Journal of Visual Languages and Computing (2001) **12,** 551–571
doi:10.1006/jvlc.2001.0219, available online at http://www.idealibrary.com on IDEAL®

DataSplash: A Direct Manipulation Environment for Programming Semantic Zoom Visualizations of Tabular Data

Allison Woodruff*, Chris Olston†, Alexander Aiken, Michael Chu, Vuk Ercegovac‡, Mark Lin, Mybrid Spalding§ and Michael Stonebraker‖

University of California, Berkeley, CA, U.S.A.

Received 1 January 2000; revised 1 June 2001; accepted 5 July 2001

We describe DataSplash, a direct manipulation system for creating semantic zoom visualizations of tabular (relational) data. DataSplash makes contributions in three areas that are key to the construction of such visualizations. First, DataSplash helps users graphically specify the visual appearance of groups of objects. Second, the system helps users visually program the way the appearance of groups of objects changes as users browse the visualization. Third, DataSplash allows users to create groups of graphical links between canvases. These direct manipulation facilities simplify the process of constructing semantic zoom applications, particularly ones that display large data sets.
© 2001 Academic Press

Keywords: database visualization, direct manipulation, information visualization, semantic zoom.

1. Introduction

Semantic zoom is a technique for displaying different graphical representations of objects [1, 2]. This technique lends itself particularly well to the visualization of large amounts of *tabular data*, which for the purposes of this paper we define as data that can be stored in the rows and columns of a relational database table. Furthermore, semantic zoom visualizations with nesting are well suited for data sets consisting of multiple related database tables. However, semantic zoom applications both with and without nesting can be difficult and tedious to program, and the difficulty increases as the amount of data increases. In this paper, we introduce DataSplash, a direct manipulation environment in which users can more easily construct and navigate semantic zoom visualizations of tabular data.

* Corresponding author. Current address: Xerox PARC, 3333 Coyote Hill Road, Palo Alto, CA 94304, U.S.A.
† Currently at Stanford University.
‡ Currently at Quiq, Inc.
§ Currently at datarebels.com.
‖ Currently at MIT.

1045-926X/01/050551+21 $35.00/0 © 2001 Academic Press

Semantic zoom is useful when different levels of detail are desirable. Semantic zoom systems typically use a spatial metaphor in which objects are displayed in a two-dimensional *canvas*. The user views the canvas as though through a camera pointed at the canvas; the user can pan and zoom this camera to view different parts of the canvas. The user's (camera's) height above the canvas is known as their *elevation*. Semantic zoom differs from simple graphical zoom in that different graphical representations are shown as the user changes elevation. The usefulness of this approach is evident in applications such as interactive atlases or Internet map services. For example, suppose a city is represented as a dot when the user is at a high elevation. When the user zooms in on the city, a text label may be added to the representation, or a city map may replace the dot. Similarly, when users view a map for an overview, they may want to see only freeways, but when they are using the map for detail, they may want to see minor roads as well.

Although semantic zoom has proven to be very useful, programming semantic zoom visualizations remains difficult. Semantic zoom has been an active research area and a number of systems support semantic zoom [1–3]. However, developing semantic zoom applications in these systems generally requires a trained programmer. This programmer must specify both the objects that appear in the visualization and their behavior (the way they change as the user zooms), i.e. the programmer must set the graphical representations and sizes for all objects for all elevations.

Not only do semantic zoom systems typically require software development, but the development process itself is often difficult. Choosing appropriate parameters can be very challenging because it can be hard to imagine how the visualization will look when viewed from different elevations and from different positions in the canvas. For example, setting the sizes or other graphical properties of objects can be time-consuming because the apparent size of an object is a function of the browser's elevation. Programmers can find it difficult to extrapolate from the view at one elevation to the view at another elevation. Therefore, they often visit many elevations and experiment with many sizes before choosing a final representation. Further, as objects occupy more display space, it is appropriate to make their graphical representations more complex. Programmers find it difficult to set the transition points between graphical representations without exploring several possibilities. Finally, it can be hard to imagine what combinations of representations will look like, e.g. how will it look to put both rivers and highways together on this map at this elevation? Again, to address this problem, programmers like to experiment with multiple possible visualizations. Because of the issues listed above, creating appealing semantic zoom applications is a highly iterative process. This process can be extremely tedious if the programmer must make a textual programmatic change every time they wish to experiment with a new value.

Tabular (relational) data sets are extremely common and useful, but they exacerbate the existing problems for programmers of semantic zoom applications. Often such data sets are large, so that specifying the graphical representation of each object individually can be extremely time-consuming. Further, the contents of the data sets may change, e.g. the user may create a visualization of today's stock market data and wish to apply it to tomorrow's stock market data when it becomes available. Moreover, the range of values may not be familiar to the programmer. For example, when they assign a mapping between object size and a data value such as price, that mapping may be inappropriate, making objects so large that they occlude each other or so small that they are not visible. Finally, when nesting is used to visualize data sets with multiple tables, the relationships

between the data tables can be complex, so finding the appropriate nested visualization representation is often difficult.

As a result of these issues raised in programming visualizations of tabular data, we determined to build a system that allowed programmers to easily specify the appearance and behavior of types of objects including nested windows. Because semantic zoom applications are hard for the programmer to visualize, and because they are often developed using an iterative process, our general strategy is to use direct manipulation to simplify the construction of semantic zoom applications with optional nesting. Shneiderman identifies the following principles of direct manipulation systems, which are highly appropriate for our task [4]:

- continuous representation of the objects of interest;
- physical actions or presses of labeled buttons instead of complex syntax; and
- rapid incremental reversible operations whose effect on the object of interest is immediately visible.

The use of direct manipulation is suggested by our experience with previous semantic zoom systems, e.g. the Tioga system [5]. Our approach is further validated by programs such as KidPad, MuSE and PadDraw, which have explored the use of direct manipulation to specify the appearance and behavior of individual objects in semantic zoom systems [6–8]. To extend this approach to tabular data, we explore the use of direct manipulation to specify the appearance and behavior of groups of objects (as opposed to individual objects) in semantic zoom systems. We also introduce a novel direct manipulation interface for generating nested visualizations of complex data sets consisting of multiple related data tables. Note that while our techniques are most effective for semantic zoom visualizations of tabular data, many of our ideas extend to other domains as well.

In this paper, we describe DataSplash, a direct manipulation system for creating and interacting with semantic zoom visualizations of tabular data. We also describe an extension to DataSplash called VIQING, a novel direct manipulation interface for creating nested visualizations. We focus on three main design issues that arise in the context of direct manipulation semantic zoom systems that incorporate optional nesting capability:

1. Users want to specify the appearance of groups of objects. To this end, DataSplash provides a *paint program interface* for creating and editing the graphical representation of groups of objects. We call these groups *layers*. This approach was suggested by experience with a previous semantic zoom system, Tioga, in which end users were frustrated because they could not specify the appearance of groups of objects. While other systems allow users to specify the visual appearance of groups of objects [9–11], to our knowledge none of these has confronted the issues that arise in the context of semantic zoom systems, e.g. the interaction between visual appearance and user navigation actions.
2. Users want to specify the behavior of groups of objects during zooming. To meet this need, DataSplash provides a visual program called the *layer manager* that specifies the elevations at which the contents of layers are displayed as the user pans and zooms. In addition to serving as a programming mechanism, the layer manager is

a helpful visualization of the contents of the application: this graphical representation guides the browser to interesting parts of the visualization, a problem identified in [12]. We introduced our first design of this construct as the *elevation map* in [13], and we have subsequently designed and implemented more mature versions in multiple research and commercial systems. Although a related construct for individual objects was introduced in [7], to our knowledge the layer manager is the only method that allows users to visually program the zooming behavior of groups of objects.

3. Users want to create groups of portals. Portals are windows between canvases, which are a convenient mechanism for nesting visualizations. To support groups of portals, the DataSplash paint program interface and the VIQING visual query interface both allow end users to create *replicated portals*. Such groups of portals yield a number of interesting nested visualizations, such as small multiples [14, 15].

Note that DataSplash and VIQING users can perform several types of activities, i.e. they can browse, program and construct complex custom visualizations. While these activities require increasing levels of sophistication on the part of the user, DataSplash and VIQING are intended to be used by individuals with no textual programming knowledge. Further, the interfaces and required skills are similar for the different types of activities, allowing users to gradually transition from simply browsing visualizations to developing custom visualizations. Therefore, we do not generally distinguish developers from browsers in this discussion, usually referring to them both simply as users.

In the remainder of this paper, we describe the major components of the DataSplash system in detail, and we share our experience about the issues that arise in the design and implementation of such a system. In Section 2, we begin with an overview of DataSplash. We then turn to the three critical design points listed above, describing the paint program, the layer manager, and replicated portals in Sections 3–5 in turn. We next discuss implementation and status in Section 6, followed by related work in Section 7. In Section 8, we conclude.

2. System Overview

In this section, we describe the DataSplash system. Our final system design was influenced by experience with building visualizations of data from a number of domains (business, scientific, census and political) as well as feedback from informal user testing. We begin our system overview by describing the canvas and the objects that it displays graphically. We next discuss how users can navigate in the canvas to view its contents. We then describe portals and discuss how portals are used to navigate between canvases. We then present the DataSplash graphical user interface with which users can create canvases and portals. Finally, we discuss the graphical save mechanism with which users can save the applications they create.

2.1. Canvases and Their Contents

In DataSplash, graphical objects appear on a two-dimensional canvas. They can be data-dependent (*splash objects*) or data-independent (*trim objects*). We discuss these types in turn.

Splash objects are generated from database tables as follows. Each row in a given table is assigned an x, y location on the canvas, i.e. the rows are scattered across the canvas, giving an effect similar to a scatterplot. The x, y locations are derived from user-specified linear functions applied to data values in the rows. The default function simply returns the values in the rows. As an example of the default function, if the user has a table of United States (US) cities with latitude and longitude columns, x and y can be assigned to the longitude and latitude values of each city, respectively. More complex linear functions are useful for scaling multiple data sets, so they will be coregistered. We found linear functions to be sufficiently powerful for virtually all our applications.

At any time, the user can create an object in DataSplash's paint program interface and duplicate that object for every row in the database table. As a result of this duplication operation, a copy of the object appears at the x, y location of every row in the table. The effect is like splashing paint across the canvas, coating every scattered row. The user may also associate display properties of objects with columns in the table, e.g. the height, width, color and rotation of each splash object can be derived from linear functions applied to values in the columns of its row. Continuing the example of a visualization of US cities, the user may specify that a circle is to be drawn at the x, y location of each city. The user may further specify that the radius of each circle be proportional to the population of that city to create a display such as that seen on the left side of Figure 1.

Unlike splash objects, trim objects do not have data-dependent position or properties and appear only once on the canvas. Trim objects are useful as, e.g. labels or axes. Trim objects can either be *sticky* (painted on the camera lens) or not sticky (embedded in the

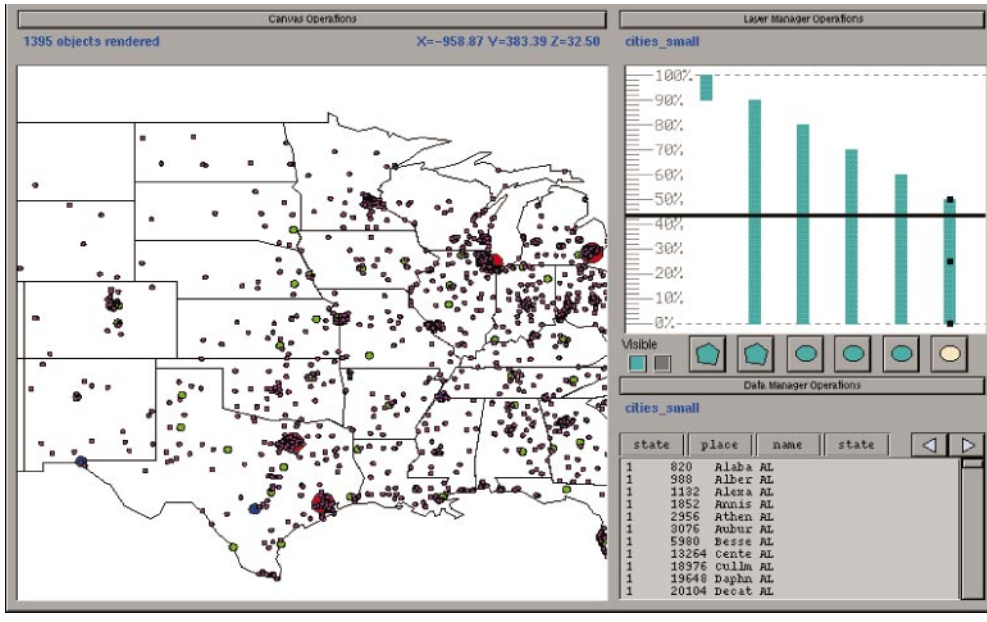

Figure 1. US cities visualization seen from a low elevation

canvas). A sticky, trim object therefore does not move during the panning or zooming actions described below.

A set of objects associated with a given database table is called a *layer*. Multiple layers may be associated with a single database table, i.e. there is a one-to-many relationship between tables and layers. A (possibly empty) set of layers may be associated with a single canvas. Layers exist independently of canvases and therefore can be shared by multiple canvases.

2.2. Canvas Navigation

In this subsection, we discuss the DataSplash navigation model that allows users to view the contents of canvases either while editing and programming the visualization or while simply browsing. In the semantic zoom metaphor, users view the canvas as though through a camera pointed straight down at the canvas. Users can pan, i.e. change the x, y camera position above the canvas. Users can also zoom, i.e. change the z camera position, or elevation above the canvas. Users can select between linear or logarithmic zoom speed. During logarithmic zooming, the zoom speed becomes slower as the camera approaches the canvas. With both linear and logarithmic zooming, objects change representation as users zoom closer to them. This effect is achieved by displaying different layers at different elevations. As discussed below, DataSplash provides a unique mechanism, the layer manager, which allows users to program this behavior.

Users can also teleport directly to specific x, y, z locations, including a *home* position. The contents of the canvas are based on database tables that may not be entirely familiar to the user, e.g. the user may not know the minimum and maximum values in the data set, and therefore, they may not have a sense of the boundaries of the canvas. Therefore, we found it helpful to provide default values for teleporting and navigation. When a new canvas is created, DataSplash samples values in the table to automatically generate a default home position and a default maximum elevation from which the sampled tuples are visible. Users can also bookmark new teleport locations on the fly. Bookmarks and other teleport destinations can be automatically displayed using portals, which are described next.

2.3. Portals

Portals are nested canvases that provide extra visual information about objects and allow users to navigate to other canvases or other locations on the same canvas. This feature is illustrated in Figure 2, which shows a DataSplash visualization of Fortune 100 data (data on America's highest revenue corporations). In the main canvas, a label and a circle represent a company. The circles vary in size by overall profit of the company in the underlying data set. On the left side of the main canvas, there is a portal (outlined in a box) to another Fortune 100 visualization in which circle size is associated with a company's growth instead of profit. The visualization suggests that overall growth is inversely proportional to overall profits for these large American companies.

While this example shows only one level of nesting, portal nesting may be arbitrarily deep. Portals are discussed further in Section 5.

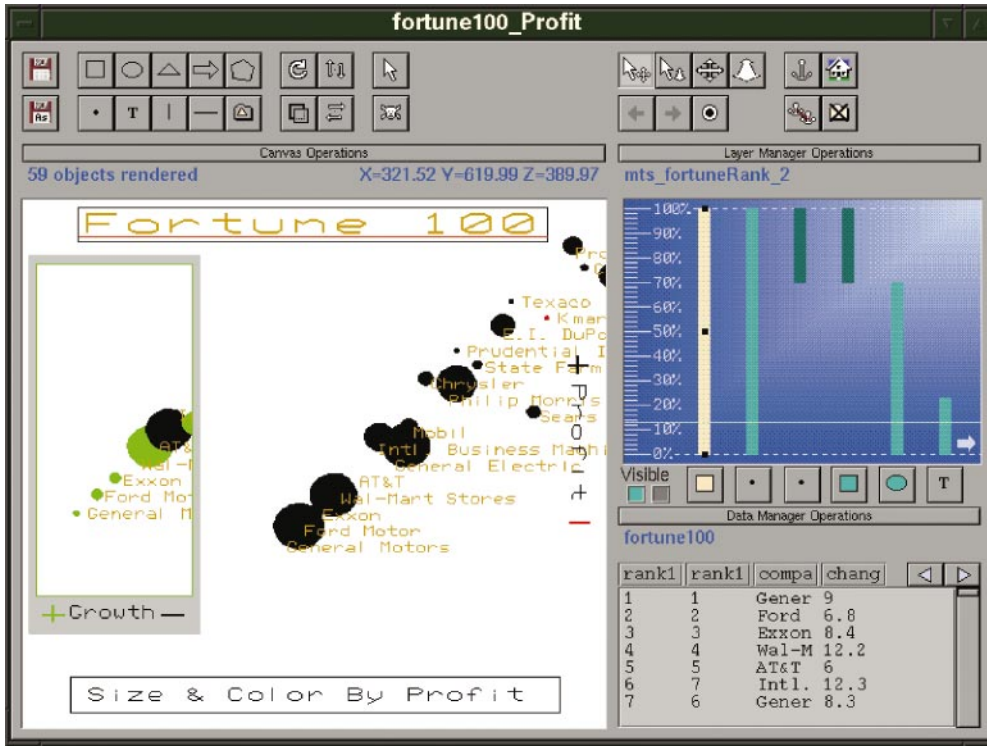

Figure 2. Fortune 100 visualization with a portal

2.4. Portal Navigation

A canvas containing a portal is called a parent canvas. Every portal goes to a canvas which we refer to as the child canvas. The parent canvas is considered to have a spatial relationship to the child canvas. Specifically, every portal is linked to a specific x, y, z coordinate in its child canvas. Therefore, the view through the portal is of the child canvas at that x, y, z location. Portals can be panned and zoomed independently of the parent canvas, allowing users to change the x, y, z offset between the parent canvas and the portal.

If the user is centered on the x, y location of the portal in the main canvas and zooms to zero elevation in the main canvas, the user will metaphorically 'pass through' the portal into the child canvas. At this point, the canvas previously shown in the portal (the child canvas) will become the user's main canvas. In some cases, the child canvas may be the same as the parent canvas. In such cases, passing through a portal is equivalent to teleporting to another x, y, z location in the same canvas. Such *recursive* portals are a convenient way for the system to visually represent bookmarks and other teleport destinations.

In addition to supporting interactive panning and zooming to enter a portal, DataSplash supports a number of facilities for portal navigation. Often users want to go into a portal, but they may find it tedious to navigate through that portal. Therefore,

DataSplash supports portal click, a shorthand for panning and then zooming to zero elevation to go into a portal canvas. Further, DataSplash maintains a list of all visited portal canvases. (This is very similar to the list of links visited in a Web browser.) Pressing the portal back tool takes the user to the previous canvas in the list; portal forward goes forward in the list.

Portals have a number of parameters: *moveable*, *dynamic* and *slaved*. The x, y, z values specifying the relationship between the child and parent canvases combined with portal parameters determine the way the portal is displayed when viewed in the context of the parent canvas. These parameters allow the user to create a number of interesting effects. A taxonomy of these parameters and the way they interact is given in [16]. We briefly highlight some key ideas here.

When a portal is moveable, moving the portal within the parent canvas updates the portal's x, y values. This can be useful to create effects like Magic Lenses [17].

When a portal is dynamic, a line-of-sight effect is created. More precisely, the portal is drawn using a true three-dimensional perspective: the user is looking through a hole in the parent canvas onto the child canvas below in three-space. To imagine this effect, suppose a viewer is standing to the left of a window and looking through it. Now, suppose the viewer walks to the right side of the same window. The viewer will see different views from these perspectives.

When a portal is slaved, the portal x, y, z coordinates are slaved to the parent canvas camera x, y, z position. That is to say, the portal canvas x, y, z values are some function of the parent canvas camera position. For example, the portal in Figure 2 is slaved to the main canvas. Consequently, when the user navigates inside the main canvas that shows profit data, the portal x, y, z coordinates will be automatically adjusted to show the corresponding growth data inside the portal window. This feature is only available for sticky, trim portals.

2.5. Graphical User Interface

In this section, we describe the DataSplash components that allow users to edit and navigate in visualizations. These components are grouped together in a window called a *studio*, which appears in, for example, Figure 2. DataSplash presents the user with one studio per visualization.

The studio contains the following four components:

Canvas: The canvas displays the current visualization. The user can pan and zoom in the canvas. The user can also create and edit objects in the canvas. The canvas is the large square area in the left side of the studio, below the toolbar.

Toolbar: The toolbar contains icon buttons with which the user can edit objects and navigate in the canvas. The toolbar buttons occupy two rows in the top of the studio.

Layer manager: The layer manager lets the user graphically program the elevations of different layers in the canvas. The user can also use the layer manager to change their current elevation in the canvas (a shortcut for zooming). The layer manager is also linked to the data manager, so that the table associated with the currently selected layer is always shown in the data manager. The layer manager, which is discussed further below, occupies a sky-blue square area in the upper right of the studio, below the toolbar. (While the layer manager is generally blue, it is white in Figures 1, 5 and 6 in this paper.)

Data manager: The data manager displays the data table associated with the currently selected layer. By default, all rows from the table are present in the associated layer. Users can specify that only a subset of the rows in the table should appear in the layer using VIQING, a simple direct-manipulation interface described in part in Section 5. Note, however that DataSplash does not currently support arbitrary restrictions, so complex filters must be specified textually. The data manager occupies a gray square area in the lower right of the studio.

2.6. Graphical Save

DataSplash allows layers to be shared among canvases, which means that modifying and saving a layer in one canvas may affect another canvas. As a result, we found it necessary to support saving of the following types of objects: the entire canvas (including the layer manager program and every individual layer), the layer manager program only (the names of layer bars in the layer manager as well as their ordering and top-and-bottom positions), or individual layers. Currently, there is no way to individually save or reuse specific objects within a layer.

In keeping with our philosophy of direct manipulation, we implemented a graphical save mechanism. Accordingly, in save mode, all unsaved components are visually highlighted by a bright pink color. Clicking the mouse on one of these pink-colored components saves it. Items that have not been modified since they have been saved are a burgundy color. When users save an item, it changes from pink to burgundy. For example, layer bars are colored according to their saved status. Users can click on them to save their contents. To save the canvas or the layer manager program, users click on the button bars associated with those components. Additionally, DataSplash supports write-protection of layers and canvases; objects that are write-protected are assigned a different color in save mode.

3. Paint Program Interface

In this section, we describe the DataSplash paint program interface. The purpose of the paint program is to allow users to easily create graphical representations of data rows. It accomplishes this by providing an intuitive, direct manipulation interface for specifying the visual appearance of these rows. We begin by discussing the drawing primitives supported by the paint program. We next discuss how users can edit graphical objects on the canvas. Finally, we discuss how users can edit the relative x, y positions of objects on the canvas.

3.1. Drawing Primitives

As in a conventional paint program, the DataSplash user is presented with a palette of displayable objects (these are contained in the toolbar described in Section 2). To draw an object, the user selects the corresponding paint primitive from the paint palette and places it in the two-dimensional canvas. By default, an object is a trim object; users may convert trim objects to splash objects.

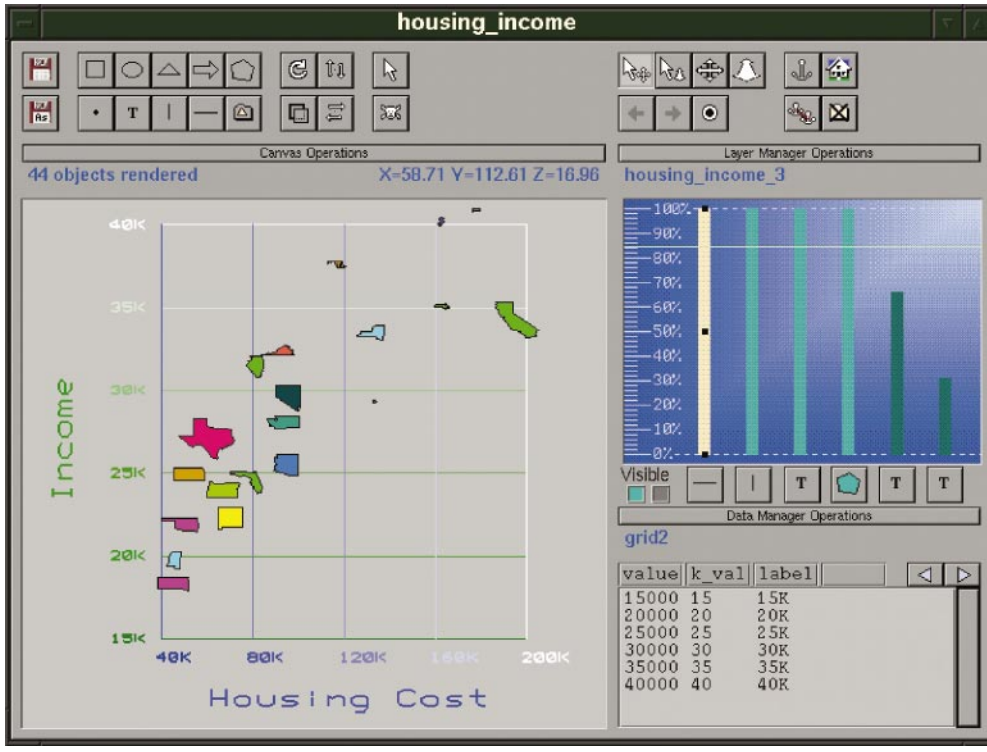

Figure 3. Scatterplot of US states shown according to housing cost and income

DataSplash supports the following drawing primitives: rectangle, point, oval, text, triangle, vertical line, horizontal line, arrow, fixed polygon/polyline and data-specific polygon/polyline. Fixed polygons/polylines are identically shaped for every row in a table, while data-specific polygons/polylines support different shapes for each row in a table. To use data-specific polygons/polylines, users specify a column in the table that holds the vertices of the shape.

Figure 3 uses data-specific polygons to show average housing cost and income for selected US states. The visualization is a scatterplot (housing cost appears on the *x*-axis and income on the *y*-axis). The position of each state in the scatterplot is shown by an icon in the shape of the state; the state icons are implemented using data-specific polygons.

3.2. Object Manipulation

After users have created objects, they may modify them using a variety of tools. Users can visually select, erase, resize, rotate, duplicate or flip objects horizontally or vertically. They can also invoke a property dialog in which they can modify values such as the data column associated with visual attributes such as color or orientation. If users modify an instance of a splash object, e.g. by resizing it when it is selected, the changes to that object are automatically propagated to all related splash objects, e.g., by scaling them proportionately.

3.3. Layer Alignment

Data values in data sets do not always 'line up'. Two different layers may be based on different underlying data tables. These data tables may be in different units or the base data values may be offset.

If the user wishes to coregister two layers, they can adjust the linear functions that specify the x, y position of each row. These adjustments can be made in one of three ways. First, they can be made non-graphically in a dialog box. Alternatively, in keeping with our direct manipulation paradigm, coregistration may be performed graphically using the layer translate or layer scale modes.

Layer translate mode can be used to coregister two different layers that have the same x, y scale but are slightly offset. The general procedure begins by selecting one object; an anchor appears over the selected object. When the user moves the anchor to a new location, the functions that dictate the x, y position of each row are updated to incorporate this change. Specifically, layer translate changes the x, y constant term of the linear functions used to determine the x, y position of each row.

Layer scale mode can be used to coregister two different layers that have different x, y scales. The general procedure begins by selecting two objects in a layer; anchors appear on each object. When the user moves either anchor to a new location, the functions that dictate the x, y position of each row are updated to incorporate this change. Specifically, layer scale changes the x, y multiplier term of the linear functions used to determine the x, y position of each row.

4. Layer Manager

As mentioned above, users can pan and zoom above the DataSplash two-dimensional canvas. When they zoom, they change their elevation above the canvas. Different layers are displayed at different elevations. In the first part of this section, we describe the layer manager, with which users graphically program the elevations at which layers are visible. We then discuss the layer manager interaction with the paint program.

4.1. Layer Manager Functionality

The layer manager allows users to control the range of elevations at which each layer is rendered. To this end, each layer is represented by a vertical bar in the *layer manager*. Figure 4 shows a close-up of the layer manager. The top of each layer bar represents the highest elevation at which objects in the layer are rendered. Similarly, the bottom of each layer bar represents the lowest elevation at which objects in the layer are rendered. The user's current elevation is shown with a horizontal *elevation line*. Any layer bar that is crossed by the horizontal elevation line is considered to be active and objects in the corresponding layer are rendered on the canvas. An icon of the type of object displayed by each layer appears in the button below its layer bar.

Users can graphically resize layer bars in the layer manager. In addition to resizing layer bars, users can also shift them up and down and add or delete new layer bars. These operations are performed in the same way as in traditional paint programs, e.g. to resize a layer bar, the user drags a resize handle.

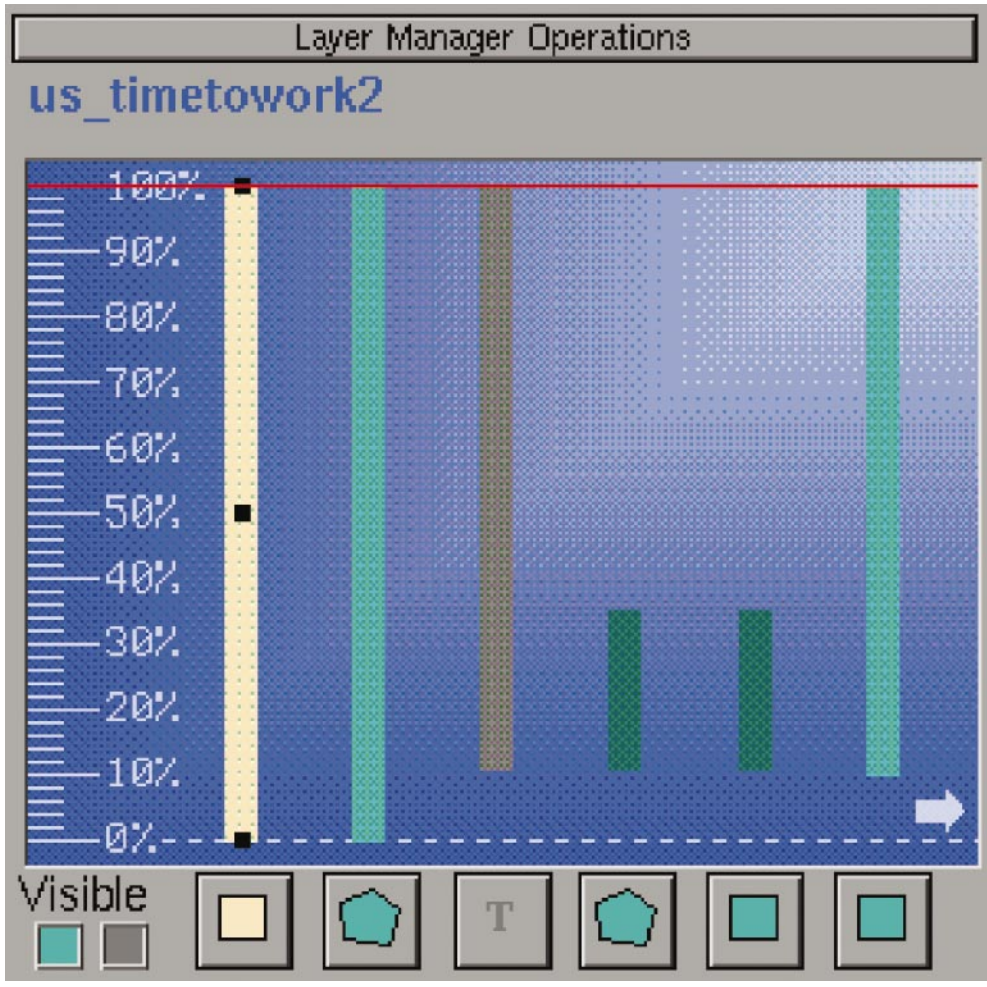

Figure 4. A detailed view of the layer manager

Figures 5 and 6 show the canvas display and layer manager of a sample application. (Figure 1 is duplicated here as Figure 6 to facilitate comparison with Figure 5.) The application contains six layers. The first layer contains an outline of the US. The second layer contains outlines of each state in the US. Note that the elevation ranges of these two representations are disjoint, so only one can be visible at a given time.

The third through sixth layers are based on city data. The data has been segmented according to population size; the third layer contains cities with the highest population, the fourth layer contains slightly smaller cities, etc. Each city in each layer is drawn as a circle and cities with higher populations are drawn as larger circles.

In Figure 5, the layer manager appears in the white rectangle on the upper right side. The horizontal elevation line indicates that the user is at a high elevation. The only layer active at this elevation is the US outline layer. The contents of this layer are rendered in the display on the left.

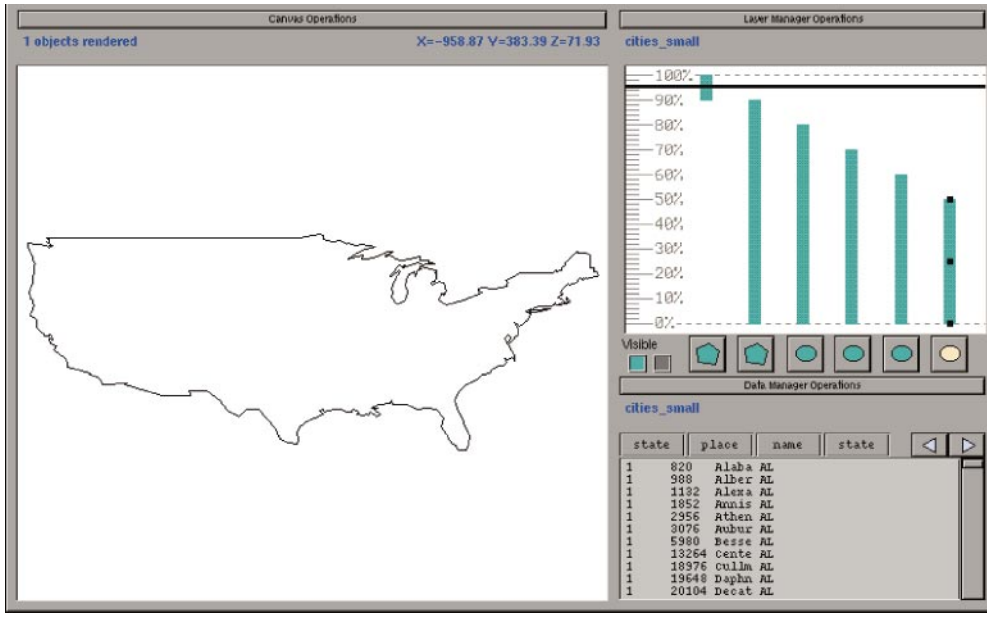

Figure 5. US cities visualization seen from a high elevation

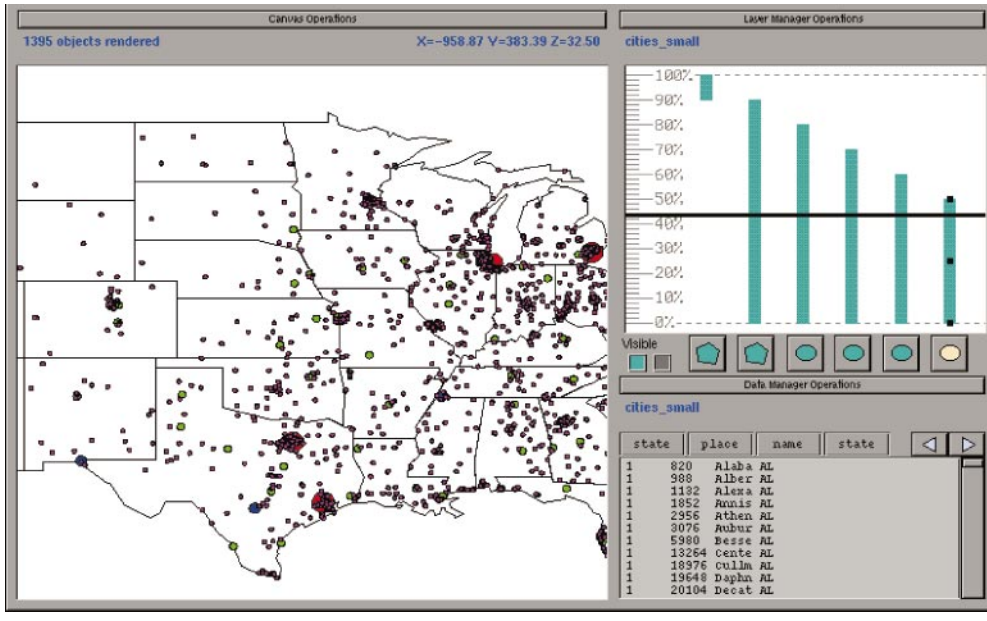

Figure 6. US cities visualization seen from a low elevation

In Figure 6, the user has zoomed to a lower elevation, as can be seen from the position of the horizontal elevation line in the layer manager. At this elevation, the state outlines and city circles layers are active. Therefore, the objects associated with these layers are all visible in the display on the left.

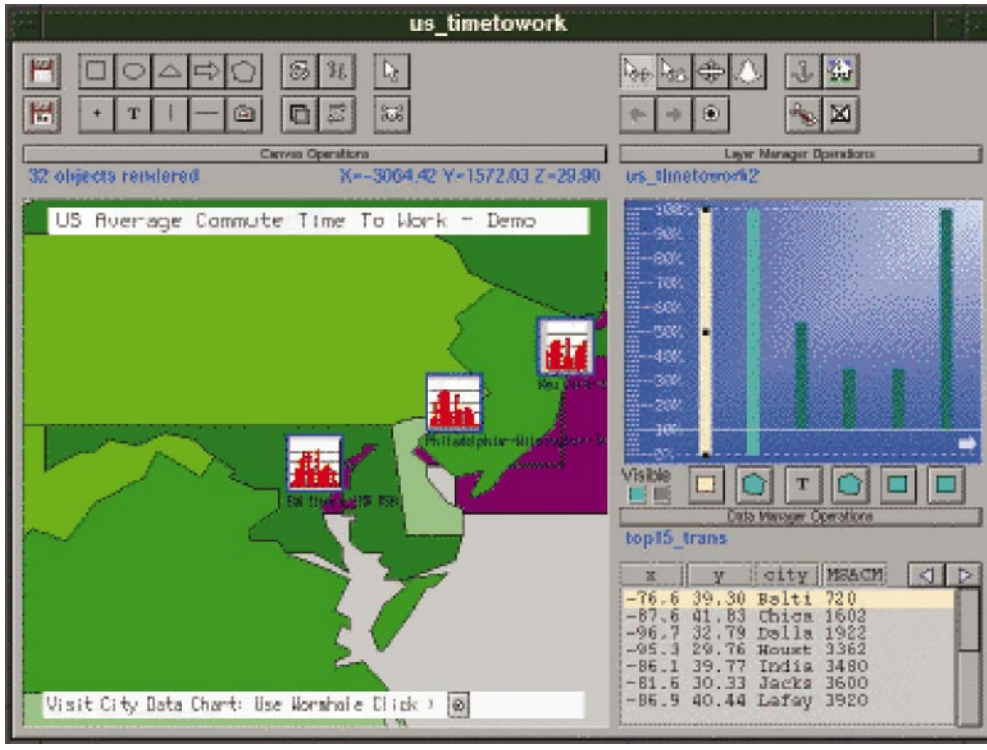

Figure 7. US map with city portals

4.2. Interaction with Paint Program

In our informal studies, user response to the layer manager was generally very positive. However, one user interface problem did arise with regard to the linking between the paint program and the conceptually and visually distinct layer manager. Specifically, we found that users often lost track of the layer in which they had drawn an object. For example, a user might not know which layer contained circles and which contained text. We addressed this issue in several ways.

First, we constrained the system so that objects could only be drawn in the currently selected layer (the sixth layer in Figures 5 and 6). This helps prevent users from drawing objects in one layer when they intend to draw them in another. To further aid in this regard, we modified the system so that if users try to paint an object on the canvas when the selected layer is not visible, an error dialog appears. Although preventing invisible layers from being selected would also resolve this issue, it has the major drawback of preventing the user from programming (moving and resizing) layer bars that do not currently cross the horizontal elevation line.

Second, to help users understand which objects are in the currently selected layer, we modified the system so that when DataSplash is in an editing mode (e.g., an object manipulation or drawing mode) as opposed to a navigation mode, objects in the non-selected layers are drawn as wireframe while objects in the selected layer are drawn normally. This makes it visually obvious which objects are in the currently selected layer.

Figure 8. Visualization of voting patterns

Third, we added icons on the visible buttons that represent the type of object in the layer (described above). This makes it easier for users to locate the layer with the content they want.

5. Replicated Portals

Recall that DataSplash provides portals (windows between canvases). Just as they can automatically create splash objects such as rectangles for every row in a database table, DataSplash users can automatically generate splash portals for every row in a database table. Suppose the user has a canvas with a map of the US. The user can easily specify that each city in the US map should have a portal that goes to a detailed map of that city. Figure 7 shows a slight variation of this nested visualization example, in which a portal in each city contains a bar chart of transportation data for that city.

The design challenge is to provide lightweight mechanisms for users to specify such portals. These splash portals (which we also refer to as replicated portals) may be specified in one of two ways. First, the user may create splash portals by creating a splash rectangle using the paint program interface and then editing the fill property to make it a splash portal. Second, the user may use the VIQING visual querying extension to DataSplash [18]. VIQING supports a direct-manipulation approach to querying: users construct queries by manipulating visualizations. The results of these graphical queries are visualizations; some classes of VIQING queries result in nested canvases with replicated portals.

In the remainder of this section, we describe the VIQING direct manipulation mechanism for specifying replicated portals. We illustrate the functionality of VIQING using a series of visualizations of US states' voting patterns in presidential elections.

We begin with Figure 8, which shows the visualization we will replicate. This visualization shows US states colored according to the party a state has voted for most often in presidential elections between 1952 and 1992. The blue-colored states (the darker states in the black and white version of this paper) favored the Republican party while the red-colored states (the lighter states in the black and white version of this paper) favored the Democratic party. The remainder of the figures show how a user can produce portals that show subsets of the data rendered with the same visualization logic.

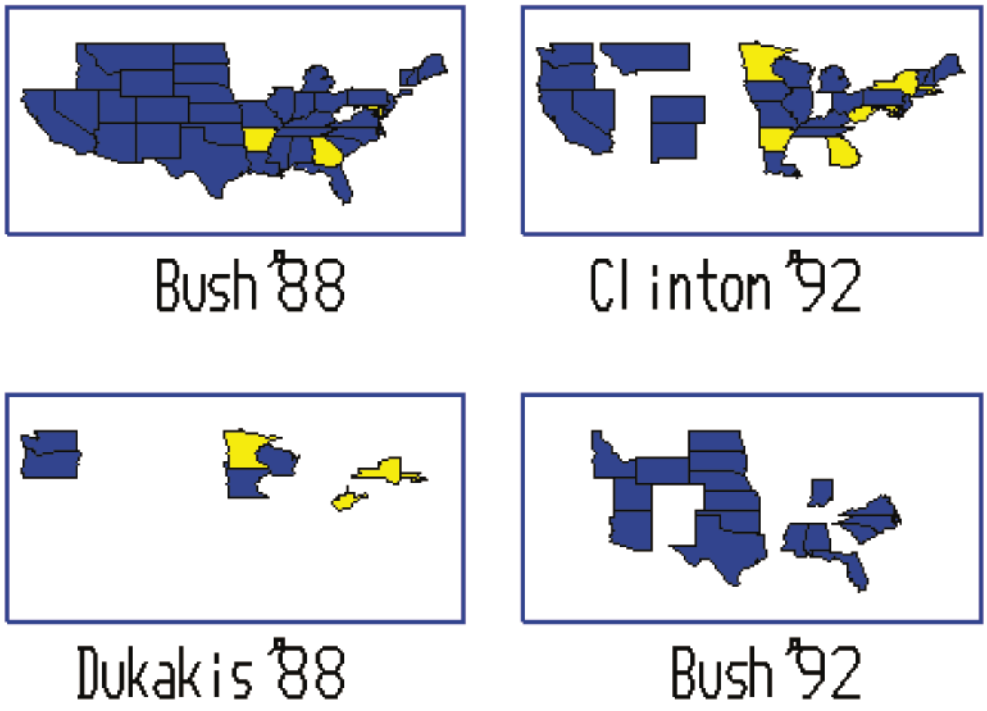

Figure 9. VIQING visual join

Figure 9 shows the result of a VIQING direct manipulation operation that combines the US states visualization of Figure 8 with a visualization of political candidates in the election years between 1952 and 1992. The layout of the candidates canvas is as follows. The *x*-axis represents the election year and the *y*-axis represents the result of the election; the winner of each election is on top. When the user performs a *visual join* operation (the interface for this is described below) to combine the US states visualization and the candidates visualization, a splash portal showing US states is replicated for each candidate. Each splash portal contains the states that voted for its associated candidate. The result is the visualization shown in Figure 9, in which additional candidates can be seen by zooming out or panning to the left or right.

Figure 10 illustrates the use of the direct manipulation visual join operation to join a subset of the visualization from Figure 9 (on the left) with a visualization of political parties (on the right). The user has used rubberband selection to identify political candidates of interest in the canvas on the left. Dragging this selection and dropping it into the political parties canvas on the right performs a visual join that replicates the selected subset of the visualization of Figure 9 for each party in the political parties canvas. The result is the three-level nested visualization shown in Figure 11. The user now has information about the party affiliation of the candidates. In addition, the user can see the election result trends for each party over time. Recall that the winning candidate for each election year is on top.

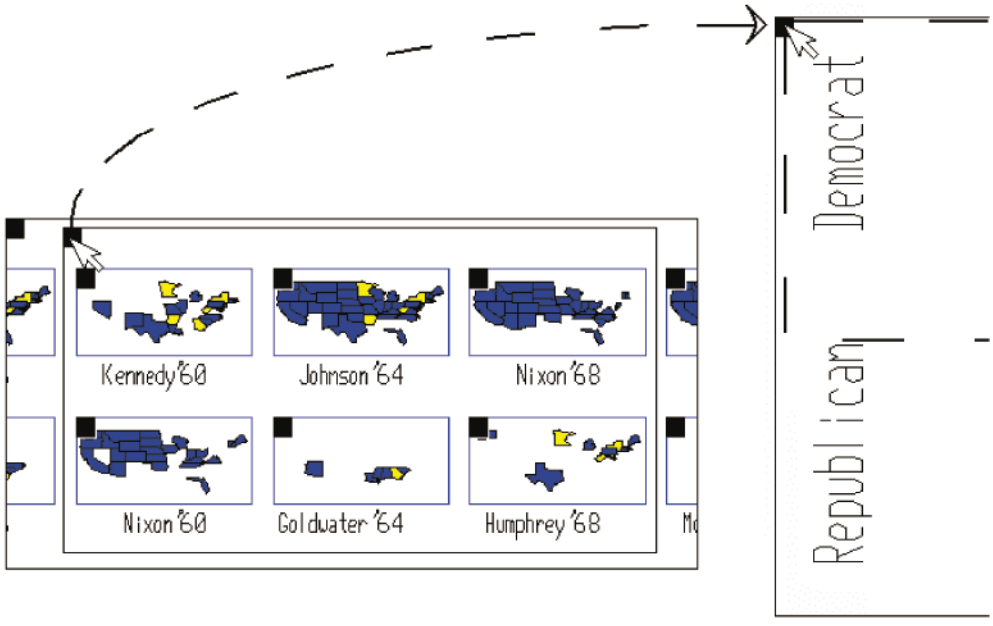

Figure 10. Creating a 3-level join in VIQING

VIQING's functionality generalizes a number of interesting data presentation metaphors, including nested report writers, master/detail forms [19] and small multiple visualizations [14, 15].

6. Implementation and Status

DataSplash is implemented in C++, the Mesa graphics programming library [20], XForms [21] and the POSTGRES object-relational database management system [22]. It has been released as freely available software [23] for a variety of UNIX platforms including Linux.

DataSplash has been used as a research platform for a number of projects in areas such as information density [24] and visual representation of online aggregation [25]. Many components of DataSplash (and its predecessor, Tioga) have been commercialized in the Illustra Object Knowledge and Informix Visionary products.

7. Related Work

Semantic zoom has been explored in a number of systems [1–3, 26]. These systems generally require expert programmers. Our purpose is to advance work in the area of semantic zoom systems by making semantic zoom applications with optional nesting easy to program visually, and by explicitly considering the application of the (nested) semantic zoom paradigm to tabular data.

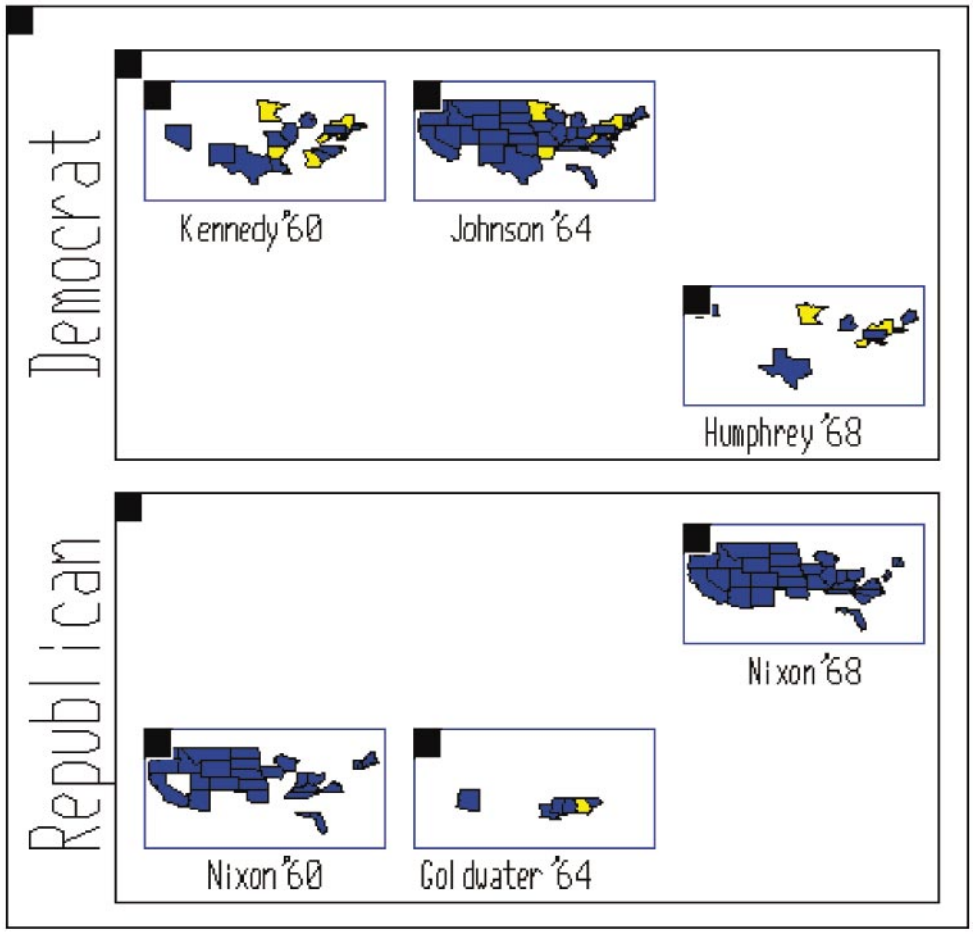

Figure 11. VIQING 3-level join

A number of systems provide a paint program or direct manipulation interface that allows users to visualize structured data. Sagebrush [11] is a data visualization system that shares DataSplash's approach to visualization programming. As in DataSplash, Sagebrush users can make the display of primitive objects dependent on data values. The mapping from data values to visual representation is also done by direct manipulation, as in DataSplash. DEVise is a similar system [9], providing a direct manipulation interface for specifying the appearance of groups of objects. GOLD takes a different approach. In contrast to Sagebrush and DataSplash, GOLD is a programming-by-demonstration [27] system that allows the user to provide exemplars and infers the desired mapping from data values to visual representation. IVEE and Spotfire also provide direct manipulation facilities for constructing visualizations, as well as tools for dynamically restricting the contents of visualizations [28, 29]. Presumably because the researchers on these projects chose to focus on other issues, none of these systems have semantic zoom facilities. Our intent is to advance the state of this class of systems by explicitly considering the issues that arise in semantic zoom applications.

Several systems provide direct manipulation interfaces for semantic zoom systems. For example, KidPad, MuSE, and PadDraw are drawing programs that allow users to graphically specify objects in semantic zoom applications [6, 7, 9]. MuSE further provides a space-scale [30] editor, a visual mechanism for specifying the elevations at which objects change representation. However, none of these programs address the issues that arise for groups of objects or portals.

Finally, [31] introduces a layout algorithm that shows objects at different levels of detail, using user-specified layout to guide heuristic decision. However, this system has quite a different research focus from ours, e.g. it does not incorporate the full functionality of semantic zoom.

8. Conclusions and Future Work

In this paper, we have presented the DataSplash database visualization system and an extension called VIQING that generates nested visualizations. DataSplash allows users to visually construct, edit, program and navigate visual representations of data using direct manipulation gestures. DataSplash specifically focuses on helping users visually program semantic zoom applications of tabular data sets, addressing issues that arise for groups of objects as opposed to individual objects. The DataSplash/VIQING environment provides graphical programming mechanisms such as the paint program to create objects, the layer manager to program the behavior of objects during zooming and the portal replication mechanisms to automatically specify the appearance of a large number of portals in nested visualizations.

A number of issues remain. For example, DataSplash does not consider data aggregation or database updates. Further, a 3D layer manager to create effects such as the 3D magic lenses in [32] would be interesting, but might require a major redesign of the existing layer manager.

Acknowledgments

We are grateful to Joe Hellerstein and Paul Aoki for many insightful discussions and suggestions. We are further indebted to Paul Aoki for helpful comments on earlier versions of this paper.

This work was sponsored in part by NSF under Grants IRI-9400773 and IRI-9411334.

References

1. B. Bederson & J. Hollan (1994) Pad ++: a zooming graphical interface for exploring alternate interface physics. In: *Proceedings of the ACM UIST'94*. ACM Press, New York, pp. 17–26.
2. K. Perlin & D. Fox (1993) Pad: an alternative approach to the computer interface. In: *Proceedings of the ACM SIGGRAPH'93*. ACM Press, New York, pp. 57–64.
3. B. Bederson, J. Meyer & L. Good (2000) Jazz: an extensible zoomable user interface graphics toolkit in Java. In: *Proceedings of the ACM UIST 2000*. ACM Press, New York, pp. 171–180.

4. B. Shneiderman (1992) *Designing the User Interface: Strategies for Effective Human–Computer Interaction*. Addison-Wesley, Reading, MA.
5. M. Stonebraker, J. Chen, N. Nathan, C. Paxson & J. Wu (1993) Tioga: providing data management for scientific visualization applications. In: *Proceedings of the 19th International Conference on Very Large Data Bases*, Dublin, Ireland, pp. 25–38.
6. A. Druin, J. Stewart, D. Proft, B. Bederson & J. Hollan (1997) KidPad: a design collaboration between children, technologists, and educators. In: *Proceedings of the ACM SIGCHI'97*. ACM ress, New York, pp. 463–470.
7. G. W. Furnas & X. Zhang (1998) MuSE: a MultiScale Editor. In: *Proceedings of the ACM UIST '98*. ACM Press, New York, pp. 107–116.
8. J. Meyer (1997) Getting started with PadDraw. http://www.cs.umd.edu/hcil/pad++/documentation/paddraw/
9. M. Livny, R. Ramakrishnan, K. Beyer, G. Chen, D. Donjerkovic, S. Lawande, J. Myllymaki & K. Wenger (1997) DEVise: integrated querying and visual exploration of large datasets. *SIGMOD Record* **26,** 301–312.
10. B. Myers, J. Goldstein & M. Goldberg (1994) Creating charts by demonstration. In: *Proceedings of the ACM SIGCHI'94*. ACM Press, New York, pp. 106–111.
11. S. Roth, J. Kolojejchick, J. Mattis & J. Goldstein (1994) Interactive graphic design using automatic presentation knowledge. In: *Proceedings of the ACM SIGCHI'94*. ACM Press, New York, pp. 112–117.
12. S. Jul & G. W. Furnas (1998) Critical zones in desert fog: aids to multiscale navigation. In: *Proceedings of the ACM UIST'98*. ACM Press, New York, pp. 97–106.
13. A. Woodruff, P. Wisnovsky, C. Taylor, M. Stonebraker, C. Paxson, J. Chen & A. Aiken (1994) Zooming and tunneling in Tioga: supporting navigation in multidimensional space. In: *Proceedings of the 10th IEEE Symposium on Visual Languages*. IEEE Computer Society Press, Los Alamitos, CA, pp. 191–193.
14. Belmont Research Inc. (2000) CrossGraphs. http://www.belmont.com/cg.html.
15. E. Tufte (1983) *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, CT.
16. C. Olston & A. Woodruff (2000) Getting portals to behave. In: *Proceedings of the IEEE Symposium on Information Visualization*. IEEE Computer Society Press, Los Alamitos, CA, pp. 15–25.
17. E. Bier, M. Stone, K. Pier, W. Buxton & T. DeRose (1993) Toolglass and magic lenses: the see-through interface. In: *Proceedings of the ACM SIGGRAPH'93*. ACM Press, New York, pp. 73–80.
18. C. Olston, M. Stonebraker, A. Aiken & J. Hellerstein (1998) VIQING: Visual Interactive QueryING. In: *Proceedings of the 14th IEEE Symposium on Visual Languages*. IEEE Computer Society Press, Los Alamitos, CA, pp. 162–169.
19. L. Rowe (1985) 'Fill-in-the-form' programming. In: *Proceedings of the 11th International Conference on Very Large Data Bases*. Morgan Kaufmann, San Mateo, CA, pp. 394–404.
20. B. Paul (2001) Mesa. http://www.mesa3d.org/
21. T. Zhao & M. Overmars (1995) XForms. http://world.std.com/~xforms/
22. M. Stonebraker & G. Kemnitz (1991) The POSTGRES next-generation database management system. *Communications of the ACM* **34,** 78–92.
23. Regents of the University of California (1997) DataSplash. http://datasplash.cs.berkeley.edu
24. A. Woodruff, J. Landay & M. Stonebraker (1999) VIDA (Visual Information Density Adjuster). In: *Proceedings of the ACM SIGCHI'99 Extended Abstracts*. ACM Press, New York, pp. 19–20.
25. J. M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth & P. J. Haas (1999) Interactive data analysis: the Control project. *IEEE Computer* **32,** 51–59.
26. C. F. Herot (1980) Spatial management of data. *ACM Transactions on Database Systems* **5,** 493–513.
27. A. Cypher, D. C. Halbert, D. Kurlander, H. Lieberman, D. Maulsby, B. A. Myers & A. Turransky (eds) (1993) *Watch What I Do: Programming by Demonstration*. The MIT Press, Cambridge, MA.
28. C. Ahlberg & B. Shneiderman (1994) Visual information seeking: tight coupling of dynamic query filters with starfield displays. In: *Proceedings of the ACM SIGCHI'94*. ACM Press, New York, pp. 313–317.

29. Spotfire Inc. (1999) Spotfire. http://www.spotfire.com/
30. G. Furnas & B. Bederson (1995) Space-scale diagrams: understanding multiscale interfaces. In: *Proceedings of the ACM SIGCHI'95*, ACM Press, New York, pp. 234–241.
31. Y. Ioannidis, M. Livny, J. Bao & E. Haber (1996) User-oriented visual layout at multiple granularities. In: *Proceedings of the 3rd International Workshop on Advanced Visual Interfaces*. ACM Press, New York, pp. 184–193.
32. P. Cignoni, C. Montani & R. Scopigno (1994) MagicSphere: an insight tool for 3D data visualization. *Computer Graphics Forum* **13,** C317–C328.