Polaris: A System for Query, Analysis and Visualization of Multi-dimensional Relational Databases

Chris Stolte and Pat Hanrahan
Computer Science Department
Stanford University

Abstract

In the last several years, large multi-dimensional databases have become common in a variety of applications such as data warehousing and scientific computing. Analysis and exploration tasks place significant demands on the interfaces to these databases. Because of the size of the data sets, dense graphical representations are more effective for exploration than spreadsheets and charts. Furthermore, because of the exploratory nature of the analysis, it must be possible for the analysts to change visualizations rapidly as they pursue a cycle involving first hypothesis and then experimentation.

In this paper we present Polaris, an interface for exploring large multi-dimensional databases that extends the well-known Pivot Table interface. The novel features of Polaris include an interface for constructing visual specifications of table-based graphical displays and the ability to generate a precise set of relational queries from the visual specifications. The visual specifications can be rapidly and incrementally developed, giving the analyst visual feedback as they construct complex queries and visualizations.

1. Introduction

In the last several years, large databases have become common in a variety of applications. Corporations are creating large data warehouses of historical data on key aspects of their operations. International research projects, such as the Human Genome Project [15] and Digital Sky Survey [23], are generating massive databases of scientific data.

A major challenge with these databases is to extract meaning from the data they contain: to discover structure, find patterns, and derive causal relationships. The analysis and exploration necessary to uncover this hidden information places significant demands on the human-computer interfaces to these databases. The exploratory analysis process is one of hypothesis, experiment and discovery. The path of exploration is unpredictable, and the analysts or scientists need to be able to rapidly change both what data they are viewing and how they are viewing that data.

The current trend is to treat multi-dimensional databases as n-dimensional data cubes [13]. Each dimension in these data cubes corresponds to one dimension in the relational schema. Perhaps the most popular interface to multi-dimensional databases is the Pivot Table [11]. Pivot Tables allow the data cube to be rotated, or pivoted, so that different dimensions of the dataset may be encoded as rows or columns of the table. The remaining dimensions are aggregated and displayed as numbers in the cells of the table. Cross-tabulations and summaries are then added to the resulting table of numbers. Finally, graphs may be generated from the resulting tables. Visual Insights recently released a new interface for visually exploring projections of data cubes using linked views of bar charts, scatterplots, and parallel coordinate displays [10].

In this paper we present Polaris, an interface for the exploration of multi-dimensional databases that extends the Pivot Table interface to directly generate a rich, expressive set of graphical displays. Polaris builds tables using an algebraic formalism involving the fields of the database. Each table consists of layers and panes, and each pane may be a different graphic. The use of tables to organize multiple graphs on a display is a technique often used by statisticians in their analysis of data [3][7][28].

The Polaris interface is simple and expressive because it is built upon a formalism for constructing graphs and building data transformations. We interpret the state of the interface as a visual specification of the analysis task and automatically compile it into data and graphical transformations. This allows us to combine statistical analysis and visualization. Furthermore, all intermediate specifications that can be created in the visual language are valid and can be interpreted to create visualizations. Therefore, analysts can incrementally construct complex queries, receiving visual feedback as they assemble and alter the specifications.

2. Related Work

The related work to Polaris can be divided into three categories: formal graphical specifications, table-based data displays and database exploration tools.

2.1. Formal Graphical Specifications

Bertin's *Semiology of Graphics* [4] is one of the earliest attempts at formalizing graphing techniques. Bertin developed a vocabulary for describing data and the techniques for encoding data in a graphic. One of his important contributions is the identification of the retinal variables (position, color, size, etc.) in which data can be encoded. Cleveland [7][8] used theoretical and experimental results to determine how well people can use these different retinal properties to compare quantitative variations.

Mackinlay's APT system [18] is one of the first applications of formal graphical specifications to computer-generated displays. APT uses a set of graphical languages and composition rules to automatically generate 2D displays of relational data. The Sage system [21] extends the concepts of APT, providing a richer set of data characterizations and generating a wider range of displays.

Livny et al. [17] describe a visualization model that provides a foundation for database-style processing of visual queries. Within this model, the relational queries and graphical mappings necessary to generate visualizations are defined by a set of relational operators. The Rivet visualization environment [6] applies similar concepts to provide a flexible database visualization tool.

Wilkinson [29] recently developed a comprehensive language for describing traditional statistical graphs and proposed a simple interface for generating a subset of the specifications expressible within his language. We have extended Wilkinson's ideas to develop a specification that can be directly mapped to an interactive interface and that is tightly integrated with the relational data model. The differences between our work and Wilkinson's will be further discussed in Section 7.

2.2. Table-based Displays

Another area of related work is visualization systems that use table-based displays. Static table displays, such as scatterplot matrices [14] and Trellis [2] displays, have been used extensively in statistical data analysis. Recently, several interactive table displays have been developed. Pivot Tables [11] allow analysts to explore different projections of large multi-dimensional datasets by interactively specifying assignments of fields to the table axes, but are limited to text-based displays. Systems such as the Table Lens [19] and FOCUS [24] visualization system provide table displays that present data in a relational table view, using simple graphics in the cells to communicate quantitative values.

2.3. Database Exploration Tools

The final area of related work is visual query and database exploration tools. Projects such as VQE [9], Visage [22], DEVise [17], and Tioga-2 [1] have focused on developing visualization environments that directly support interactive database exploration through *visual queries*. Users can construct queries and visualizations directly through their interactions with the visualization system interface. These systems have flexible mechanisms for mapping query results to graphs, and all of the systems support mapping database records to retinal properties of the marks in the graphs. However, none of these systems leverages table-based organizations of their visualizations.

3. Overview

Polaris has been designed to support the interactive exploration of large multi-dimensional relational databases. Relational databases organize data into tables where each row in a table corresponds to a basic entity or fact and each column represents a property of that entity [26]. We refer to a row in a relational table as a *tuple* or *record*, and a column in the table as a *field*. A single relational database will contain many heterogeneous but interrelated tables.

We can characterize fields in a database as nominal, ordinal or quantitative [4][25]. Polaris reduces this categorization to ordinal and quantitative by assigning an ordering to the nominal fields and subsequently treating them as ordinal.

The fields within a relational table can also be partitioned into two types: dimensions and measures. Dimensions and measures are similar to independent and dependent variables in traditional analysis. For example, a product name or type would be a dimension of product, and the product price or size would be a measure. The current implementation of Polaris treats all nominal fields as dimensions and all quantitative fields as measures.

In many important business and scientific databases there are often many dimensions identifying a single entity. For example, a transaction within a store may be identified by the time of the sale, the location of the store, the type of product, and the customer. In most data warehouses, these multidimensional databases are structured as n-dimensional data cubes [26]. Each dimension in the data cube corresponds to one dimension in the relational schema.

To effectively support the analysis process in large multidimensional databases, an analysis tool must meet several demands:

- **Data-dense displays:** The databases typically contain a large number of records and dimensions. Analysts need to be able to create visualizations that will simultaneously display many dimensions of large subsets of the data.
- **Multiple display types:** Analysis consists of many different tasks such as discovering correlations between variables, finding patterns in the data, locating outliers and uncovering structure. An analysis tool must be able to generate displays suited to each of these tasks.
- **Exploratory interface:** The analysis process is often an unpredictable exploration of the data. Analysts must be able to rapidly change what data they are viewing and how they are viewing that data.

Polaris addresses these demands by providing an interface for rapidly and incrementally generating table-based displays. In Polaris, a table consists of a number of rows, columns, and layers. Each table axis may contain multiple nested dimensions. Each table entry, or *pane*, contains a set of records that are visually encoded as a set of marks to create a graphic.

Several characteristics of tables make them particularly effective for displaying multi-dimensional data:

- **Multivariate:** Multiple dimensions of the data can be explicitly encoded in the structure of the table, enabling the display of high-dimensional data.
- **Comparative:** Tables generate *small-multiple* displays of information, which, as Tufte [28] explains, are easily compared, exposing patterns and trends across dimensions of the data.
- **Familiar:** Table-based displays have an extensive history. Statisticians are accustomed to using tabular displays of graphs, such as scatterplot matrices and Trellis displays, for analysis. Pivot Tables are a common interface to large data warehouses.

Figure 1 shows the user interface presented by Polaris. In this example, the analyst has constructed a matrix of scatterplots showing sales versus profit for different product types in different quarters. The primary interaction technique is to drag-and-drop fields from the database schema onto shelves throughout the display. We call a given configuration of fields on shelves a *visual specification*. The specification determines the analysis and visualization operations to be performed by the system, defining:

- The mapping of data sources to layers. Multiple data sources may be combined in a single Polaris visualization. Each data source maps to a separate layer or set of layers.
- The number of rows, columns, and layers in the table and their relative orders (left to right as well as back to front). The database dimensions assigned to rows are specified by the fields on the *x* shelf, columns by fields on the *y* shelf, and layers by fields on the *layer* (z) shelf. Multiple fields may be dragged onto each shelf to show categorical relationships.
- The selection of records from the database and the partitioning of records into different layers and panes.
- The grouping of data within a pane and the computation of statistical properties and aggregates. Records may also be sorted into a given drawing order.

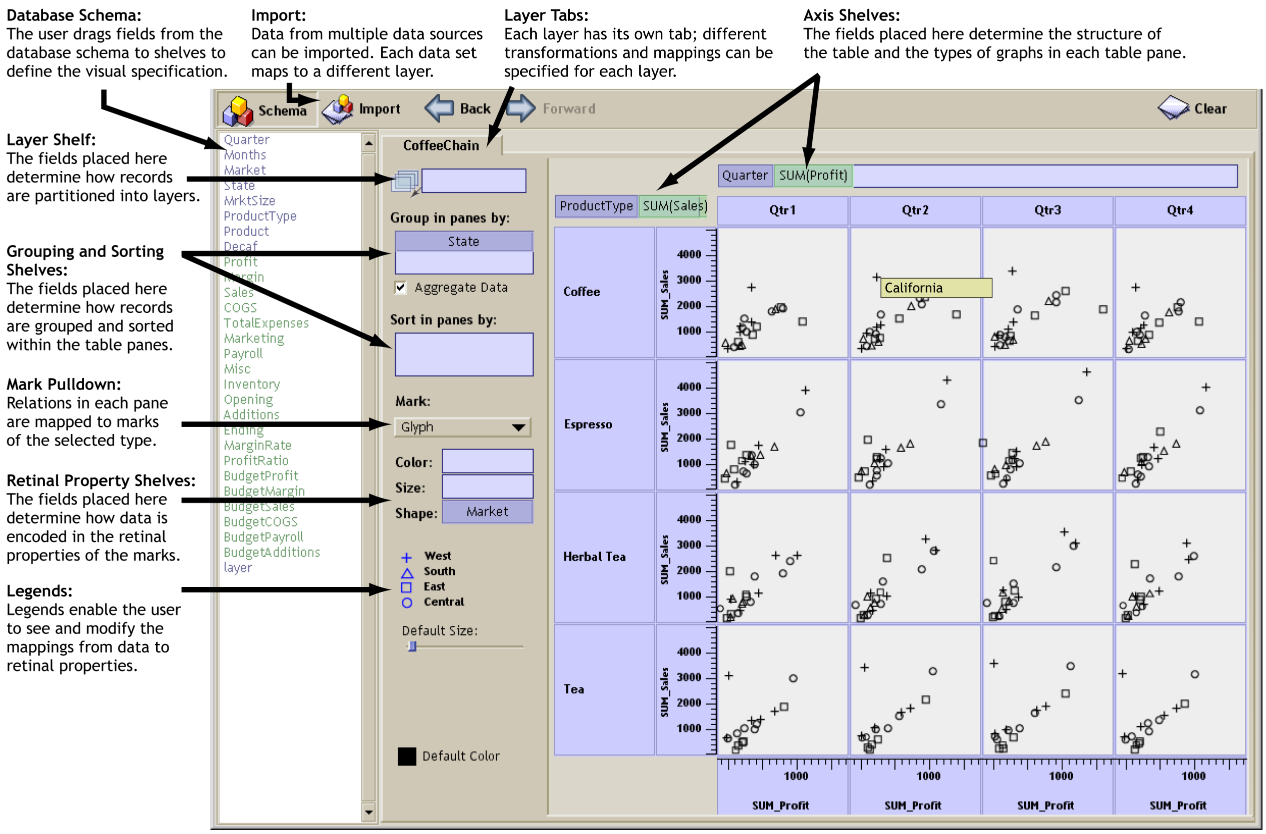

Figure 1: *The Polaris user interface. Analysts construct table-based displays of relational data by dragging fields from the database schema onto shelves throughout the display. A given configuration of fields on shelves is called a visual specification. The specification unambiguously defines the analysis and visualization operations to be performed by the system to generate the display.*

- The type of graphic displayed in each pane of the table. Each graphic consists of a set of marks, one mark per record in that pane.
- The mapping of data fields to retinal properties of the marks in the graphics. The mappings used for any given visualization are shown in a set of automatically generated legends.

Analysts can interact with the resulting visualizations in several ways. Selecting a single mark in a graphic by clicking on it pops up a window displaying user-specified field values for the tuple corresponding to that mark. Analysts can draw rubberbands around a set of marks to brush records. Brushing can be performed within a single table or between multiple Polaris displays.

In the next section, we describe how the visual specification is used to generate graphics. In the following section, we describe how it is used to generate the database queries and to perform statistical analysis.

4. Generating Graphics

The visual specification consists of three components: (a) the specification of the different table configurations, (b) the type of graphic inside each pane, and (c) the details of the visual encodings. We discuss each of these in turn.

4.1. Table Algebra

We need a formal mechanism to specify table configurations, and we have defined an algebra for this purpose. When the analysts place fields on the axis shelves, as shown in Figure 1, they are implicitly creating expressions in this algebra.

A complete table configuration consists of three separate expressions in this table algebra. Two of the expressions define the configuration of the *x* and *y* axes of the table, partitioning the table into rows and columns. The third expression defines the *z* axis of the table, which partitions the display into *layers*.

The operands in this table algebra are the names of the ordinal and quantitative fields of the database. We will use *A*, *B*, and *C* to represent ordinal fields and *P*, *Q*, and *R* to represent quantitative fields. We assign ordered sets to each field symbol in the following manner: to ordinal fields we assign the members of the ordered domain of the field, and to quantitative fields we assign the single element set containing the field name.

$$A = domain(A) = \{a_1, \ldots, a_n\}$$
$$P = \{P\}$$

This assignment of sets to symbols reflects the difference in how the two types of fields will be encoded in the structure of the tables. Ordinal fields will partition the table into rows and columns, whereas quantitative fields will be spatially encoded as axes within the panes.

A valid expression in our algebra is an ordered sequence of one or more symbols with operators between each pair of adjacent symbols, and with parentheses used to alter the precedence of the operators. The operators in the algebra are cross (×), nest (/) and concatenation (+), listed in order of precedence. The precise se-

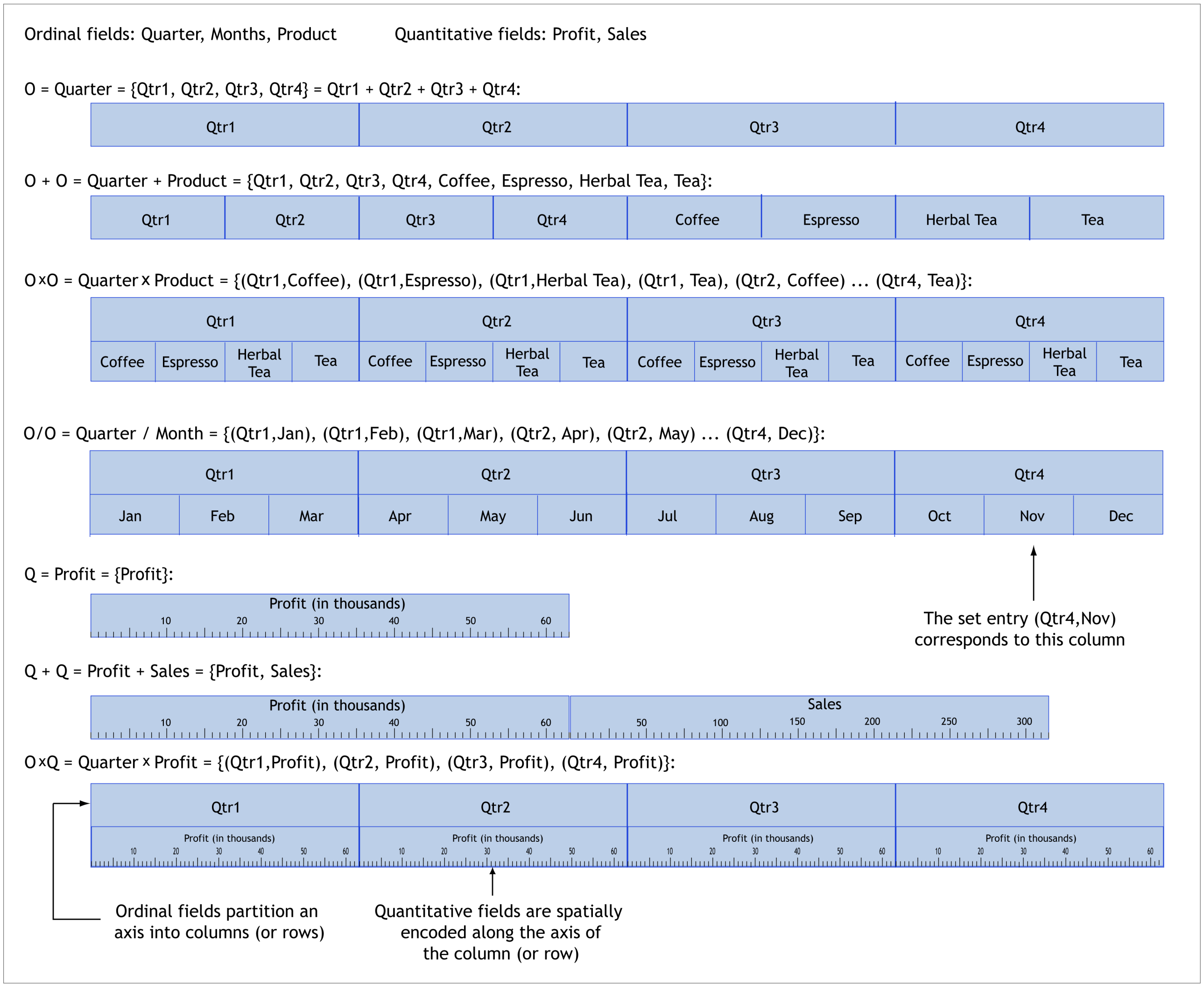

Figure 2: *The graphical interpretation of several expressions in the table algebra. Each expression in the table algebra can be reduced to a single set of terms, and that set can then be directly mapped into a configuration for an axis of the table.*

mantics of each operator is defined in terms of its effects on the operand sets.

Concatenation

The concatenation operator performs an ordered union of the sets of the two symbols:

$$A+B = \{a_1, \dots, a_n\} + \{b_1, \dots, b_m\} = \{a_1, \dots, a_n, b_1, \dots, b_m\}$$

$$A+P = \{a_1, \dots, a_n\} + \{P\} = \{a_1, \dots, a_n, P\}$$

$$P+Q = \{P\} + \{Q\} = \{P, Q\}$$

Cross

The cross operator performs a Cartesian product of the sets of the two symbols:

$$A \times B = \{a_1, \dots, a_n\} \times \{b_1, \dots, b_m\} = \{a_1b_1, \dots, a_1b_m, a_2b_1, \dots, a_2b_m, \dots, a_nb_1, \dots, a_nb_m\}$$

$$A \times P = \{a_1, \dots, a_n\} \times P = \{a_1P, \dots, a_nP\}$$

Nest

The nest operator is similar to the cross operator, but it only creates set entries for which there exist records with those domain values. If we define *R* to be the dataset being analyzed, *r* to be a record, and *A(r)* to be the value of the field *A* for the record *r*, then we can define the nest operator as follows:

$$A/B = \{a_ib_j \mid \exists r \in R \text{ st } A(r)=a_i \;\&\; B(r)=b_j\}$$

The intuitive interpretation of the *nest* operator is “B within A”. For example, given the fields *quarter* and *month*, the expression *quarter / month* would be interpreted as those months within each quarter, resulting in three entries for each quarter. In contrast, *quarter* × *month* would result in 12 entries for each quarter.

Using the above set semantics for each operator, every expression in the algebra can be reduced to a single set, with each entry in the set being an ordered concatenation of zero or more ordinal values with zero or more quantitative field names. We call this set evaluation of an expression the *normalized set form*. The normalized set form of an expression determines one axis of the table: the table axis is partitioned into columns (or rows or layers) so that there is a one-to-one correspondence between set entries in the normalized set and columns. Figure 2 illustrates the configurations resulting from a number of expressions.

Analysts can also combine multiple data sources in a single Polaris visualization. When multiple data sources are imported, each data source is mapped to a distinct layer (or set of layers). While all data sources and all layers share the same configuration for the *x* and *y* axes of the table, each data source can have a different expression for partitioning its data into layers.

4.2. Types of Graphics

After the table configuration is specified, the next step is to specify the type of graphic in each pane. One option, typical of most charting and reporting tools, is to have the user select a chart type from a predefined set of charts. Polaris allows analysts to flexibly construct graphics by specifying the individual components of the graphics. However, for this approach to be effective, the specification must balance flexibility with succinctness. We have developed a taxonomy of graphics that results in an intuitive and concise specification of graphic types.

When specifying the table configuration, the user also implicitly specifies the axes associated with each pane. We have structured the space of graphics into three families by the type of fields assigned to their axes:

- Ordinal-Ordinal
- Ordinal-Quantitative
- Quantitative-Quantitative

Each family contains a number of variants depending on how records are mapped to marks. For example, selecting a bar in an ordinal-quantitative pane will result in a bar chart, whereas selecting a line mark results in a line chart. The mark set currently supported in Polaris includes the rectangle, circle, glyph, text, Gantt bar, line, polygon and image.

Following Cleveland [8], we further structure the space of graphics by the number of independent and dependent variables. For example, a graphic where both axes encode independent variables is different than a graphic where one axis encodes an independent variable and the other encodes a dependent variable ($y=f(x)$). By default, dimensions of the database are interpreted as independent variables and measures as dependent variables.

Finally, the precise form of the data transformations, in particular how records are grouped and whether aggregates are formed, can affect the type of graphic. Some graphic types best encode a single record, whereas others can encode an arbitrary number of records.

We briefly discuss the defining characteristics of the three families within our categorization.

Ordinal-Ordinal Graphics

The characteristic member of this family is the table, either of numbers or of marks encoding attributes of the source records.

In ordinal-ordinal graphics, the axis variables are typically independent of each other, and the task is focused on understanding patterns and trends in some function $f(O_x,O_y) \rightarrow R$, where R represents the fields encoded in the retinal properties of the marks. This can be seen in Figure 3(a) where the analyst is studying sales and margin as a function of product type, month and state for the items sold by a hypothetical coffee chain. Figure 3(b) presents another example of an ordinal-ordinal graphic. In this figure, the analyst is investigating the performance of a graphics rendering library. The number of cache misses attributable to each line of source code have been plotted as a function of the ordinal variables line number and file name.

The cardinality of the record set in each pane has little effect on the overall structure of the table. When there is more than one record per pane, multiple marks are shown in each display, with a one-to-one correspondence of mark to record. The marks are stacked in a specified drawing order, and the spatial placement of the marks within the pane conveys no additional information about the record's data.

Ordinal-Quantitative Graphics

Well-known representatives of this family of graphics are the bar chart, possibly clustered or stacked, the dot plot and the Gantt chart.

In an ordinal-quantitative graphic, the quantitative variable is often dependent on the ordinal variable, and the analyst is trying to understand or compare the properties of some set of functions $f(O) \rightarrow Q$. Figure 6(c) illustrates a case where a matrix of bar charts is used to study several functions of the independent variables product and month. The cardinality of the record set does affect the structure of the graphics in this family. When the cardinality of the record set is one, the graphics are simple bar charts or dot plots. When the cardinality is greater than one, additional structure may be introduced to accommodate the additional records (e.g., a stacked bar chart).

The ordinal and quantitative values may be independent variables, such as in a Gantt chart. Here, each pane represents all the events in a category; each event has a type as well as a begin and end time. In Figure 3(c), major wars over the last five hundred years are displayed as Gantt charts, categorized by country. An additional layer in that figure displays pictures of major scientists plotted as a function of the independent variables country of birth and date of birth. Figure 3(d) shows a table of Gantt charts, with each Gantt chart displaying the thread scheduling and locking activity on a CPU within a multiprocessor computer.

Quantitative-Quantitative Graphics

Graphics of this type are used to understand the distribution of data as a function of one or both quantitative variables and to discover causal relationships between the two quantitative variables. Figure 6(a) illustrates a matrix of scatterplot graphics used to understand the relationships between a number of attributes of different products sold by a coffee chain.

Figure 3(e) illustrates a another example of a quantitative-quantitative graphic: the map. In this figure, the analyst is studying how flight scheduling varies with the region of the country the flight originated in. Data about a number of flights between major airports has been plotted as a function of latitude and longitude, and composed with two other layers which plot the location of airports and display the geography of each state as a polygon.

It is extremely rare to use this type of graph with a cardinality of one, not because it is not meaningful, but because the density of information in such a graphic is very low.

4.3. Visual Mappings

Each record in a pane is mapped to a mark. There are two components to the visual mapping. The first component, described

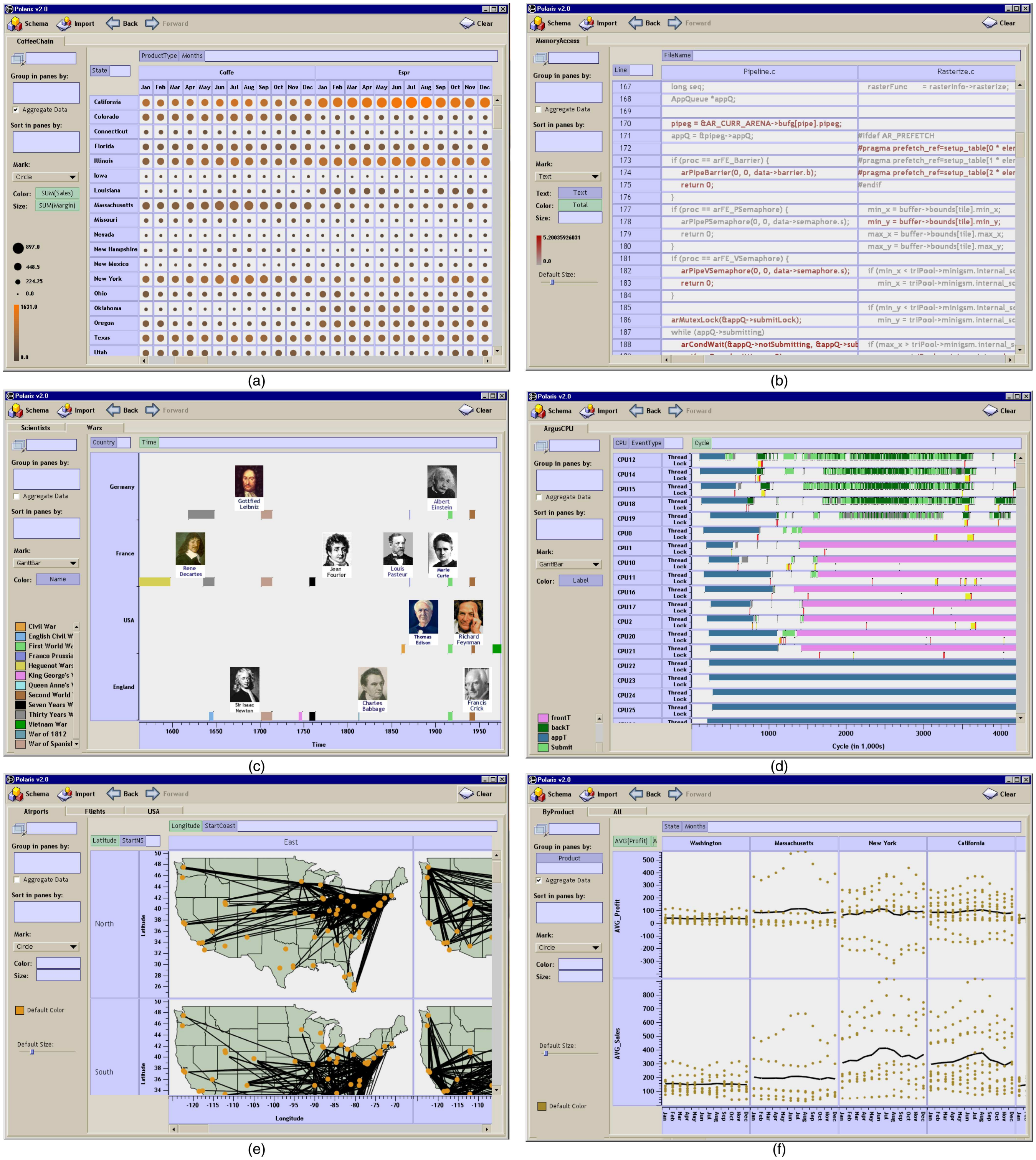

Figure 3: *Examples of the different graph types that can be constructed in Polaris. The graphical table (a), showing sales and margin as a function of product type and state for a hypothetical coffee chain, is an example of the ordinal-ordinal family of graphics. The source code display (b), which is displaying the source code for a graphics rendering library with the code colored by the number of cache misses attributable to each line of code, is another example of the ordinal-ordinal family. Gantt charts, used in (c) to display major wars in several countries over the last five hundred years, and in (d) to display the thread scheduling for a graphics application, are an example of the ordinal-quantitative family. Maps, as in (e) which shows flights between several major airports categorized by the region of the country they departed from, are an example of the quantitative-quantitative family of graphics. Line charts, as in (f) which shows average sales and profit as a function of time for the coffee chain, are another example of the quantitative-quantitative family.*

property	marks	ordinal/nominal mapping	quantitative mapping
shape	glyph	○ □ + △ S U	
size	rectangle, circle, glyph, text		
orientation	rectangle, line, text		
color	rectangle, circle, line, glyph, y-bar, x-bar, text, gantt bar	...	min max

Figure 4: *The different retinal properties that can be used to encode fields of the data and examples of the default mappings that are generated when a given type of data field is encoded in each of the retinal properties.*

in the previous section, determines the type of mark. The second component encodes fields of the records into visual or retinal properties of the selected mark. The visual properties in Polaris are based on Bertin's retinal variables [4]: shape, size, orientation, color (value and hue), and texture (not supported in the current version of Polaris).

Allowing analysts to explicitly encode different fields of the data to retinal properties of the display greatly enhances the data density and the variety of displays that can be generated. However, in order to keep the specification succinct, analysts should not be required to construct the mappings. Instead, they should be able to simply specify that a field be encoded as a visual property. The system should then generate an effective mapping from the domain of the field to the range of the visual property. These mappings are generated in a manner similar to other visualization systems. We discuss how this is done for the purpose of completeness. The default mappings are illustrated in Figure 4.

Shape

Polaris uses the set of shapes recommended by Cleveland for encoding ordinal data [7]**.** We have extended this set of shapes to include several additional shapes to allow a larger domain of values to be encoded.

Size

Size can be used to encode either an ordinal or quantitative field. When encoding a quantitative domain as size, a linear map from the field domain to the area of the mark is created. The minimum size is chosen so that all visual properties of a mark with the minimum size can be perceived [16]. If an ordinal field is encoded as size, the domain needs to be small, at most four or five values, so that the analyst can discriminate between different categories [4].

Orientation

A key principle in generating mappings of ordinal fields to orientation is that the orientation needs to vary by at least 30° between categories [16], thus constraining the automatically-generated mapping to a domain of at most six categories. For quantitative fields, the orientation varies linearly with the domain of the field.

Color

When encoding an ordinal domain, we use a predefined palette to select the color for each domain entry. The colors in the palette are well separated in the color spectrum, predominantly on hue [27]. We have ordered the colors to avoid adjacent colors with different brightness or substantially different wavelengths in an attempt to include harmonious sets of colors in each palette [4][16][27]. We additionally reserve a saturated red for highlighting items that have been selected or brushed.

When encoding a quantitative variable, it is important to vary only one psychophysical variable, such as hue or value. The default palette we use for encoding quantitative data is the isomorphic colormap developed by Rogowitz [20].

5. Generating Database Queries

Obviously, the visual specifications determine the final visualization. However, just as importantly, the visual specification generates queries to the database that (a) select subsets of the data for analysis, then (b) filter, sort and group the results into panes, and then finally (c) group, sort and aggregate the data before passing it to the graphics encoding process.

Figure 5 shows the overall data flow in Polaris. We can precisely describe the transformations in each of the three phases using SQL queries.

Step 1: Selecting the Records

The first phase of the data flow retrieves records from the database, applying user-defined filters to select subsets of the database.

For an ordinal field *A*, the user may specify a subset of the domain of the field as valid. If *filter(A)* is the user-selected subset, then a relational predicate expressing the filter for *A* is:

```
A in filter(A)
```

For a quantitative field *P*, the user may define a subset of the field's domain as valid. If *min(P)* and *max(P)* are the user-defined extents of this subset, then a relational predicate expressing the filter for *P* is:

```
(P ≥ min(P) and P ≤ max(P))
```

We can define the relational predicate *filters* as the conjunction of all of the individual field filters. Then, the first stage of the data transformation network is equivalent to the SQL statement:

```
SELECT *
WHERE {filters}
```

Step 2: Partitioning the records into panes

The second phase of the data flow partitions the retrieved records into groups corresponding to each pane in the table. As we discussed in Section 4.1, the *normalized set form* of the table axis expressions determines the table configuration. The table is partitioned into rows, columns, and layers corresponding to the entries in these sets.

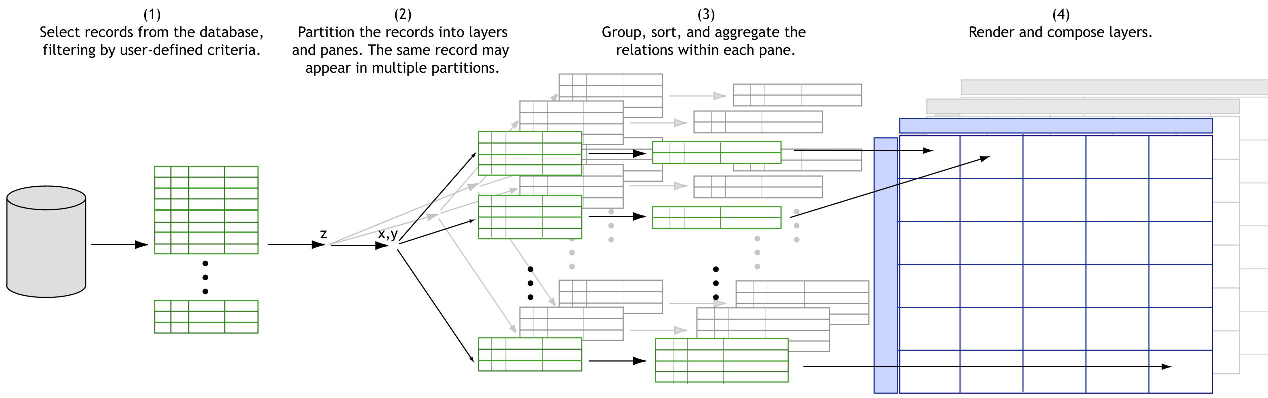

Figure 5: *The transformations and data flow within Polaris. The visual specification generates queries to the database to select subsets of the data for analysis, then to filter, sort, and group the results into panes, and then finally to group, sort and aggregate the data within panes.*

The ordinal values in each set entry define the criteria by which records will be sorted into each row, column and layer. Let *Row(i)* be the predicate that represents the selection criteria for the i^{th} row, *Column(j)* be the predicate for the j^{th} column, and *Layer(k)* the predicate for the k^{th} layer. For example, if the *y*-axis of the table is defined by the normalized set:

$$\{a_1b_1P,\ a_1b_2P,\ a_2b_1P,\ a_2b_2P\}$$

then there are four rows in the table, each defined by an entry in this set, and *Row* would be defined as:

$$\begin{aligned}
\mathrm{Row}(1) &= (A = a_1 \text{ and } B = b_1)\\
\mathrm{Row}(2) &= (A = a_1 \text{ and } B = b_2)\\
\mathrm{Row}(3) &= (A = a_2 \text{ and } B = b_1)\\
\mathrm{Row}(4) &= (A = a_2 \text{ and } B = b_2)
\end{aligned}$$

Given these definitions, the records to be partitioned into the pane at the intersection of the i^{th} row, the j^{th} column, and the k^{th} layer can be retrieved with the following query:

```
SELECT *
WHERE {Row(i) and Column(j) and
       Layer(k)}
```

To generate the groups of records corresponding to each of the panes, we must iterate over the table executing this SELECT statement for each pane. There is no standard SQL statement which enables us to perform this partitioning in a single query. We note that this is a same problem that motivated the CUBE [13] operator; we will revisit this issue in the discussion section.

Step 3: Transforming Records within the Panes

The last phase of the data flow is the transformation of the records in each pane. If the visual specification includes aggregation, then each measure in the database schema must be assigned an aggregation operator. If the user has not selected an aggregation operator for a measure, that measure is assigned the default aggregation operator (SUM). We define the term *aggregates* as the list of the aggregations that need to be computed. For example, if the database contains the quantitative fields Profit, Sales and Payroll, and the user has explicitly specified that the average of Sales should be computed, then *aggregates* is defined as:

```
aggregates =
SUM(Profit),AVG(Sales),SUM(Payroll)
```

Aggregate field filters (for example, SUM(Profit) > 500) could not be evaluated in Step 1 with all of the other filters because the aggregates had not yet been computed. Thus, those filters must be applied in this phase. We define the relational predicate *filters* as in Step 1 for unaggregated fields.

Additionally, we define the following lists:

G : the field names in the grouping shelf,

S : the field names in the sorting shelf, and

dim : the dimensions in the database.

The necessary transformation can then be expressed by the SQL statement:

```
SELECT {dim},{aggregates}
GROUP BY {G}
HAVING {filters}
ORDER BY {S}
```

If no aggregate fields are included in the visual specification, then the remaining transformation simply sorts the records into drawing order:

```
SELECT *
ORDER BY {S}
```

6. Results

Polaris is useful for performing the type of exploratory data analysis advocated by statisticians such as Bertin [3] and Cleveland [8]. We demonstrate the capabilities of Polaris as an exploratory interface to multi-dimensional databases by considering the following scenario.

The chief financial officer (CFO) of a national coffee store chain has just been told to cut expenses. To get an initial understanding of the situation, the CFO creates a table of scatterplots showing the relationship between marketing costs and profit categorized by product type and market (Figure 6(a)). After studying the graphics, the CFO notices an interesting trend: certain products have high marketing costs with little or no return in the form of profit.

To further investigate, the CFO creates two linked displays: a table of scatterplots showing profit and marketing costs for each state and a text table itemizing profit by product and state (Figure 6(b)). The two views are linked by the state field: if records are selected in either display, then all records with the same state

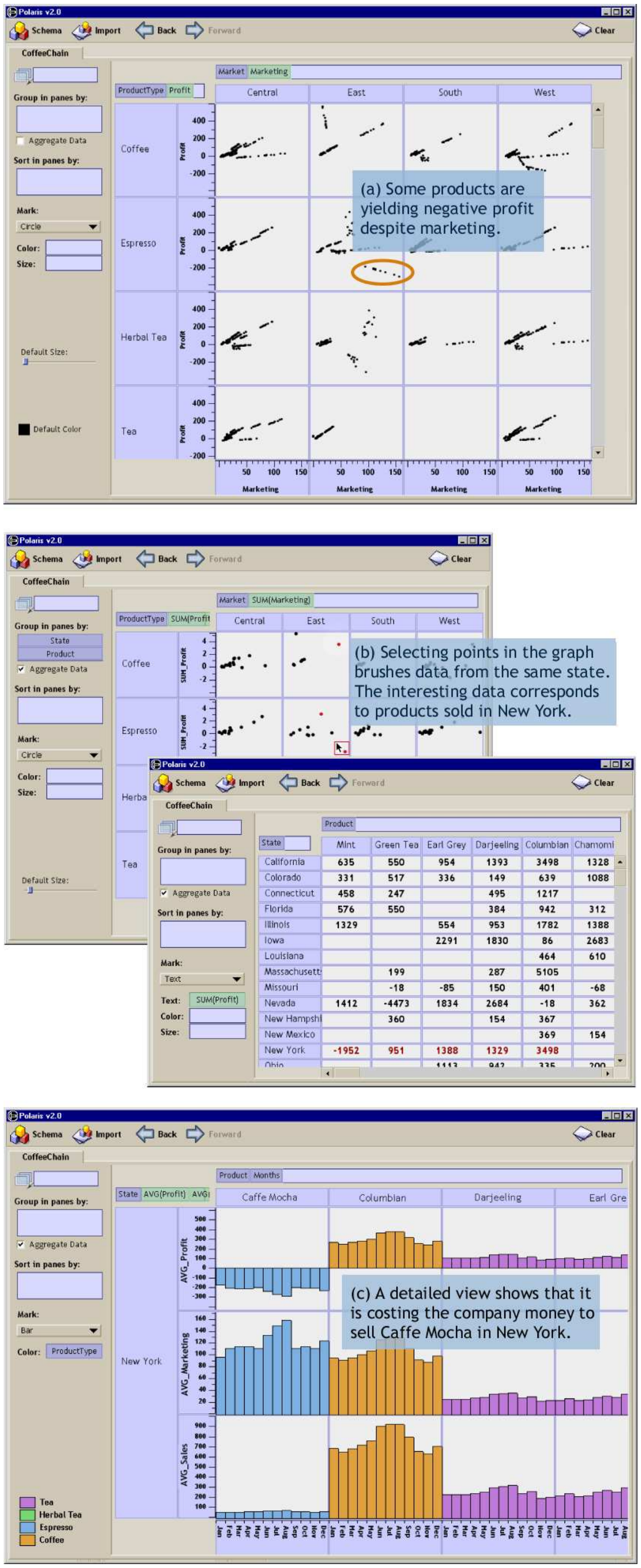

Figure 6: *An example scenario demonstrating the capabilities of Polaris for exploratory analysis of multi-dimensional databases. The data displayed is for a hypothetical coffee store chain, and the analyst is searching for ways to reduce the company's expenses.*

value as the selected records are highlighted. The CFO is able to use these linked views to determine that in New York, several products are offering very little return despite high expenditures.

The CFO creates a third display (Figure 6(c)): a set of bar charts showing profit, sales, and marketing for each product sold in New York, broken down by month. In this view, the CFO can clearly see that Caffé Mocha's profit margin does not justify its marketing expenses. With this data, the CFO can change the company's marketing and sales strategies in this state.

This example illustrates several important points about the exploratory process. Throughout the analysis, both the data users want to see and how they want to see it change continually. Analysts first form hypotheses about the data and then create new views to perform tests and experiments to validate or disprove those hypotheses. Certain displays enable an understanding of overall trends, whereas others show causal relationships. As the analysts better understand the data, they may want to drill-down in the visible dimensions or display entirely different dimensions.

Polaris supports this exploratory process through its visual interface. By formally categorizing the types of graphics, Polaris is able to provide a simple interface for rapidly generating a wide range of displays. This allows analysts to focus on the analysis task rather than the steps needed to retrieve and display the data.

7. Discussion

Several of the ideas in our specification are extensions of Wilkinson's [29] efforts to develop a grammar for statistical graphics. His grammar encapsulates both the statistical transformation of datasets and their mapping to graphic representations.

The primary distinctions between Wilkinson's system and ours arise because of differences in the data models. We chose to focus on developing a tool for multi-dimensional relational databases and we decided to build as much of the system as possible using relational algebra. All of the data transformations required by our visual specifications can be precisely interpreted as standard SQL queries to OLAP servers. Wilkinson instead intentionally uses a data model that is not relational, citing shortcomings in the relational model's support for statistical analysis. Consequently, his specification defines operations and function in terms of his own data model consisting of variable sets and indexed variables.

The differences in design are most apparent in the table algebra. As in our system, Wilkinson's table algebra performs two functions: it provides database services such as set operations and it specifies the layout of the tables and graphs. Since we use relational algebra for all our database services, our algebra is different. For example, his blend operator performs both set union and may partition the axes of a table; our concatenation operation is different since it just performs partitioning. Another difference is in his cross and nest operators: cross generates a 2D graphic and nest only a 1D graphic. We use a different mechanism (shelves) to specify axes of the graphic. Overall, whether our system is better than Wilkinson's is hard to judge completely, and will require more experience using the system to solve practical problems. One major advantage of our approach is that it leverages existing database systems and as a result was very easy to implement.

Another interesting issue is the interpretation of our visual specifications as database queries. When database queries are generated from the visual specifications in Polaris, it is necessary to generate a SQL query per table pane. This problem is similar to the one that motivated Gray et al. to develop the CUBE operator [13]. The CUBE operator generalizes the queries necessary to develop cross-tab and Pivot Table displays of relational data into a single, more efficient operator. However, the CUBE operator cannot be applied in our situation because it assumes that the sets of relations partitioned into each table pane do not overlap. In several possible Polaris table configurations, such as scatterplot matrices, there can be considerable overlap between the relations partitioned into each pane. One can imagine generalizing the CUBE operator to handle these overlapping partitions.

Another major limitation of the CUBE operator is its method for computing aggregates. Usually only aggregates based on sums are allowed. More complex aggregation operators requiring ranking, such as computation of medians and modes, are not part of the current specification, although they are available in some commercial systems. These operators are very useful for data mining applications.

8. Conclusions and Future Work

We have presented Polaris, an interface for the exploration and analysis of large multi-dimensional databases. Polaris extends the well-known Pivot Table interface to display relational query results using a rich, expressive set of graphical displays. A second contribution of this system is a succinct visual specification for describing table-based graphical displays of relational data and the interpretation of these visual specifications as a precise sequence of relational database operations.

We have many plans for extending this system. The current version of Polaris does not leverage the hierarchical structure of many multi-dimensional databases. We are currently exploring interaction techniques for navigating these types of hierarchies and developing specifications that describe graphics derived from hierarchical data.

Furthermore, because graphical marks in Polaris directly correspond to tuples in the relational databases, it is possible to generate database tables from a selected set of graphical marks. This technique can be used to develop lenses, similar to the Magic Lens [5], that can perform much more complex transformations because they operate in data space rather than image space. This technique can also be used to compose Polaris displays, using a selected mark set in one display as the data input to another. We are exploring these techniques and believe it is possible to develop a relational spreadsheet by composing Polaris displays in this manner.

Acknowledgments

The authors especially thank Robert Bosch and Diane Tang for their contributions to the design and implementation of Polaris, their reviews of manuscript drafts, and for many useful discussions. The air traffic data is courtesy William F. Eddy and Shingo Oue. The coffee chain data is courtesy Stephen Eick and Visual Insights. This work was supported by the Department of Energy through the ASCI Level 1 Alliance with Stanford University.

References

[1] A. Aiken, J. Chen, M. Stonebraker, and A. Woodruff. Tioga-2: A Direct Manipulation Database Visualization Environment. In *Proc. of the 12th International Conference on Data Engineering*, February 1996, pp. 208-217.

[2] R. Becker, W. S. Cleveland and R. Douglas Martin. Trellis Graphics Displays: A Multi-Dimensional Data Visualization Tool for Data Mining. *3rd Annual Conference on Knowledge Discovery in Databases*, August 1997.

[3] J. Bertin. *Graphics and Graphic Information Processing.* Walter de Gruyter, Berlin, 1980.

[4] J. Bertin. *Semiology of Graphics.* The University of Wisconsin Press, Madison, Wisconsin, 1983. Translated by W. J. Berg.

[5] E. A. Bier, M. Stone, K. Pier, W. Buxton and T. DeRose. Toolglass and Magic Lenses: The See-Through Interface. In *SIGGRAPH '93 Proceedings*, August 1993, pp. 73-80.

[6] R. Bosch, C. Stolte, D. Tang, J. Gerth, M. Rosenblum, and P. Hanrahan. Rivet: A Flexible Environment for Computer Systems Visualization. In *Computer Graphics,* February 2000, pp. 68-73.

[7] W. S. Cleveland. *The Elements of Graphing Data.* Wadsworth Advanced Books and Software, Pacific Grove, California, 1985.

[8] W. S. Cleveland. *Visualizing Data.* Hobart Press, New Jersey, 1993.

[9] M. Derthick, J. Kolojejchick and S. F. Roth. An Interactive Visualization Environment for Data Exploration. In *Proc. of Knowledge Discovery in Databases*, August, 1997, pp. 2-9.

[10] S. Eick. Visualizing Multi-Dimensional Data. In *Computer Graphics,* February 2000, pp. 61–67.

[11] *Microsoft Excel – User's Guide,* Microsoft, Redmond, WA, 1995.

[12] J. Goldstein, S. F. Roth, J. Kolojejchick, and J. Mattis. A Framwork for Knowledge-based Interactive Data Exploration. In *Journal of Visual Languages and Computing*, December 1994, pp. 339-363.

[13] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, H. Pirahesh, and F. Pellow. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. In *Proc. of the 12th International Conference on Data Engineering*, February 1996, pp. 152-159.

[14] J.A. Hartigan. Printer graphics for clustering. *Journal of Statistical Computation and Simulation, 4,* pp. 187-213.

[15] Human Genome Project. [online] Available: http://www.ornl.gov/hgmis/about.html, cited March 2000.

[16] S. M. Kosslyn. *Elements of Graph Design*. W.H. Freeman and Co., New York, NY, 1994.

[17] M. Livny, R. Ramakrishnan, K. Beyer, G. Chen, D. Donjerkovic, S. Lawande, J. Myllymaki and K. Wenger. DEVise: Integrated Querying and Visual Exploration of Large Datasets. In *Proc. of ACM SIGMOD*, May, 1997.

[18] J.D. Mackinlay. Automating the Design of Graphical Presentations of Relational Information. In *ACM Trans. of Graphics,* April 1986, pp. 110-141.

[19] R. Rao and S. Card. The Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus+Context Visualization for Tabular Information. In *Proc. of SIGCHI '94,* pp. 318-322.

[20] B. Rogowitz, and L. Treinish. How NOT to Lie with Visualization. *Computers in Physics*, May/June 1996, pp. 268-274.

[21] S.F. Roth, J. Kolojejchick, J. Mattis and J. Goldstein. Interactive Graphic Design Using Automatic Presentation Knowledge. In *Proc. of SIGCHI '94*, April 1994, pp. 112-117.

[22] S.F. Roth, P. Lucas, J.A. Senn, C.C. Gomberg, M.B. Burks, P.J. Stroffolino, J. Kolojejchick and C. Dunmire. Visage: A User Interface Environment for Exploring Information. In *Proc. of Information Visualization,* October 1996, pp. 3-12.

[23] Sloan Digital Sky Survey. [online] Available: http://www.sdss.org/, cited March 2000.

[24] M. Spenke, C. Beilken and T. Berlage. FOCUS: The Interactive Table for Product Comparison and Selection. In *Proc. of the ACM Symposium on User Interface Software and Technology*, November 1996.

[25] S.S. Stevens. On the theory of scales of measurement. *Science, 103,* pp. 677-680.

[26] E. Thomsen. *OLAP Solutions: Building Multidimensional Information Systems*. Wiley Computer Publishing, New York, 1997.

[27] D. Travis. *Effective Color Displays: Theory and Practice.* Academic Press, London, 1991.

[28] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Box 430, Chesire, Connecticut, 1983.

[29] L. Wilkinson. *The Grammar of Graphics.* Springer, New York, New York, 1999.