Building Highly-Coordinated Visualizations in Improvise

Chris Weaver*

Computer Science Department
University of Wisconsin–Madison

ABSTRACT

Improvise is a fully-implemented system in which users build and browse multiview visualizations interactively using a simple shared-object coordination mechanism coupled with a flexible, expression-based visual abstraction language. By coupling visual abstraction with coordination, users gain precise control over how navigation and selection in a visualization affects the appearance of data in individual views. As a result, it is practical to build visualizations with more views and richer coordination in Improvise than in other visualization systems. Building and browsing activities are integrated in a single, live user interface that lets users alter visualizations quickly and incrementally during data exploration.

CR Categories: D.2.2 [Software Engineering]: Design Tools and Techniques—User Interfaces; H.2.3 [Information Systems]: Database Management—Languages; H.5.2 [Information Systems]: Information Interfaces and Presentation—User Interfaces

Keywords: coordinated queries, coordination, exploratory visualization, multiple views, visual abstraction language

1 INTRODUCTION

Visualization systems have matured into full-featured development environments that enable users to build visualizations with multiple coordinated views rapidly. In these systems, users coordinate views either by using a small set of pre-defined coordinations or by writing scripts. The choice between these two approaches is a tradeoff between simplicity and flexibility.

Improvise is an environment for building and browsing visualizations of relational data. Like other visualization systems, Improvise enables users to load data, create views, specify visual abstractions, and establish coordinations interactively. Unlike other systems, Improvise provides a rich visual abstraction language that can be coupled with a shared-object coordination mechanism, thereby increasing the expressive power of both.

The primary goal of Improvise is to enhance data exploration by offering users fine-grain control over the appearance of visualized data while preserving their ability to work quickly and easily. Improvise combines a simple, direct coordination mechanism called *live properties* with a more powerful, indirect coordination mechanism called *coordinated queries*. The combination is a significant improvement over existing coordination approaches because it enables users to define complex interactive dependencies between the appearance and behavior of views.

Live properties coordinates views using a symmetric update and notification mechanism to link views and other controls through shared objects. Controls interpret the shared objects as basic formatting information (colors, fonts, etc.), limits to abstract spatial extent (ranges, angles, etc.), data, and data querying operations (projections, filters, etc.) Visual abstractions are created by projecting and filtering data using expressions that can themselves be defined in terms of shared objects. By editing the expressions used by views to project and filter data, users are able to customize the visual and spatial abstractions that specify what data to draw, how to draw it, and where to draw it in a visualization.

Improvise is written in Java. Visualizations are saved to and loaded from disk as serialized XML documents. Improvise and the documents it produces are platform-independent.

This paper proceeds as follows. Section 2 reviews coordination approaches in several visualization systems similar to Improvise. Section 3 describes live properties and coordinated queries. Section 4 describes construction of visual abstractions and coordinations in two Improvise visualizations. Section 5 concludes.

2 RELATED SYSTEMS

Improvise is similar to DEVise [12], Snap-Together Visualization [15], and other visualization systems based on the relational data model. These systems follow a recent trend toward interactive construction approaches that employ simple combinations of well-known coordination and view types. Although building small visualizations in these systems is very easy, the number of views and the degree of coordination that can be practically incorporated during data exploration is limited.

LinkWinds [10] uses non-view controls to append clauses to queries for filtering purposes. The ability to reference shared objects in coordinated queries has a similar flavor. Whereas LinkWinds has a "track" mode in which mouse drags continuously update coordinated views, views in Improvise are always tightly coupled over all coordinations, including all coordinated queries.

Given an input table, IVEE [2] automatically selects appropriate controls (such as range sliders or checkboxes) for each data attribute. Users can create one or more views (scatterplots, geographic maps, or cluster views) and specify projections that map attributes into view parameters. The conjunction of slider selections is used to filter the contents of all views (as well as the contents of the sliders themselves). In Improvise, views can be filtered independently using filter expressions that depend on navigation or selection in any combination of sliders or views.

Tioga-2 [23] uses a data flow model to support advanced navigation features such as tunneling (wormhole-like hyperlinks), view cloning, and nested views. DataSplash [17] adds end-user visualization construction in the form of tuple painting and a zoom layer manager for editing how tuples appear at different levels of magnification. VIQING [16] is an extension of DataSplash that allows users to express queries by conjoining views into the visual equivalent of projections, selections, and joins. The elements of Improvise visualizations are declarative rather than procedural; users can generate nested views and semantic zoom, but approaches like tunneling and layer management have to be built into views as fixed features that allow little or no customization by the user.

Polaris [21] automatically generates multiscale visualizations and the queries needed to draw them using a formal specification language. Zooming in Polaris is conceptually equivalent to traversing an edge of a *zoom graph* [20] in which each node corresponds

*e-mail: weaver@cs.wisc.edu

IEEE Symposium on Information Visualization 2004
October 10-12, Austin, Texas, USA
0-7803-8779-1/04/$20.00 ©2004 IEEE

to a particular visual representation in a data cube. Nodes are drawn in a graphical notation that describes the visual query at that point in the graph. Because Improvise users build coordinated queries using custom-defined expressions, complex or unusual visual representations are possible, but simple or common ones cannot be manipulated as quickly or as easily as in Polaris.

In the coordination model prototyped in CViews [5], explicit *coordination objects* in a *coordination space* manage visual parameters and access data using a dataflow model to define a particular type of coordination, such as brushing. Views connect to coordination objects through translation functions. The equivalent space in Improvise consists of coordinated query graphs that connect views through navigational parameters, selections, data, and expressions. In this space, coordinations exist implicitly as recognizable patterns of interactive dependence between views, rather than as explicit objects. Translation (spatial transformation, rendering, and so on) is an inherent function of views and is not user-customizable.

Snap-Together Visualization uses a relational data model that coordinates views using *primary key actions*. When two views are coordinated, invoking an action in one view causes the other view to perform its corresponding action. Actions are extensible and include loading (of a relation), selection (of tuples), and scrolling (over a list of tuples). Coordinated queries create similar interactive dependencies between views, but allow fine-grain user customization of dependencies between visual encodings as well as data.

DEVise uses a relational data model to coordinate multiple views of large datasets. Users can create, destroy, coordinate, and specify the contents of views interactively. Its only view—the scatterplot—and few coordination types—*cursor*, *visual link*, *record link*, and *set link*—are quite powerful. However, reproducing common visualization constructions in DEVise frequently involves convoluted chains of linked scatterplots (many of which are undesirable artifacts that must be intentionally hidden offscreen). Coordination graphs of DEVise visualizations reveal that all four coordination types can be reproduced by treating the X and Y ranges of scatterplots as shared objects or as dynamic parameters in simple query expressions. This discovery motivated the design of live properties and coordinated queries in Improvise.

3 IMPROVISE

This section presents the two major parts of the architecture and implementation of user-definable coordination in Improvise. The first, live properties, is a direct coordination mechanism that uses a simple shared object model to create interactive dependencies between views. The second, coordinated queries, is an indirect coordination mechanism in which the data and visual encodings that determine each view's appearance are calculated from user-definable expressions that can depend on interaction in other views.

3.1 Live Properties

Live properties is a user interface architecture for directly coordinating *controls*—including views, sliders, and other widgets—through shared objects called *variables*. Each control defines one or more *live properties*, each of which can bind to at most one variable. Live properties may be either active (access and modify variables) or passive (access only). Changes to variables are propagated to controls via their live properties, as shown in figure 1.

Live properties serve two purposes. First, they are value slots that a control uses to determine its appearance and behavior. For instance, a scatterplot has two range properties that specify which region of the cartesian plane to show, and a color property that specifies the color used to fill its background. Second, live properties act as ports through which controls communicate with each other as a result of interaction. Variables and live properties are strongly typed, and binding is type-matched. Each live property also has a default value which is used by the control when the live property is not bound to any variable.

Live properties may be thought of as an instance of the Model-View-Controller architecture [11] with many small models. Similarly, the Abstraction-Link-View paradigm (ALV) [8] employs an encapsulated communications mechanism between views and data to link views shared by one or more users. However, live properties is not a constraint model (as in ThingLab [4]). By implementing controls so as to change their live properties only in response to user interaction, potential cycles and deadlocks are avoided.

Athena MUSE [9] defines integer values in bounded ranges as global parameters on views ("multidimensional information") and uses bidirectional equality constraints to link parameters to view attributes through reversible linear functions ("declarative constraints"). Variables are like these parameters, but are neither bounded nor limited to integers. Bindings between variables and live properties are like declarative constraints limited to the identity function.

3.2 Coordinated Queries

Coordinated queries is a visual abstraction language based on the relational database model. An *expression* is a tree of operators that calculates the value of an output field using the fields of a input record. Expressions make up query operations that views use to visually encode data records into graphical attributes (as in [13]):

- *Filters* use a single expression to calculate a boolean value for each input record. Views draw records for which this value is true.
- *Projections* use one or more expressions to calculate successive fields of an output record for each input record. Views draw records using graphical information (such as position and color) contained in the fields of the output record. This information is often encapsulated in view-specific glyphs.

Projections and filters are constructed in a tree-based expression editor (figure 2). The user builds each expression top-down, by choosing an operator for each position in the tree. Subexpressions are automatically appended whenever the chosen operator takes arguments. Editing this way takes a little getting used to, but has the advantage of being syntactically constrained. Editing is live; the visualization reflects changes immediately.

Expressions are composed of eight different kinds of operators. *Function operators* perform a variety of duties including object construction, type casting, member access, arithmetic, and statistics. *Value operators* evaluate to a user-edited value of a particular type. *Attribute operators* evaluate to the value of an input record field. In addition to these three basic kinds of operators, *aggregate operators* allow the calculation of simple aggregates on tables, *constant operators* provide easy access to frequently used fixed values

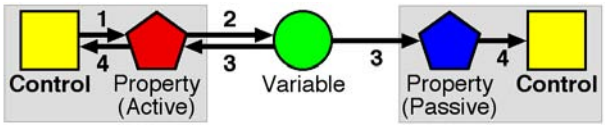

Figure 1: Direct coordination. (1) A control modifies the value of one of its (active) live properties in response to interaction. (2) The live property assigns the new value to its bound variable. (3) The variable sends a change notification to all live properties bound to it. (4) The live properties notify their respective parent controls of the change. The controls update themselves appropriately.

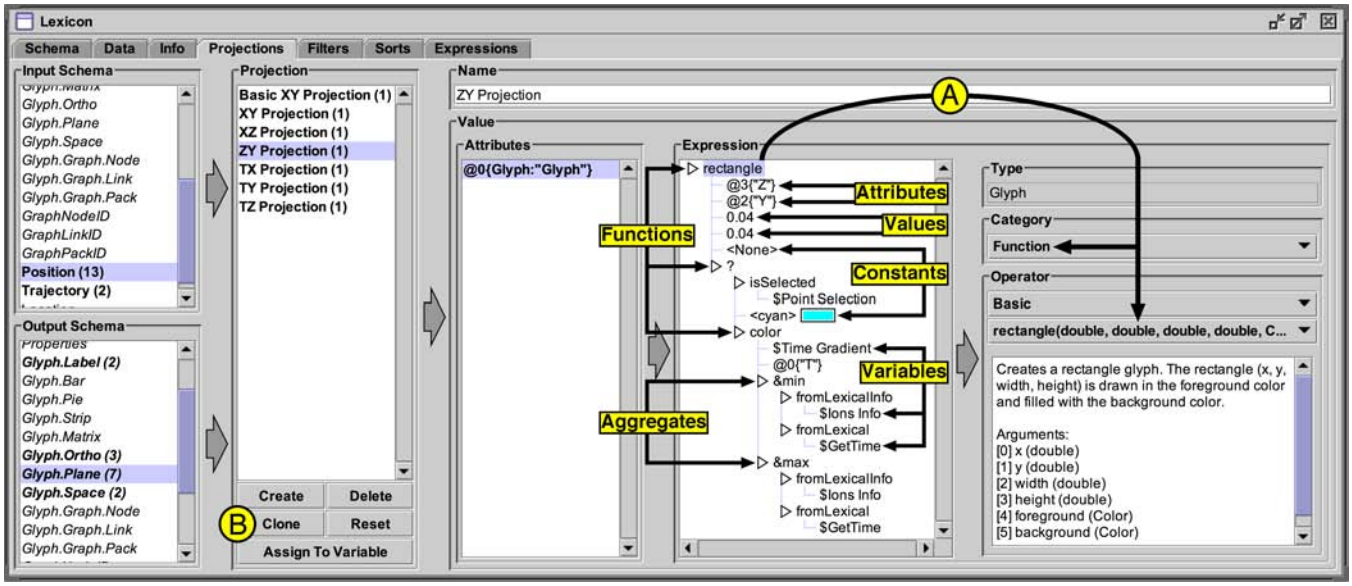

Figure 2: The lexicon editor, showing a projection that generates rectangle scatterplot glyphs. Users can select individual operators in an expression for modification (A), or copy-and-paste/drag-and-drop entire subexpressions. Cloning whole objects (B) allows users to experiment with variations of expressions quickly and reversibly. Large libraries can be built up for reuse or rapid switching during data exploration.

(e.g. pi), and *conversion operators* perform common numeric conversions between units (e.g. feet to meters). *Index operators* provide indexed data lookups, by mapping a primary key value of an attribute to a foreign key value of an attribute in a different dataset.

Indirect coordinations are created using *variable operators*. Whenever an expression is evaluated, variable operators take on the current value of their corresponding variable. When a control depends directly on a variable that contains a projection or filter, it also depends indirectly on any variables referenced by the expressions of the projection or filter. Through variable operators, expressions can depend not only on the navigation and selection parameters of a visualization, but also on its projections, filters, and datasets. This multi-stage dependence is used for aggregation, grouping, indexing, nested views, and other kinds of queries. Figure 3 shows how interaction propagates through one level of dependence in an indirect coordination. A shortcoming of the current implementation is that it does not detect or handle cycles in coordinated query graphs.

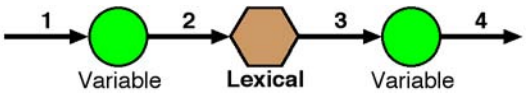

Figure 3: Indirect coordination. (1) An upstream object propagates a value change to a variable. (2) The variable notifies all lexical values that contain expressions which reference the variable. (3) Each expression notifies variables to which it is assigned as a value. (4) The variable sends a change notification to all downstream objects. Upstream and downstream objects can be live properties (as in figure 1), or other lexical values.

Each Improvise visualization stores schemas, data, and query operations in a central respository called the *lexicon*. To access these objects, views define live properties that bind to variables of the desired lexical type. The user assigns the value of a lexical variable by choosing from available objects of the same type in the lexicon. To specify the visual abstraction of a view, the user binds lexical variables to the view's data, projection, and filter properties.

4 Building Coordinated Queries

The combination of live properties and coordinated queries enables Improvise users to build highly-coordinated visualizations with complex visual encodings. This section describes how a variety of well-known coordination types are reproduced in the construction of two typical Improvise visualizations, shown in figures 4 and 11. Proceeding from navigation coordinations through selection coordinations to semantic zooming, each coordination is introduced in terms of related research. Figures 5-10 and 12-16 capture the corresponding coordinated query graphs used in Improvise to help users visualize coordination structure as they work.

Sliders and other controls are often useful for manipulating individual parameters of a visualization. In Dynamic Queries [1], non-spatial data attributes can be manipulated using range sliders. LinkWinds provides controls that can be coordinated with views for dynamic filtering. Users browse Improvise visualizations by interacting with views and non-data controls such as checkboxes, textfields, and sliders. Improvise axis controls are independent of scatterplots, but perform the usual roles of marking, labeling, and handling interaction in one dimension. Figure 5 shows how horizontal and vertical axes can be coordinated with a scatterplot.

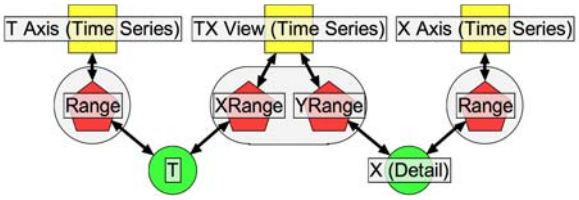

Figure 5: Coordination graph for a scatterplot with axis controls (see figure 4A). Panning or zooming in the T (or X) axis changes the value of the T (or X) range variable, which causes the plot to translate or stretch horizontally (or vertically). Manipulating the plot changes both variables, causing both axes to update appropriately.

Views can also be coordinated with each other. Synchronized scrolling is a common form of coordination in which two views

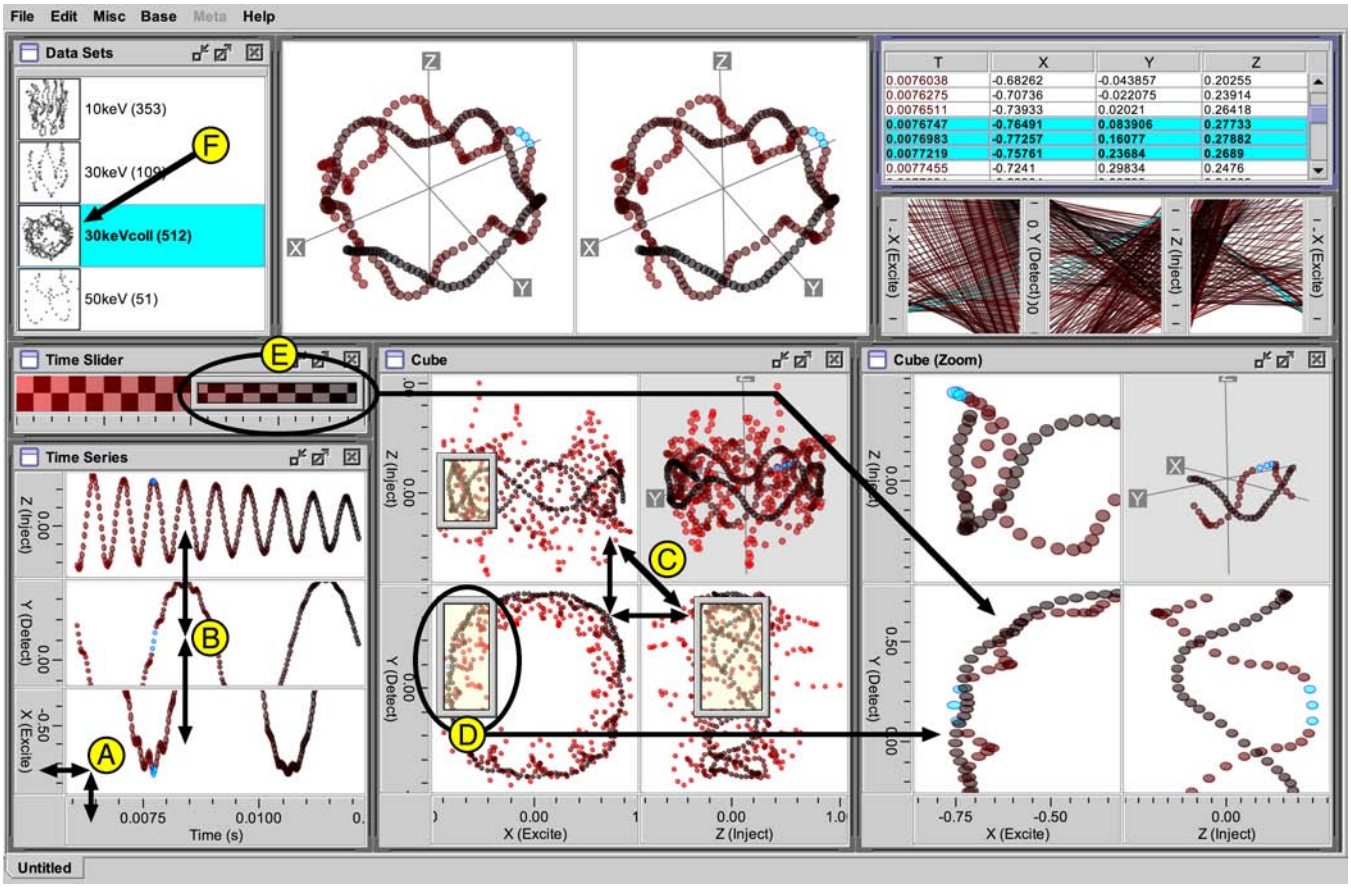

Figure 4: Visualization of a simulated ion trajectory in a cubic ion trap. (A) Axis controls label a plot and provide a way to change X and time independently. (B) Horizontal synchronized scrolling coordinates three time series plots showing the X, Y, and Z positions of ions over time. (C) A scatterplot matrix shows the trajectory as seen from three orthogonal sides of the ion trap. (D) An overview uses a portal (circled) to select the extent of a detail view. (E) A perceptual slider enables users to select a visible range of time using a color gradient instead of numeric values. (F) The names of the available trajectory datasets are accompanied by nested views that project each trajectory into a 3-D view.

are constrained to show the same data items or the same region of a coordinate space. For instance, scatterplots in DEVise can be coordinated with *visual links* to show the same range of X and/or Y. In Snap-Together Visualization, synchronous scrolling between lists of items is achieved by coordinating their *scroll actions*.

Plots can be coordinated with each other in the same way that they coordinate with axis controls: through their range properties. Figure 6 shows three plots in which scrolling is synchronized horizontally. This is done by binding the same range variable to the X range property of all three views. The flexibility of property-variable binding makes it simple to construct numerous variations of synchronized scrolling, including two-dimensional (sharing X and Y ranges), horizontal (sharing only the X range), vertical (sharing only the Y), and crossed (one view shows XY, the other YX).

Scatterplot matrices [3] show an N-dimensional space as a stairstep arrangement of 2-D scatterplots. Synchronized scrolling in this case is complicated by the need to invert the coordinates of some plots in order to produce the expected navigation behavior. Figure 7 shows how inverting the coordinates of a plot is a simple matter of swapping the range variables bound to its live properties. (Building plot matrices is straightforward but tedious; Improvise provides shortcuts for creating common multiview constructions.)

Coordination using the overview+detail [19] technique differs from synchronized scrolling in that the entire area shown in a detail view is synchronized with a subarea of an overview. *Cursors* in

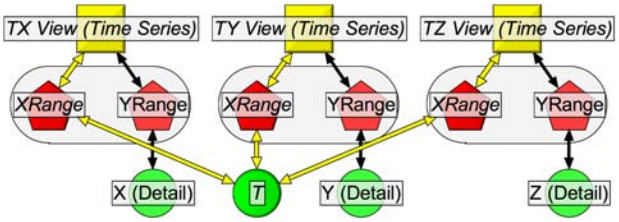

Figure 6: Three scatterplots with synchronized horizontal scrolling but independent vertical scrolling (see figure 4B). All three plots update in unison whenever the value of T changes.

DEVise are an example of this technique in which a selection box in a scatterplot has the same X and Y ranges as some other scatterplot.

In Improvise, *portals* (not to be confused with portals in DataSplash) are draggable controls for selecting a rectangular region. (Portals can also draw data, acting as lenses above the plots that contain them.) Figure 8 shows how the X and Y ranges of a detail plot are coordinated with the ranges of a portal inside an overview plot. This construction can be chained to create multiple levels of detail (as in [18]). Omitting the two X (or two Y) range variables produces vertical (or horizontal) versions of overview+detail.

Another use of one-dimensional portals is in a *perceptual sliders*,

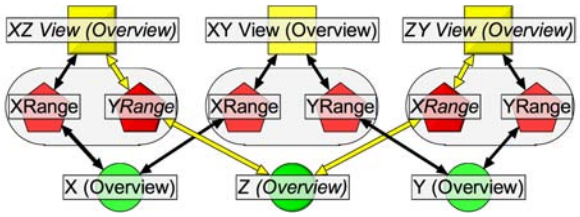

Figure 7: Coordination graph for a 3-D scatterplot matrix (see figure 4C). The shared Z variable synchronizes vertical navigation in the XZ plot with horizontal navigation in the ZY plot.

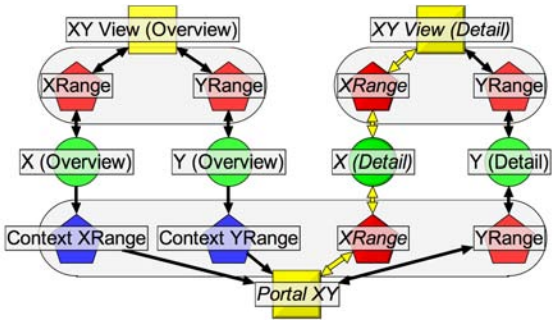

Figure 8: Coordination graph for overview+detail (see figure 4D). The portal covers the region in the overview (its context) that corresponds to the full region visible in the detail view.

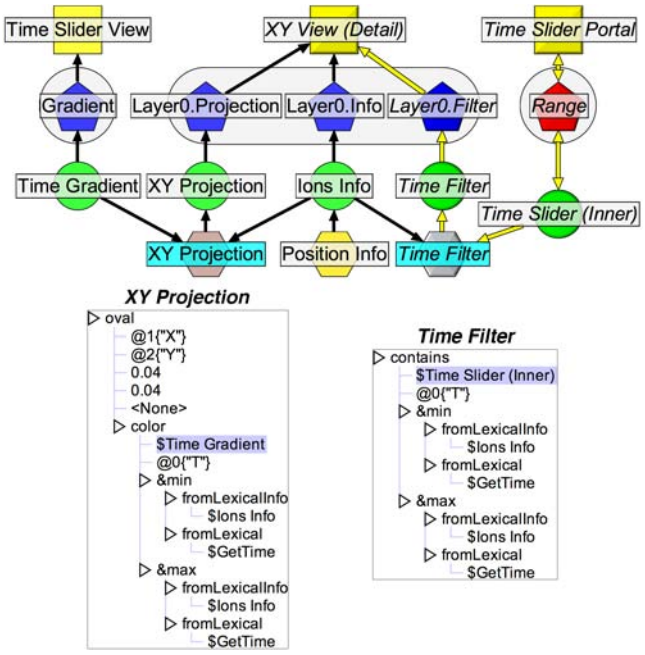

Figure 9: Coordination between a scatterplot and a gradient slider (see figure 4E). The scatterplot draws ovals colored by mapping time into a color gradient, relative to minimum and maximum values, but only for relative times in the range selected by the slider portal.

which allow users to select data by thinking visually while acting spatially. Figure 9 shows how a plot is coordinated with a portal in a gradient view to create a perceptual slider based on color. The projection expression used by the plot visually encodes points along an ion trajectory by mapping (normalized) time into the same color gradient shown in the gradient view. The filter expression used by the plot elides points that would fall outside the range of color selected by the portal. Although the user perceives the position of the portal as a selection on color, the selection is actually on a range of time values. (Perceptual sliders are similar to visualization sliders [7], but present a set of possible perceptual values instead of a distribution of values that actually occur in the data.)

Nested views enable exploration of a group of related datasets by displaying each dataset in its own view, all of which are contained in a larger view. In DataSplash, portals are clickable hyperlink windows into other data spaces. In Improvise, nested views are special glyphs in which the value being visually encoded is an entire dataset. Because all Improvise views use projection expressions to generate glyphs, they all can contain nested views. In figure 10, a list visually encodes data files as a filename next to an icon which shows the data as 3-D points. The projection expression that draws each list item generates a nested 3-D view glyph by applying a second projection to the data from the corresponding file.

North and Shneiderman [14] have described a taxonomy of strategies for coordinating multiple views in terms of selecting items and navigating views. In Improvise, a selection is a bitstring that indicates selected records by the integer identifiers assigned to them when data is accessed during visualization. Decoupling selections from data in this way separates coordination of views on data from coordination of views on selections. This approach makes it possible to coordinate multiple views using multiple independent selections of the same dataset in a single visualization.

Shared selection is a form of brushing that allows the user to select an item in a view, and see the corresponding item in other views. In XGobi [6], users can brush items in multiple scatterplots

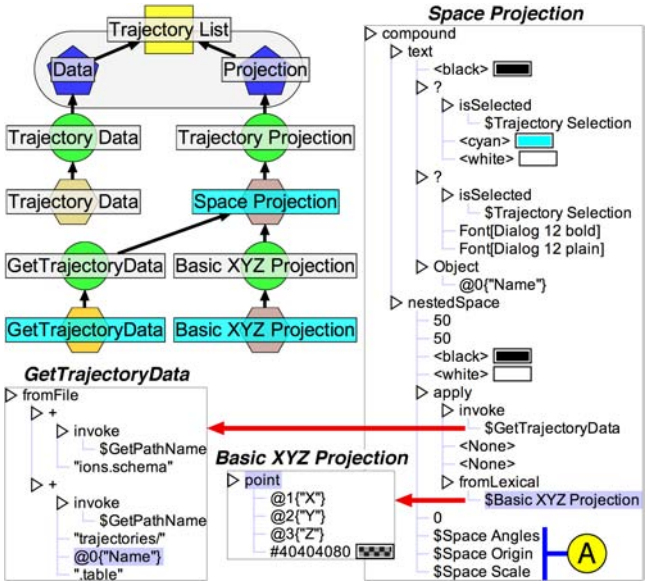

Figure 10: Coordinated query graph for a list of available datasets (see figure 4F). Each item in the list consist of a formatted file name and a nested 3-D plot. These plots are navigationally coordinated with the main 3-D stereogram through variable operators (A).

of high-dimensional data. Brushing-and-linking in Snap-Together Visualization uses *select actions* to coordinate selections in two views of the same data. Figure 12 shows how two Improvise views are coordinated to share a selection over data that describes the 83 counties in Michigan. The scatterplot draws counties as polygons read from shapefiles; the table view draws each county as a row of text with a nested bar plot. Selecting items in either view (by click-

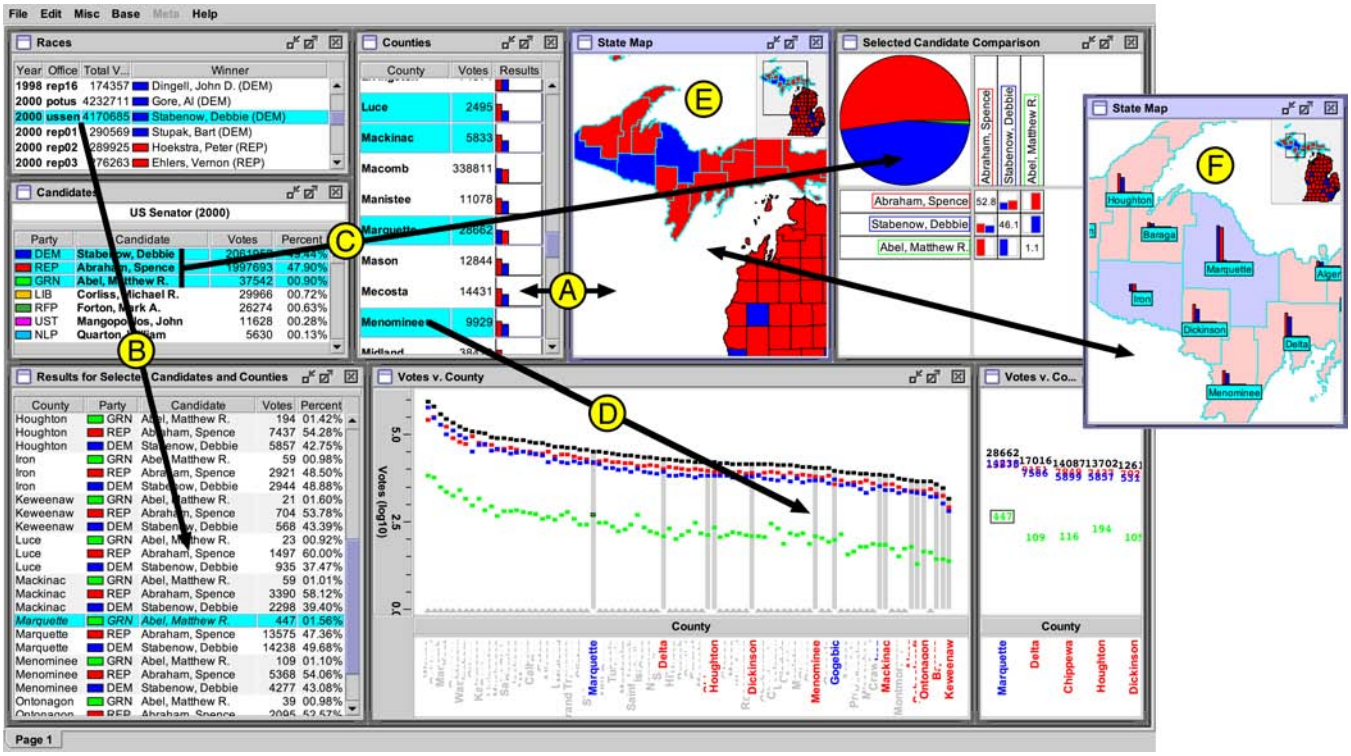

Figure 11: Visualization of election results in Michigan from 1998 to 2002. (A) Shared selection of counties between a table view and a map. (B) Selecting a race causes the election results for that race to be loaded (from a file) and shown throughout the visualization. (C) A pie chart uses a filter to compare results for selected candidates only. (D) A scatterplot highlights selected counties with gray bars. (E) A four-layer scatterplot colors counties by winning candidate party. (F) Semantic zoom labels counties with nested bar plots at sufficient zoom.

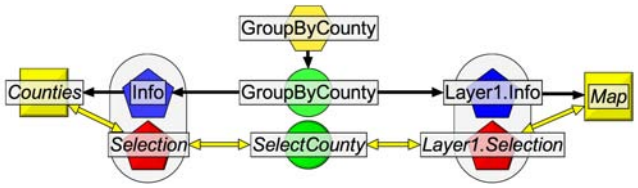

Figure 12: Coordination graph for shared selection between a table view and a scatterplot (see figure 11A). Selection of items in either view causes both views to redraw their shared data.

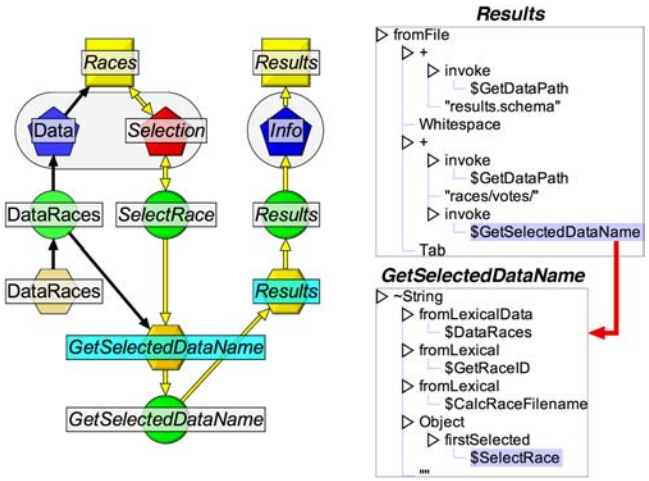

Figure 13: Coordinated query graph for selection-dependent loading of data (see figure 11B). An index on the races dataset maps the record identifier of the first selected race into a filename. The "Results" view displays an *info* (an expression that calculates a dataset) that reads data from the corresponding file of voting results.

ing shapes or rows) changes the selection variable, causing both views to redraw with the selected items highlighted.

Users often want to select from multiple related datasets (or subsets of one large dataset) in a single visualization, such as during analysis of a sequence of experiments. Selecting a dataset in one view to show in another view is a form of drill-down. For instance, Snap-Together Visualization supports drill-down by coordinating a *select action* in one view with a *load action* in another view.

Selection-dependent loading of data in Improvise is performed using an expression that is defined in terms of (1) data that lists the names of (or otherwise identifies) loadable datasets, and (2) a selection on that data. In figure 13, the election results for each office are stored in separate files. The expression constructs the name of a file to load using the name of the selected office. Whenever the user selects an office, the visualization loads data from the corresponding file. Using expressions, the user can specify a file, URL, or database as the source of data to visualize.

Selection-dependent filtering is an asymmetric version of shared selection in which the filtered view differentiates between selected and unselected items by not drawing unselected items instead of highlighting selected ones. DEVise uses this technique in the form of *record links* that cause a "destination" view to render only those tuples that are visible in a "source" view.

Whereas selection-dependent filtering determines the visibility of items, selection-dependent projection determines the appearance

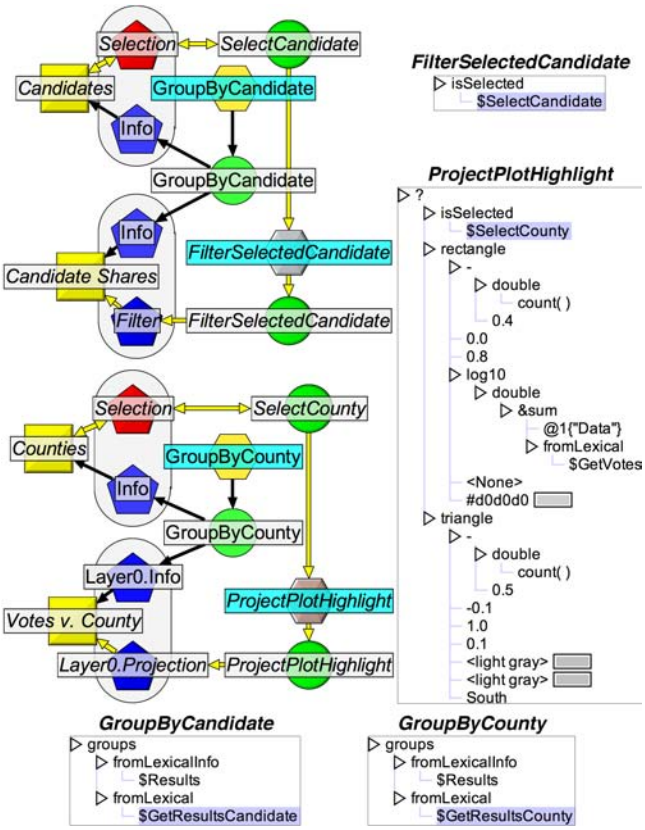

Figure 14: Views can be indirectly coordinated through filters or projections that depend on selection variables (see figure 11C, 11D). The filter expression states that "for each candidate, draw it only if it is selected." The projection expression states that "for each county, draw a rectangle if it is selected, a triangle otherwise." The height of each rectangle is an aggregate of the dataset created by grouping the overall election results by the corresponding county.

of items. Most visualization systems can coordinate two views so as to highlight the items in one view that correspond with items selected in the other view. Highlighting is usually a fixed function of the type of view, typically implemented as a special background color. In XGobi, points and lines in scatterplots can be brushed using glyphs as well as color.

By using expression-based projections to determine the entire visual encoding of items in views, highlighting in Improvise is a user-customizable visual differentiation of selected and unselected items. Highlighting of items can therefore appear as a special background color, reverse video, a special font, or just about any variation on color or other visual attributes the user can dream up. Customizable highlighting can also be used to avoid conflict with normal visual encoding of items. In figure 14, the "Candidate Shares" pie chart shows vote shares for candidates selected in the "Candidates" table view. Although both views display the same data, the filtered view elides unselected candidates using an expression defined in terms of the selection. The result is a kind of multi-item details-on-demand that allows comparison of details for selected subsets of items. The "Votes v. County" scatterplot highlights counties based on whether they are selected in the "Counties" table view. Although the filter and projection expressions in this example depend on independent selections (one of candidates, one of counties), it is easy to extend them to depend on conjunctions or disjunctions of selections. The effect would be similar to additive encoding of selection highlighting in interactive externalizations [22].

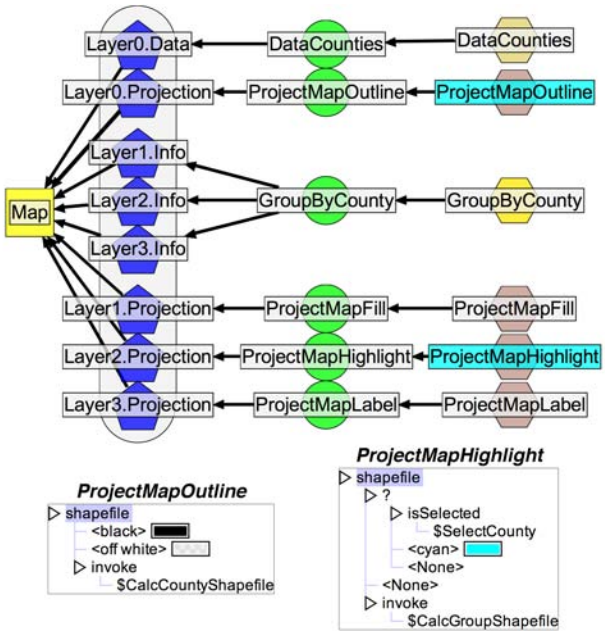

Figure 15: Coordinated query graph for a four layer plot (see figure 11E). The top three layers draw different projections of the same data. All four layers invoke user-defined expressions (not shown) to load county shapefiles for drawing.

Layered plots (such as *piles* in DEVise) enable users to visualize multiple datasets using different visual encodings in a single plot. A common use of layering is to visualize a single dataset using a layer to highlight selected items in a lower layer. In Improvise, scatterplots have multiple layers each defined by its own data, projection, and filter properties. Figure 15 shows how a four layer scatterplot draws a map using four different projections of two data sets. The bottom layer draws all counties. The top three layers fill, highlight, and label only the counties which are involved in the selected election. Drawing labels in the highest layer keeps them from being obscured by shapes in underlying layers. The combination of layering and compound glyphs provides extensive control over the z-order of items drawn in plots.

Semantic zoom is a form of details on demand that lets the user see different amounts of detail in a view by zooming in and out. For instance, the layer manager in DataSplash allows the user to select the amount of detail by changing a view's "altitude". The view draws data using the visual encodings visible at the chosen altitude. Semantic zoom in Improvise uses expressions that calculate glyphs as a function of a plot's own X and Y ranges. Figure 16 shows how the county map plot depends on two ranges both directly and indirectly. Although this example demonstrates synchronized zoom between plot layers, the expressions could be edited to make the layers change detail at different zoom levels. One-dimensional zooming and multiple levels of detail are also straightforward.

5 Conclusion

In Improvise, users interactively build and browse multiview visualizations using a simple shared-object coordination mechanism coupled with a flexible, expression-oriented visual abstraction language. Improvise is a fully-implemented, self-contained Java application that has been used to create complex visualizations of election results, particle trajectories, network loads, county maps, music collections, the chemical elements, and even the dynamic coordination structure of its own visualizations during construction

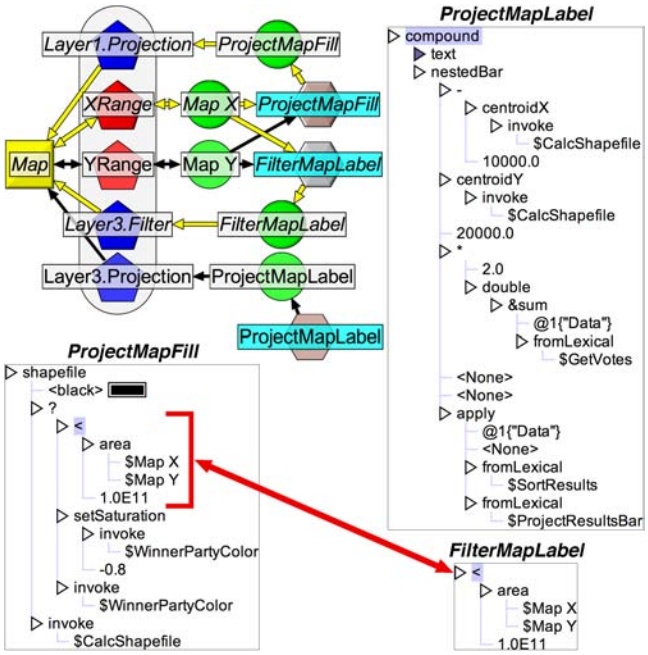

Figure 16: Semantic zoom in the county map (see figure 11F). At sufficient zoom, the top layer draws a centered label and a scaled, nested bar plot for all counties. To make the top layer easier to read, the fill layer reduces the saturation of the winning candidate's party color at the same zoom level.

and data exploration.

Highly-coordinated visualizations appear to be much easier to build in Improvise than other visualization systems because views are connected indirectly through a coordination model in which navigational parameters, selections, data, and visual encodings are shared objects that can be edited on-the-fly. Unlike other visualization systems, there is no need to link views pairwise or in sequence to achieve complex coordination semantics. By way of example, the bottom half of the visualization in figure 4 contains ten scatterplots, four portals, and 13 axes but uses only eight numeric ranges for navigational coordination.

A major goal for Improvise has been to increase coordination flexibility substantially without significantly decreasing ease-of-use, as compared to similar systems like DEVise and Snap; to make simple coordinations (like synchronized scrolling) easy, and complex coordinations (like semantic zoom) possible. Although comparative user studies will be needed to determine if this goal has been achieved, the possibilities for highly-coordinated visualization in Improvise appear to be limited only by user creativity.

6 Acknowledgments

Thanks to Miron Livny and Raghu Ramakrishnan for support and advice, and Kevin Beyer and Kent Wenger for many discussions.

References

[1] Christopher Ahlberg and Ben Shneiderman. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *Proceedings of CHI '94 Conference: Human Factors in Computing Systems*, pages 313–317, 479–480, Boston, MA, April 1994. ACM.

[2] Christopher Ahlberg and Erik Wistrand. IVEE: An environment for automatic creation of dynamic queries applications. In *Proceedings of CHI '95*, pages 15–16, Denver, CO, May 1995. ACM.

[3] Richard A. Becker, P. J. Huber, William S. Cleveland, and A. R. Wilks. Dynamic graphics for data analysis. *Stat. Science*, 2, 1987.

[4] Alan Borning. The programming language aspects of ThingLab, a constraint-oriented simulation laboratory. *ACM Transactions on Programming Languages and Systems*, 3(4):353–367, October 1981.

[5] Nadia Boukhelifa and Peter J. Rodgers. A model and software system for coordinated and multiple views in exploratory visualization. *Information Visualization*, 2003(2):258–269, September 2003.

[6] Andreas Buja, Dianne Cook, and Deborah F. Swayne. Interactive high-dimensional data visualization. *Journal of Computational and Graphical Statistics*, 5(1):78–99, 1996.

[7] Stephen G. Eick. Data visualization sliders. In *Proceedings of UIST '94*, pages 119–120, Monterey, CA, November 1994. ACM Press.

[8] Ralph D. Hill. The abstraction-link-view paradigm: Using constraints to connect user interfaces to applications. In *Proceedings of CHI '92*, pages 335–342, Monterey, CA, May 1992. ACM.

[9] Matthew E. Hodges, Russell M. Sasnett, and Mark S. Ackerman. A construction set for multimedia applications. *IEEE Software*, 6(1):37–43, January 1989.

[10] Allan S. Jacobson, Andrew L. Berkin, and Martin N. Orton. LinkWinds: Interactive scientific data analysis and visualization. *Communications of the ACM*, 37(4):43–52, April 1994.

[11] G. E. Krasner and S. T. Pope. A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, 1(3):26–49, August 1988.

[12] Miron Livny, Raghu Ramakrishnan, Kevin Beyer, G. Chen, Donko Donjerkovic, Shilpa Lawande, Jussi Myllymaki, and Kent Wenger. DEVise: Integrated querying and visualization of large datasets. In *Proceedings of SIGMOD '97*, pages 301–312, Tucson, AZ, 1997. ACM.

[13] Jock D. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2):110–141, April 1991.

[14] Chris North and Ben Shneiderman. A taxonomy of multiple window coordinations. Technical Report CS-TR-3854, University of Maryland Department of Computer Science, 1997.

[15] Christopher Loy North. *A User Interface for Coordinating Visualization Based On Relational Schemata: Snap-Together Visualization*. PhD thesis, University of Maryland, 2000.

[16] Chris Olston, Michael Stonebraker, Alexander Aiken, and Joseph M. Hellerstein. VIQING: Visual Interactive QueryING. In *Proceedings of the 14th IEEE Symposium on Visual Languages*, Halifax, Nova Scotia, Canada, September 1998. IEEE.

[17] Chris Olston, Allison Woodruff, Alexander Aiken, Michael Chu, Vuk Ercegovac, Mark Lin, Mybrid Spalding, and Michael Stonebraker. DataSplash. In *Proceedings of SIGMOD '98*, pages 550–552, Seattle, WA, June 1998. ACM.

[18] Catherine Plaisant, David Carr, and Ben Shneiderman. Image browser taxonomy and guidelines for designers. *IEEE Software*, 12(2):21–32, March 1995.

[19] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of IEEE Symposium on Visual Languages '96*, pages 336–343, Boulder, CO, September 1996. IEEE.

[20] Chris Stolte, Diang Tang, and Pat Hanrahan. Multiscale visualization using data cubes. In *Proceedings of Infovis 2002*, pages 7–14, Boston, MA, Oct 2002. IEEE.

[21] Chris Stolte, Diang Tang, and Pat Hanrahan. Polaris: A system for query, analysis, and visualization of multi-dimensional relational databases. *Transactions on Visualization and Computer Graphics*, 8(1):52–65, Jan 2002.

[22] Lisa Tweedie. Characterizing interactive externalizations. In *Proceedings of CHI '97*, pages 375–382, Atlanta, GA, March 1997. ACM.

[23] Allison Woodruff, Alan Su, Michael Stonebraker, Caroline Paxson, Jolly Chen, Alexander Aiken, Peter Wisnovsky, and Cimarron Taylor. Navigation and coordination primitives for multidimensional visual browsers. In S. Spaccapietra and R. Jain, editors, *Proceedings of the 3rd IFIP 2.6 Working Conference on Visual Database Systems*, pages 360–371, Lausanne, Switzerland, March 1995. Chapman & Hall.