Multiscale Visualization Using Data Cubes

Chris Stolte, Diane Tang, Pat Hanrahan
Stanford University

Abstract

Most analysts start with an overview of the data before gradually refining their view to be more focused and detailed. Multiscale pan-and-zoom systems are effective because they directly support this approach. However, generating abstract overviews of large data sets is difficult, and most systems take advantage of only one type of abstraction: visual abstraction. Furthermore, these existing systems limit the analyst to a single zooming path on their data and thus a single set of abstract views.

This paper presents: (1) a formalism for describing multiscale visualizations of data cubes with both data and visual abstraction, and (2) a method for independently zooming along one or more dimensions by traversing a zoom graph with nodes at different levels of detail. As an example of how to design multiscale visualizations using our system, we describe four design patterns using our formalism. These design patterns show the effectiveness of multiscale visualization of general relational databases.

1 Introduction

When exploring large datasets, analysts often work through a process of "Overview first, zoom and filter, then details-on-demand" [14]. Multiscale visualizations are an effective technique for facilitating this process because they change the visual representation to present the data at different levels of abstraction as the user pans and zooms. At a high level, because a large amount of data needs to be displayed, it is highly abstracted. As the user zooms, the data density decreases and thus more detailed representations of individual data points can be shown.

The two types of abstraction performed in these multiscale visualizations are *data abstraction* and *visual abstraction*. Data abstractions (e.g., aggregation or selection) change the underlying data before mapping them to visual representations. Visual abstractions change the visual representation of data points (but not the underlying data itself) to provide more information as the user zooms; e.g., an image may morph from a simplified thumbnail to a full-scale editable version. Existing systems, such as DataSplash [21] and Pad++ [2], focus primarily on visual abstractions with support for data abstractions limited to simple filtering and the ability to add or switch data sources. In addition, these systems primarily only allow for a single zooming path.

Our goal is to develop a system for describing and developing multiscale visualizations that support multiple zoom paths and both data and visual abstraction. We want to support multiple zoom paths because many large data sets today are organized using multiple hierarchies that define meaningful levels of aggregation (i.e., detail). Data cubes are a commonly accepted method for abstracting and summarizing relational databases. By representing the database with a data cube, we can switch between different levels of detail using a general mechanism applicable to many different data sets. Combining this general mechanism for performing meaningful data abstraction with traditional visual abstraction techniques enhances our ability to generate abstract views of large data sets, a difficult and challenging problem.

Previously, we presented Polaris, a tool for visually exploring relational databases [15] and later extended for hierarchically structured data cubes [16]. In this paper, we describe a multiscale visualization system using data cubes and Polaris. Specifically, we present:

- **Zoom graphs:** We present zoom graphs as a formal notation for describing multiscale visualizations of hierarchically structured data that supports multiple zooming paths and both data and visual abstraction. We also present a system based upon this formalism in which we can easily implement these visualizations.
- **Design patterns:** While these graphs and our system provide a general method for describing and developing multiscale visualizations of hierarchically structured data, designing such visualizations remains a hard and challenging problem. We use our formalism to enumerate four design patterns in the style of Gamma et al. [10] that succinctly capture the critical structure of commonly used multiscale visualizations. In addition, these patterns illustrate the use of small multiples and tables in multiscale visualizations.

Note that we are using data cubes not only because they provide a powerful mechanism for data abstraction, but also because many large and important data sets are already stored in relational databases and data cubes.

The layout of the rest of this paper is as follows. In Section 2, we survey existing approaches to multiscale visualization. Next, we describe in Section 3 how multiscale visualizations can be expressed as graphs using our Polaris formalism and data cubes and then implemented in Rivet [5]. We then present our design patterns in Section 4 before concluding with some discussion and directions for future work in Section 5.

2 Related Work

In this section, we review several existing multiscale visualization systems, focusing on how the systems perform both data and visual abstraction. *Data abstraction* refers to transformations applied to the data before being visually mapped, including aggregation, filtering, sampling, or statistical summarization. *Visual abstraction* refers to abstractions that change the visual representation (e.g., a circle at an overview level versus a text string at a detailed level), change how data is encoded in the retinal attributes of the glyphs (e.g., encoding data in the size and color of a glyph only in detailed views), or apply transformations to the set of visual representations (e.g., combining glyphs that overlap).

Multiscale Visualization in Cartography

Cartography is the source of many early examples of multiscale visualizations. Cartographic generalization [19] refers to the process of generating small scale maps by simplifying and abstracting large scale source material and consists of two steps: (1) employing selection to decide which features should be shown and (2) simplifying the visual representations of the selected features. A map series developed using this process and depicting a single geographic region at varying scales is a multiscale visualization. While the initial selection process is a specialized form of data abstraction, the subsequent manipulations are all visual abstractions.

Multiscale Information Visualization

Several information visualization systems provide some form of zooming or multiscale interface. Given our goal in expressing general multiscale visualizations, we only discuss general systems; domain-specific tools may apply both data and visual abstraction but their abstractions are not generally applicable.

The Pad series of interfaces (Pad++ [2] and Jazz [3]) are among the earliest examples of multiscale visualization in information visualization. These systems were developed not as data exploration tools but as alternate desktops, although they have been applied to other domains such as web histories [12]. Given this goal, their focus has been on interaction and applying visual abstractions for "semantic zooming" rather than easily applying data abstractions.

DataSplash [21] is the first multiscale visualization system focused on data exploration. It provides the layer manager, a novel interface mechanism for easily constructing multiscale visualizations of graphs. Each individual graph can have multiple layers, with each layer activated at different viewing elevations. As the user zooms, the set of active layers change. Layers can be used to change the visual representation of relations and to add or remove data sources. Although DataSplash provides mechanisms for zooming on a single graph, it does not provide mechanisms for zooming on tables or small multiples of graphs, nor does it provide for multiple zooming paths on a single graph.

XmdvTool [13] provides multiscale views using hierarchical clusters that structure the underlying data into different levels of abstraction; widgets such as as structured brushes [9] provide a mechanism for zooming. XmdvTool is limited to this single method for providing data abstraction and does not provide visual abstraction capabilities.

Eick and Karr also present a survey of common visual metaphors and associated interaction techniques and motivate the need for both data and visual abstractions from perceptual issues [6]. These issues drive the ADVIZOR system, which uses multiple visual metaphors, each with a single zoom path based on with the visual and data abstractions given in their survey. They do not provide a system for exploring other types of zooms nor a formal notation for describing multiscale visualizations.

3 Multiscale Visualizations

In this section, we present our system for describing multiscale visualizations that support multiple zoom paths and both data and visual abstraction. Rather than considering multiscale visualizations as simply a series of linear zooms, we think of multiscale visualizations as a graph, where each node corresponds to a particular set of data and visual abstractions and each edge is a zoom. Zooming in a multiscale visualization is equivalent to traversing this graph. Each node in this graph can be described using a Polaris specification that identifies the visual representation and abstraction and can be mapped to a unique projection of the data cube, which is a data abstraction of the underlying relational data.

In the remainder of this section, we first review the two technologies we use to perform data abstraction (data cubes) and visual abstraction (Polaris). When reviewing Polaris, we also introduce a graphical notation for describing the key elements of a specification. Finally, we present how we can create a zoom graph of Polaris specifications to describe a multiscale visualization of a hierarchical data set, as well as how we can easily implement such visualizations within our system.

3.1 Data Abstraction: Data Cubes

Not only are data cubes widely used, but they also provide a powerful mechanism for performing data abstraction that we can leverage. Specifically, data cubes quickly provide summaries of the underlying data at different meaningful levels of detail, rather than arbitrary summarizations such as aggregating every two records. This goal is achieved by building a lattice of data cubes to represent the data at different levels of detail according to a semantic hierarchy and providing mechanisms for then summarizing each cube. We first describe an individual data cube before describing the lattice.

Data cubes categorize information into two classes: dimensions and measures, corresponding to the independent and dependent variables, respectively. For example, U.S. states are a dimension, while the population of each state is a measure. Within a cube, the data is abstractly structured as an n-dimensional data cube. Each axis corresponds to a dimension in the data cube and consists of every possible value for that dimension. For example, an axis corresponding to states would have fifty values, one for each state. Every "cell" in the data cube corresponds to a unique combination of values for the dimensions. For example, if we had two dimensions, State and Product, then there would be a cell for every unique combination of the two (e.g., one cell each for (California, Oranges), (California, Coffee), (Florida, Oranges), (Florida, Coffee), etc.). Each cell contains one value per measure of the data cube; e.g., if we wanted to know about product production and consumption, then each cell would contain two values, one for the number of products of each type consumed in that state, and one for the number of products of each type produced in that state.

Thus far, we have considered dimensions to be flat structures. However, most dimensions have a hierarchical structure. For example, rather than having a single dimension "state", we may have a hierarchical dimension "location" that has levels for country, state, and county. If each dimension has a hierarchical structure, then the data must be structured as a lattice of data cubes, where each cube is defined by the combination of a level of detail for each dimension.

Data abstraction in this model means choosing a meaningful summary of the data. Choosing a data abstraction corresponds to choosing a particular *projection* in this lattice of data cubes: (a) which dimensions we currently consider relevant and (b) the appropriate level of detail for each relevant dimensional hierarchy. Specifying the level of detail identifies the cube in the lattice, while the relevant dimensions identifies which projection (from n-dimensions down to the number of relevant dimensions) of that cube is needed. Figure 1 shows a simple lattice and projection.

While identifying a specific projection in the data cube corresponds to specifying the desired data abstraction of the raw data, in multiscale visualizations we need to specify both the data and visual abstractions; both sets of information are contained in a Polaris specification.

3.2 Visual Abstraction: Polaris

Previously, we presented the Polaris database exploration tool [15], consisting of three parts: (1) a formal specification language for describing table-based visualizations, (2) a user interface for automatically generating instances of these specifications, and (3) a method for automatically generating the necessary database queries to retrieve the data to be visualized by a specification. We later extended all three parts to support hierarchically structured data cubes [16].

In this paper, we only use the specification language from the previous papers. We use this language to describe a node within the zoom graph identifying a multiscale visualization. In this section, we briefly review the components of a Polaris specification and introduce a graphical notation that succinctly captures the data and visual abstractions in table-based visualizations of hierarchically structured data.

A Polaris specification uses a formal table algebra to specify the table configuration of the visualization. Each expression in the table algebra defines an axis of the table: how the table is divided into rows or columns. The main components of an expression are the operands and the operators. Each operand is the name of a field and can be one of two types: a dimension is an ordinal operand (O) while a measure is a quantitative operand (Q). (Note that nominal fields become ordinal fields since the values must be drawn in some order.) The type of the operand determines how the field is encoded into the structure of the table: ordinal fields partition the table into rows and columns while quantitative fields are spatially encoded as axes within the table panes.

A valid expression in the algebra is an ordered sequence of one

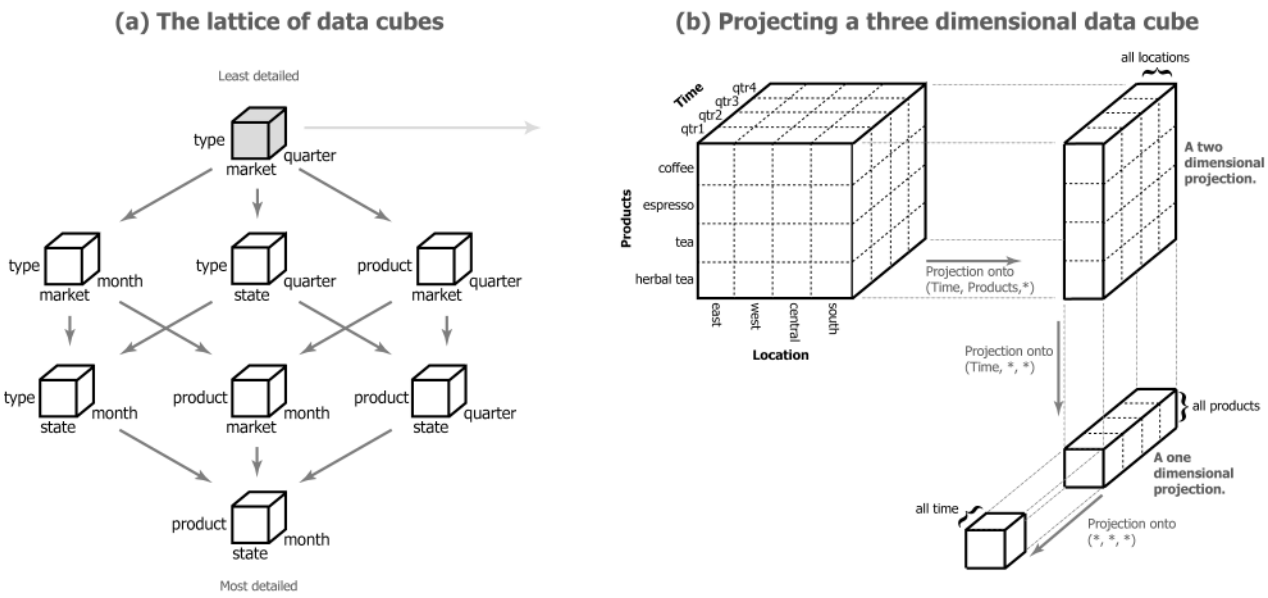


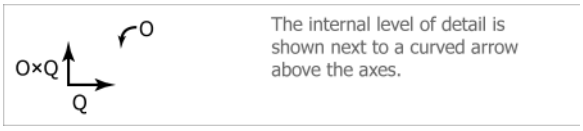
Figure 1: (a) The lattice of data cubes for a data base with three dimensions: Products (with levels Type and Product), Time (with levels Quarter and Month), and Location (with levels Market and State). (b) Several projections of the least detailed data cube in the lattice.

or more operands with an operator between each pair of adjacent operands. The four types of operators in this algebra, in order of precedence, are dot (.), cross (x), nest (/), and concatenate (+). Parentheses can be used to alter the precedence of the operators. Each operand can be interpreted as an ordered set and the precise semantics of each operator are defined in terms of their effects on these operand sets. The dot operator specifies the desired level of detail within a dimensional hierarchy. The cross and nest operator behave like a cross-product of two vectors (nest only produces pairs for which data exist), and the concatenate operator yields the union of two sets. The full details of these operators are given in our previous papers [15][16].

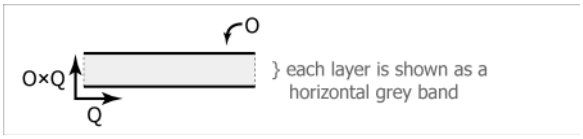
The table algebra is only one of five portions comprising a complete Polaris specification. We now briefly describe each part and its corresponding portion in a graphical notation (inspired by Bertin's notation for describing charts and diagrams [4]) for succinctly communicating a Polaris specification.

- **The table structure**: Two expressions in the table algebra, one each for the x- and y-axis, define (1) the rows and columns of the table and (2) how data is spatially encoded within each pane. Changing the expressions changes the data abstraction; for example, using the dot operator on an operand identifies a different data cube.

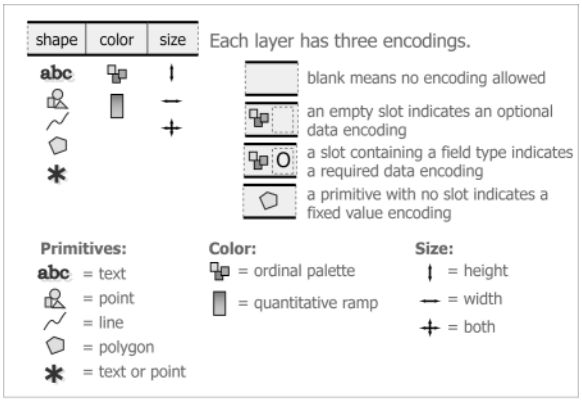
- **Internal level of detail**: This portion of the specification identifies any dimensions that are needed but not already encoded in the table structure. Together, the complete list of dimensions uniquely identify the desired projection of the data cube. Changing the internal level of detail changes the data abstraction.



- **The mapping of data sources to layers**: Multiple data sources may be combined within a single Polaris visualization, with each source mapped to a separate layer. All layers share the same table structure and are composited together back-to-front to generate the final visualization.



- **The visual representation for tuples**: Both the mark type and the retinal attributes of each mark can be specified. While the current graphical notation encodes only color and size, it is easily extended to include other retinal attributes such as shape, orientation, or texture.



We will use this graphical notation throughout the rest of the paper to describe our design patterns for multiscale visualizations. Note that this notation can be used describe both a class of visualizations (a *visual template*) or a specific visualization (*visual instance*).

3.3 Zoom Graphs

We describe a multiscale visualization as a graph because we want to allow for multiple zoom paths from any given point; this ability is necessary for exploring data cubes that commonly have multiple independent hierarchical dimensions. An individual zoom can either change the data abstraction, the visual abstraction, or both. The zooming actions can be tied to an axis, for example allowing zooms

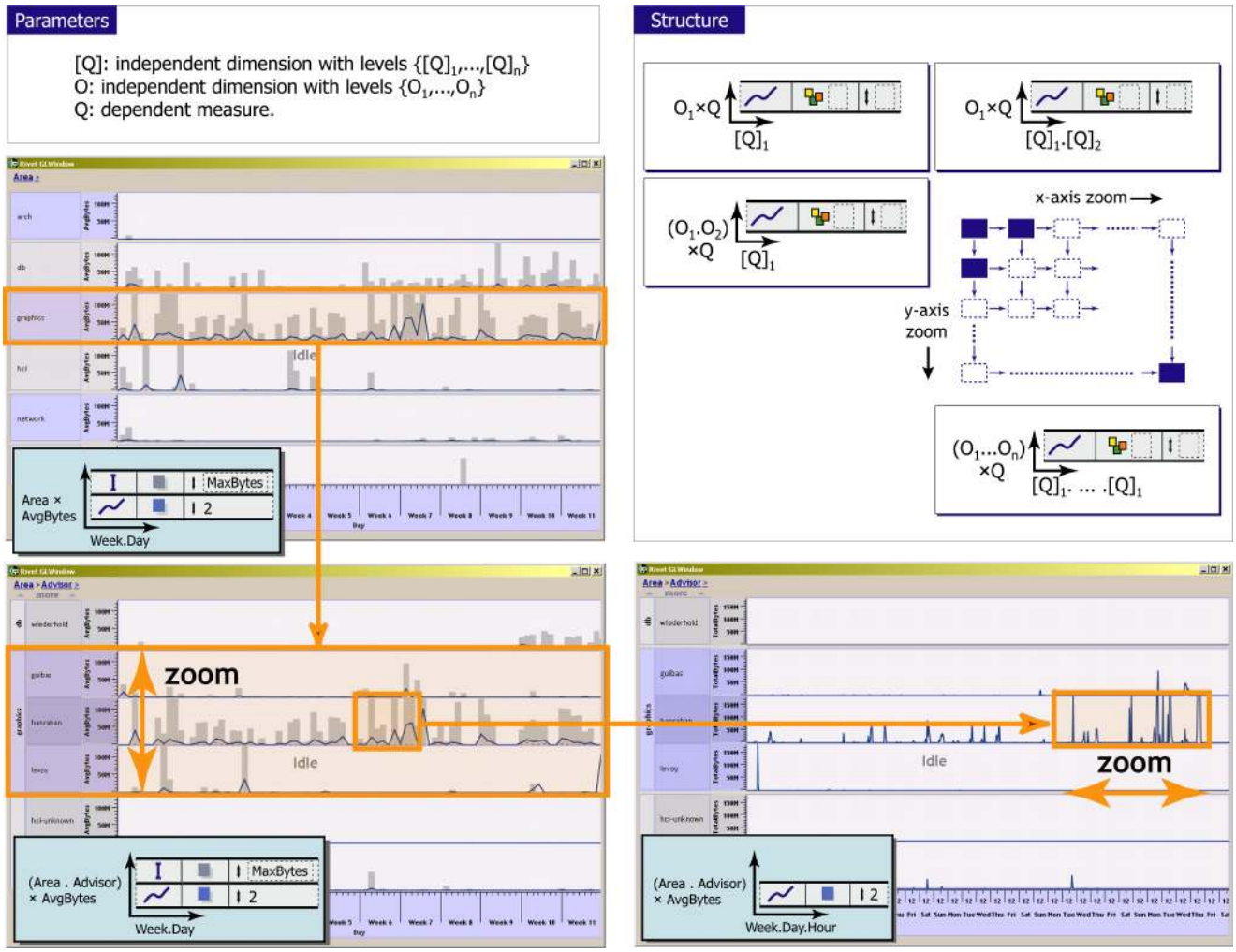


Figure 2: The Zoom Graph for the Chart Stacks Pattern as well as screenshots of a visualization of a 12-week trace of an in-building mobile network developed using that pattern. The top visualization shows a line chart of average bytes/hour for each day for each research area. The line charts are layered above a high-low bar encoding the maximum and minimum bytes/hour. In the next visualization, the user has zoomed in on the y-axis, breaking apart the charts to create a chart for each advisor within the research groups. In the final visualization, the user has zoomed on the x-axis, increasing the granularity of the line chart to hourly values from daily values.

along the x- and y-axis independently, or they may be triggered by interacting with an external widget.

The previous sections describe how to express a node in the graph using a Polaris specification and how a specification corresponds to a particular projection of a data cube. Using the graphical notation we introduced, we can describe and design these zoom graphs. In this section, we explain how we implement the multiscale visualization corresponding to a zoom graph within Rivet [5], a visualization environment designed for rapidly prototyping interactive visualizations. The main components of any implementation of a multiscale visualization are the nodes, the edges, and how user interaction can trigger a transition (i.e., an edge traversal).

Nodes: Each node in a zoom graph is abstractly described using a Polaris specification. In our implementation, we can concretely describe a Polaris specification using XML. Rivet contains an interpreter that parses the XML to create a visualization, which includes automatically generating both the necessary queries (i.e., SQL or MDX queries) and drawing operations.

Edges and Interaction: Visualizations are created in Rivet by writing a script. An abstract zoom graph can be implemented as a finite state machine within a Rivet script. We use Rivet's event binding mechanism to bind user interaction events (e.g., mouse movement events) to procedures within the script to trigger transitions between states (i.e., nodes).

The notation described in this section is very general and can be used to describe many different graphs, i.e., many different multiscale visualizations. Many of these visualizations are easily implemented within Rivet. However, designing effective multiscale visualizations is still a challenging task, and in the next section, we present four patterns using our graphical notation that describe common multiscale visualizations and encapsulate the changes in abstraction that occur as the user zooms.

4 Multiscale Design Patterns

Even though implementing a multiscale visualization is simplified using our system, designing such a visualization is still, in general, a hard and challenging problem. One way to help solve this problem is to capture zoom structures that have been effective as patterns that can be reused in the design of new visualizations. In this section, we present four standard zooms and express them using our formal notation for zoom graphs. These zooms have traditionally been used in domain-specific applications, and while we also give specific examples for each, our notation expresses each pattern as a general class of multiscale visualizations. Each zoom is described

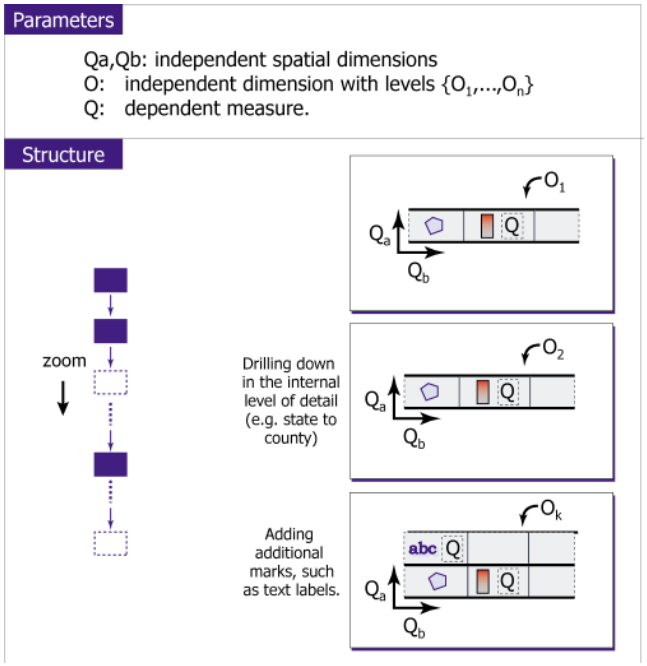


Figure 3: Zoom graph for Pattern 2: Thematic Maps

in the style of Gamma et al. [10], and the goal is not only to provide some guidance to others when designing multiscale visualizations, but also to provide a formal way for exchanging design knowledge and discussing multiscale visualizations (i.e., which data and visual abstractions to apply).

Pattern 1: Chart Stacks

This first pattern applies when analysts are trying to understand how a dependent measure (such as profit or number of network packets) varies with two independent hierarchical ordinal dimensions, one derived from continuous data (such as time). This type of data can be effectively visualized using a vertically stacked small multiple of line charts (e.g., a single column of charts) [20]. The hierarchy derived from continuous data is encoded in the x-axis of each chart while the other hierarchy determines the y-axis structure of the table (e.g., the order and number of rows). The y-axis for each individual chart encodes the dependent measure. The zooming in this pattern is inspired both by the types of visualizations created in ADVIZOR [6] as well as in our own analyses of this type of data [17].

The main thing to note in this pattern is that the analyst can independently zoom along either the x- or y-axis, leading to a graph describing the multiscale visualization; the analyst can choose any path through this graph. Each zoom corresponds to changing the data abstraction: the dot operator is applied to the table algebra expression corresponding to the relevant axis. Zooming along the x-axis changes the granularity of each individual chart while zooming along the y-axis changes the number of charts. The zoom graph for this pattern is shown in Figure 2.

Figure 2 also shows how we applied this pattern to a 12-week trace of every packet that entered or exited the mobile network in the Gates Computer Science building at Stanford University [18]. Each packet is categorized by the time it was sent (one hierarchy) and the user who sent the packet (the second hierarchy); the schema of this data is described in Figure 9. To transition between the different specifications, the user can trigger the y-zoom by clicking on the arrow at the top of the y-axis to introduce a new level of detail, and we animate the transition by growing one chart before breaking it into multiple charts, and similarly animate the x-zoom by growing a bar before showing its breakdown.

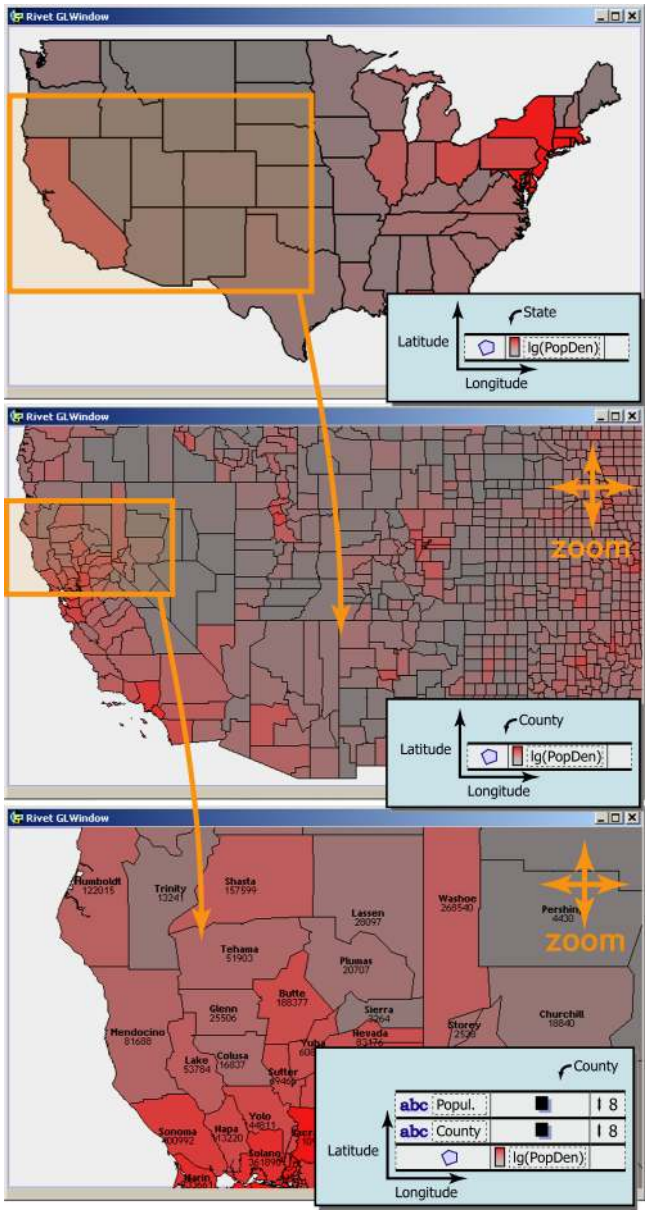


Figure 4: A series of screenshots of a multiscale visualization of the population of the USA, developed using the "Thematic Maps" pattern. The initial view is at the state level of detail, with each state colored by population density. As the user zooms in, with the x and y dimensions lock-stepped together, the visualization changes data abstraction, drilling down to the county level of detail. As the user zooms in further, the visual abstraction changes as layers are added to display more details: both the county name and population values are displayed as text.

Pattern 2: Thematic Maps

This pattern is applicable when visualizing geographically-varying dependent measures that can be summarized at multiple geographic levels of detail (such as county or state). Thus, the data contains an ordinal dimension hierarchy that characterizes the geographic levels of detail, two independent spatial dimensions (e.g., latitude and longitude), and some number of dependent measures. Examples of this type of data are census or election data. Typically, this type of data is visualized as a map with measures encoded in the color of the area features or as glyphs layered on the map.

Unlike the previous pattern, where the user could zoom independently on x and y, in this pattern, the user must zoom on both si-

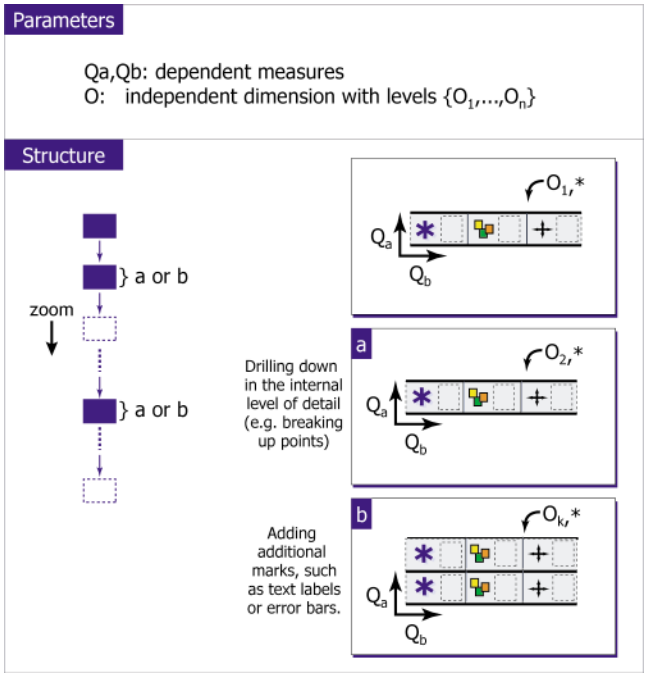


Figure 5: Zoom graph for Pattern 3: Dependent Quantitative-Dependent Quantitative Scatterplots

multaneously. Thus, zooming in this pattern is like a fly-through: as the viewer zooms, more detail is displayed. There are two types of zooms in this pattern: the data abstraction can change by changing the specification's internal level of detail or the visual abstraction can change by adding details in additional layers. The zoom graph for this pattern is shown in Figure 3.

To illustrate this pattern, we show in Figure 4 a series of zooms on a thematic map where the measure of interest is population density. The schema is shown in Figure 9. In this example, the user can zoom in by moving the mouse up or zoom out by moving the mouse down. As the user zooms in, the map zooms in; when a predetermined elevation is reached, the script switches to a different specification, i.e., a different node in the zoom graph.

Pattern 3: Dependent Quantitative-Dependent Quantitative Scatterplots

This pattern (inspired by the types of visualizations created in DataSplash [21]) is very similar to the previous pattern in that the main visualization again has two quantitative axes. However, the primary distinction between the two patterns is that in this pattern, the axes have no inherent mapping to the physical world; instead, they spatially encode an abstract quantity, thus freeing many constraints imposed in the previous pattern. Thus, the data used in this type of visualization can be any set of abstract measurements that can be categorized according to some set of hierarchies. Many corporate data warehouses fall into this category.

Like the previous example, there are two types of zooms in this pattern. The data abstraction can change by either adding or removing fields or changing the level of detail of the fields listed in the internal level of detail portion of the specification. Changing this portion of the specification changes the number of tuples, thus changing how many marks are displayed.

Alternatively, the visual abstraction can change, either by adding retinal encodings to the current layers or by adding information in additional layers. Note that while the map pattern must keep a layer with a polygonal mark, this pattern has considerably more flexibility. The zoom graph for this pattern is shown in Figure 5.

To illustrate this pattern, we use constructed data from a hypothetical chain of coffee shops (the schema is shown in Figure 9). A multiscale visualization of this data set is shown in Figure 6.

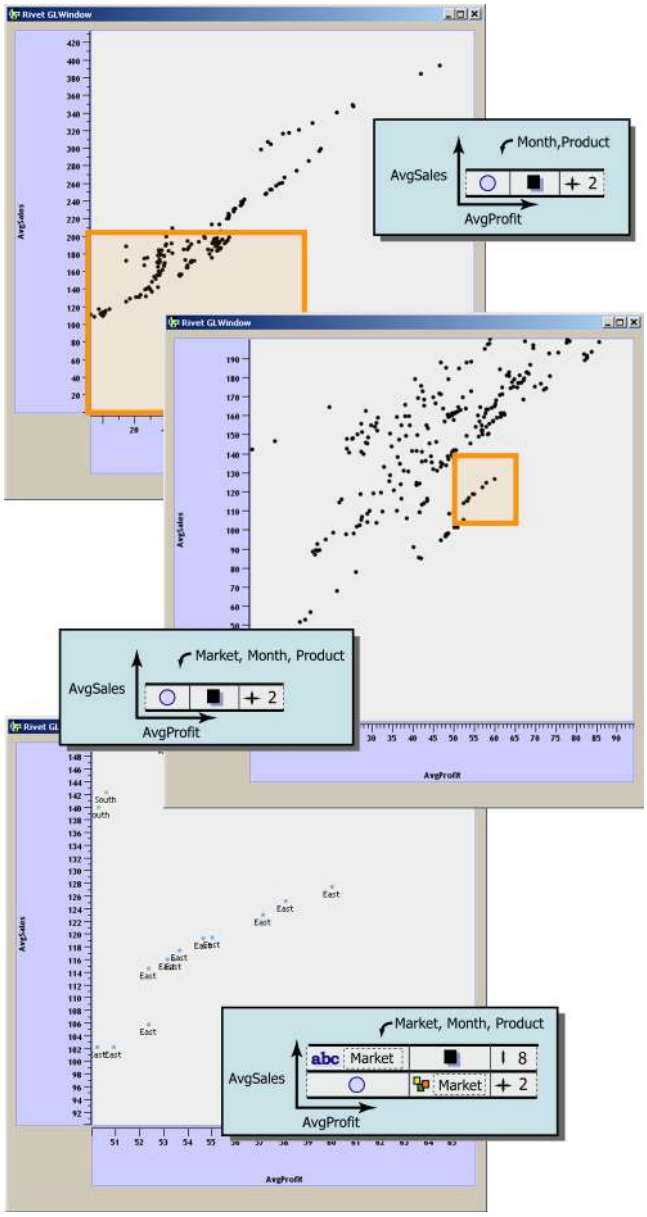


Figure 6: A series of screenshots of a multiscale visualization of average sales versus average profit over a two-year period for a hypothetical coffeeshop chain. In the first visualization, each point represents profit and sales for a particular month and product, summed over all locations. In the next visualization, the user zooms, changing the data abstraction: points that were originally aggregated over all locations are now broken down by market, resulting in four points for every original point. As the user zooms in further, the visual abstraction changes as layers are added to display more details: each point is colored according to market and a text label is added to redundantly encode the market name.

Pattern 4: Matrices

Our final pattern applies when the analyst is exploring how a dependent measure varies with the values of two independent dimension hierarchies and is motivated by Abello's work in visualizing call density [1]. This type of data can be effectively visualized as a table, where the rows encode one hierarchy while the columns encode a different hierarchy and a glyph in each cell depicts the measure.

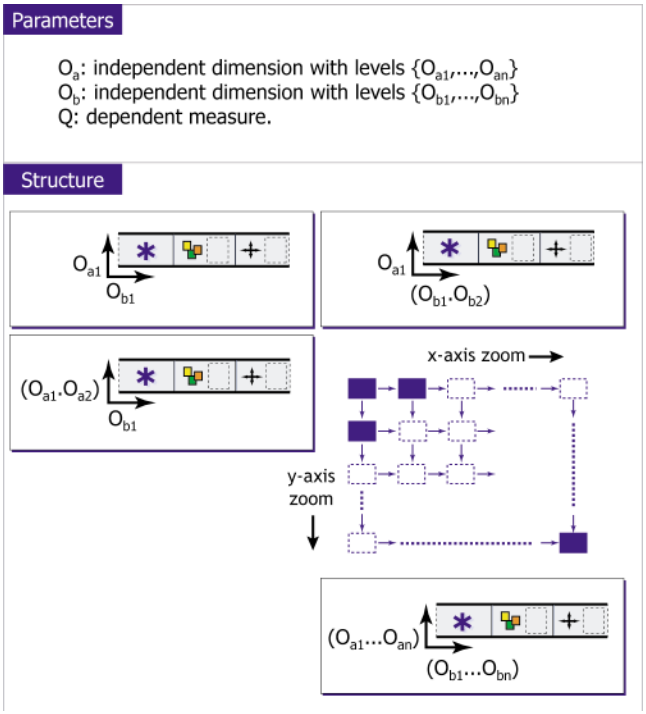


Figure 7: Zoom graph for Pattern 4: Matrices

Zooming in this graph involves either aggregating rows (or columns) or breaking a single row (or column) down into multiple rows (or columns). In other words, the zooms are changes in the data abstraction: the user can change the level of detail requested on either the x- or y-axis (by applying the dot operator), either independently or together. The zoom graph for this pattern is shown in Figure 7.

One type of data that fits this type of display particularly well is DNA microarray data (the schema is shown in Figure 9), where a series of microarray experiments are performed, each experiment measuring the expression level of different genes. The genes can be clustered to form one hierarchy and the experiments can also be clustered to form another hierarchy. We illustrate this pattern using publicly available yeast gene expression data [7] and Eisen's publicly available clustering software [8]. A visualization for this data based on this pattern is shown in Figure 8.

5 Discussion and Future Work

This paper presents (1) a formalism for describing multiscale visualizations of data cubes with both data and visual abstraction, and (2) a method for independently zooming along one or more dimensions by traversing a zoom graph with nodes at different levels of detail. As an example of how to design multiscale visualizations using our system, we describe four design patterns using our formalism. These design patterns show the effectiveness of multiscale visualization of general relational databases.

One of the key insights behind the system is the importance of performing *both* data and visual abstraction using general mechanisms, especially since many of the multiscale design patterns rely heavily on data abstraction. Data cubes are a commonly accepted method for abstracting and summarizing relational databases, much like how wavelets are used to abstract continuous functions. By representing the database with a data cube, we can switch between different levels of detail using a general mechanism applicable to many different data sets. Previous multiscale visualization systems performed data generalization using special-purpose mechanisms, and hence are only applicable to their specific domain.

Given the Polaris formalism and data cubes, it was relatively

Figure 8: A series of screenshots of a multiscale visualization of yeast microarray data developed using the Matrix pattern. The first visualization shows the highest level gene clusters on the y-axis, the microarray experiment clusters on the x-axis, and the average gene expression in each cell. In the next visualization, the user zooms on both axes to show more detailed information for both gene and array clusters. In the final visualization, the user has zoomed to show the original measurements for each gene in each microarray experiment.

easy to construct a visualization of a particular level of detail in a hierarchical database [16]. It turns out to be much more complicated, however, to construct continuous zooms into the data. One issue is that the user can zoom along different axes, since, unlike previous systems, we allow both independent and coupled zooming on the x- and y-axis. However, we do not allow pivoting during a zoom, i.e., the dimension displayed along an axis cannot change during a zoom. We also currently restrict zooms to always follow the same linear hierarchy rather than allowing arbitrary branching hierarchies (i.e., snowflake schemas) in which a category might belong to multiple hierarchies. Developing more flexible zooms is

Mobile Network Trace

Time ([Q])
Independent hierarchical dimension (derived from continuous data) describing when packets were sent.
Levels: Week, Day, Hour, Minute.

User (O)
Independent hierarchical dimension describing the user of the machine that sent the packet.
Levels: Area ("research area", e.g., graphics), Advisor, Project, Username

PacketCount (Q)
Dependent measure indicating the number of packets sent.

Population Data

Latitude, Longitude (Q)
Independent quantitative dimensions.

PopDensity, Population (Q)
Dependent measures.

SpatialLOD (O)
Independent hierarchical ordinal dimension describing the available geographic levels of detail.
Levels: State, County

Coffee Shop Data

Products (O)
Independent hierarchical ordinal dimension describing the products sold.
Levels: Product Type, Product

Time ([Q])
Independent hierarchical ordinal dimension derived from continuous data describing when a business transaction took place.
Levels: Year, Quarter, Month

Location (O)
Independent ordinal dimension describing the location of stores.
Levels: Market (e.g., "east"), State

Profit, Sales (Q)
Dependent measures describing business activity.

Yeast Microarray Data

Gene Clusters (O)
Independent hierarchical ordinal dimension describing a hierarchical clustering of the gene data created using Eisen's Cluster software.
Levels: GeneRoot, Level 1-12, Genes

Array (Experiment) Clusters (O)
Independent hierarchical ordinal dimension describing a hierarchical clustering of the array data created using Eisen's Cluster software.
Levels: ArrayRoot, Level 1-8, Arrays

Expression (Q)
Dependent measure describing the amount of gene expression on a microarray.

Figure 9: The schemas for the example data sets.

one important area for future work.

Even with these restrictions, there are many ways to zoom, many of which are not very effective. We have described four fairly simple patterns that are effective and that we have used in several applications. These patterns were motivated by previous work, but are still quite general. Many incremental extensions of these patterns are possible. For example, the thematic map pattern should work whenever there is a fixed mapping between the spatial encodings and the physical world. And, while we use line charts for the vertically stacked charts pattern, other chart types (e.g., histograms, strip charts, and Gantt charts) might work equally well. We should also be able to embed one pattern within another. There are also completely different patterns that also might work; developing an extensive repository of zoom graphs would be another good direction for future work.

Another critical issue when designing a zooming interface is in making natural transitions between levels of detail, requiring visualizations to clearly communicate the parent-child relationships. We have found visual cues such as color and padding to be effective in indicating the hierarchy. Another difficulty in transitioning between different views occurs when using categorical hierarchies with non-uniform branching factors. This situation means that more space is needed to zoom into some nodes than others. We have explored several transition mechanisms, including animating the transition and gradually fading between the two views to avoid the disconcerting "popping" that can happen.

A final area of future work is to develop the systems infrastructure so that very large datasets may be visualized in real-time. In this paper, we have used small to moderate-sized data sets so that the zooms are interactive. Larger datasets could viewed at interactive rates if we optimized the data cube queries using a combination of prefetching and caching.

6 Acknowledgments

The authors especially thank Maneesh Agrawala, Francois Guimbretiere, Tamara Munzner, Maureen Stone, and Barbara Tversky for many useful discussions. This work was supported by the US Department of Energy through the ASCI Level 1 Alliance with Stanford University.

References

[1] J. Abello and J. Korn. MGV: A System for Visualizing Massive Multidigraphs. In IEEE Trans. on Visualization and Computer Graphics, 8(1), January 2002, pp. 21-38.

[2] B. Bederson, J. Hollan, K. Perlin, J. Meyer, D. Bacon, and G. Furnas. Pad++: A Zoomable Graphical Sketchpad for Exploring Alternate Interface Physics. In J. of Visual Languages and Computing, 7, 1996, pp. 3-31.

[3] B. Bederson, J. Meyer, and L. Good. Jazz: An Extensible Zoomable User Interface Graphics Toolkit in Java. In Proc. UIST 2000, 2(2), 2000, pp. 171-180.

[4] J. Bertin. Semiology of Graphics: Diagrams, Networks, Maps. Univ. of Wisconsin Press, 1983.

[5] R. Bosch, C. Stolte, D. Tang, J. Gerth, M. Rosenblum, and P. Hanrahan. Rivet: A Flexible Environment for Computer Systems Visualization. In Computer Graphics, 34(1), February 2000.

[6] S. Eick and A. Karr. Visual Scalability. In J. of Computational and Graphical Statistics, 11(1), March 2002, pp. 22-43.

[7] M. Eisen, P. Spellman, P. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. In Proc Natl. Acad. Sci. USA 95, 1998, pp. 14863-8.

[8] M. Eisen. Cluster and Treeview. http://rana.lbl.gov.

[9] Y. Fua, M. Ward, and E. Rundensteiner. Structure-based Brushes: A Mechanism for Navigating Hierarchically Organized Data and Information Spaces. In IEEE Trans. on Visualization and Computer Graphics, June 2000.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1995.

[11] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. In J. of Data Mining and Knowledge Discovery, 1(1), 1997, pp. 29-53.

[12] R. Hightower, L. Ring, J. Helfman, B. Bederson, and J. Hollan. Graphical Multiscale Web Histories: A Study of PadPrints. In Proc. ACM Conference on Hypertext, 1998, pp. 58-65.

[13] E. Rundensteiner, M. Ward, J. Yang, and P. Doshi. XmdvTool: Visual Interactive Data Exploration and Trend Discovery of High-dimensional Data Sets. In Proc. ACM SIGMOD 2002, June 2002.

[14] http://www.cs.umd.edu/hcil/research/visualization.shtml

[15] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A System for Query, Analysis, and Visualization of Multi-dimensional Relational Databases. In IEEE Trans. on Visualization and Computer Graphics, 8(1), January 2002, pp. 52-65.

[16] C. Stolte, D. Tang, and P. Hanrahan. Query, Analysis, and Visualization of Hierarchically Structured Data using Polaris. In Proc. ACM SIGKDD 2002, July 2002.

[17] D. Tang. Analyzing Wireless Networks. Ph.D. Dissertation, October 2000.

[18] D. Tang and M. Baker. Analysis of a Local-Area Wireless Network. In Proc of the 6th International Conference on Mobile Computing and Networking, August 2000, pp. 1-10.

[19] F. Topfer and W. Pillewizer. The principles of selection, a means of cartographic generalization. In Cartographic J., 3(1), 1966, pp. 10-16.

[20] E. Tufte. The Visual Display of Quantitative Information. Cheshire, Connecticut: Graphics Press, 1983.

[21] A. Woodruff, C. Olston, A. Aiken, M. Chu, V. Ercegovac, M. Lin, M. Spalding, and M. Stonebraker. DataSplash: A Direct Manipulation Environment for Programming Semantic Zoom Visualizations of Tabular Data. J. of Visual Languages and Computing, Special Issue on Visual Languages for End-user and Domain-specific Programming, 12(5), October 2001, pp. 551-571.