VIQING: Visual Interactive QueryING

Chris Olston, Michael Stonebraker, Alexander Aiken, Joseph M. Hellerstein
{cao, mike, aiken, jmh}@cs.berkeley.edu
Department of Electrical Engineering and Computer Sciences
University of California at Berkeley

Abstract

This paper presents VIQING, an environment for expressing queries via direct manipulation of data visualizations. VIQING provides a simple graphical interface for connecting visualizations, and has the expressive power of the basic relational operators select, project and join. VIQING has been implemented in the Tioga DataSplash visualization system to provide a seamless integration of querying and browsing. The resulting system is unique in providing a unified visual interface for developing database applications, encompassing both querying and data visualization.

Key Words: database visualization, graphical query languages, direct-manipulation interfaces

1. Introduction

Database systems are hard to use. Untrained users often find query interfaces frustrating and even trained database users frequently have difficulty analyzing the results of queries. Despite over 25 years of research in this area, these problems persist today.

The goal of the Tioga project [14] has been to address these problems by designing a simple graphical interface for querying a database and visually browsing the results of queries. The initial Tioga design (Tioga-1 [13]) used a "boxes-and-arrows" paradigm for constructing queries, and provided minimal support for developing visualizations of query results. A revised design (Tioga-2 [1], implemented as the Tioga DataSplash system [9]) discarded the boxes-and-arrows paradigm (which was difficult to use), and focused on direct-manipulation mechanisms for constructing and browsing custom visualizations of query results. Users have found that while DataSplash is very effective for the interpretation of query results, it provides essentially no support for query specification in its current implementation.

This paper presents a technique called VIQING (Visual Interactive QueryING) which extends DataSplash with a visual, interactive interface for query specification. VIQING differs from previous graphical query tools in that it supports a "direct-manipulation" approach to querying: users construct queries by manipulating visual representations of entire data sets, as opposed to representations of schemas or example records. The combination of VIQING with the DataSplash architecture results in a seamless and, we believe, intuitive system in which querying and data browsing are unified into a single metaphor: the direct manipulation of data visualizations.

1.1. Related Work

The goal of developing simple easy-to-use relational query tools is an old one -- interesting work goes back as far as the 1970's [7,18], and is embodied in modern products like Microsoft Access [8]. Tools for developing data presentations form a more recent body of research (e.g. Pad [10] and DEVise [6], as well as so-called 4GLs and report writers). Outside of the relational environment, tools like Khoros [11] and AVS [16] have been developed to construct data manipulations roughly analogous to queries; visualization concepts like Magic Lenses [4,5] have been developed for viewing data.

To put VIQING and DataSplash into perspective with previous work, it is useful to consider the matrix in Figure 1, which categorizes a variety of ostensibly visual tools for data analysis.

In the lower left cell we list systems that require traditional textual specification (e.g. SQL) for composing queries; the cell above it lists systems that provide visual query interfaces. In the bottom of the center column, we list systems that require traditional programming (e.g. C++ and the MS Windows API) to construct data visualizations; the upper center contains systems that simplify this chore to varying extents.

Visual Programming	QBE, Cupid, Tioga-1, AVS, Khoros, Access, DEVise	4GLs, Tioga-1, AVS, Khoros, Access, DEVise, Magic Lenses	DataSplash/VIQING
Traditional Programming	4GLs, Magic Lenses	QBE, Cupid	PLs + widget libraries
	Query Component	**Visualization Component**	**Unified Query/Visualization**

Figure 1. Categorization of visual programming tools for databases.

Note that Tioga-1, AVS, Khoros, MS-Access, and DEVise contain visual components for both querying and visualization. However, in each of these examples the visualization and query components operate under *two distinct metaphors*. That is, there are two separate graphical conceptions for querying and visualization, and users must learn both and switch between them to design an application. This complexity makes these systems difficult to use for interactive querying and visualization.

The rightmost column in the matrix represents systems that unify querying and visualization into a single metaphor. The widget libraries available for programming languages do this in the textual domain, but the overhead of programming queries and visualizations in a language like C++ is prohibitive. To our knowledge, the VIQING/DataSplash environment is the first example of a system in which querying and browsing are combined in a seamless manner.

In Section 2, we describe the DataSplash environment. Section 3 details the VIQING paradigm. In Section 4, we present a direct-manipulation interface for creating and modifying visual queries. Section 5 describes how VIQING generalizes other paradigms. Status and future work is discussed in Section 6. We conclude in Section 7.

2. DataSplash Background

The DataSplash system provides a direct-manipulation interface for database visualization. Using a simple paint program interface, users create graphical objects on a 2-dimensional canvas. Each canvas is associated with a database table. Users can create **splash objects**, which are replicated for each tuple (record) in the table (*i.e.*, one instance of the object is drawn for each tuple). Graphical properties of a splash object, such as its location on the canvas, shape and color, can be functions of attributes of the underlying tuple. Thus, splash objects are graphical representations of database tuples. DataSplash supports ad-hoc visualization of arbitrary data fields, making it useful for spatial and non-spatial data.

Figure 2 shows a sample DataSplash visualization of a table of states. Each state is an instance of a splash object. Each state tuple is mapped to a location on the canvas according to its geographical position. The shape and color of each state is determined by table attributes.

Once a user has created a collection of canvases, s/he can create **portals** to visually link them. A portal is a sub-area within one canvas (the **parent canvas**) that contains a second canvas (the **child canvas**). To illustrate the concept of portals, we ask that you skip ahead momentarily to Figure 4, which shows a canvas that contains four portals. Portals serve two purposes. First, they display the child canvas within the parent canvas, allowing for nested visualizations. Second, portals can be used to navigate between DataSplash canvases in the same way that hyperlinks are used to go between World-Wide-Web pages. DataSplash displays an **outermost canvas**, which may contain portals. Users click on a portal to "go through" it. At that point, the child canvas inside the portal becomes the outermost canvas. A history mechanism allows users to return to the parent canvas. As is the case for all DataSplash objects, portals can be replicated per tuple; such portals are called **splash portals**.

DataSplash canvases are infinitely pannable in the X and Y dimensions. In addition, canvases can be zoomed in and out to adjust the level of magnification. Thus, arbitrarily large data sets can be represented, and different portions can be accessed via panning and zooming. The same applies to portals, because child canvases can be panned and zoomed independently from their parent canvases.

DataSplash provides no explicit query composition capabilities. However, projections are supported implicitly in the DataSplash environment. If graphical properties of splash objects only reflect certain table attributes, then the table is effectively projected onto only those attributes in the visualization.

3. The VIQING Paradigm

In this section, we describe how VIQING extends DataSplash by providing a query manipulation paradigm. In our initial discussion we focus on the way that DataSplash presents the *results* of queries. We defer discussion of how VIQING queries are actually specified until Section 4. For the remainder of this paper, we refer to VIQING queries as "visual queries."

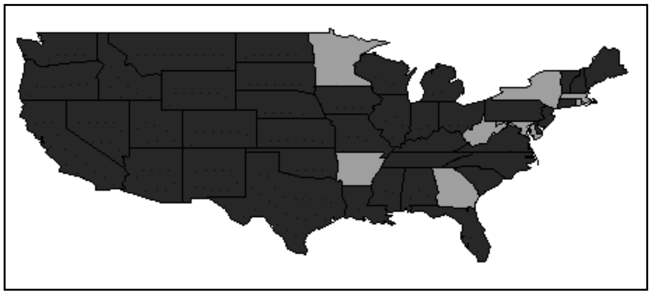

Figure 2. A basic DataSplash visualization.

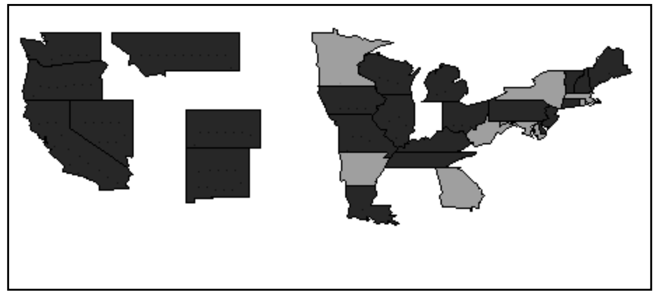

Figure 3. A visual selection.

3.1. Visual Selections

A visual selection query is represented as a portal that performs a selection on the graphical tuples in the child canvas. Thus, a canvas viewed through a selection portal contains a subset of its tuples. This functionality serves two purposes. First, it allows the user to eliminate uninteresting tuples from a canvas, thus reducing data density and isolating data of interest to the user interested. Second, it allows the user to explore trends that occur in specific subsets of the data.

Figure 2 shows a visualization of U.S. states colored according to the party a state has voted for most often in presidential elections between 1952 and 1992. The dark colored states favored the Republican Party while the light colored states favored the Democratic Party. Figure 3 shows the same visualization, but with only the states that voted Democratic in 1992 selected. This reveals that all the states that favored the Democratic Party from 1952 to 1992 voted Democratic in 1992. In addition, we see that many states that traditionally vote Republican voted Democratic in 1992.

The following database schema is used:

```
     states (sname, shape, latitude,
             longitude, favored_party)
    parties (party_abrev, party_name)
 candidates (cname, party, year,
             won_or_lost)
      votes (year, sname, party)
```

The state table contains each state's name (sname), shape, latitude and longitude, and the party it has voted for most often in recent elections (favored_party). The parties table contains the abbreviations (party_abrev) and names of each party. The candidates table contains each candidate's name (cname) and party affiliation, along with his year of candidacy and whether he won or lost (won_or_lost). Finally, the votes table contains one tuple for each vote cast for a party in each year by each state (sname). For the purpose of illustration we define a view called state_votes that contains states joined with votes with the join predicate `votes.sname = states.sname`. Section 4 presents a direct-manipulation operation that can be used to generate the state_votes view.

To support visual selections, we associate a **selection filter** with portals. The selection filter limits which tuples of the child canvas appear inside the portal. The SQL query that retrieves these tuples has the form:

```
SELECT *
  FROM <child table>
 WHERE <filter>
```

The selection filter used in Figure 3 to display only those states that voted Democratic in 1992 was:

```
state_votes.year = 92 AND
state_votes.party = 'D'
```

In Section 4 we present a visual technique for specifying these filters

3.2. Visual Joins

Splash portals can be used to visually join two DataSplash canvases. A database join combines information from two tables by correlating tuples in one table with tuples in the other. Similarly, a visual join correlates graphical representations of tuples.

Figure 4 shows a visual join. By joining a parent table of presidential candidates with a child table of states, we have replicated the visualization of Figure 3 for the candidates in election years between 1952 and 1992. Each candidate has a portal instance that contains the states that voted for him. (The other candidates can be seen by zooming out or panning to the left or right.) The layout of the candidates canvas is as follows. The X-axis represents the election year and Y-axis represents the result of the election – the winner of each election is on top.

A join parameter specifies the relationship between the parent and child tables. In a visual join, it determines which child tuples relate to each parent tuple. The join parameter used in Figure 4 is:

```
State_votes.year = candidates.year
AND state_votes.party =
candidates.party
```

In this case, the relationship is that states vote for candidates.

A visual **join portal** is a visual selection portal that is replicated for each tuple of the parent canvas (*i.e.* a *splash portal*). Each instance of the splash portal has a different selection filter. Each selection filter must retrieve the child tuples that relate (via the join parameter) to the parent tuple. We determine the selection filter for each parent tuple by "plugging in" data from the parent tuple into the join parameter. This results in an SQL query of the form:

```
SELECT <child>.*
  FROM <child>, <parent>
 WHERE <join parameter>
   AND <parent>.<primary key>
       = ___________
```

We submit this query once for each parent tuple to retrieve the contents of its portal instance. Each time, we replace the blank with the parent tuple's primary key (a set of attributes that uniquely identifies a tuple). [1]

For the lower left portal instance in Figure 4, we plugged "Dukakis '88" into the selection filter. The SQL query to retrieve the child tuples (states) for that portal instance is:

```
SELECT state_votes.*
  FROM states_votes, candidates
 WHERE (state_votes.year =
       candidates.year
   AND state_votes.party =
       candidates.party)
   AND (candidates.name = 'Dukakis'
   AND candidates.year = 1988)
```

Recall that the child canvas is a view that contains states joined with votes. If such a view did not exist, it could be composed as we describe in Section 4.

Visualizing a join in this manner offers several benefits over simply visualizing a single canvas representing the entire join space. First, a visual join joins two *existing* canvases by simply adding a portal between them. On the other hand, specifying the join separately and visualizing the result on a single canvas would require creating a new visualization from scratch. Second, because a visual join contains several nested canvases, more attributes can be represented than on a single flat canvas. This is due to the fact that each canvas has a limited number of graphical attributes (*i.e.:* x, y, shape, color, etc.).

Joins of more than two tables are common and can be represented with visual joins. By joining the child or parent canvas of a visual join with a third canvas, we can create a three-table visual join.

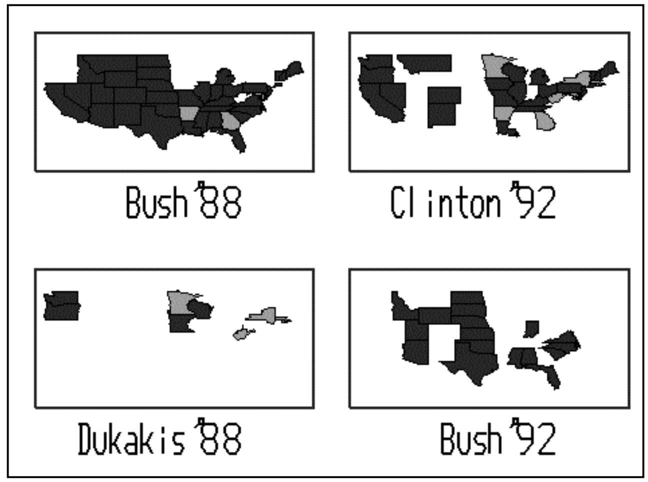

Figure 4. A visual join.

4. Specifying Visual Queries

In this section, we present a direct-manipulation interface for creating and modifying VIQING queries. Then we discuss how users specify join predicates.

4.1. User Interface

We now introduce a direct-manipulation interface for specifying VIQING queries. We present three operations which provide extensive functionality for creating and modifying visual queries: visual select, visual join, and reorder.

4.1.1 Visual Selection. First, we describe the visual select operation. While using the visual selection tool, "rubber-banding" an area of the canvas creates a selection portal. The selection filter of the new portal is a range selection on the x and y attributes. The endpoints of this range are defined by the rubber band. Selections can be

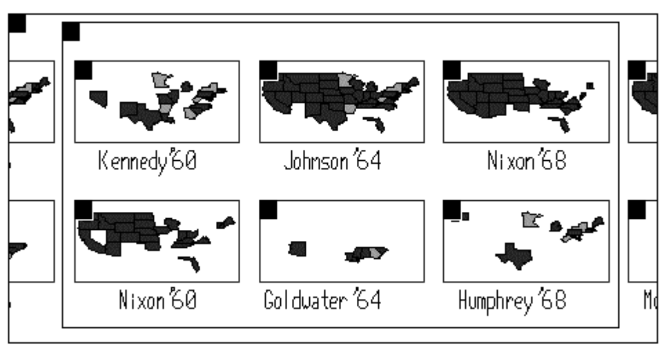

Figure 5. The visual select operation.

[1] Note that this is a naïve algorithm presented for illustrative purposes. Our implementation only submits one join query, with the join order determined by the DBMS.

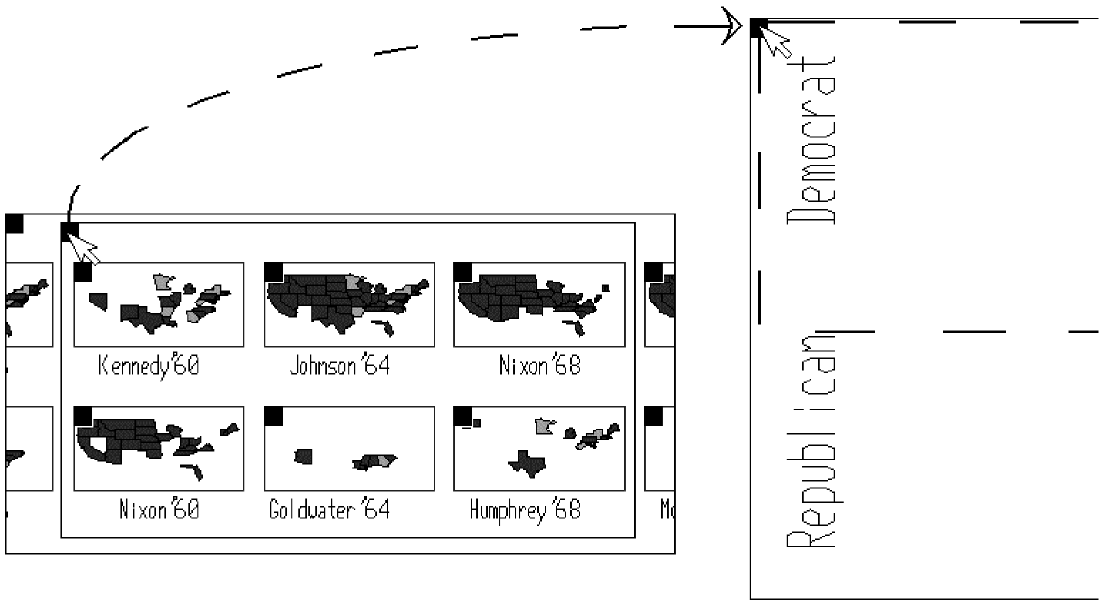

Figure 6. The visual join operation.

performed on any two attributes because users can change the x and y attributes via simple pull-down menus. In addition, zooming allows users to rubber-band a range of any size. Once the portal has been generated, it can be easily transferred to another canvas using the cut and paste features provided by DataSplash.

Figure 5 shows the immediate result of a visual selection on candidates who ran for office in the 1960's. The selection portal can now be used to visually join another canvas with just the 1960's candidates. The selection filter associated with the selection portal is:

```
Candidates.year > 1958 AND
candidates.year < 1970
```

As another example, consider specifying the visual selection described in Section 3 (Figure 3). To do so, we first change the x attribute to the party voted for in 1992. This results in a visualization with states that voted Democratic on the left and states that voted Republican on the right. Then, we rubber-band the states that voted Democratic. This generates a selection portal containing only those states. Then, we enter the portal (to make it the outermost canvas). Finally, we change the x and y attributes back to longitude and latitude, respectively.

4.1.2 Visual Join. We now introduce a drag and drop interface for creating join portals. Consider dragging a canvas S over another canvas R, and dropping S into R. This results in a modified version of canvas R, containing a join portal over canvas S. Note that canvases R and S could be portals themselves. Once a join predicate has been determined,[2] the join parameter of the new join portal becomes the conjunction of the join predicate with the selection filter of canvas S. Thus, if canvas S performs a selection, then the new join portal only joins with the selected canvas S tuples.

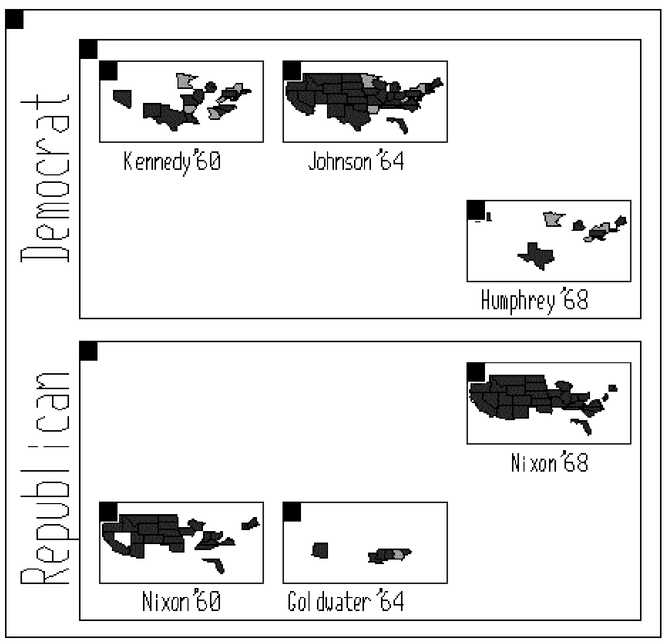

Figure 7. The result of the visual join operation.

[2] We discuss how join predicates are inferred or specified in Section 4.2.

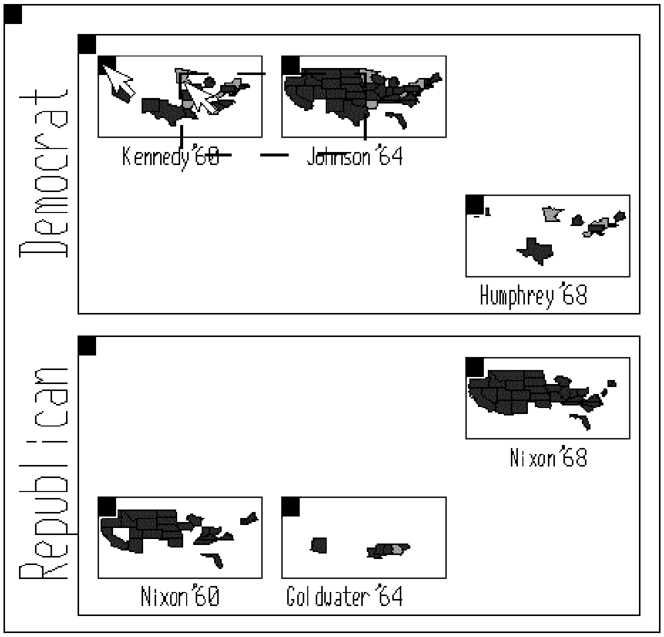

Figure 8. The reorder operation.

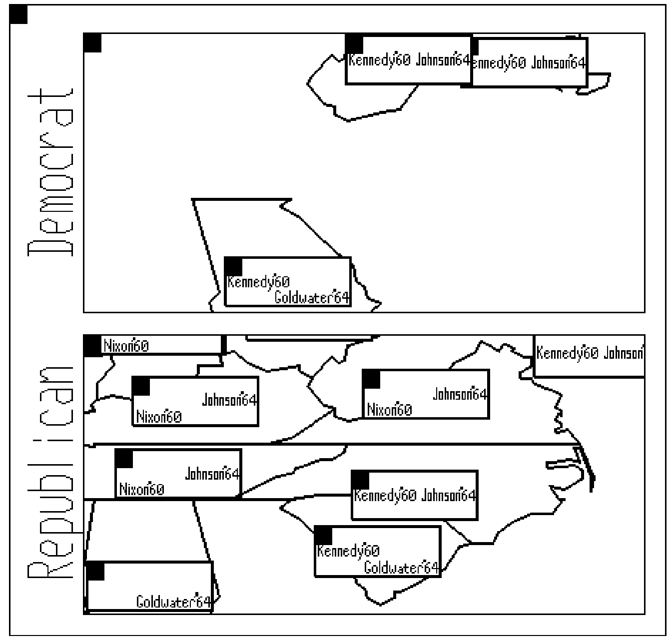

Figure 9. The result of the reorder operation.

Figure 6 illustrates the visual join operation. (The dashed line represents the position of the dragged canvas when it is dropped.) Picking up the 1960's candidates canvas and dropping it into a canvas which visualizes political parties results in the three-level visual join shown in Figure 7. We now have information about the party affiliation of the candidates. In addition, we can see the election result trends for each party over time. Recall that the winning candidate for each election year is on top. The join parameter for the portal in the parties canvas containing to the candidates canvas is

```
candidates.party =
parties.party_abrev.
```

4.1.3 Visual Reordering. In VIQING, joins of more than two canvases have an ordering associated with them. If canvas R joins with canvas S which in turn joins with canvas T, then canvas R is the outermost canvas, canvas S is in the middle, and canvas T is the innermost canvas in the join hierarchy. This ordering affects important nesting properties of the VIQING query. We introduce a **reorder** operation, which permits the user to alter the ordering of a VIQING query once it has been constructed. If the user drags and drops a portal onto a tuple of its child canvas, the relative ordering of the parent and child canvases is reversed.

Figure 8 illustrates the reorder operation on the three-level visual join from parties to candidates to states. In this example, an arbitrarily chosen candidate portal (Kennedy '60) is being dropped on an arbitrary tuple of its child states (Minnesota), as illustrated by the dashed line. Figure 9 shows the resulting visualization in which the new ordering is <parties, states, candidates>. States are now joined with their traditionally favored parties. As before, the candidates are arranged by year on the X-axis and the victors on top. From this new ordering, we can see that Georgia, a state that has generally favored Democrats, was one of the few states that voted for Goldwater, the Republican candidate in 1964.

Note that reordering in this context has direct effect on the visualization, but no effect on the query specification itself. In particular, this implies that the reordering of the tables in the visualization has no effect on query optimization or performance.

4.1.4 Discussion. We believe that VIQING queries are easier to formulate than SQL for several reasons. First, users do not have to know exactly what they want in advance because VIQING lets them incrementally build and refine queries. At each step, the user gets useful feedback that guides the next query manipulation action. In this way, complex queries can be built by combining simpler query pieces. Second, VIQING integrates querying with visualization. Query manipulations are performed on graphical representations of data that are generally easier to understand than text representations. Finally, VIQING eliminates the need to learn any SQL for most query formulation by providing a simple direct-manipulation interface. The drag and drop visual join operation requires no understanding of SQL or the database schema to formulate most queries. Figure 10 presents a summary of VIQING support for the base relational operations.

Relational Operation	Equivalent VIQING Operation
Projection	Simply assigning a graphical representation to the desired attributes.
Selection	Rubber-banding a set of tuples.
Join	Dragging and dropping one canvas into another to create a set of splash portals.

Figure 10. Summary of how VIQING supports the base relational operations.

4.2. Join Predicates

We now address the manner in which join predicates are specified. In most cases, two tables join via a single equality predicate. The system can infer the join predicate by inspecting the key/foreign key associations between the tables. Alternatively, the user can manually specify a more general join (i.e., a range join). We could implement a tool like the one provided by MS Access [8] to make manual specification of the join predicate more natural.

Often, two tables join via an intermediate table. The intermediate table may not be interesting to visualize. For example, the candidates table joins with the states table via the votes table. The visualization in Figure 4 has candidates joining directly with states, bypassing a votes canvas. Since this is a common case, we introduce a new operation to allow the user to eliminate an intermediate canvas from a visual join. To perform the **remove intermediate** operation, the user simply clicks on the intermediate canvas' handle (black rectangle in the upper left corner) while in remove intermediate mode. Use of this operation eliminates the need to create the state_votes view in advance.

5. Generalizing Other Work

We believe that VIQING is easy to use, and provides a powerful metaphor for presenting relational queries. To illustrate the latter point, this section describes how VIQING generalizes some commonly-used data presentation metaphors. Space constraints prevent us from illustrating these points with detailed examples, though such examples are quite easy to construct.

VIQING generalizes the functionality of *nested report writers*. Each level of nesting of a nested report is represented as a canvas. A join portal connects each category with a set of sub-categories represented in another canvas. The drill-down operation is performed by entering an instance of the join portal. Roll-up is performed by going back in the portal history to a higher level in the nesting hierarchy.

Master/detail forms [12] are generalized in the same way. A parent canvas of a VIQING join represents the master and the child canvas represents the detail. Users navigate between master and detail forms via portals. Data-entry with master/detail forms could be supported by extending DataSplash to allow users to update the underlying data by manipulating its graphical representation.

VIQING can be used without modification to create *small multiple* [15] visualizations. These visualizations consist of several views of the same data, indexed by changes in a separate data element. In VIQING, this is essentially a visual join of the original data and a table of groups. To generate such a small multiple visualization using VIQING queries, we first create a data canvas, which graphs all the data. This data set should contain an attribute or set of attributes that will be used to group the data into the desired "small multiples." Then, we create a group canvas, which visualizes the set of all groups; this can be done with a visual selection on the grouping attributes. Finally, we create a VIQING join portal in the group canvas that contains the data canvas with an equality predicate on the grouping attributes. The result is that each instance of the join portal contains the data for its group.

Using VIQING to create small multiple visualizations generalizes the functionality of the CrossGraphs [3] commercial system for graph replication. While CrossGraphs supports many built-in visualization types, DataSplash with VIQING supports ad-hoc visualizations without programming.

6. Status and Future Work

An implementation of VIQING as an extension to DataSplash is complete. The resulting system is a unified direct-manipulation interface that combines query specification and data visualization. To our knowledge this is the only system that provides a single high-level metaphor for both of these operations. We believe that the result is natural and easy to use. However, we feel that several improvements could be made.

First, in terms of expressibility, VIQING currently supports only selection, projection and join; it does not support aggregates or nested subqueries. Extending VIQING to support a larger class of queries could be beneficial. In addition, a direct-manipulation method for performing visual selections on graphical attributes other than x and y would improve functionality.

Second, the current implementation of VIQING lacks an automatic way to expose important meta-data (*e.g.*, the database schema) to the user. Meta-data would help users understand the meaning of a visual query.

Finally, direct-manipulation querying relies on an effective and efficient system for visualizing large data sets. The techniques of this paper are largely orthogonal to the techniques for visualizing large amounts of data, but the latter set of techniques are a subject of active research in our group [2,17]. As the technology for visualizing massive data sets matures, we intend to integrate VIQING with it.

7. Conclusions

We have introduced VIQING, a direct-manipulation replacement for the query component of the Tioga boxes-and-arrows environment. By combining querying with visualization, VIQING simplifies and generalizes nested reports and master/detail relationships. VIQING can also be used to generate small multiple visualizations. This generalizes the functionality of CrossGraphs.

The VIQING/DataSplash environment supports the three commonly-used relational operators. Projections can be performed in DataSplash simply by not mapping certain attributes to graphical representations. VIQING supports selections via the rubber-band visual selection operation. Joins are specified visually by dragging and dropping canvases to form join portals.

We discussed VIQING as an extension to DataSplash, but VIQING could also be implemented on other database visualization systems that have portals and a direct-manipulation interface such as Pad.

Acknowledgements

We would especially like to thank Allison Woodruff for extremely valuable discussion and draft editing. Mybrid Spalding also provided numerous helpful comments. We would also like to thank Raghu Ramakrishnan for helpful information about DEVise.

This work was sponsored by the NSF under grants IRI-9400773 and IRI-9411334.

References

[1] Aiken, A., J. Chen, M. Stonebraker and A. Woodruff, "Tioga-2: A Direct Manipulation Database Visualization Environment," *Data Engineering 1996*, New Orleans, Louisiana, February 1996, pp. 208-217.

[2] Avnur, R., J. M. Hellerstein, B. Lo, C. Olston, V. Raman, T. Roth, K. Wylie. "CONTROL: Continuous Output at Navigation Technology with Refinement Online". *SIGMOD 1998*, Seattle, Washington, June 1998.

[3] Belmont Research, Inc., "CrossGraphs: Multidimensional Graphical Reporting and Data Visualization," white paper.

[4] Bier, E., M. Stone, K. Pier, W. Buxton, T. DeRose, "Toolglasses and Magic Lenses: The See-through Interface," *SIGGRAPH 1993*, Anaheim, California, August 1993, pp. 73-80.

[5] Fishkin, K., and M. Stone, "Enhanced Dynamic Queries via Movable Filters," *ACM Conference on Human Factors in Computing Systems*, Denver, Colorado, May 1995, pp. 415-420.

[6] Livny, M., R. Ramakrishnan, K. Beyer, G, Chen, D. Donjerkovic, S. Lawande, J Myllymaki and K. Wenger, "DEVise: Integrated Querying and Visual Exploration of Large Data sets," *ACM SIGMOD 1997*, Tucson, Arizona, May 1997, pp. 301-312.

[7] McDonald, N., M. Stonebraker, "Cupid – The Friendly Query Language," *Electronics Research Laboratory, University of California at Berkeley Technical Report No. ERL-M487*, Berkeley, California, 1974.

[8] Microsoft Access Relational Database Management System, Microsoft Corp., Redmond, Washington.

[9] Olston, C., A. Woodruff, A. Aiken, M. Chu, V. Ercegovac, M. Lin, M. Spalding, M. Stonebraker, "DataSplash," *SIGMOD 1998*, Seattle, Washington, June 1998.

[10] Perlin, K., and S. Fox, "Pad: An alternative approach to the computer interface," *ACM SIGGRAPH 1993*, Anaheim, CA, August 1993, pp. 57-64.

[11] Rasure, J., M. Young, "An Open Environment for Image Processing Software Development," *The 1992 SPIE Symposium on Electronic Image Processing*, San Jose, California, February 1992.

[12] Rowe, L., "'Fill-in-the-Form' Programming," *VLDB 1985*, Stockholm, Sweden, August 1985, pp. 394-404.

[13] Stonebraker, M., J. Chen, N. Nathan, C. Paxson, J. Wu, "Tioga: Providing Data Management Support for Scientific Visualization Applications," *VLDB 1993*, Dublin, Ireland, August 1993, pp. 25-38.

[14] The Tioga Database Visualization Research Group, "Tioga Project Home Page: http://datasplash.cs.berkeley.edu," *Electronics Research Laboratory, University of California at Berkeley*, Berkeley, California.

[15] Tufte, E. R., *The Visual Display of Quantitative Information*, Graphics Press, Cheshire, Connecticut, 1983.

[16] Upson, C., T. Faulhaber Jr., D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, A. VanDam, "The Application Visualization System: A Computational Environment for Scientific Visualization," *IEEE Computer Graphics and Applications*, 9(4), July 1989, pp. 32-40.

[17] Woodruff, A., J. Landay. and M. Stonebraker. "Constant Information Density in Zoomable Interfaces," *Advanced Visual Interfaces 1998*, May 1998.

[18] Zloof, M., "Query-by-Example: A Data Base Language," *IBM Systems Journal*, 16(4), 1977, pp. 324-343.