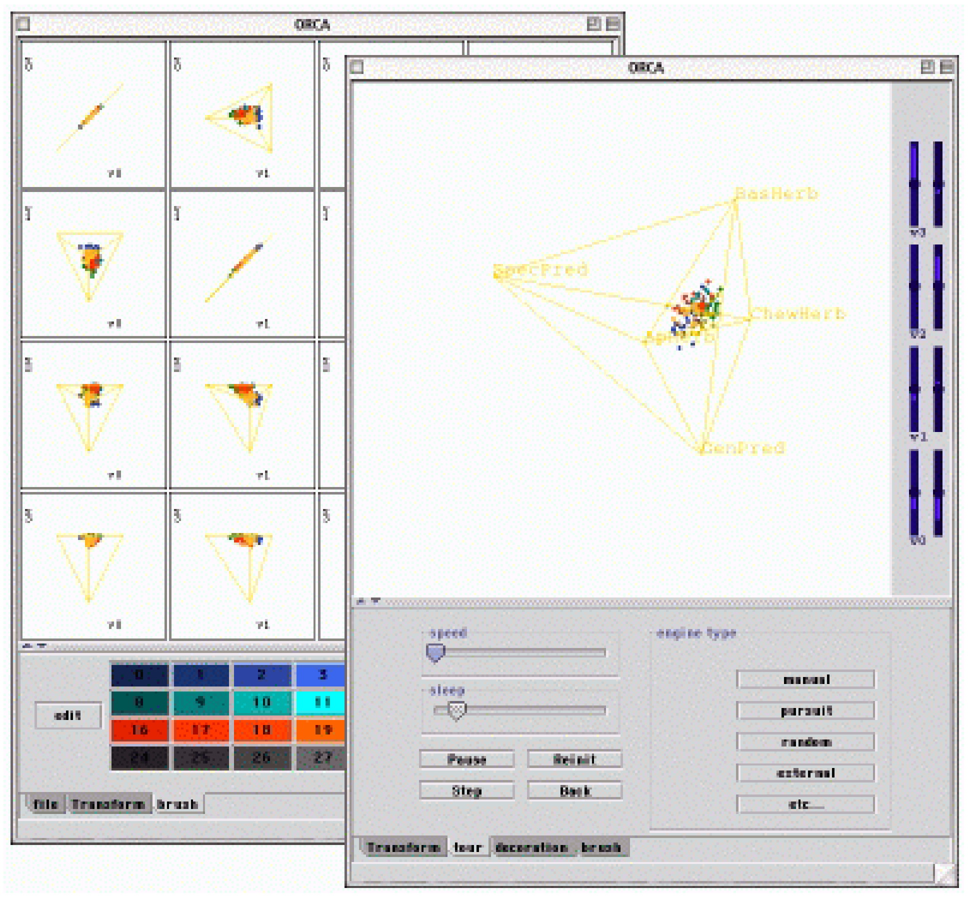

Figure 6: Scatterplot matrix and tour viewers for the insects population data.

4.2 Multivariate Time Data

The Tropical Atmosphere Ocean (TAO) project consists of 70 moored buoys in equatorial Pacific, measuring air temperature, relative humidity, surface winds, sea surface temperatures. The TAO buoy data is 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 7 March 80 to 3 May 98, giving 178080 observations. Ongoing measurements are available from:

`http://www.pmel.noaa.gov/toga-tao/home.html`.

We reduce the data set further to monthly averages of meridian winds, zonal winds and sea surface temperatures from buoys moored at -2^o lat, and $(-110^o, -140^o)$ 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.

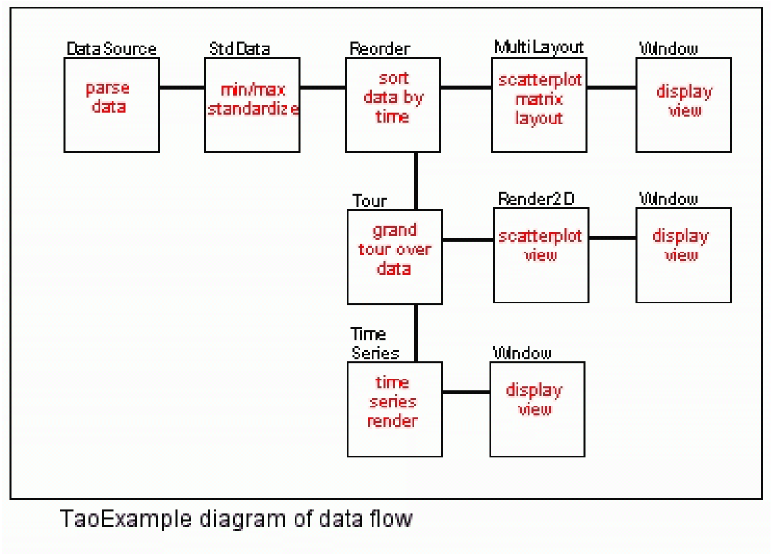

Figure 7: Pipe diagram for the TAO example.

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 which 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. If you look at the variable axes you can work out 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 other years.

From the scatterplot matrix we can see: 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 scatter plot to the time plot shows that these points all correspond to 1998.

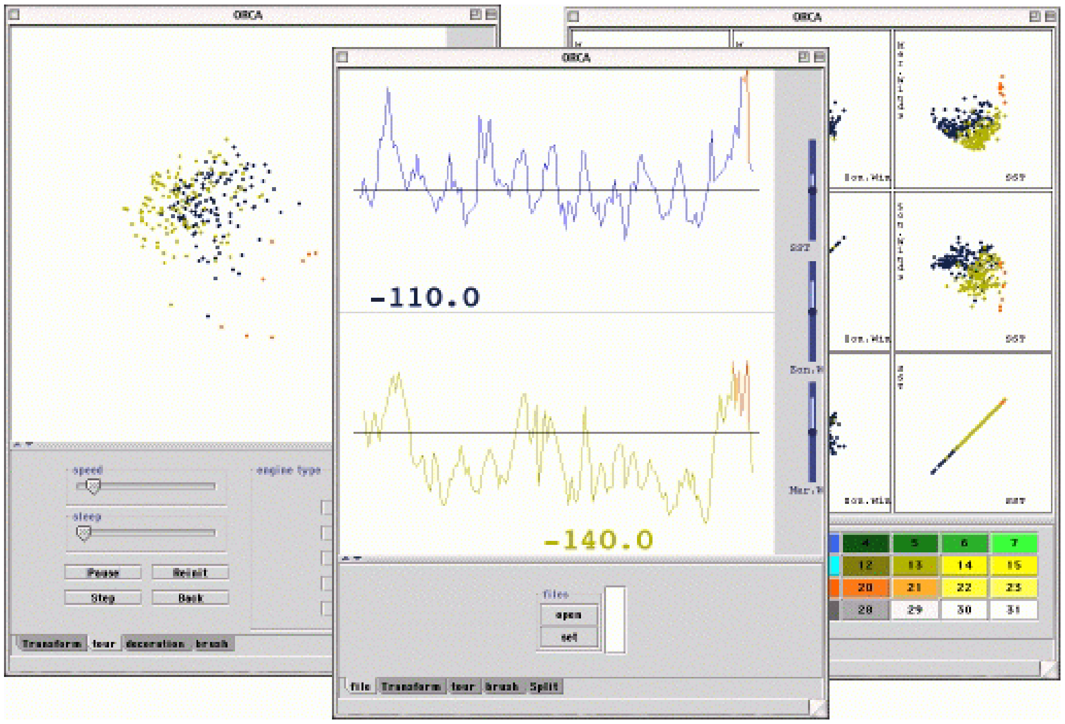

Figure 8: Scatterplot matrix, multivariate tour, and time-constrained tour viewers for the TAO buoy data.

From the multivariate tour we can see: The difference in location is even more pronounced in the 3 dimensional space, so that there is almost a boundary between the two buoys measurements on the 3 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 data set (1985-1997).

5 Current and Future Developments

Three current applications being developed are clustering, mapping and graph layout. In the clustering application, two issues are being explored: dynamic updating of the appearance of points, based on a running clustering algorithm and secondly, 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. The graph layout application is being used for examining statistical models for social network analysis.

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 in Wilks (1995) 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 need 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 finishing, 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 wish to thank Peter Guttorp, Deborah Swayne, Andreas Buja, Duncan Temple-Lang, Julie Dickerson and Graham Wills for helpful discussions in relation to the work discussed in this paper. In addition, we would like to thank Dean Billheimer for providing us with the insect population data. This paper was started while the fifth 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
`http://pyrite.cfas.washington.edu/orca/`.
It 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

The code is available through anonymous CVS from the web site. The code is released under the Lesser Gnu 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.

References

Asimov, D. (1985), 'The Grand Tour: A Tool for Viewing Multidimensional Data', *SIAM Journal of Scientific and Statistical Computing* **6**(1), 128–143.

Becker, R., Cleveland, W. S. & Shyu, M.-J. (1996), 'The Visual Design and Control of Trellis Displays', *Journal of Computational and Graphical Statistics* **6**(1), 123–155.

Buja, A., Asimov, D., Hurley, C. & McDonald, J. A. (1988), Elements of a Viewing Pipeline for Data Analysis, *in* W. S. Cleveland & M. E. McGill, eds, 'Dynamic Graphics for Statistics', Wadsworth, Monterey, CA, pp. 277–308.

Buja, A., Cook, D., Asimov, D. & Hurley, C. (1997), Dynamic Projections in High-Dimensional Visualization: Theory and Computational Methods, Technical report, AT&T Labs, Florham Park, NJ.

Buja, A., Cook, D. & Swayne, D. (1996), 'Interactive High-Dimensional Data Visualization', *Journal of Computational and Graphical Statistics* **5**(1), 78–99. See also `www.research.att.com/~andreas/xgobi/heidel/`.

Carr, D. B., Wegman, E. J. & Luo, Q. (1996), ExplorN: Design Considerations Past and Present, Technical Report 129, Center for Computational Statistics, George Mason University.

Chang, J. (1970), 'Real-time Rotation', ASA Statistical Graphics Video Lending Library (`http://www.bell-labs.com/topic/societies/asagraphics/`).

Cook, D., Buja, A., Cabrera, J. & Hurley, C. (1995), 'Grand Tour and Projection Pursuit', *Journal of Computational and Graphical Statistics* **4**(3), 155–172.

Eick, S. G. (1994), 'Graphically Displaying Text', *Journal of Computational and Graphical Statistics* **3**(2), 127–142.

Fisherkeller, M., Friedman, J. H. & Tukey, J. (1974), PRIM-9: An Interactive Multidimensional Data Display and Analysis System, Technical Report SLAC-PUB-1408, Stanford Linear Accelerator Center, Stanford, CA.

Kruskal, J. B. (1970), ‘Multidimensional Scaling’, ASA Statistical Graphics Video Lending Library (`http://www.bell-labs.com/topic/societies/asagraphics/`).

Mardia, K. V., Kent, J. T. & Bibby, J. M. (1979), *Multivariate Analysis*, Academic Press, London.

McDonald, J. A. (1982), Interactive Graphics for Data Analysis, Technical Report Orion II, Statistics Department, Stanford University.

Murrell, P. (1998), Investigations in Graphical Statistics, PhD thesis, University of Auckland, Auckland, New Zealand.

Swayne, D. F., Cook, D. & Buja, A. (1998), ‘XGobi: Interactive Dynamic Graphics in the X Window System’, *Journal of Computational and Graphical Statistics* **7**(1), 113–130.

Ward, M. (1994), XmdvTool: Integrating Multiple Methods for Visualizing Multivariate Data, *in* ‘Proccedings of Visualization ’94’, IEEE Computer Society Press, Los Alamitos, CA, pp. 326–333.

Wegman, E. J. & Carr, D. B. (1993), Statistical Graphics and Visualization, *in* C. R. Rao, ed., ‘Handbook of Statistics, Vol. 9’, Elsevier Science Publishers, Amsterdam, pp. 857–958.

Wilks, D. S. (1995), *Statistical Methods in the Atmospheric Sciences*, Academic Press, San Diego, CA.