Multiscale Visualization Using Data Cubes

Chris Stolte, Diane Tang, and Pat Hanrahan

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 to 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.

Index Terms—Multiscale visualization, database visualization, graphic formalism.

1 Introduction

When exploring large datasets, analysts often work through a process of "Overview first, zoom and filter, then details-on-demand" [21]. 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 [29] 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 [22] and later extended for hierarchically structured data cubes [23]. 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 zoom 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. [13] 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.

• *The authors are with the Department of Computer Science, Stanford University, Gates Computer Science Building, Stanford, CA 94305-4070. E-mail: {cstolte, dtang, hanrahan}@graphics.stanford.edu.*

Manuscript received 12 Nov. 2002; accepted 5 Dec. 2002.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number SI0005-1102.

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 or simplifying polygons).

2.1 Multiscale Visualization in Cartography

Cartography is the source of many early examples of multiscale visualizations. Cartographic generalization [27] 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.

Several data structures that are similar to data cubes have been developed in efforts to develop interactive systems for exploring (or simply producing) map series. Representative models are the Multiscale Tree [11] and Map Cube [26], both of which utilize a tradeoff between storage and computation: Redundant storage replaces all steps in the cartographic generalization process that are difficult to automate. The general approach is to pregenerate (either by hand or algorithmically) a tree that stores generalizations of objects at different levels of detail. A selection process generates output by extracting generalizations from this tree until a specified information density is reached. While these structures are generated using a precomputation process similar to data cubes, the process focuses entirely on visual abstraction rather than data abstraction.

2.2 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 [16]. Given this goal, their focus has been on interaction and applying visual abstractions for "semantic zooming" rather than easily applying data abstractions.

DataSplash [29] 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 [20] provides multiscale views using hierarchical clusters that structure the underlying data into different levels of abstraction; widgets such as structured brushes [12] 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 [8]. 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 (stored in a base *fact table*) 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

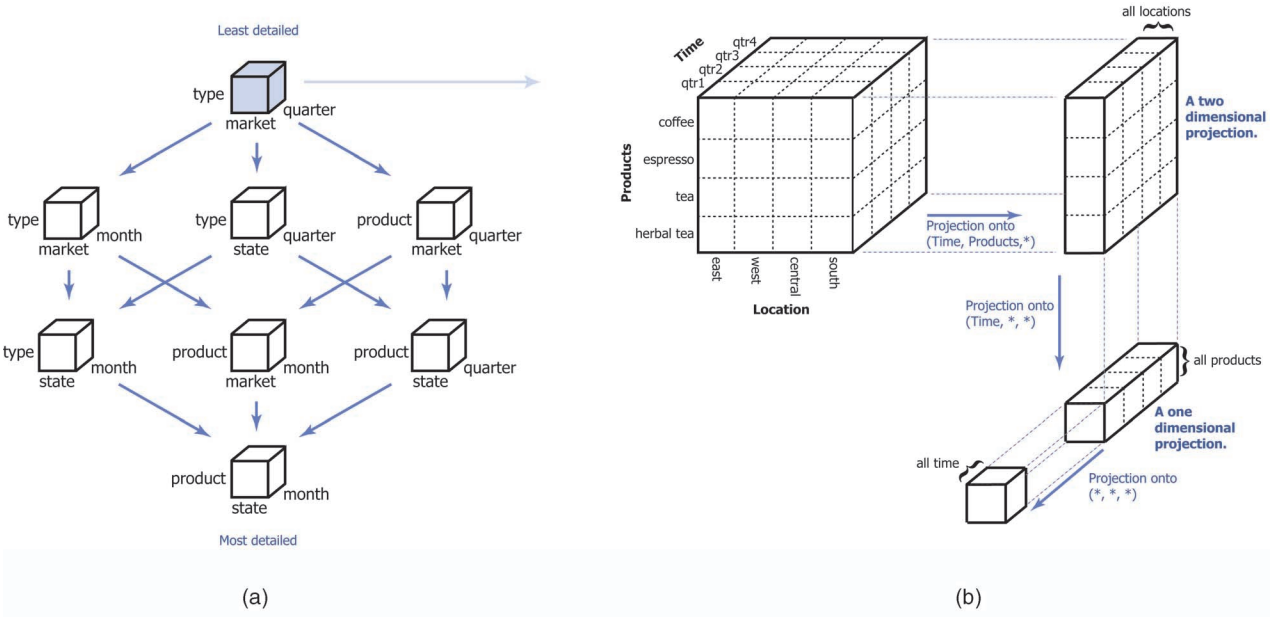


Fig. 1. (a) The lattice of data cubes for a data base with three dimensions: Products (with levels Type and Product), Time (with levels Quarter and Month), and Location (with levels Market and State). (b) Projecting a three-dimensional data cube: several projections of the least detailed data cube in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

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, US 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 50 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 (i.e., a single ordered list of values). However, most dimensions have a hierarchical structure (stored in a *dimension table*). 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: 1) which dimensions we currently consider relevant and 2) 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. Fig. 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 [22], 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 [23].

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.

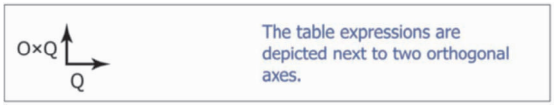
A Polaris specification uses a formal table algebra to specify the table configuration of the visualization. Each expression in the table algebra defines an axis of the table: how the table is divided into rows or columns. The main components of an expression are the operands and the operators. Each 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 or more operands with an operator between each pair of adjacent operands. 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.

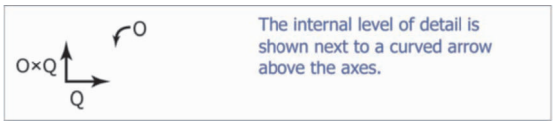
The Polaris algebra presented in the original Polaris paper [22] had three operators: cross, next, and concatenate (in order of precedence). The cross and nest operator behave like a cross-product of two vectors (nest only produces pairs for which data exist) and the concatenate operator yields the union of two sets. In order to fully support and expose the hierarchical structure in data cube dimensions, we had to redefine the algebra to introduce another operator, the dot (".") operator, that has the highest precedence and is similar to the nest operator, but operates relative to the dimension tables rather than the fact table, and is thus "hierarchy-aware." The dot operator specifies the desired level of detail within a dimensional hierarchy and can thus be used to traverse the lattice of data cubes. The precise semantics of the dot operator are defined in the hierarchical Polaris paper [23].

The table algebra is only one of four 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.

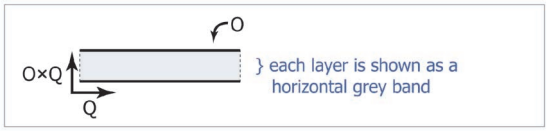
- **The table structure:** Two expressions in the table algebra, one each for the x- and y-axis, define 1) the rows and columns of the table and 2) how data is spatially encoded within each pane. Changing the expressions changes the data abstraction, for example, using the dot operator on an operand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



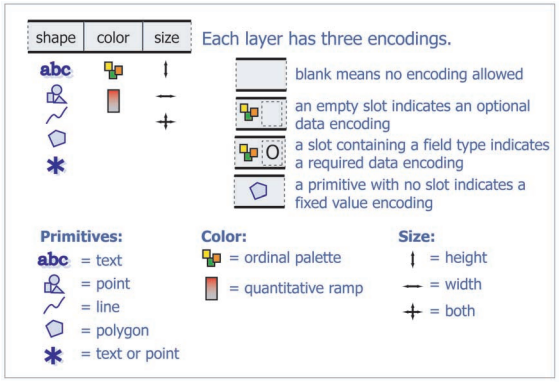
- **Internal level of detail:** This portion of the specification identifies any dimensions that are needed but not already encoded in the table structure. Together, the complete list of dimensions uniquely identify the desired projection of the data cube. Changing the internal level of detail changes 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. This notation can be used to describe either a *visual instance*, which corresponds to a specific visualization, or a *visual template*, which captures the structure of a class of visualizations. Whereas visual instances indicate the specific fields to be encoded in attributes of the display, visual templates parameterize the field types (quantitative, ordinal, etc.) required for each encoding present in the visualization. Visual templates provide a concise, encapsulated description of a visualization, thus enabling an easy way for comparison and discussion.

3.3 Zoom Graphs

In a typical analysis, the user starts at a high level overview and successively zooms in on areas of interest; as he zooms, he also changes the visual representation of the data being displayed. In the Polaris interface, we store the analysis history as a graph of Polaris specifications with each specification representing a subsequent visualization created during the analysis. This history captures not only the visualizations created, but also the ad hoc visual and data abstraction selected by the analyst (the analyst selects the data abstraction as well as the visual because we can

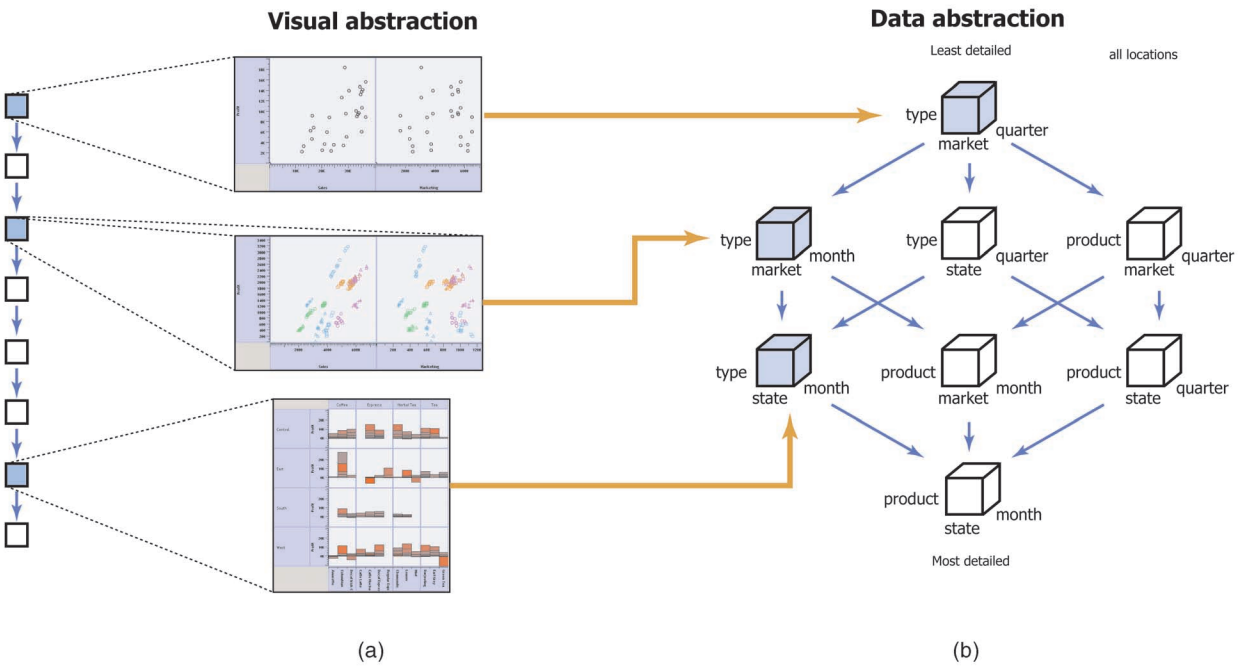


Fig. 2. (a) Analysis as a graph of specifications: In the Polaris interface, we store the analysis history as a path or graph of Polaris specifications with each specification representing a visualization created during the analysis. Shown here is the history of an analysis of sales data for a hypothetical coffee chain that was presented in the Polaris OLAP paper [23]. Examining the graph, we can see that, during the analysis, the user implicitly performs visual abstraction as they change display types to test hypotheses and investigate areas of interest. (b) Mapping analysis to the lattice of cubes: If we consider the mapping of Polaris specifications to data cubes in the lattice of cubes, it is apparent that the analysis also involves implicit data abstraction. This insight leads to the realization that general multiscale visualizations can be modeled using a graph of specifications.

map each specification to a particular projection of a specific cube). This path of exploration is depicted in Fig. 2.

Our insight is that we can use this model not only to record the analysis process, but also to describe multiscale visualizations supporting both visual and data abstraction. Specifically, we can link specifications together into a graph with neighboring specifications corresponding to a change in abstraction (visual, data, or both) and with the links representing possible zooms.

This zoom graph model supports multiple zoom paths from any given point since each node can have multiple incoming and outgoing links (each corresponding to a different path). This flexibility is needed for exploring data cubes that commonly have multiple independent hierarchical dimensions. An individual zoom can change either the data abstraction, the visual abstraction, or both. The zooming actions can be tied to an axis, for example, allowing zooms 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

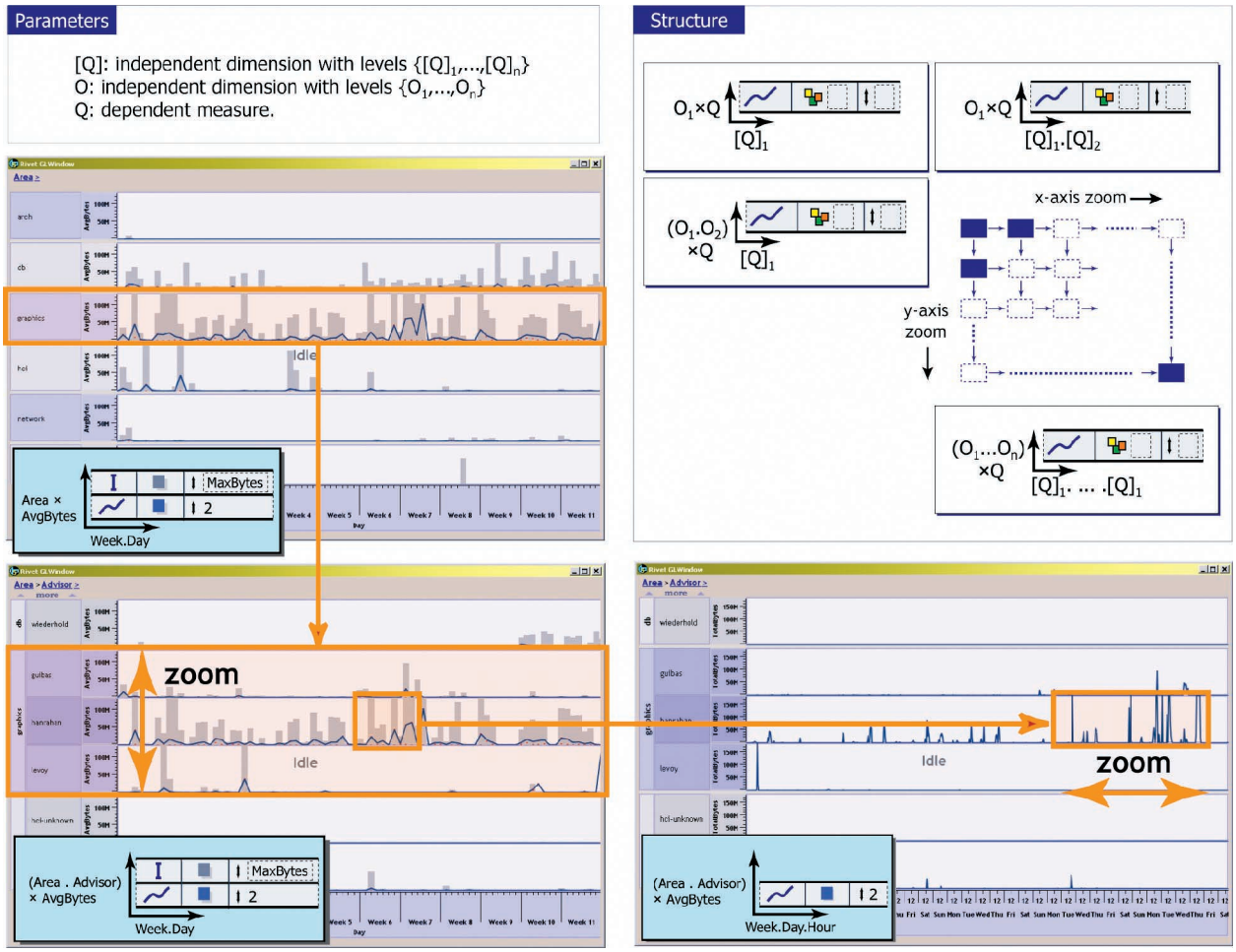


Fig. 3. The zoom graph for the chart stacks pattern as well as screenshots of a visualization of a 12-week trace of an in-building 25 network developed using that pattern. The top visualization shows a line chart of average bytes/hour for each day for each research area. The line charts are layered above 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.

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 in the style of Gamma et al. [13], 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).

4.1 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) [28]. 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 [8] as well as in our own analyses of this type of data [24].

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 Fig. 3.

Fig. 3 also shows how we applied this pattern to a 12-week trace of every packet that entered or exited the 25 network in the Gates Computer Science building at Stanford University [25]. 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 Fig. 11. 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.

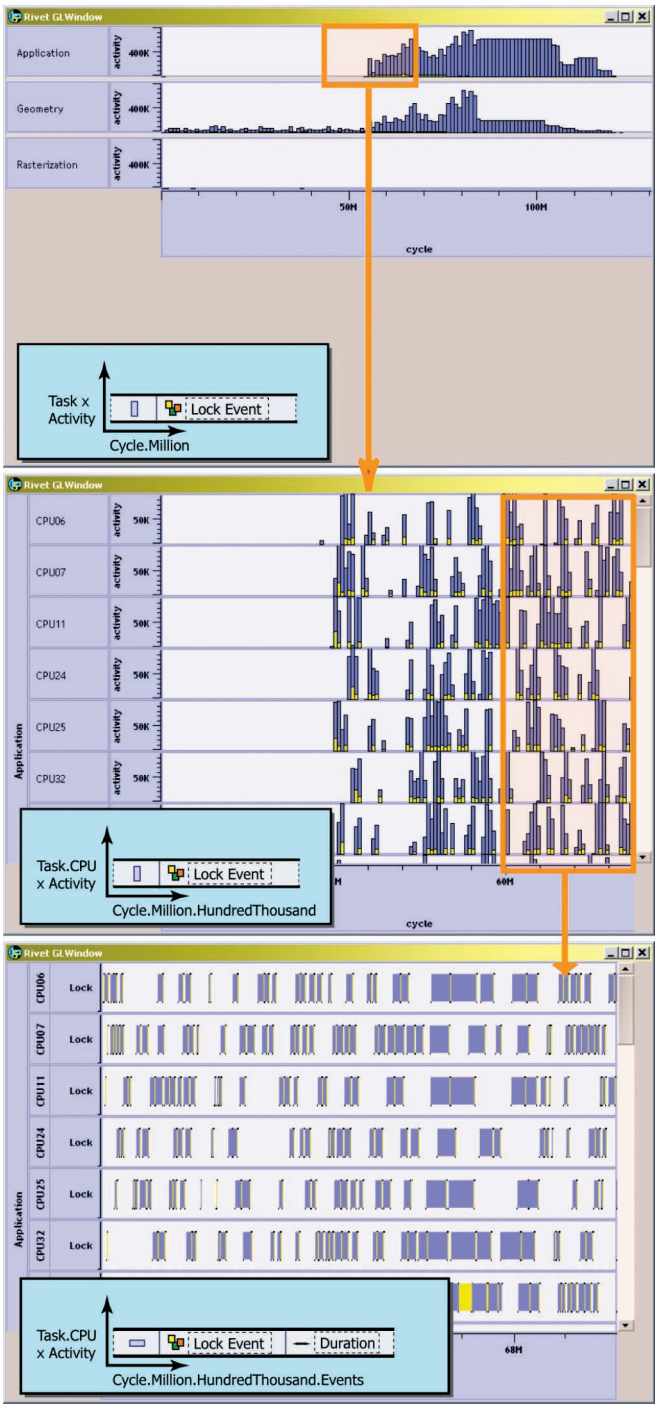


Fig. 4. A variation on the chart stack pattern: A visualization of kernel lock activity (lock requests are shown in blue; time holding a lock is shown in yellow) collected from a 125 million cycle simulation of the Argus parallel graphics library [17]. The top visualization shows a histogram of average time spent requesting or holding a kernel lock. A time interval corresponds to one million cycles and CPUs are grouped by their primary task (e.g, processing geometry, rasterization, etc.). In the next visualization, the user has zoomed in on the y-axis, breaking apart the task charts to create a chart for each CPU, and has zoomed on the x-axis, changing the time granularity to 100,000 cycles per interval. In the final visualization, the user has zoomed further on the x-axis, resulting in a change in visual abstraction from strip charts to Gantt charts displaying individual events.

Fig. 4 shows how we applied the chart stack pattern to kernel lock events collected from a simulation of a parallel graphics application. In this application, rather than using line charts, we use both strip charts and Gantt charts. As the user zooms, not only does the data abstraction change, but the visual abstraction changes from using high-level strip charts summarizing across time to using detailed Gantt charts depicting individual events.

4.2 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 simultaneously. 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 Fig. 5.

To illustrate this pattern, we show in Fig. 6 a series of zooms on a thematic map where the measure of interest is population density. The schema is shown in Fig. 11. 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.

4.3 Pattern 3: Dependent Quantitative-Dependent Quantitative Scatterplots

This pattern (inspired by the types of visualizations created in DataSplash [29]) is similar to the previous pattern in that the main visualization again has two quantitative axes. However, there is an important distinction between the two patterns. In this pattern, the axes have no inherent mapping to the physical world; instead, they spatially encode abstract quantities, thus freeing many constraints imposed in the previous pattern. For example, it would make sense to zoom independently on the axes in this pattern, whereas that operation is not allowed in the Thematic Map 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.

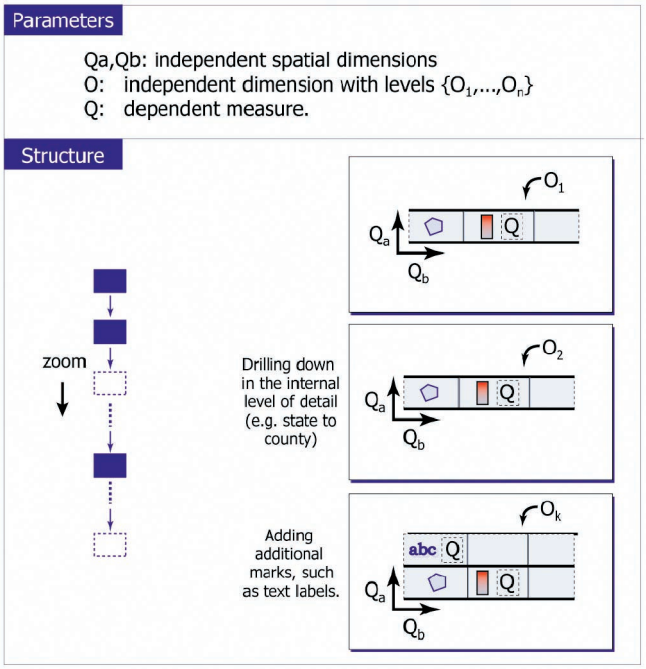


Fig. 5. Zoom graph for Pattern 2: Thematic Maps.

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 Fig. 7.

To illustrate this pattern, we use constructed data from a hypothetical chain of coffee shops (the schema is shown in Fig. 11). A multiscale visualization of this data set is shown in Fig. 8.

4.4 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 and Korn'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.

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 Fig. 9.

One type of data that fits this type of display particularly well is DNA microarray data (the schema is shown in Fig. 11), 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 [9] and Eisen's publicly available clustering software [10]. A visualization for this data based on this pattern is shown in Fig. 10.

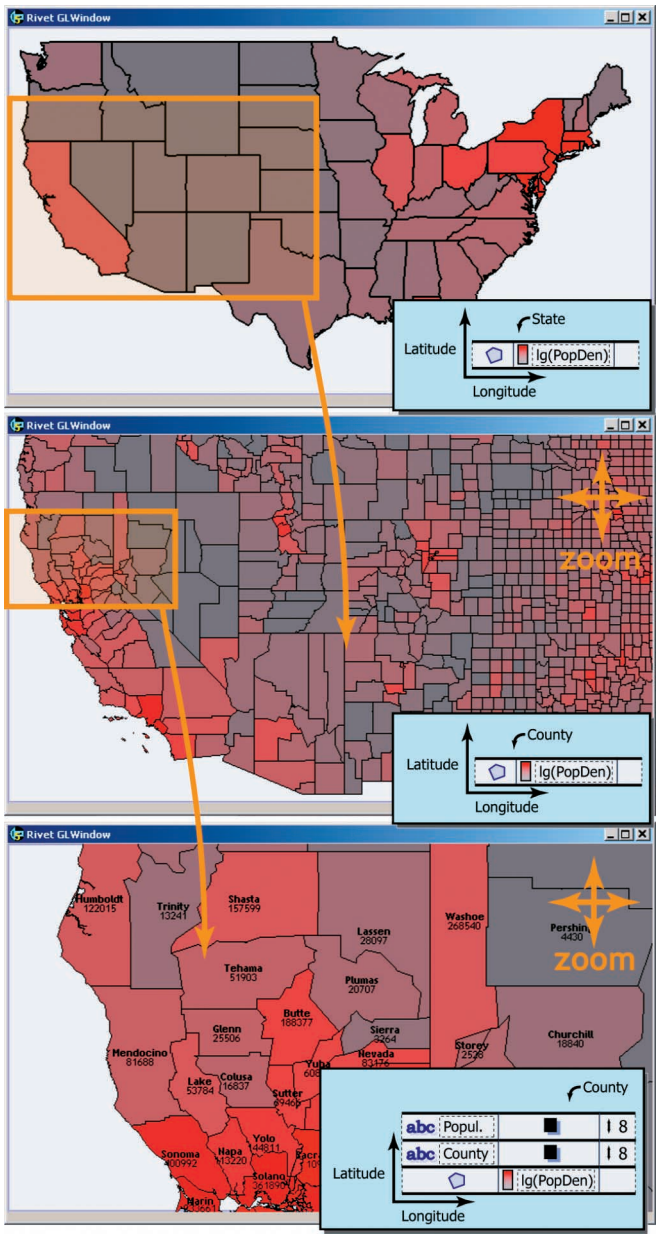


Fig. 6. A series of screenshots of a multiscale visualization of the population of the USA, developed using the "Thematic Maps" pattern. The initial view is at the state level of detail, with each state colored by population density. As the user zooms in, with the x and y dimensions lock-stepped together, 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.

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.

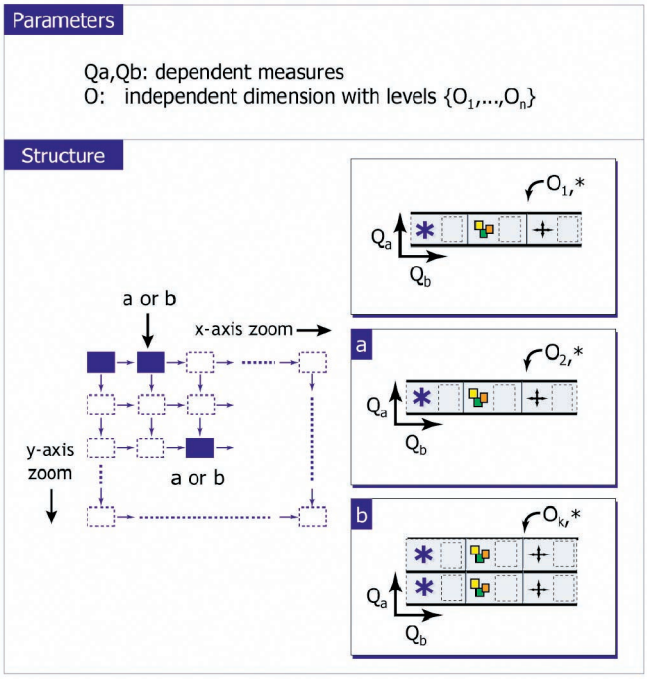


Fig. 7. Zoom graph for Pattern 3: Dependent Quantitative-Dependent Quantitative Scatterplots.

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.

In this section, we discuss the issue of more sophisticated hierarchies, such as branching hierarchies, and future work in supporting these hierarchies, as well as extending this research to find more design patterns and design effective transitions between zooms.

5.1 Multiple Hierarchies

Data cubes are inherently multidimensional, with each dimension modeled with a hierarchy defining the levels of detail to use when aggregating the base fact table. In the simple case, these hierarchies are simple, uniform, and nonbranching so that there is only a single way to define the levels of detail for any particular dimension, i.e., a single path for zooming along that dimension. This type of data is commonly modeled using a star schema and is the type of data we have used in this paper to show how users can independently zoom on multiple hierarchies within a single visualization by associating hierarchies with the axes of a visualization.

Data warehouses can also contain intersecting, nonuniform, or branching hierarchies that can be modeled by snowflake schemas or directed acyclic graphs. In other

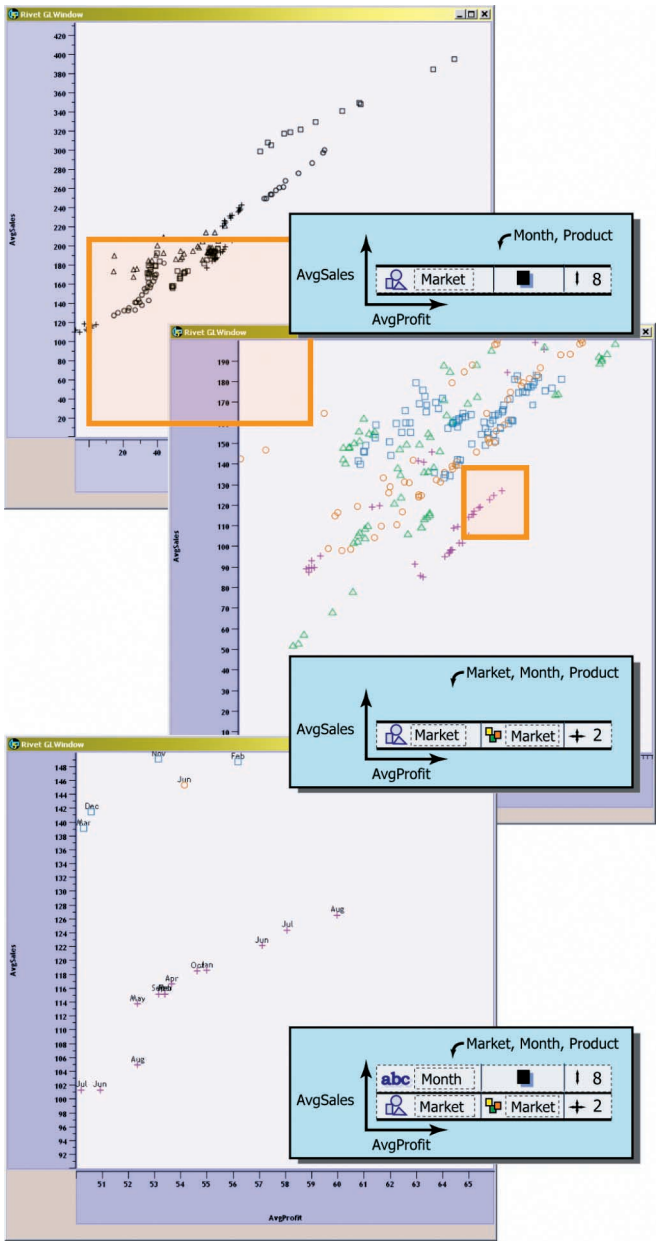


Fig. 8. A series of screenshots of a multiscale visualization of average sales versus average profit over a two-year period for a hypothetical coffeeshop chain. In the first visualization, each point represents profit and sales for a particular month and product, summed over all locations. In the next visualization, the user zooms, changing both 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) and the visual abstraction (points are now colored by market). As the user zooms in further, the visual abstraction changes as a layer is added to display more details: a text label is added to redundantly encode the market name.

words, for a given dimension, there are multiple ways to define the levels of aggregation. We can develop multiscale visualizations on these hierarchies by choosing a single path through the branching hierarchy (i.e., superimposing a star schema view on a subset of a schema, which is always possible) and then constructing a zoom graph. Although this method supports branching hierarchies, it restricts the zooms to always following a single linear path, chosen a priori, within the branching hierarchy. The key thing to note

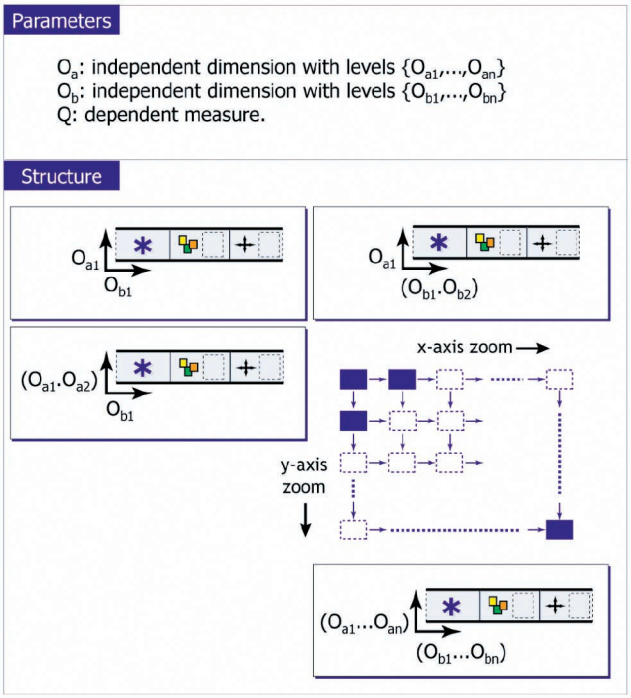


Fig. 9. Zoom graph for Pattern 4: Matrices.

is that it is not our formalism nor zoom graphs that restricts us to a single linear path, but rather the user interface for determining which branch to follow. A zoom graph can be arbitrarily sophisticated, allowing for different representations along the different branches of the hierarchies or even pivots to different dimensions. The main question is finding a user interface technique that enables the user to select an explicit branch to follow or even to change the hierarchy they are zooming on. Furthermore, zoom graphs can model zooming visualizations where the zooming is not tied to an axis. Developing more flexible zooms is one important area for future work.

Within the field of information visualization, several researchers have focused on developing visualizations for presenting and exploring multiple hierarchies. These multiple hierarchies are often called "polyarchies" or "multitrees." Conklin et al. presented a system [7] for exploring a simplified polyarchy scheme that can be modeled in data cubes and is addressable with our formalism. Robertson et al. have described more complex polyarchies [19]; however, these polyarchies are not a basis for aggregation. Instead, they reflect an organization among the nodes with each node representing a distinct data entity. With this type of polyarchy, the concern is not leveraging the hierarchies for abstraction, but rather communicating the multiple hierarchies and their intersections.

5.2 Design Patterns

Given the Polaris formalism and data cubes, it was relatively easy to construct a visualization of a particular level of detail in a hierarchical database [23]. It turns out to be much more complicated, however, to construct continuous zooms into the data. There are many ways to zoom, many of which are not very effective. We have described

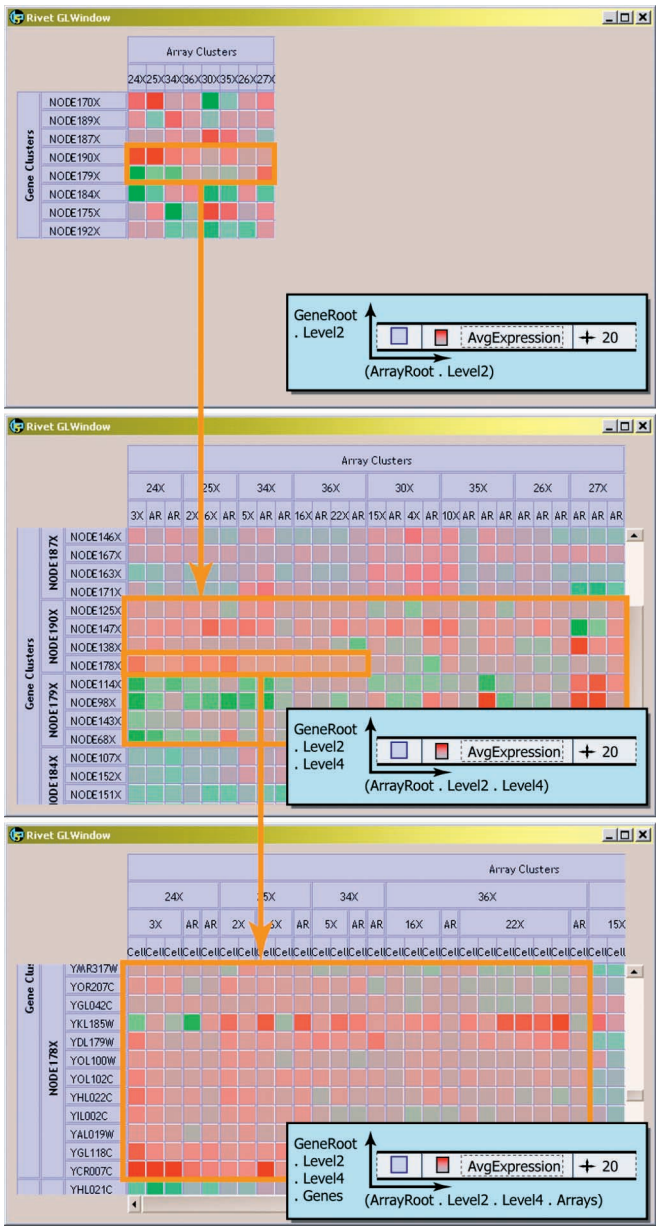


Fig. 10. A series of screenshots of a multiscale visualization of yeast microarray data developed using the Matrix pattern. The first visualization shows the highest level gene clusters on the y-axis, the microarray experiment clusters on the x-axis, and the average gene expression in each cell. In the 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.

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. Another example is the chart stack pattern in which we showed a version using line charts and a variation using a different chart type (strip charts and Gantt charts). Other chart types, such as histograms, would work equally well. Other possible extensions include embedding

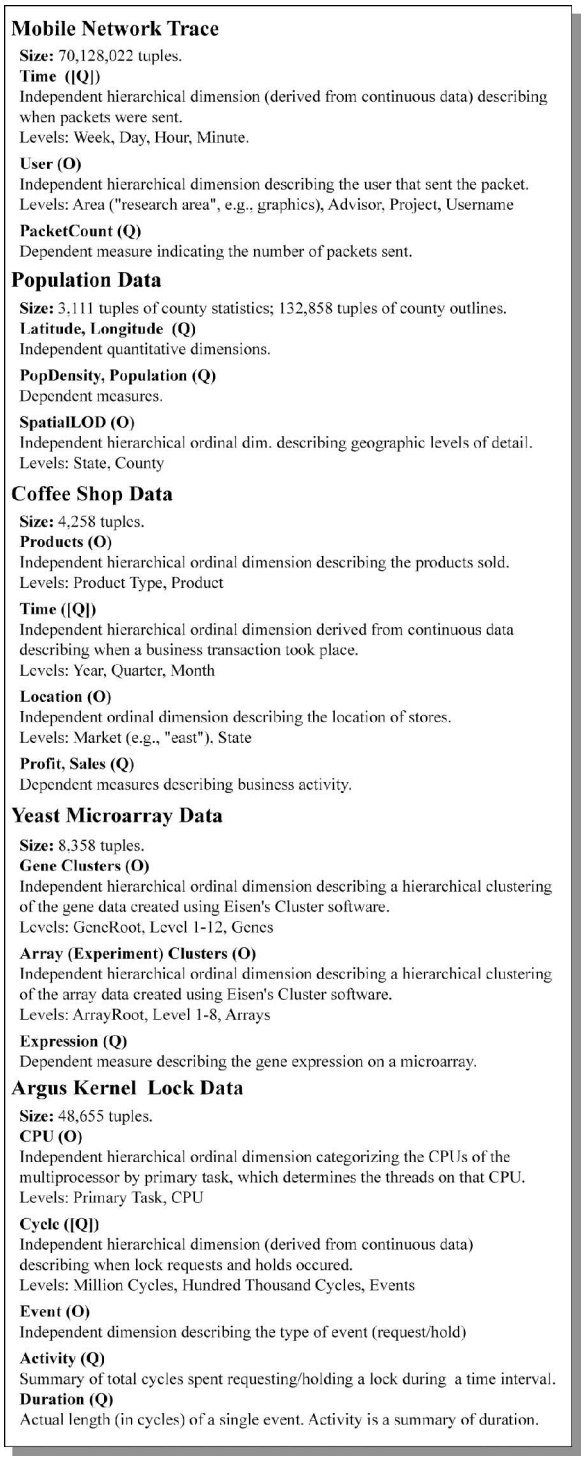


Fig. 11. The schemas for the example data sets.

one pattern within another. There are also completely different patterns that might also work.

Developing an extensive repository of design patterns is another promising direction for future work. One approach for finding new patterns is to use the taxonomy of graph types presented in the original Polaris paper [22]. That taxonomy, similar to Cleveland's [6], structured the space of graphics into three families according to the types of fields assigned to their axes; these families are decomposed further, both by analysis task and by whether the fields were independent or dependent variables. In the design patterns we present in this paper, we have used these types of parameters to describe the patterns; for example, both the Thematic Maps and the Dependent Quantitative-Dependent Quantitative Scatterplots pattern fall into the QQ family, differing primarily in the independence of the axis variables. Thus, one possibility is to use this structure in developing new design patterns since each unique combination of field types, analysis task, and independent/dependent variables is a likely source for at least one design pattern.

5.3 Transitions

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 nonuniform 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 data sets may be visualized in real time. In this paper, we have used small to moderate-sized data sets that vary in size from several thousand tuples (the coffee shop data) to seventy million tuples (the 25 network data). For these databases, the performance is acceptable, with most queries executing in less then one second. Query performance for a multiscale application depends on many factors, ranging from the visualization itself (e.g., the average number of tuples in any given display) to the database organization (e.g., the size of the database, the number of materialized views, and the number and type of indices). Larger data sets could be viewed at interactive rates if we optimized the data cube queries using a combination of prefetching and caching. Another possibility is to use data sampling techniques to choose what to draw [15] and importance metrics to determine in what order to draw the data [18].

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.

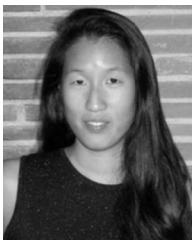
References

[1] J. Abello and J. Korn, "MGV: A System for Visualizing Massive Multidigraphs," *IEEE Trans. Visualization and Computer Graphics*, vol. 8, no. 1, pp. 21-38, Jan.-Mar. 2002.

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

[3] B. Bederson, J. Meyer, and L. Good, "Jazz: An Extensible Zoomable User Interface Graphics Toolkit in Java," *Proc. 13th Ann. ACM Symp. User Interface Software and Technology (UIST 2000),* vol. 2, no. 2, pp. 171-180, 2000.
[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," *Computer Graphics,* vol. 34, no. 1, Feb. 2000.
[6] W. Cleveland, *Visualizing Data.* Summit, N.J.: Hobart Press, 1993.
[7] N. Conklin, S. Prabhakar, and C. North, "Multiple Foci Drill-Down through Tuple and Attribute Aggregation Polyarchies in Tabular Data," *Proc. IEEE Symp. Information Visualization,* Oct. 2002.
[8] S. Eick and A. Karr, "Visual Scalability," *J. Computational and Graphical Statistics,* vol. 11, no. 1, pp. 22-43, Mar. 2002.
[9] M. Eisen, P. Spellman, P. Brown, and D. Botstein, "Cluster Analysis and Display of Genome-Wide Expression Patterns," *Proc Nat'l Academy of Science USA,* vol. 95, pp. 14863-14868, 1998.
[10] M. Eisen, "Cluster and Treeview," http://rana.lbl.gov, 1998.
[11] A. Frank and S. Timpf, "Multiple Representations for Cartographic Objects in a Multi-Scale tree—An Intelligent Graphical Zoom," *Computers and Graphics,* special issue: modeling and visualization of spatial data in geographical information systems, vol. 18, no. 6, pp. 823-830, 1995.
[12] Y. Fua, M. Ward, and E. Rundensteiner, "Structure-Based Brushes: A Mechanism for Navigating Hierarchically Organized Data and Information Spaces," *IEEE Trans. Visualization and Computer Graphics,* vol. 6, no 2, pp. 150-159, Apr.-June 2000.
[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software.* Reading, Mass.: Addison-Wesley, 1995.
[14] 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," *J. Data Mining and Knowledge Discovery,* vol. 1, no. 1, pp. 29-53, 1997.
[15] J. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth, and P. Haas, "Interactive Data Analysis: The CONTROL Project," *Computer,* pp. 51-59, Aug. 1999.
[16] R. Hightower, L. Ring, J. Helfman, B. Bederson, and J. Hollan, "Graphical Multiscale Web Histories: A Study of PadPrints," *Proc. ACM Conf. Hypertext,* pp. 58-65, 1998.
[17] H. Igehy, G. Stoll, and P. Hanrahan, "The Design of a Parallel Graphics Interface," *Proc. ACM SIGGRAPH 1998,* pp. 141-150, Aug. 1998.
[18] T. Munzner, "Drawing Large Graphs with H3Viewer and Site Manager," *Proc. Graph Drawing 1998,* pp. 384-393, Aug. 1998.
[19] G. Robertson, K. Cameron, M. Czerwinski, and D. Robbins, "Polyarchy Visualization: Visualizing Multiple Intersecting Polyarchies," *Proc. ACM SIGCHI 2002,* pp. 423-430, Apr. 2002.
[20] E. Rundensteiner, M. Ward, J. Yang, and P. Doshi, "XmdvTool: Visual Interactive Data Exploration and Trend Discovery of High-Dimensional Data Sets," *Proc. ACM SIGMOD 2002,* June 2002.
[21] http://www.cs.umd.edu/hcil/research/visualization.shtml, 2002.
[22] C. Stolte, D. Tang, and P. Hanrahan, "Polaris: A System for Query, Analysis, and Visualization of Multi-Dimensional Relational Databases," *IEEE Trans. Visualization and Computer Graphics,* vol. 8, no. 1, pp. 52-65, Jan.-Mar. 2002.
[23] C. Stolte, D. Tang, and P. Hanrahan, "Query, Analysis, and Visualization of Hierarchically Structured Data Using Polaris," *Proc. ACM SIGKDD 2002,* July 2002.
[24] D. Tang, "Analyzing Wireless Networks," PhD dissertation, Oct. 2000.
[25] D. Tang and M. Baker, "Analysis of a Local-Area Wireless Network," *Proc. Sixth Int'l Conf. Mobile Computing and Networking,* pp. 1-10, Aug. 2000.
[26] S. Timpf, "Map Cube Model—A Model for Multi-Scale Data," *Proc. Eighth Int'l Symp. Spatial Data Handling,* pp. 190-201, 1998.
[27] F. Topfer and W. Pillewizer, "The Principles of Selection, a Means of Cartographic Generalization," *Cartographic J.,* vol. 3, no. 1, pp. 10-16, 1966.
[28] E. Tufte, *The Visual Display of Quantitative Information.* Cheshire, Conn.: Graphics Press, 1983.
[29] 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. Visual Languages and Computing,* special issue on visual languages for end-user and domain-specific programming, vol. 12, no. 5, pp. 551-571, Oct. 2001.

Chris Stolte received the BSc degree in computer science in 1996 from Simon Fraser University. He is currently a PhD student in the Department of Computer Science at Stanford University. His research interests include information visualization and computer graphics. His current research projects include the Rivet project for studying computer systems and the Polaris project on visual interfaces for the exploration and analysis of large relational databases.



Diane Tang received the AB degree in computer science in 1995 from Harvard/Radcliffe University and the MS and PhD degrees in computer science in 2000 from Stanford University. She is a research associate in the Department of Computer Science at Stanford University. Her interests include information visualization, especially with regard to level-of-detail issues, mobile networking, and distributed systems. She is currently working on the Rivet and Polaris projects on interactive visual exploration of large data sets. She is a recipient of the NPSC Fellowship.

Pat Hanrahan is the Canon USA Professor of Computer Science and Electrical Engineering at Stanford University, where he teaches computer graphics. His current research involves visualization, image synthesis, and graphic systems and architectures. Before joining Stanford, he was a faculty member at Princeton University. He has also worked at Pixar, where he developed volume rendering software and was the chief architect of the RenderMan Interface—a protocol that allows modeling programs to describe scenes to high quality rendering programs. Previous to Pixar, he directed the 3D Computer Graphics Group in the Computer Graphics Laboratory at the New York Institute of Technology. Professor Hanrahan has received three university teaching awards. In 1993, he received an Academy Award for Science and Technology, the Spirit of America Creativity Award, and the SIGGRAPH Computer Graphics Achievement Award; he was recently elected to the National Academy of Engineering.