DOI:10.1145/1400214.1400234

Polaris: A System for Query, Analysis, and Visualization of Multidimensional Databases

By Chris Stolte, Diane Tang, and Pat Hanrahan

Abstract

During the last decade, multidimensional databases have become common in the business and scientific worlds. Analysis places significant demands on the interfaces to these databases. It must be possible for analysts to easily and incrementally change both the data and their views of it as they cycle between hypothesis and experimentation.

In this paper, we address these demands by presenting the Polaris formalism, a visual query language for precisely describing a wide range of table-based graphical presentations of data. This language compiles into both the queries and drawing commands necessary to generate the visualization, enabling us to design systems that closely integrate analysis and visualization. Using the Polaris formalism, we have built an interactive interface for exploring multidimensional databases that analysts can use to rapidly and incrementally build an expressive range of views of their data as they engage in a cycle of visual analysis.

1. INTRODUCTION

Nowadays, structured databases are widely used. Corporations store every sales transaction in large data warehouses. International research projects such as the Human Genome Project and Digital Sky Survey are generating massive scientific databases. Organizations such as the United Nations are making a wide range of global indicators on issues ranging from carbon emission to the adoption of technology publicly available via the Internet.

Unfortunately, our ability to collect and store data has rapidly exceeded our ability to analyze it. A major challenge in computer science is how to extract meaning from data: to discover structure, find patterns, and derive causal relationships. An analytical session cycles between hypothesis, experiment, and discovery. Often the path of exploration is unpredictable, and thus analysts need to be able to rapidly change both what data they are viewing and how they are viewing that data. This exploratory analysis process places significant demands on the human–computer interfaces to these databases. Few good tools exist.

In this paper, we present a formal approach to building visualization systems that addresses these demands. The first contribution is the Polaris formalism, a declarative visual query language that specifies a wide range of 2D graphic displays. The three key components of the formalism are (1) a table algebra that captures the structure of tables and spatial encodings, (2) a graphic taxonomy that results in an intuitive specification of graphic types, and (3) a system for effective visual encoding. This language allows for easily changing between different graphic displays as well as adding or removing data.

The second main contribution is the combination of this visual query language with the underlying database queries needed. This allows us to combine both visualization as well as the underlying data transformations to support the exploratory process.

The final contribution is the Polaris interface that allows users to incrementally construct a visual specification by dragging fields onto "shelves" (see Figure 1). Each intermediate specification is valid and corresponds to a graphical data display, giving the user quick visual feedback to support this analysis. This interface is built on top of the visual query language that specifies both the data and graphical transformations needed, thus combining statistical analysis and visualization. Polaris enables visual analysis by allowing an analyst to answer a question by composing a picture of what they want to see.

It has been 6 years since this work was originally published. In that time, the technology has been commercialized by Tableau Software as Tableau Desktop and is currently in use by thousands of companies and tens of thousands of users. As a result, we have gained considerable experience that has validated the effectiveness of the visual query language and interface and resulted in extensions and revisions to both.

2. OVERVIEW

Polaris has been designed to support the interactive exploration of large multidimensional relational databases or data cubes. 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.[18] We refer to a row in a relational table as a *tuple* or *record*, and a column as a *field*. A single database will contain many heterogeneous but interrelated tables.

The authors dedicate this article to the memory of Jim Gray, whose pioneering work inspired this research.

A previous version of this paper was published in IEEE's *Transactions on Visualization and Computer Graphics*, vol 8, issue 1 (Jan. 2002), pp. 52–65.

We can classify fields in a database as nominal, ordinal, quantitative, or interval.[4,16] This classification is the field's **scale**. Polaris reduces this categorization to ordinal and quantitative by treating intervals as quantitative and assigning an ordering to the nominal fields to treat them as ordinal. A field's scale affects its visual representation. Quantitative fields are continuous, and are shown as axes or smoothly varying values. Ordinal scales are represented discretely, as headers or different classes.

The fields within a relational table can also be partitioned into two types: dimensions and measures. This classification is the field's **role**. Dimensions and measures are similar to independent and dependent variables in traditional analysis. For example, a product name or type would be a dimension while the product price or size would be a measure. The field's role determines how the query is generated. Measures are computed using an aggregation function and dimensions form the groups to be aggregated.

Polaris originally treated ordinal fields as dimensions and quantitative fields as measures. With experience, however, we have found that a field's scale and role are orthogonal, and may change depending on the question. For example, when asking the question "What is the average age of people purchasing a product?" the field Age is acting as a measure. However, when asking the question "What is the average amount spent classified by customer age?" then Age is acting as a dimension. The current implementation of Tableau uses simple heuristics based on a field's data type and domain cardinality to determine a field's default role and scale, but allows both to be easily changed.

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

- **Exploratory interface:** Analysts must be able to rapidly and incrementally change what data they are viewing and how they are viewing that data as they explore hypotheses.
- **Multiple display types:** Analysis consists of different tasks such as discovering correlations between variables, finding patterns, and locating outliers. An analysis tool must be able to generate displays suited to these disparate tasks.
- **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.

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 multidimensional data:

- **Multivariate:** The table structure can encode multiple dimensions, enabling the display of high-dimensional data.
- **Comparative:** Tables generate *small-multiple* displays of information, which are easily compared to expose patterns and trends across dimensions.[20]
- **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.[2,7,20]

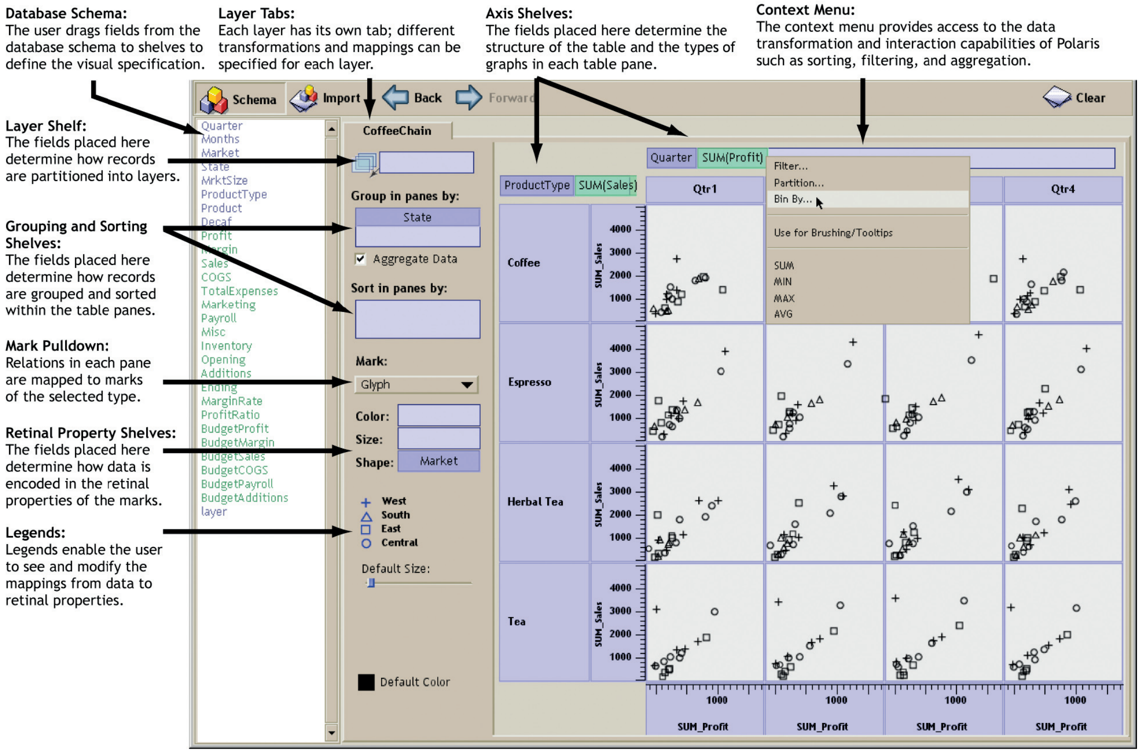
Figure 1 shows the Polaris user interface. In this example, the analyst has constructed a matrix of scatterplots showing sales versus profit for different product types in different quarters. The primary interaction technique is to drag-and-drop fields from the database schema onto shelves throughout the display. We call a given configuration of fields on shelves a *visual specification*. The visual specification is converted into the language using simple operations. 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 *y* shelf, columns by fields on the *x* 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, aggregates, and other derived fields. Records may also be sorted into a given drawing order.
- The type of graphic displayed in each table pane. 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. Each mark represents a tuple, so selecting a single mark in a graphic by clicking on it pops up a detail window that displays user-specified field values for the tuples corresponding to that mark. The tuples represented by a set of marks can be cut and pasted into a spreadsheet by selecting the marks representing the tuples. Analysts can draw rubber bands around a set of marks to brush or highlight related records, either within a single table or between multiple Polaris displays.

In Section 3, we describe how the visual specification is used to generate graphics. In Section 4, we describe the supported data transformations and how the visual specifications are

Figure 1: The Polaris user interface. Analysts construct table-based displays of 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.



Database Schema:
The user drags fields from the database schema to shelves to define the visual specification.

Layer Tabs:
Each layer has its own tab; different transformations and mappings can be specified for each layer.

Axis Shelves:
The fields placed here determine the structure of the table and the types of graphs in each table pane.

Context Menu:
The context menu provides access to the data transformation and interaction capabilities of Polaris such as sorting, filtering, and aggregation.

Layer Shelf:
The fields placed here determine how records are partitioned into layers.

Grouping and Sorting Shelves:
The fields placed here determine how records are grouped and sorted within the table panes.

Mark Pulldown:
Relations in each pane are mapped to marks of the selected type.

Retinal Property Shelves:
The fields placed here determine how data is encoded in the retinal properties of the marks.

Legends:
Legends enable the user to see and modify the mappings from data to retinal properties.

used to generate the database queries for statistical analysis.

3. GENERATING GRAPHICS

The visual specification consists of three components: (a) the specification of the table configuration, (b) the type of graphic inside each pane, and (c) the details of the visual encodings (for more details, see[17]). We discuss each of these in turn.

3.1. Table algebra

We define an algebra as a formal mechanism to specify table configurations. When analysts place fields on 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 *x*, *y*, and *z* expressions form clauses in the language.

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 sequences of values 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.

Ordinal and quantitative fields generate tables with different structures. Ordinal fields partition the table into rows and columns using headers, whereas quantitative fields generate axes.

$$
\begin{aligned}
A &= domain(A) = \{a_1, \ldots, a_n\} \\
P &= \{P\}
\end{aligned}
$$

A valid expression in our algebra is one or more field 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 semantics of each operator is defined in terms of its effects on sequences.

Concatenation: The plus operator concatenates two sequences:

$$
\begin{aligned}
A+B &= \{a_1, \ldots, a_n\} + \{b_1, \ldots, b_m\} \\
&= \{a_1, \ldots, a_n, b_1, \ldots, b_m\} \\
A+P &= \{a_1, \ldots, a_n\} + \{P\} \\
&= \{a_1, \ldots, a_n, P\} \\
P+Q &= \{P\} + \{Q\} \\
&= \{P, Q\}
\end{aligned}
$$

Cross: The cross operator performs a Cartesian product of elements in the two sequences:

Nest: The nest operator is similar to the cross operator, but it only creates sequence entries for which there exist records

$$
\begin{aligned}
A\times B &= \{a_1,\ldots,a_n\}\times\{b_1,\ldots,b_m\}\\
&= \{a_1b_1,\ldots,a_1b_m,\\
&\quad a_2b_1,\ldots,a_2b_m,\ldots,\\
&\quad a_nb_1,\ldots,a_nb_m\}\\
A\times P &= \{a_1,\ldots,a_n\}\times P\\
&= \{a_1P,\ldots,a_nP\}
\end{aligned}
$$

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. Data cubes represent hierarchies explicitly and there is no need to compute the nest relationship.

Using the above semantics for each operator, every expression in the algebra can be reduced to a single sequence, with each entry in the sequence being an ordered term consisting of zero or more ordinal values with zero or more quantitative field names. We call this set evaluation of an expression the *normalized form*. The normalized 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.

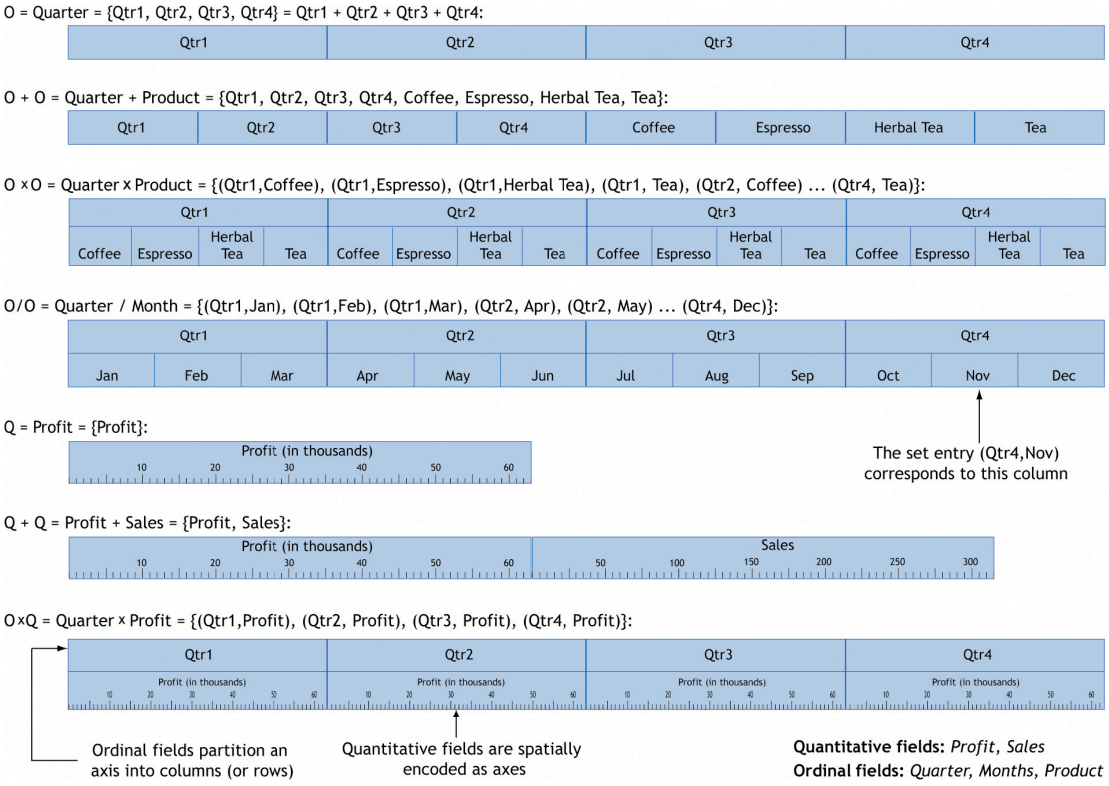
In retrospect, the Polaris table algebra is very similar to the operations in the MDX query language for data cubes.

3.2. Types of graphics

Given a table configuration, the next step is to specify the type of graphic in each pane. We have developed a taxonomy of graphics that results in an intuitive and concise specification of graphic types. This taxonomy is based on both the axes within each pane (implicitly specified from the table configuration via the role and scale of the innermost field in the sequence) as well as the mark type representing a tuple (e.g., text, shape, bar, etc.). We group this taxonomy into three families (illustrated in Figure 3) based on the axes: ordinal–ordinal, ordinal–quantitative, and quantitative–quantitative.

Each family contains a number of variants depending

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


O = Quarter = {Qtr1, Qtr2, Qtr3, Qtr4} = Qtr1 + Qtr2 + Qtr3 + Qtr4:
Qtr1
Qtr2
Qtr3
Qtr4
O + O = Quarter + Product = {Qtr1, Qtr2, Qtr3, Qtr4, Coffee, Espresso, Herbal Tea, Tea}:
Qtr1
Qtr2
Qtr3
Qtr4
Coffee
Espresso
Herbal Tea
Tea
O ×O = Quarter x Product = {(Qtr1,Coffee), (Qtr1,Espresso), (Qtr1,Herbal Tea), (Qtr1, Tea), (Qtr2, Coffee) ... (Qtr4, Tea)}:
Qtr1
Qtr2
Qtr3
Qtr4
Coffee
Espresso
Herbal Tea
Tea
O/O = Quarter / Month = {(Qtr1,Jan), (Qtr1,Feb), (Qtr1,Mar), (Qtr2, Apr), (Qtr2, May) ... (Qtr4, Dec)}:
Qtr1
Qtr2
Qtr3
Qtr4
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
The set entry (Qtr4,Nov) corresponds to this column
Q = Profit = {Profit}:
Profit (in thousands)
10
20
30
40
50
60
Q + Q = Profit + Sales = {Profit, Sales}:
Profit (in thousands)
10
20
30
40
50
60
Sales
50
100
150
200
250
300
O×Q = Quarter x Profit = {(Qtr1,Profit), (Qtr2, Profit), (Qtr3, Profit), (Qtr4, Profit)}:
Qtr1
Qtr2
Qtr3
Qtr4
Profit (in thousands)
Ordinal fields partition an axis into columns (or rows)
Quantitative fields are spatially encoded as axes
Quantitative fields: Profit, Sales
Ordinal fields: Quarter, Months, Product

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. There are two types of marks; single tuple marks and multituple marks. Multituple marks form a single graphical entity from a set of marks; an example is a polygon mark where each vertex of the polygon is a single tuple.

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. We briefly discuss the defining characteristics of the three families within our categorization. It should be noted that these rules allow us to automatically choose a default mark given the types of the fields on the axes.

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 $f(O_x,O_y) \to R$, where R represents the fields encoded in the retinal properties of the marks. This can be seen in the heatmap in Figure 3, where the analyst is studying gene expression as a function of experiment and gene. Figure 6(a) shows another example where lines of source code are color-encoded with the number of cache misses attributable to that line.

Ordinal–Quantitative Graphics: Well-known representatives of this family of graphics are the bar chart, 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 $f(O) \to Q$. The cardinality of the record set affects 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 or clustered bar chart).

The ordinal and quantitative values may be independent variables, such as in a Gantt chart. Here, each pane represents all events in a category; each event has a type as well as a beginning and end time. Figure 6(c) shows a table of Gantt charts, with each Gantt chart displaying the thread scheduling and locking activity on a CPU within a multiprocessor computer.

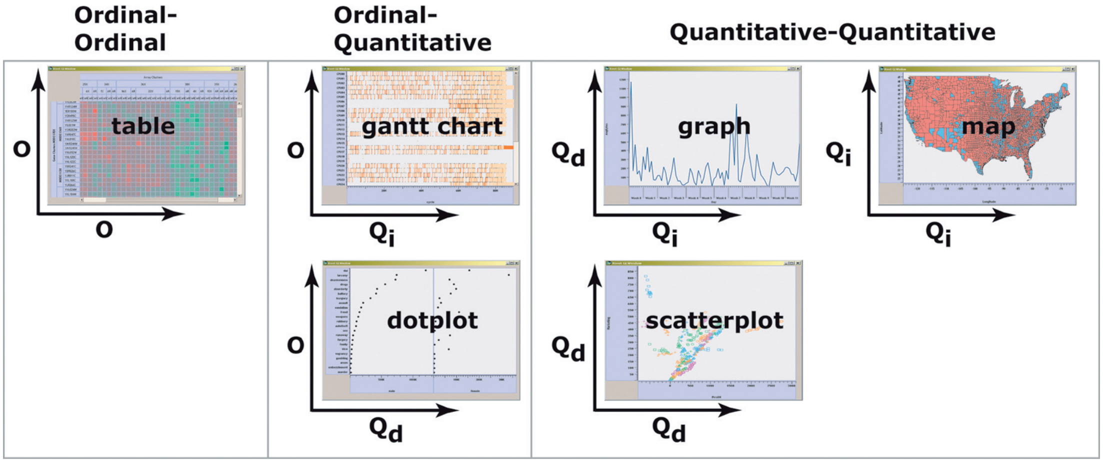
Quantitative–Quantitative Graphics: Graphics of this type are used to understand the distribution of data as a function of one or both quantitative variables and to discover causal relationships between the two quantitative variables, such as in a scatterplot matrix. Figure 3 illustrates another example of a quantitative–quantitative graphic: the map. In this figure, the analyst is studying election results by county.

3.3. Visual mappings

Each record in a pane is mapped to a mark. There are two components to the visual mapping. The first component, described in Section 3.2, determines the type of graphic and 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.

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

Figure 3: The families of graphics within our taxonomy with examples of well-known charts from each family. The taxonomy structures the space of graphics into three families by the types of fields assigned to their axes and then further structures each family by the number of independent and dependent variables. Using this taxonomy we can derive the type of graphic within each pane from the table axes expressions and the mark type.



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. While generating effective default mappings is not a fundamental aspect of the language, it has proven to be important and is discussed in more detail in Section 7. Here, we briefly discuss how effective mappings are generated for each retinal property. 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.

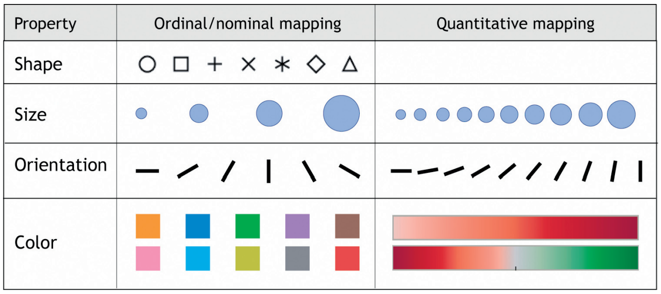
Size: Analysts can use size to encode either an ordinal or quantitative field. When encoding a quantitative domain as size, a linear map from the field domain to the *area* of the mark is created. The minimum size is chosen so that all visual properties of a mark with the minimum size can be perceived.[10] 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,[10] 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.[19] 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.[4,10,19] 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 originally used for encoding quantitative data was the isomorphic colormap developed by Rogowitz.[13] We have since had palettes hand-designed by a color expert in the HSV space that balance perceptual properties with aesthetics.

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.



Property	Ordinal/nominal mapping	Quantitative mapping
Shape	○ □ + × ∗ ◇ △	
Size	● ● ● ●	● ● ● ● ● ● ● ● ● ●
Orientation	— ⁄ / | \ ⟍	— — ⁄ ⁄ / / / / | |
Color	🟧 🟦 🟩 🟪 🟫 🟥 🟦 🟨 ⬛ 🟥	▬ ▬

4. DATA TRANSFORMATIONS, VISUAL QUERIES, AND GENERATING DATABASE QUERIES

An important aspect of the Polaris formalism is the unification of graphics and data transformations. A single visual specification must *completely* specify both the data retrieval and the data presentation. Thus, the formalism must support the complete range of data transformations possible in a query language such as SQL,[17] including the common relational operators: selection, filtering, grouping and aggregation, and sorting. It can be shown that any query expressible in SQL can be expressed as a specification in the Polaris formalism.[17]

The Polaris interface exposes all capabilities of the underlying database query language: the state of the interface generates both a visual specification and a statement in the visual query language. All fields on shelves are inserted into a select statement. Measure fields are aggregated while dimension fields are inserted into a GROUPBY statement, with additional dimension fields specified in a Level-of-Detail shelf. Each dimension can also be sorted, and different aggregation functions can be associated with each measure; these options are chosen by drop-down menus on each field on the shelf. There is also a filter shelf that represents items in the WHERE clause. Finally, dialog boxes expose general calculations and joins.

Figure 5 shows the overall data flow in Polaris.

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 and computing any user-defined calculations.

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))
```

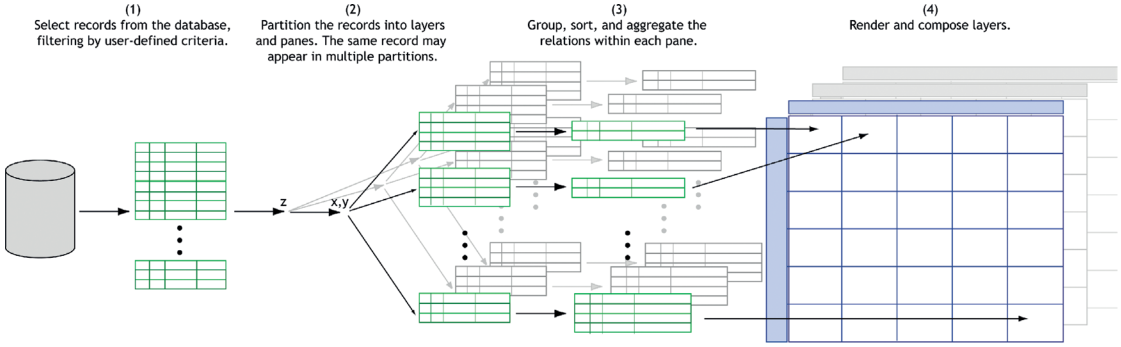
We can define the relational predicate *filters* as the conjunction of all of the individual field filters. Then, the first stage of the data transformation network is equivalent to the SQL statement:

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

It is possible within the complete formalism to define more sophisticated filtering, such as filters on the cross-product of multiple fields or filters with ordering dependencies (filter A is computed relative to filter B).

Step 2: Partitioning the Records into Panes: The second

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.


(1)
Select records from the database, filtering by user-defined criteria.
(2)
Partition the records into layers and panes. The same record may appear in multiple partitions.
(3)
Group, sort, and aggregate the relations within each pane.
(4)
Render and compose layers.
z
x,y

phase of the data flow partitions the retrieved records into groups corresponding to each pane in the table. As we discussed in Section 3.1, 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.

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, which is clearly nonoptimal. Various optimizations are discussed in.[17]

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}
```

5. 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 multidimensional databases by considering the following scenario.

At Stanford, researchers developing Argus,[9] a parallel graphics library, found that its performance had linear speedup when using up to 31 processors, after which its performance diminished rapidly. Using Polaris, we recreate the analysis they performed using a custom-built visualization tool.[5]

Initially, the developers hypothesized that the diminishing performance was a result of too many remote memory accesses, a common performance problem in parallel programs. They collected and visualized detailed memory statistics to test this hypothesis. Figure 6(a) shows a visualization constructed to display this data. The visualization is composed of two linked Polaris instances. One displays a bird's eye view of multiple source code files with each line of code represented by a single pixel height bar and the other displays the detailed source-code. In both views, the hue of each line of code encodes the number of cache misses suffered by that line. Upon seeing these displays, they could tell that memory was in fact not the problem.

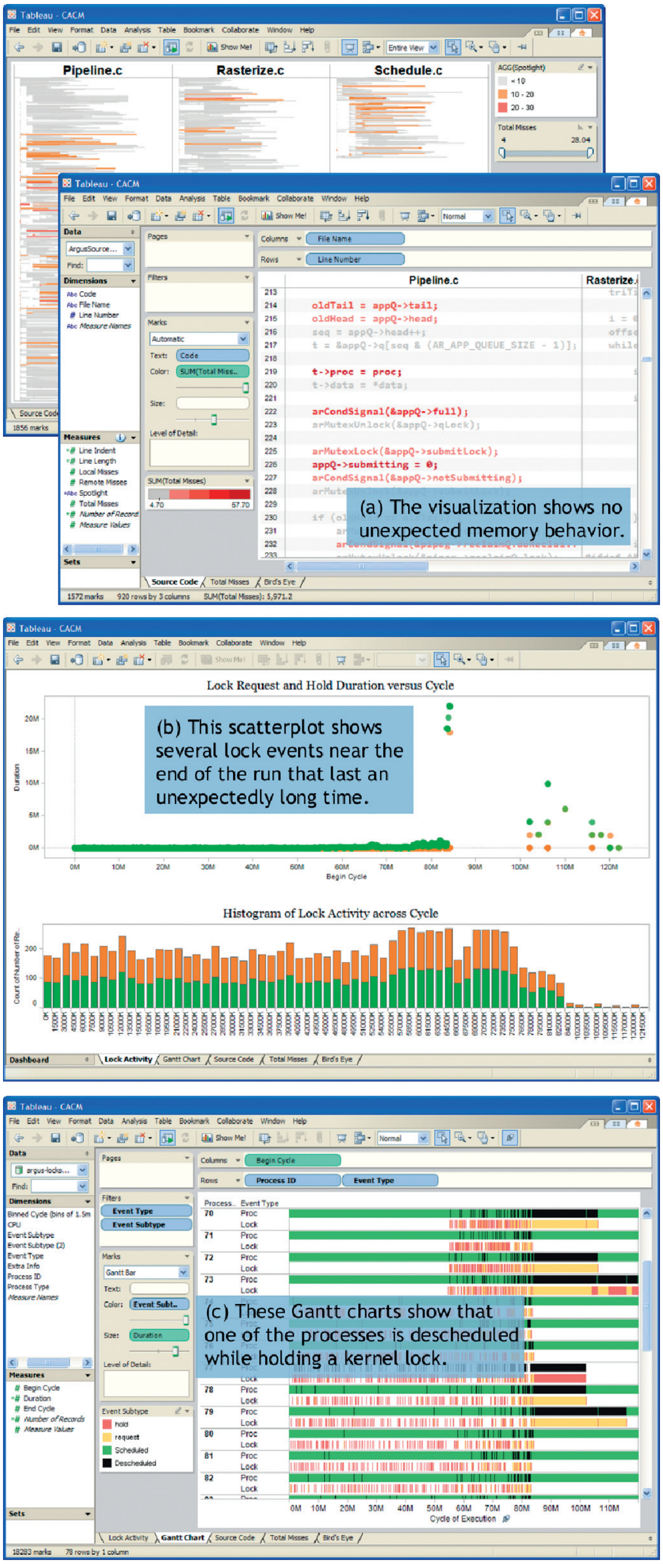
The developers next hypothesized that lock contention might be a problem, so they reran Argus and collected detailed lock and scheduling information. The data is shown in Figure 6(b) using a dashboard within Polaris to create a composite visualization with two linked projections of the same data. One projection shows a scatterplot of the start cycle versus cycle duration for the lock events (requests and holds). The second shows a histogram over time of initiated lock events. The scatterplot shows that toward the end of the run, the duration of lock events (both holds and requests) was taking an unexpectedly long time. That observation correlated with the histogram showing that the number of lock requests peaked and then tailed off towards the end of the run indicated that this might be a fruitful area for further investigation.

A third visualization, shown in Figure 6(c), shows the same data using Gantt charts to display both lock events and process-scheduling events. This display shows that the long lock requests correspond to descheduled periods for most processes. One process, however, has a descheduled period corresponding to a period during which the lock was held. This behavior, which was due to a bug in the operating system, was the source of the performance issues.

This example illustrates several important points about the exploratory process. Throughout the analysis, both the data that 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 test 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, allowing analysts to focus on the analysis task rather than interface.

Figure 6: A scenario demonstrating the use of Polaris to analyze the performance of a parallel graphics library.



```
oldTail = appQ->tail;
oldHead = appQ->head;
seq = appQ->head++;
t = &appQ->q[seq & (AR_APP_QUEUE_SIZE - 1)];

t->proc = proc;
t->data = *data;

arCondSignal(&appQ->full);
arMutexUnlock(&appQ->qLock);

arMutexLock(&appQ->submitLock);
appQ->submitting = 0;
arCondSignal(&appQ->notSubmitting);
```

(a) The visualization shows no unexpected memory behavior.

(b) This scatterplot shows several lock events near the end of the run that last an unexpectedly long time.

(c) These Gantt charts show that one of the processes is descheduled while holding a kernel lock.

6. EXPERIENCE

In the 6 years since this work was originally published, we have gained considerable experience with the formalism and the interface. In that time, the technology has been commercialized and extended by Tableau Software as Tableau Desktop and is used by thousands of companies and tens of thousands of users. The system has also been adapted to the web, so it is possible to perform analysis within a browser. The uses are diverse, ranging from disease research in the jungles of Central America to marketing analysis in Fortune 50 companies to usability analysis by video game designers, and the data sizes range from small spreadsheets to billions of rows of data. The many types of users indicate the ubiquity of data and the demand for new tools. This experience has emphasized three points to us: (1) the importance of a formal approach, (2) the importance of an architecture that leverages database technology rather than replaces it, and (3) the importance of building effective defaults into the graphical interface.

One question early on was: "do we need a formalism?" Most visualization systems have predefined types of charts and use wizards to help the user construct graphs. Having a language allows us to generate an unlimited number of different types of graphics. Restricting the set of views to a small set limits the power of visualization; this would be like building into a query language a small set of predefined queries. Experience has shown that this flexibility makes it possible to *incrementally* build new views, which is key to smoothly supporting the analysis process. Both of these aspects of Polaris are enabled by the formal nature of the algebra, where every addition or deletion leads to a new algebraic statement.

The formalism also enables us to unify the specification of the visualization with the database query: users can change the query used to fetch the data and their view of it simultaneously. In subsequent work, we have proved that the language is complete; that is, it is possible to generate any statement in the relational algebra. A major problem with many visual interfaces is that they restrict the types of queries that can be formed.

This unification of visualization and database queries is also a key architectural decision that makes it possible to use our system as a front-end to large parallel database servers. This makes it easy to access important data in existing data sources, to leverage high performance database technology (e.g., database appliances, massively parallel computation, column stores), and to avoid data replication and application-specific data silos. Why move a terabyte of data if you don't have to?

One potential issue with a compositional language is that it creates a large space of possible visualizations. While many are effective and aesthetically pleasing, many are not. Thus, choosing default graphics is an important part of any production system and allows for additional succinctness in the language. However, the issue is not just with choosing default graphics. Generating effective visual mappings (e.g., color, shape) by default is not a fundamental aspect of the language, but is equally important. Effective defaults enable users to focus on their task and questions rather than the details of color or shape selection, especially since many users are not trained as graphic designers or psychologists.

7. RELATED WORK

The related work to Polaris can be divided into two categories: formal graphical specifications and database exploration tools.

7.1. Formal graphical specifications

We have built on the work of several researchers' insights into the formal properties of graphic communication, such as Bertin's *Semiology of Graphics*,[4] Cleveland's experimental results on the perception of data,[7,8] Wilkinson's formalism for statistical graphics,[22] and Mackinlay's APT system.[12] However, the Polaris formalism is innovative in several ways. One key aspect of our approach is that all specifications can be compiled directly into queries. Existing formalisms do not consider the generation of queries to be related to the presentation of information. Another innovation is the use of an algebra to describe table-based displays. Tables are particularly effective for displaying multidimensional data, as multiple dimensions of the data can be explicitly encoded in the structure of the table. Finally, our formalism is the basis for several interactive tools for analyzing and exploring large data warehouses and this usage has affected the development of the formalism.

7.2. Database exploration tools

The second area of related work is visual query and database exploration tools. Academic projects such as Visage,[14] DEVise,[11] and Tioga-2[1] have focused on developing visualization environments that support interactive database exploration through *visual queries*. Users construct queries and visualizations through their interactions with the visualization system interface. These systems have flexible mechanisms for mapping query results to graphs, and support mapping database records to retinal properties. However, none of these systems is based on an expressive formal language for graphics nor do they leverage table-based organizations of their visualizations.

Finally, existing systems, such as XmdvTool,[21] Spotfire,[15] and XGobi[6] have taken the approach of providing a set of predefined visualizations, such as scatterplots and parallel coordinates. These views are augmented with interaction techniques, such as brushing and zooming, which can be used to refine the queries. We feel that this approach is much more limiting than providing the user with a set of building blocks that can be used to interactively construct and refine a wide range of displays to suit any analysis task.

8. CONCLUSIONS

We have presented Polaris, a visual query language for databases and a graphical interface for authoring queries in the language. The Polaris formalism uses succinct visual specifications to describe a wide range of table-based visualizations of multidimensional information. Visual specifications can be compiled into both the queries and the drawing commands necessary to generate the displays, thus unifying analysis and visualization into a single visual query language.

Using the Polaris formalism, we have built the Polaris interface. The Polaris interface directly supports the cycle of analysis. Analysts can incrementally create sophisticated visualizations using simple drag-and-drop operations to construct a visual specification. This interface has been commercialized and extended by Tableau Software and is now in use by tens of thousands of users in thousands of companies.

Acknowledgments

The authors especially thank Robert Bosch for his contribution to the design and implementation of Polaris, his review of manuscript drafts, and for many useful discussions. The authors also thank Maneesh Agrawala for his insightful reviews and discussions on early drafts of this paper. This work was supported by the Department of Energy through the ASCI Level 1 Alliance with Stanford University.

References

1. Aiken, A., Chen, J., Stonebraker, M., and Woodruff, A. Tioga-2: A direct manipulation database visualization environment. *Proceeding of the 12th International Conference on Data Engineering*, February 1996, pp. 208–217.
2. Becker, R., Cleveland, W. S., and Douglas Martin, R. Trellis graphics displays: A multi-dimensional data visualization tool for data mining. *3rd Annual Conference on Knowledge Discovery in Databases*, August 1997.
3. Bertin, J. *Graphics and Graphic Information Processing*. Walter de Gruyter, Berlin, 1980.
4. Bertin, J. *Semiology of Graphics*. The University of Wisconsin Press, Madison, WI, 1983. Translated by W. J. Berg.
5. Bosch, R., Stolte, C., Stoll, G., Rosenblum, M., and Hanrahan, P. Performance analysis and visualization of parallel systems using SimOS and Rivet: A case study. *Proceedings of the Sixth Annual Symposium on High-Performance Computer Architecture*, January 2000, pp. 360–371.
6. Buja, A., Cook, D., and Swayne, D. F. Interactive high-dimensional data visualization. *Journal of Computational and Graphical Statistics*, 5(1):78–99, 1996.
7. Cleveland, W. S. *The Elements of Graphing Data*. Wadsworth Advanced Books and Software, Pacific Grove, CA, 1985.
8. Cleveland, W. S. *Visualizing Data*. Hobart Press, New Jersey, 1993.
9. Igehy, H., Stoll, G., and Hanrahan, P. The design of a parallel graphics interface. *Proceedings of SIGGRAPH*, 1998, pp. 141–150.
10. Kosslyn, S. M. *Elements of Graph Design*. W.H. Freeman and Co., New York, NY, 1994.
11. Livny, M., Ramakrishnan, R., Beyer, K., Chen, G., Donjerkovic, D., Lawande, S., Myllymaki, J., and Wenger, K. DEVise: Integrated querying and visual exploration of large datasets. *Proceeding of ACM SIGMOD*, May, 1997.
12. Mackinlay, J. D. Automating the design of graphical presentations of relational information. *ACM Transaction of Graphics*, April 1986, pp. 110–141.
13. Rogowitz, B. and Treinish, L. How NOT to lie with visualization. *Computers in Physics*, May/June 1996, pp. 268–274.
14. Roth, S. F., Lucas, P., Senn, J. A., Gomberg, C. C., Burks, M. B., Stroffolino, P. J., Kolojejchick, J., and Dunmire, C. Visage: A user interface environment for exploring information. *Proceeding of Information Visualization*, October 1996, pp. 3–12.
15. Spotfire Inc. [online] Available: http://www.spotfire.com, cited September 2001.
16. Stevens, S. S. On the theory of scales of measurement. *Science*, 103:677–680.
17. Stolte, C. Query, analysis, and visualization of multidimensional databases. PhD Dissertation, Stanford University, June 2003.
18. Thomsen, E. *OLAP Solutions: Building Multidimensional Information Systems*. Wiley Computer Publishing, New York, 1997.
19. Travis, D. *Effective Color Displays: Theory and Practice*. Academic Press, London, 1991.
20. Tufte, E. R. *The Visual Display of Quantitative Information*. Graphics Press, Box 430, Cheshire, CT, 1983.
21. Ward, M. XmdvTool: Integrating multiple methods for visualizing multivariate data. *Proceedings of Visualization*, October 1994, pp. 326–331.
22. Wilkinson, L. *The Grammar of Graphics*. Springer, New York, NY, 1999.

Chris Stolte is founder and vice president of Tableau Software, Seattle, WA. **Diane Tang** is a software researcher at Google, Inc., Mountain View, CA. **Pat Hanrahan** is Canon USA Professor in the School of Engineering at Stanford University, Stanford, CA.