SCI INSTITUTE
TECHNICAL REPORT

DEFOG: A System for Data-Backed Visual Composition

Lauro Lins, David Koop, Juliana Freire and Claudio Silva

UUSCI-2011-003

Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT 84112 USA

March 2, 2011

Abstract:

As users analyze data, visualization is important for both generating insight during exploration and displaying information for presentation purposes. While visualization systems have been very successful at the latter, their full potential as an aid for data exploration has yet to be realized. Existing systems have often focused on techniques for displaying information, requiring data to conform to specific file formats or allowing very limited manipulation of the data. In addition, most systems that provide a stronger link between data and visualization sacrifice the freedom of arbitrary composition. In this paper, we present DEFOG, a system that aims to more tightly integrate data exploration and visualization. DEFOG was designed to manipulate objects and as such, it is able to handle a wide range of data types. In addition, it allows flexible data manipulation through both visual and programmatic operations, all integrated in the same environment. By allowing both objects and their relationships to be displayed and combined, DEFOG provides great freedom for visual composition. To that end, it combines information visualization with infrastructure to support a variety of data including in-memory objects and operations found in vector-based graphics editors like select, move, scale, group, and copy-and-paste. We show that this system provides a natural way to use and combine data visualization techniques to explore a variety of data types, and present case studies that demonstrate its use in analyzing collections of workflow specifications and biochemical pathways.

THE UNIVERSITY OF UTAH

DEFOG: A System for Data-Backed Visual Composition

Lauro Lins David Koop Juliana Freire Cláudio Silva

Abstract— As users analyze data, visualization is important for both generating insight during exploration and displaying information for presentation purposes. While visualization systems have been very successful at the latter, their full potential as an aid for data exploration has yet to be realized. Existing systems have often focused on techniques for displaying information, requiring data to conform to specific file formats or allowing very limited manipulation of the data. In addition, most systems that provide a stronger link between data and visualization sacrifice the freedom of arbitrary composition. In this paper, we present DEFOG, a system that aims to more tightly integrate data exploration and visualization. DEFOG was designed to manipulate *objects* and as such, it is able to handle a wide range of data types. In addition, it allows flexible data manipulation through both visual and programmatic operations, all integrated in the same environment. By allowing both objects and their relationships to be displayed and combined, DEFOG provides great freedom for visual composition. To that end, it combines information visualization with infrastructure to support a variety of data including in-memory objects and operations found in vector-based graphics editors like select, move, scale, group, and copy-and-paste. We show that this system provides a natural way to use and combine data visualization techniques to explore a variety of data types, and present case studies that demonstrate its use in analyzing collections of workflow specifications and biochemical pathways.

1 INTRODUCTION

Our ability to acquire and generate large volumes of digital data is growing at a startling pace. However, the same cannot be said of our ability to analyze and present data. There has been substantial work, in different fields, on techniques to help users to more effectively manipulate and explore data: programming languages have been designed with simpler syntax and large libraries of routines so as to increase productivity; graphics tools are available that allow users to quickly compose and present a large amount of information; and information visualization systems have helped users obtain insight into complicated datasets. While there have been substantial advances toward the goal of streamlining data exploration, different solutions have focused on different aspects of the problems.

A user often needs to interact with several tools when exploring a dataset. For example, a user may select a subset of data using a command in a programming language, then display that data using a visualization system, and finally touch-up the image using an illustration application. More concretely, journalists and media visualization experts note that while they may use visualization tools to initially explore data, they often turn to illustration applications or custom tools to create the final presentation [12, 14]. Because data exploration is an inherently iterative process, having to use several tools and manage the data which flows through them is both time consuming and error prone. With DEFOG, we suggest an integrated framework that combines the capabilities of traditional vector-based graphics editors with higher-level information visualization techniques and programmatic data manipulation. Our goal is to support the data analysis and visualization process from exploration to publication.

At the root of DEFOG is a simple, yet powerful model that couples graphical displays or *faces* with data objects. Thus, an element of a scene contains standard information about graphical display like position and size, but it also can be linked to a data object. This link can inform the properties of the graphical element; instead of specifying a static color, we might color a node based on a categorical attribute of the linked object. These data objects can contain a variety of attributes and serve to group other data objects, and they can also represent connections between other elements. In addition, faces are not limited to simple glyphs or textures. When a data object contains other data objects, the associated face might be configured to display, for example, a scatterplot of the child nodes. Thus, the DEFOG model provides a framework to quickly create complex visualizations but also tweak each property on an individual element.

DEFOG provides a variety of tools to support the entire data exploration, visualization, and publication process. There are tools to ingest data and display the available attributes of the objects. For quick data manipulation, DEFOG provides a Python console that also interacts with the objects displayed on the canvas. It provides high-level programs to generate visualizations and plots, while also allowing changes to the display of individual scene elements. Most importantly, almost all parameters can be set using expressions that reference attributes of the data. Selection can be accomplished manually or using the powerful expressions, and there is a suite of alignment and layout commands to polish presentation graphics.

While DEFOG can implement standard visualizations, those results need not be the end of the story. A visualization in DEFOG is not static and can be modified by moving or changing the representation of *individual* objects. For example, as illustrated in Figure 1, a user may find that a node-link diagram doesn't offer the quantitative comparisons needed. In order to better understand the local trends in each, one might select each subset and plot the variables individually. Note that this resembles some of the focus+context [8] ideas, but here, there are no constraints. At the same time, if the initial layout obscured some important labels, one can modify the visualization by moving entities, or scale text.

DEFOG also allows users to interact with data programmatically. By including a native Python environment with extensions that allow interaction with the canvas elements, users can easily manipulate and transform data while analyzing data. Instead of performing preprocessing offline to generate values to be examined, users can apply functions or scripts to compute values *during* data exploration, and interleave them with visualization operations. DEFOG also supports intuitive expressions for combining different properties of the data.

By integrating visualization techniques with the ability to manipulate both data and graphical representations, DEFOG allows more freedom to investigate and explore data. While users can construct complex visualizations using simple faces, the set of faces to represent data visually is extensible. In addition, with DEFOG it is possible to use specialized ways to visualize data that are not available in standard visualization tools. We show that such representations lead to unique views of data that can potentially lead to better insight.

The remainder of the paper is organized as follows. In Section 2, we review work that has addressed different aspects of the DEFOG model and system. The DEFOG model, its components and manipulation language are described in Section 3. The implementation details and unique features of our system are presented in Section 4. In Section 5,

Lauro Lins, David Koop, Juliana Freire and Cláudio Silva are with the Scientific Computing and Imaging (SCI) Institute at the University of Utah. email: {lauro, dakoop, juliana, csilva}@sci.utah.edu.

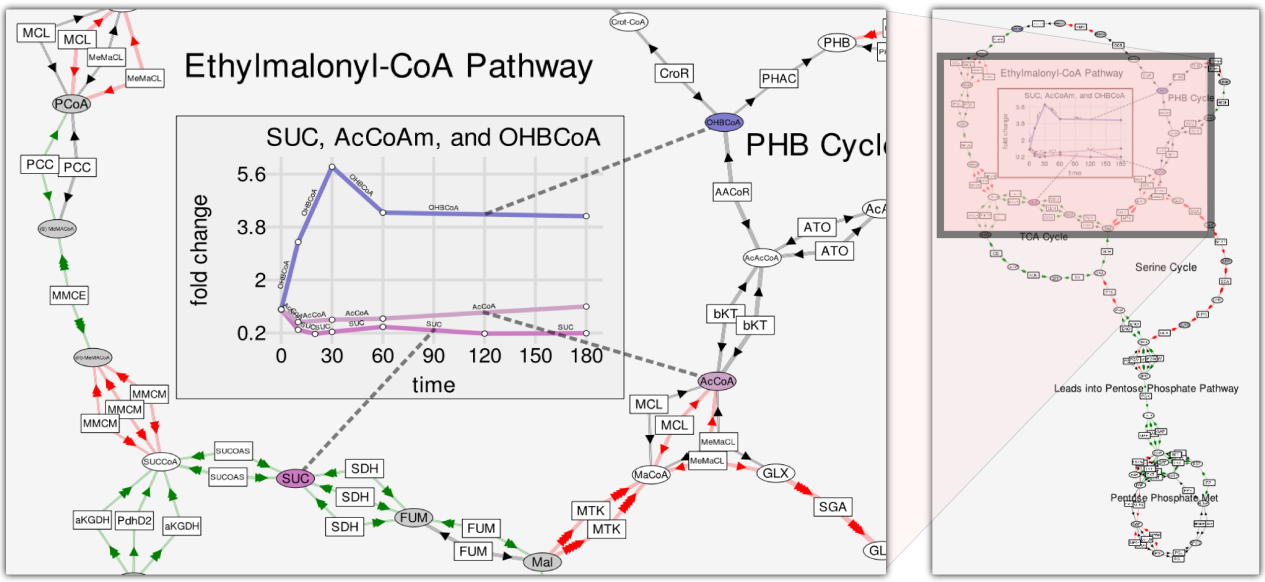

Fig. 1. Node and link representation for biopathways combined with a plot of the concentration fold change (relative to time zero) for three metabolites.

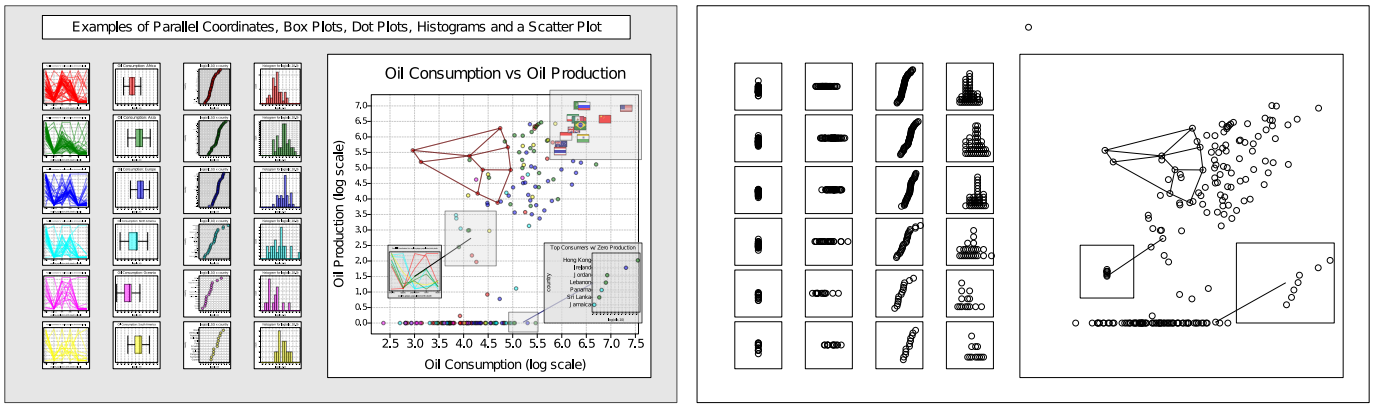

Fig. 2. Faces play a major role in defining how data is visualized. The visual representation of the canvas using different visualization techniques as faces for the various groups is shown on the left. The corresponding faceless representation used by the DEFOG model to represent this information is shown on the right.

we discuss scenarios and give specific examples where DEFOG has helped in obtaining insights. In Section 6, we conclude and discuss directions for future work.

2 Related Work

There has been much significant work in the individual areas of data analysis and manipulation, visualization systems, and frameworks and applications for creating and editing graphical scenes. However, there has been less work on the integration between these areas. Yet building a visualization of data in graphical design programs like Inkscape or Adobe Illustrator requires much tedious conversion from data values to graphical primitives. Customizing the display of visualizations produced using visualization tools like Tableau or DEVise is often impossible without using another program. Complicated analyses and data manipulation can often be easily accomplished in statistical or programming languages like R or python, but mapping the transformed data to visualizations often requires many extra libraries or exporting the data to another tool. With DEFOG, we desire to provide high-level visualization capabilities while maintaining the low-level ability to manipulate and explore data as well as customize the graphical display. Figure 3 illustrates the space we believe DEFOG fills; note that ease of use is with respect to data visualization.

Data Access & Visualization. One key issue with early visualization systems was access to data, especially when it lived in relational storage. In addition, while databases supported powerful query and filter capabilities, visualization systems were often not built to take advantage of these features. These problems have been addressed by work on visual interfaces for relational data [22,25,36]. DEVise coordinates multiple views of large datasets [22], and Improvise adds live properties and coordinated queries [36]. Snap-Together Visualization coordinates views via primary key actions [25] while Polaris automatically generates multiscale visualizations through data cubes [32]. Unlike these systems, DEFOG adopts a more general, object-oriented data model and it also provides greater flexibility in manipulating the visual representations.

Illustration & Visualization Interfaces. Other systems have sought to integrate of data-centric visualization interfaces with concepts from illustration and design [30, 37]. Visage introduced the *information-centric* interface for presenting information where the basic currency is a data element, not a file or document [30]. The system is built to facilitate the coordination of multiple analysis tools in a visual environment. To that end, visualizations are not static pictures but rather compositions of elements, and users control visualization by dragging and dropping elements in and across frames instead

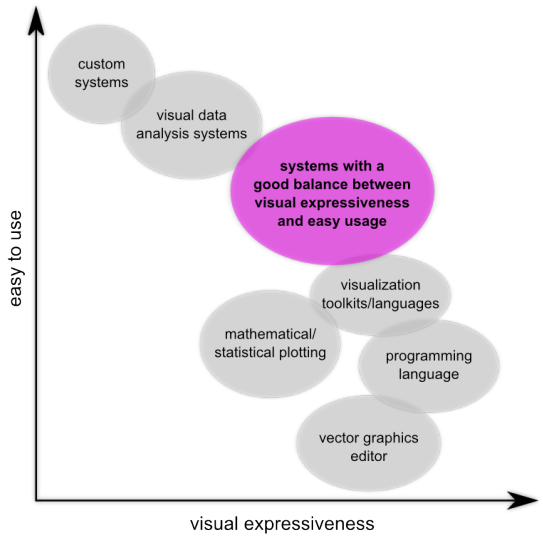

Fig. 3. We believe there is a place for more systems in the highlighted region: a good tradeoff between what data visualizations can be expressed and how easy is to get to those visualizations.

of programming scripts. DEFOG follows this information-centric approach in its design, but uses a single infinite canvas rather than discrete frames.

Tioga-2 introduced a visual environment for coordinating the display of relational data, ensuring that users could see the results of each action and program by direct manipulation [2]. In addition, it introduced the concept of *wormholes* which allow visualizations to be embedded in other visualizations. DataSplash extended this environment with a paint program interface with layers, portals, and layout management [37]. In addition, it defined a difference between data-dependent (splash) and data-independent (trim) objects. The system included the VIQING interface which adds nested visualizations and graphical methods for visually querying and selecting data according to relational principles [26]. Many of the goals identified by these systems overlap with DEFOG's, but DEFOG allows more direct interaction with objects, is not tied to the relational model, and integrates connections into visualizations.

Frameworks & Languages for Visualization. Another key advance in simplifying visualization creation has been made possible by programming toolkits that allow great control over appearance and interaction of data visualizations. The InfoVis toolkit [15] and prefuse [17] have both integrated a variety of techniques into standard, unified frameworks. Many Eyes has extended this concept to a web-based system where visualizations can be created and shared [35]. GUESS allows users to explore graph data and provides users with the ability interact with arbitrary data both visually and via the simplified Gython language [1]. More recently, Protovis has sought to bridge the gap between powerful but complicated graphical languages and easy-to-use high-level visualization systems [7]. DEFOG shares this goal but seeks a solution by coupling language with high-level interfaces so users can also visually interact with the data they are exploring.

One key difference between DEFOG and other data-centric systems is how the system was designed: instead of retrofitting an information visualization tool to allow more flexibility and customization, we started with a vector-based graphics editor, and added information visualization along with data-driven drawing methods. This has naturally led to novel features, such as for example, the ability to specify faces for connections (see Section 3). Another important feature of DEFOG is the ability to programmatically manipulate data using Python, providing users a powerful language to customize access to a wide variety of data sets. Last, but not least, DEFOG integrates and extends a number of features that have been adopted in other systems and that are extremely useful in the context of data exploration and presentation (see details below).

Visualization Models. The DEFOG model builds on work in the modeling of visualization and visual exploration. Mackinlay developed expressiveness criteria and a composition algebra for relational information in graphical presentations [23] rooted in Bertin's graphical objects and relationships [6]. Lee and Grinstein examined different types of visual explorations in the context of databases and classified interactions according to query and visualization operations [21]. They also noted that visualizations can be changed without returning to the data and new queries can be defined from the visualizations. Chi and Reidl presented an operator model for visualization that placed emphasis on the difference between operators that change views versus those that modify the underlying values [11]. Chi also developed a taxonomy of visualization techniques according to the data state reference model and suggested that systems could benefit from standardized operations [10]. Tang et al. addressed data model considerations, generalized interfaces for data access, data transformations concerns, and the difference between scripting and interactive specification [34]. Like this work, the DEFOG model draws on a separation between data and view, but it also maintains a low-level link between graphical primitives and individual data items.

Connecting Visualizations. Using multiple views in order to better understand data is a well-established technique, but accurately displaying and coordinating has been the subject of study. Roberts has argued that multiple views for visualizing the same data can lead to better understanding of data, but many systems were designed with multiple views as an added feature rather than a base component of the system, effectively discouraging users from exploiting them [28]. Brushing and linking across different visualizations has proven effective to coordinate views [4]. In addition, Card et al. have noted that animation can help users track changes in visualizations [9, 29]. Mackinlay et al. developed the spiral calendar and time lattice to maximize screen space by providing multiple views in a single canvas [24]. DEFOG seeks to help users understand relationships between different visualizations by displaying faces for such links and animating transformations in the data.

DEFOG maintains a single, zoomable canvas with all information displayed visually to simplify interaction and allow a user to store and view all of their information in a single view. The Spatial Data Management System implements a visual interface for large collections of data with zoomable views and pictorial displays [13, 19, 20]. Pad and Pad++ provide an infinite, multiscale, hierarchical sketchpad to aid in information creation, sharing, and retrieval [5, 27]. Stolte et al. addressed multiscale visualizations with zoom graphs and the changes in visual and data abstractions as the graph is traversed [33]. The use of the single, infinite canvas also allows users to compose the objects to create images for presentation purposes.

Hierarchies. A hierarchical representation of data is an effective means of organizing large volumes of data, especially when there is a natural categorization. Beyond normal graph representations of hierarchies, there are a number of techniques that effectively organize hierarchical data visually. Treemaps provide a space-filling alternative to graph-based layouts of hierarchical data [31]. Archambault et al. introduce the idea of open, cut, and hidden metanodes for topologically preserving hierarchical data [3]. DEFOG represents hierarchies as groups nested in groups and allows users to construct and modify hierarchies interactively. In addition, it provides open and closed group states, mirroring the ideas from metanodes.

3 MODEL

The DEFOG system is supported by a flexible and extensible model that emphasizes visual display derived from data objects. Like illustration systems, the DEFOG model has a *scene* composed of elements; each *element* has a visual display type and a set of parameters and programmatic specifications to configure appearance. Currently, elements are either *nodes* with position and size information or *connections* which relate two elements—a connection may link connections—and

contain a path between those elements. In addition, any node may contain a set of child nodes.

Unlike standard illustration systems, each element may be linked to an *object*, a set of attributes and methods that encapsulate an entity, as defined in object-oriented programming. Because of these references, the appearance of any element can be configured using expressions that use the attributes and methods of the object. Graphical properties are not defined by points, centimeters, or inches, but by expressions that utilize information from objects. Thus, nodes linked to data objects with country information might be colored according to their continent and sized according to the log of population. We show that this simple model for visualization leads to intuitive and powerful modes of interaction for data exploration and presentation.

3.1 Scene

Like many visualization and illustration frameworks, DEFOG has a *scene* that contains all of its elements. Because the scene is displayed as a single, infinite canvas, the scene may contain elements that represent a variety of data objects from different sources. Each *element* has a *face*, which defines its graphical display; the face may be constructed by a *program* and is configured via a set of *parameters*. In addition, an element may have a link to an *object* with defined data attributes and methods. Then, the face, its program, and its parameters can be configured using attributes and methods from the object. For example, we might configure the face of an element that represents a country with a program that combines the text of is name with an image of its flag. The `flag` and `name` parameters of this program can be set by the `flag_image` method, which reads the flag image based on a country code attribute and returns the image, and the `name` attribute of the referenced object.

Scene elements can be further classified as either nodes or connections. A node may contain other nodes as its children, just as an object may contain other objects. The face, the visual representation of the node, might display individual child nodes or choose to aggregate the information from these nodes in its display. A node also contains a *position* in the scene as well as a *size* measure. Connections relate two elements of the scene; the linked elements might be nodes, connections, or one of each. Like nodes, connections have configurable faces and can be backed by an object. In addition, they may be directed with one element designated as the *source* and the other as the *destination*. Finally, a connection contains a *path* which defines a sequence of points from the source to the destination.

In addition to configuring the faces of individual elements of the scene, users can manipulate the scene by changing positions and paths of single or multiple elements. A user might, for example, choose to move an entire set of nodes to a different position in the scene. Note that manipulation of elements in the scene can affect other elements. For example, changing the position of a node that is connected to another element has the effect of modifying the path of that connection element.

Formally a scene without connections is a sentence in the language generated by the following productions:

$$\begin{aligned} S &\rightarrow ES \mid \varepsilon \\ E &\rightarrow N \mid N(S)S \\ N &\rightarrow n_1 \mid n_2 \mid n_3 \mid \ldots \end{aligned}$$

where S is the scene, E is an element, N is a node. S is the start symbol, and we add the restriction (not in the production rules) that no node is repeated in a single sentence. The parentheses after a node indicates that the nodes inside the parentheses are children of that node. This means that the structure of the nodes of a scene is equivalent to a rooted, ordered tree; the parentheses simply allow us to write this linearly.

3.2 Nodes

As described earlier, a node in DEFOG is composed of two main ingredients: a face and a link to a data object. The graphical *face* includes a geometric shape along with position, size, color, label, and texture. Note that one object might be referenced by multiple nodes, each with its own face, leading to multiple visual encodings of the same object in the scene. A *face* is configured by selecting a program that generates the graphical representation and setting parameters that control the display. A *program* defines the computations used to generate the face including it attributes like geometry, position, and color. DEFOG includes, for example, programs for generating scatterplots, histograms, and tag clouds. Programs may also be user-defined, providing a flexible means to add new visualization techniques to DEFOG without requiring changes the interface. *Parameters* may be exposed in the definition of the program and set using arbitrary expressions that may reference the underlying object. While a face can be configured statically, the power of the model comes from being able to utilize attributes and methods from the object. One might configure a group of nodes to be colored according to any attribute or the result of some method called for each object.

Another important feature of nodes is their ability to encapsulate other nodes. As in object-oriented programming, a node can include lists of child objects. In addition, users may identify such groups manually by selecting a set of nodes in the scene and performing a group operation. In the scene description sentence $n_1(n_2n_3(n_4))$, the children of n_1 are n_2 and n_3; n_2 has no children; and n_3 has a single child, n_4. A face is *open* when both the face and its children are rendered, and *closed* children are not rendered. Figure 4 illustrates the idea of open and closed nodes for a histogram. Note that open faces allow individual child nodes to be selected, moved, or modified.

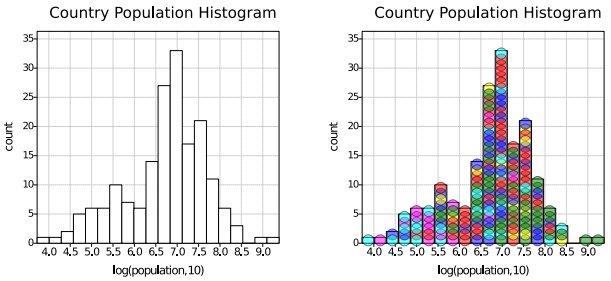

Fig. 4. Example of histogram as a closed node (left) and as an open node (right).

Figure 5 is an example of a face of n_1 configured with a scatterplot program. This scatterplot program expects the two parameters *x_value* and *y_value* to be configured. In the example, we define these expressions to be $\log_{10}(\text{mean}(oilc))$ and $\text{mean}(explife)$, respectively. Note that the scatterplot program evaluates and positions the *children* of n_1 according to the values of these expressions; some programs use the object itself while others use the object's children. The face generated for n_1 according to the program contains axis lines, labels, and a title, and positions the children nodes according to the evaluated expressions. Figure 6 shows the result of applying a parallel coordinates program configured with the expressions $\text{mean}(explife)$, $\text{mean}(oilc)$ and $\text{mean}(gdppc)$ to the scene shown on the left of Figure 5. In this case, the face and transformation of the seven nodes were modified to reflect the requirements of a parallel coordinates plot.

Note that programs can utilize data from a node's linked object, its children, or even its ancestors. In the programs we have implemented, only properties of the linked object or the node's children are used, but in theory, the ancestors of a node could also be used. This would allow propagation of properties so that data could be normalized. For example, with a dataset where each country node contains its provinces as children, we could define programs that color provinces according to their countries, specified by their parent nodes. In addition, because the parameters are separate from the program specification, a user can easily update the expressions used without changing the program. Finally, some operations including scaling, moving, or rotating individual nodes can be accomplished manually, without running a program.

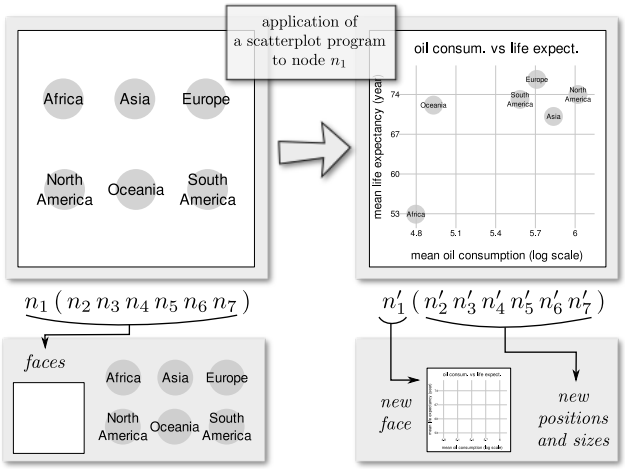

Fig. 5. Application of a scatterplot program to node n_1 configured to plot the mean oil consumption in log scale and mean life expectancy for the children of n_1. The resulting scene is shown on the right. Note that n_1 was updated to n_1' with a new *face* and the children nodes were updated to new locations and sizes.

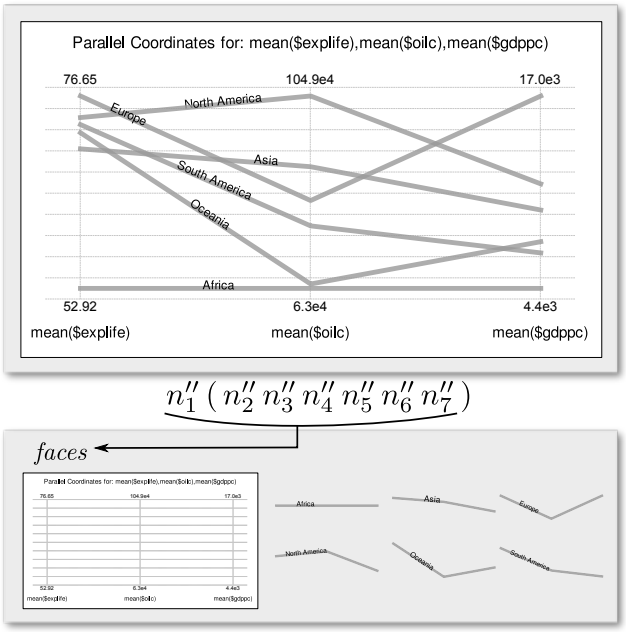

Fig. 6. Application of a parallel coordinates program to node n_1 of the left scene on Figure 5. The program is configured to plot the mean life expectancy, mean oil consumption and mean GDP per capita for the children of n_1. Note that all nodes had their faces modified.

3.3 Connections

Visualizations often encode relationships between elements, and DEFOG provides support for customized display of these connections. The most basic representation of a connection is a line connecting the faces of the two linked objects, but DEFOG allows connections to be displayed according to properties of the objects they connect or properties of the relationship itself. For this reason, a connection, like a node, is an element with a face as well as a link to the object defining the relation. In addition, it contains pointers to the *source* and *target* elements—either may be a node or connection. Also, the source and target may be the same element; Figure 7 contains such loops. While a connection's face is similar to that of a node, it also contains a *path* describing the geometry of a route between the connected elements. Including path definition as a separate attribute allows users to manually alter the route and programs to incorporate and stylize the existing path. For example, a program for a connection might configure the width of a path and add perpendicular slashes based on parameters that are set using attributes and methods from the underlying object.

Figure 7 shows a visualization generated as part of a cancer study where scientists are searching for evidence of correlations between pairs of cancer types. In this example, a connection relates two cancer types (ordering is important) and has a measure of the strength of the relationship based on observed evidence. The figure shows these relationships both as *connections* in a node-link diagram (right) and as *nodes* in a dot-plot (left) of each relationship against the strength of evidence. Because connections are elements, and have a linked data object, they can also be treated as nodes. In addition, this example shows *cross-visualization* identity relationships; the magenta lines show how where top three relationships in the dot-plot are located in the node-link diagram. Finally, the face for the connections in the node-link diagram is a labeled line, and we have lines in both directions because the relationships are ordered.

To specify the order in which connections should be rendered we associate a node and an *order* parameter ("before" or "after") to each connection. When rendering the scene with connections, before rendering a node, we check if there are connections associated with that node and draw them before or after that node is rendered depending on the order parameter. Connections will move up or down in the rendering pipeline by updating their associated node or order.

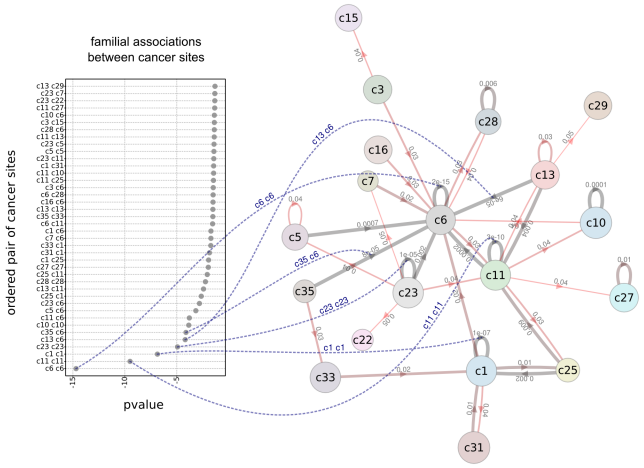

Fig. 7. Relation between cancer sites represented as a node element inside the dot plot and as a connection element on the node-and-link diagram on the right. The blues connections link the two representations for the cancer sites relations with the five smallest p-values. The blue connections are the result of a DEFOG query of which elements are representations of the same entity (object) for the bottom five nodes on the dot plot.

3.4 Expressions and Scripting

As noted earlier, one key feature of DEFOG is that faces can be configured based on data attributes. Furthermore, these parameters can be computed using arbitrarily complex expressions that reference one or more attributes from the underlying object or even objects referenced by that object. This flexibility is possible because DEFOG manipulates objects and attributes as they exist in a programming language rather than using limited representations like those for relational data or specific file formats (see *e.g.,* [36]). Thus, it is irrelevant how or

even if data exists on disk—DEFOG can be used to explore transient, in-memory objects created during the course of analysis. DEFOG provides a set of load routines for ingesting data, and these can be easily extended by users to support other formats. Additionally, the full functionality of the underlying language can be utilized for manipulating objects in addition to specifying parameters of the graphical representation. For example, temperature data read in Fahrenheit might be converted to Celsius before attempting any visualization by scripting that conversion in DEFOG.

Allowing arbitrary data manipulation and parameter specification provides a large amount of power to DEFOG, but we also wish to balance this power with a simple syntax for non-programmers. Thus, the DEFOG model works best with languages that allow expressions to be written in familiar and straightforward syntax like, for example, Python. In order to further simplify syntax, we amend the language with built-in shortcuts to access the current object, its attributes, and its children. The object can be accessed with the underscore character (_) so if that object represents a country, its population attribute can be accessed as `_.population`. Furthermore, we provide `$` as a shortcut for `_.` to make the syntax more friendly. Children of a node can be accessed via the `_children` attribute via `$_children`. Then, a user might compute the log of the mean oil consumption for a county as

```
log(mean($oilc))
```

where `oilc` is the field of the object that stores the oil consumption.

4 IMPLEMENTATION

The DEFOG system implements the model described in Section 3 and provides a set of tools and commands to manipulate the data and visualizations. In this section, we present an overview of the interface, describe some of the available interactions, and highlight significant features of the system.

The DEFOG user interface is built around a single, infinite canvas used to the scene defined by the model. As Figure 8 illustrates, the canvas is surrounded by a list of available objects, available attributes of the currently selected objects, a list of available faces, an interactive console, widgets for configuring color and size, and a list of commands and keyboard shortcuts. In addition, there is a tool bar at the top of the window with common operations and a status bar at the bottom of the window that displays information about the canvas, the current selection, and recently executed commands. All of these widgets can be hidden in order to maximize the screen space for the canvas.

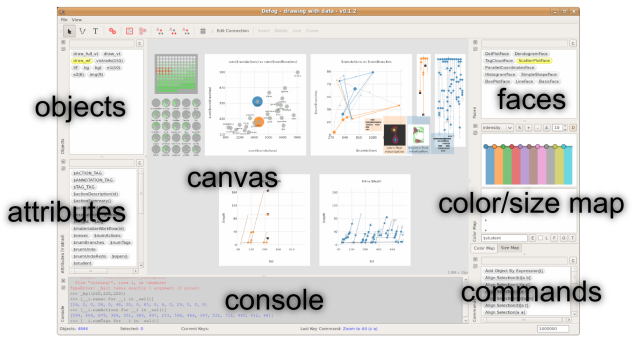

Fig. 8. Screenshot of DEFOG prototype with main widgets: Canvas, Console, Objects, Attributes, Faces, Commands, Color/Size Map.

When data is loaded as the result of either a defined load routine or an assignment in the interactive console, it is displayed in the object list. Users then can select data from that list and drag it into the canvas. By default, the data appears as a group of circles, but user can change this appearance by configuring the faces of the elements and applying any of a set of layout routines to the elements. Upon selecting any element, DEFOG displays the available attributes of the object. As described in Section 3, attributes can be used to configure color, size, or specific parameters in more complex programs. When more than one element is selected, DEFOG displays the intersection of the attributes from each element. Thus, if elements from different datasets share similar attributes, they can be customized at the same time.

While more structured visualizations require programs, DEFOG provides accessible controls for configuring the textual label, color, and size of faces. Labels can be easily added to selected elements by identifying the attribute to serve as the label. The key component of both of the color and size configuration widgets is the expression field which allows arbitrary expressions that can involve attributes of the selected elements. A user can drag and drop attributes from the attribute palette to the expression field. After identifying an expression, the user can view the ordered distribution of the values evaluated for the elements, and configure colors or sizes based on this distribution. After obtaining a satisfactory map from values to colors or sizes, the user can apply it to the selected elements.[1]

Manipulating Sets of Elements. DEFOG also has selection operations, copy-and-paste, and grouping mechanisms, more often seen in illustration tools than visualization systems. Users can select data from the scene directly or use filters than leverage the expression syntax described in Section 3.4 to compute selections. In addition, users can easily explore visualization configurations by copying the current version, pasting a second version, and then changing the modality, color schemes, or any other properties. Finally, DEFOG provides methods to define hierarchical structure on nodes by allowing users to group any selection and create a new node whose children are the selected nodes. While some data may natively exist in these hierarchies, for flat data, users may wish to define groupings to allow programs do better aggregation.

Since all visualizations live in a single canvas, selections can span more than one visualization. Like other systems (*e.g.,*Polaris [32,36]), we allow users to both interactively select canvas elements and query for elements that represent data with certain properties. However, we also allow such queries to further refine already selected entities. A user can then select the set of elements in a single visualization on the canvas and run a query the selects a subset of those points that match specified criteria. Note that if there is no selection, the query is global and will select *all* elements whose underlying objects satisfy the query. Selections can also be copied and pasted to other regions of the canvas allow for non-destructive exploration.

In addition, a selection in DEFOG can be organized as a group which functions in a manner similar to illustration programs. The system allows a user to select the direct children or parents of items in the selection. Additionally, they can select orphan items (those that do not belong to any group) and leaves (objects that are not groups). This gives a user an intuitive method for drilling down into a visualization. For example, consider two groups that contain a set of plots whose configuration we wish to update. By first selecting the two groups and then selecting the children of the selection, we will have selected all of the plots and can then configure all of them at the same time. If a face is open and we have selected both a group and its children, we can select only the group by sub-selecting the orphans.

Layout and Navigation. While the default layout of data elements is an ordered grid, DEFOG provides an array of methods for layout and arrangement. Users can manually align elements as they would in an illustration program, but can also use layout algorithms that respect connections to generate trees, graphs, and other node-link diagrams. Such layout tools make it possible to integrate plots and graphs with faces based on customized graphics.

Finally, DEFOG contains commands to navigate the canvas. While a single, infinite canvas provides an unlimited working environment, it can also lead to information overload. To alleviate these issues, DEFOG contains zoom operations that allow users to focus on current work or jump to a particular region in addition to the normal zoom in/out commands. In addition, there are scaling operations to ensure that visualizations can be resized to align with others that may have been created at a different resolution.

[1]The video shows how this works.

Customized Visualizations. Many visualization systems facilitate analysis and produce initial results but require additional polishing in an illustration tool for presentation. Because DEFOG allows a wide range of customization for the appearance of both individual data elements and relations, users can design presentation graphics as refinements to existing visualizations. Even during exploration, it can be beneficial to annotate, highlight, and even move elements. Annotations can help draw attention to specific points or tell a story about a particular data point. DEFOG allows simple annotations that can aid in chronicling the exploration process. Highlights serve to draw attention to specific regions of a visualization, and can help show specific pieces in the context of the entire visualization. Finally, DEFOG allows users to define and import path specifications to construct faces with complex glyphs.[2]

Because the model defines a relationship between two canvas elements (*e.g.,*two objects) according to the data they represent, DEFOG can locate and link equivalent entities. There has been a significant amount of work on linking multiple-view visualizations and a number of techniques that help draw attention to equivalent data points in different views. For example, brushing allows an unobtrusive means to interactively locate equivalent data across views [4]. Our approach is explicit, as we represent relationships by drawing lines between equivalent data. Note that this is possible because all visualizations live in a single canvas. Such lines resemble parallel coordinate visualizations as shown in Figure 7. This allows users to find, locate, and save relationships between equivalent data in *different visualizations*.

Programming Language Support. DEFOG is designed to take advantage of support from existing programming languages. Rather than define new syntax, users can interact with their data via syntax from an existing language. The associated data of an element can therefore be stored as an object in the language. In addition, DEFOG provides a console where users can manipulate the data using the full power of the language, and move analysis from the visual domain to the scripting domain and vice versa. Any variable is automatically displayed as an available element, and users can use commands like `_sel` to obtain the objects that are selected in the canvas. Our implementation of DEFOG extends Python for language support, but it should be possible to add support for other languages as well.

5 EVALUATION

5.1 Biochemical Pathways

Scientists who are collecting and analyzing biochemical pathways have used DEFOG to simplify the creation of visualizations showing the relationships between the pieces of the pathways. Among chemical reactions occurring within a cell, some sequences of reactions behave as a production chain: an initial substrate (molecule) is transformed by some reaction, its product (another molecule) is transformed by a second reaction and so on. This process continues until a final product is obtained. A sequence of biochemical reactions that exhibit this behavior is called a *biochemical* or *metabolic pathway*. Scientists often use node-and-link diagrams to analyze this kind of data where a variety of information is encoded using arrows to indicate pathway direction and color on both nodes and links to represent attributes like reaction intensities and molecule concentrations. In addition, they use statistical plots including scatterplots and histograms to analyze specific relationships between attribute values.

Figure 1 shows an example outcome when analyzing and visualizing this data in DEFOG. In the figure, rectangular nodes are reactions and oval nodes are metabolites (molecules). The number of arrows in a connection indicates how intense the reaction is compared to a baseline at time zero (red indicates higher intensity, green shows lower values, and black is equivalent). This diagram was generated by loading the nodes and connections data and using the Neato layout from GraphViz [16], available in DEFOG. The figure also shows a plot which examines the fold change of three specific metabolites over time. Because DEFOG allows access to objects contained in objects via the attributes list, we can extract measurements over time from the three metabolites (`SUC`, `AcCoAm`, and `OHBCoA`) as new nodes in the scene. Then, by grouping the extracted measurements, we can generate a scatterplot that relates the time attribute of each measurement with the fold change. By connecting the nodes, we can emphasize trends in each metabolite, converting the scatterplot to a line plot. Finally, to better present this information, we have connected the nodes in our original node-and-link diagram to the corresponding lines in the line plot.

The scientists were thrilled that instead of taking days to manually create a diagram in PowerPoint as they were before, they could use DEFOG to create the diagrams in seconds and also further explore the data in the context of these diagrams. At the same time, learning to harness the power of DEFOG took some adjustment, and we aided their transition by providing extensions to simplify data specification and visual encoding. Because much of the pathways data was already saved in spreadsheets, we wrote a specific load routine to ingest this data and pull out annotations that indicated connectivity information. To simplify the examination of module concentrations and reaction intensities over time, we extended DEFOG with widgets to explore these variables over time. Both of these extensions streamlined the exploration and visualization process, and we believe that such extensions can be important aids for domain-specific research.

5.2 Histories of Workflow Design Processes

A workflow is a description of computational steps to transform input data into output data. To evaluate DEFOG we used it to analyze a collection of such descriptions. The collection was generated by a workflow management system where all the modifications a user did, starting from an empty workflow, to a solve given tasks was recorded. For this reason we say it is a collection of histories of workflow design processes. The dataset consisted of 150 histories from 25 users and 6 tasks. It has been shown that such histories can be used to gather information about patterns of use [18].

A workflow description in this dataset is modeled as a directed acyclic graph and each history is modeled as tree structure whose nodes contain workflow graphs. This kind of "complicated" data types makes it hard to explore such a dataset with any other tool besides a low level programming language or custom application. It turned out that, as soon as we created objects to represent this data, DEFOG was ready to analyze and generate visualizations like the one shown in Figure 9.

On the top left of this figure each of the 150 histories is represented by a circle with its color meaning the task: from task one (white) to task six (the most saturated green). On the bottom left we copied and pasted the 150 top left circles/histories and put them into groups: one for each student. However the circles this time were not the same size, they were scaled to match the number of workflows contained in each history. In the context of this application this number is also referred as: number of actions. On the leftmost plot we copied the student groups and put them in a scatterplot using for each group the sum of the number of workflows and the number of branches for the six different task-history trees within each group. In this plot we highlighted the users Lon and Kaylee because they had a similar total number of actions, but Lon had significantly less branches than Kaylee. On the second plot (from left to right) we show the actual number of workflows (or actions) in each of the six histories for both Lon and Kaylee. The colored arrow lines are there to show the task progress: from one to six. The nodes also grow from task one to task six. The gray lines connect the histories with the same task number. Note how the blue dots are always above the corresponding orange nodes: more branches for Kailee. On the left right we the actual history tree for task six of Lon and Kaylee. The one that is indicated by the gray curved line on the second scatterplot. On the tree we highlight by using larger nodes what are called "branch" nodes and the ones that actually contained a successfully executed workflow. This were actually represented by a thumbnail image of the outcome of the workflow execution. In this collection the six tasks were all to design scientific visualization workflows. The final visualization of Lon and Kaylee for task six are high-

[2]See the video for an example.

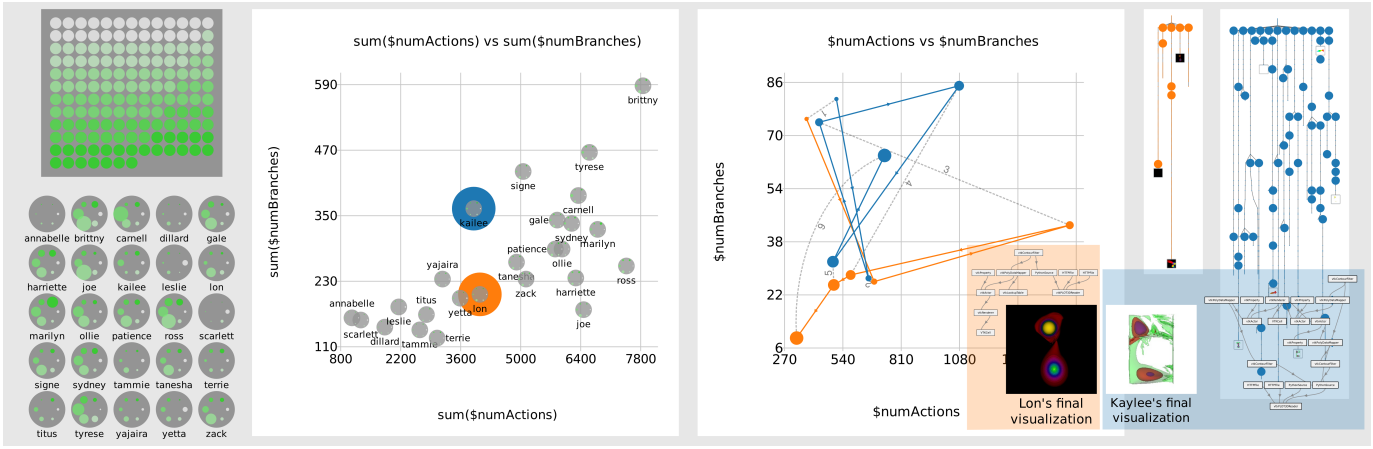

Fig. 9. DEFOG scene of a visual analysis session on a dataset of histories of workflow design: multiple levels of abstraction investigated in the same place.

lighted together with the actual workflow graphs that generated it.

Observing Lon and Kailee's visualizations we noticed that Kailee couldn't get it right. Although she generated a larger history tree and more branches than Lon, we suppose that this was due to confusion. Another question we raised by this example of Lon and Kaylee was if trees that contain a deeper single branch correlate with better workflows. Although Kaylee had more workflows she never had a branch as deep as one branch Lon had.

6 CONCLUSIONS

DEFOG provides a flexible environment for building, composing, and tweaking visualizations, helping users effectively explore data. It is built upon a simple model that supports a significant amount of freedom to manage and explore data visually. This model is coupled with a single, infinite canvas where graphical primitives backed by data can be manipulated and a variety of powerful commands that allow users to organize and present data according to these visual representations. We have shown that this implementation can help users better understand their data in two different case studies.

While we believe DEFOG provides good abstractions and interactions to simplify the creation of visualizations, rampant customization must require time and we cannot expect to eliminate user-interaction. We can allow a user to customize each element individually, but such operations will not leverage the group operations and mappings that make DEFOG more efficient for grouped customization. In addition, while DEFOG seeks to occupy a space that offers high-level as well as low-level manipulations, extensions or applications biased to one side or another devolve into existing approaches. Thus, depending on use, DEFOG can span large space in the graph shown in Figure 3.

We are working to improve DEFOG by adding features including better layout support and provenance capture. For layout, better support for alignment as well as level-of-detail configuration [37] would allow users to organize and layer information. We also wish to add provenance capture of the data-oriented exploration in DEFOG. It has been shown that information about the steps followed during exploration can be used not only to recreate plots but for analysis [18]. In addition, such provenance can be used to apply similar processing via analogy or suggest processes for analysis automatically. We believe that DEFOG can benefit in similar ways, leading to a better understanding of exploration that can help improve both the tool itself and also visualization techniques.

Acknowledgments. The research and development of the Defog system has been funded by the National Science Foundation under grants IIS-0905385, IIS-0844546, IIS-0746500, CNS-0751152, the Department of Energy (SciDAC VACET and SDM centers), and National Institutes of Health (NCRR ARRA).

REFERENCES

[1] E. Adar. Guess: a language and interface for graph exploration. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 791–800, New York, NY, USA, 2006. ACM.

[2] A. Aiken, J. Chen, M. Stonebraker, and A. Woodruff. Tioga-2: a direct manipulation database visualization environment. In *Data Engineering, 1996. Proceedings of the Twelfth International Conference on*, pages 208–217, Feb-1 Mar 1996.

[3] D. Archambault, T. Munzner, and D. Auber. GrouseFlocks: Steerable Exploration of Graph Hierarchy Space. *IEEE Transactions on Visualization and Computer Graphics*, pages 900–913, 2008.

[4] R. Becker and W. Cleveland. Brushing Scatterplots. *Dynamic Graphics for Statistics*, pages 201–224, 1987.

[5] B. Bederson and J. Hollan. Pad++: a zooming graphical interface for exploring alternate interface physics. In *Proceedings of the 7th annual ACM symposium on User interface software and technology*, pages 17–26. ACM New York, NY, USA, 1994.

[6] J. Bertin. *Semiology of graphics*. University of Wisconsin Press, 1983.

[7] M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15:1121–1128, 2009.

[8] S. K. Card, J. D. Mackinlay, and B. Shneiderman, editors. *Readings in information visualization: using vision to think*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[9] S. K. Card, G. G. Robertson, and J. D. Mackinlay. The information visualizer, an information workspace. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 181–186, New York, NY, USA, 1991. ACM.

[10] E. Chi. A taxonomy of visualization techniques using the data state reference model. In *Proc. of IEEE Information Visualization*, pages 69–75, 2000.

[11] E. H.-H. Chi and J. Riedl. An operator interaction framework for visualization systems. In *Proc. of IEEE Information Visualization*, pages 63–70, 1998.

[12] S. Cohen, J. Cukier, and M. Wattenberg. Panel: Changing the world with visualization. InfoVis 2009.

[13] W. Donelson. Spatial management of information. In *Proceedings of 1978 ACM SIGGRAPH Conference*, pages 203–209. Press, 1978.

[14] M. Ericson. Keynote: Visualizing data for the masses: Information graphics at The New York Times. InfoVis 2007.

[15] J.-D. Fekete. The infovis toolkit. In *Proc. of IEEE Information Visualization*, pages 167–174, 2004.

[16] E. R. Gansner and S. C. North. An open graph visualization system and its applications to software engineering. *Softw. Pract. Exper.*, 30(11):1203–1233, 2000.

[17] J. Heer, S. K. Card, and J. A. Landay. prefuse: a toolkit for interactive information visualization. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 421–430, New York, NY, USA, 2005. ACM.

[18] J. Heer, J. Mackinlay, C. Stolte, and M. Agrawala. Graphical histories for

visualization: Supporting analysis, communication, and evaluation. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1189–1196, Nov–Dec 2008.

[19] C. Herot. Spatial management of data. *ACM Transactions on Database Systems (TODS)*, 5(4):493–513, 1980.

[20] C. Herot, R. Carling, M. Friedell, and D. Kramlich. A prototype spatial data management system. *ACM SIGGRAPH Computer Graphics*, 14(3):63–70, 1980.

[21] J. Lee and G. Grinstein. An architecture for retaining and analyzing visual explorations of databases. In *IEEE Visualization*, pages 101–108, 1995.

[22] 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. *SIGMOD Rec.*, 26(2):301–312, 1997.

[23] J. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Trans. Graph.*, 5(2):110–141, 1986.

[24] J. D. Mackinlay, G. G. Robertson, and R. DeLine. Developing calendar visualizers for the information visualizer. In *UIST '94: Proceedings of the 7th annual ACM symposium on User interface software and technology*, pages 109–118, New York, NY, USA, 1994. ACM.

[25] C. North and B. Shneiderman. Snap-together visualization: a user interface for coordinating visualizations via relational schemata. In *AVI '00: Proceedings of the working conference on Advanced visual interfaces*, pages 128–135, New York, NY, USA, 2000. ACM.

[26] C. Olston, M. Stonebraker, A. Aiken, and J. Hellerstein. VIQING: Visual Interactive QueryING. *Visual Languages, IEEE Symposium on*, 0:162, 1998.

[27] K. Perlin and D. Fox. Pad: An alternative approach to the computer interface. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 57–64. ACM New York, NY, USA, 1993.

[28] J. Roberts. On encouraging multiple views for visualization. In *Proc. of IEEE Information Visualization*, pages 8–14, 1998.

[29] G. G. Robertson, S. K. Card, and J. D. Mackinlay. Information visualization using 3d interactive animation. *Commun. ACM*, 36(4):57–71, 1993.

[30] S. Roth, P. Lucas, J. Senn, C. Gomberg, M. Burks, P. Stroffolino, J. Kolojejchick, and C. Dunmire. Visage: a user interface environment for exploring information. *Proceedings of Information Visualization*, pages 3–12, 1996.

[31] B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Trans. Graph.*, 11(1):92–99, 1992.

[32] C. Stolte, D. Tang, and P. Hanrahan. Polaris: a system for query, analysis, and visualization of multidimensional relational databases. *Visualization and Computer Graphics, IEEE Transactions on*, 8(1):52–65, Jan/Mar 2002.

[33] C. Stolte, D. Tang, and P. Hanrahan. Multiscale visualization using data cubes. *Visualization and Computer Graphics, IEEE Transactions on*, 9(2):176–187, April-June 2003.

[34] D. Tang, C. Stolte, and R. Bosche. Design choices when architecting visualizations. In *Proc. of IEEE Information Visualization*, pages 41–48, 2003.

[35] F. Viegas, M. Wattenberg, F. van Ham, J. Kriss, and M. McKeon. Manyeyes: a site for visualization at internet scale. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1121–1–128, Nov.-Dec. 2007.

[36] C. Weaver. Building highly-coordinated visualizations in improvise. In *Proc. of IEEE Information Visualization*, pages 159–166, 2004.

[37] A. Woodruff, C. Olston, A. Aiken, M. Chu, V. Ercegovac, M. Lin, M. Spalding, and M. Stonebraker. Datasplash: A direct manipulation environment for programming semantic zoom visualizations of tabular data. *Journal of Visual Languages & Computing*, 12(5):551–571, 2001.