Zooming and Tunneling in Tioga: Supporting Navigation in Multidimensional Space

Allison Woodruff*, Peter Wisnovsky†, Cimarron Taylor†, Michael Stonebraker*, Caroline Paxson*, Jolly Chen* and Alexander Aiken*

** Department of Electrical Engineering and Computer Sciences*
University of California, Berkeley
Berkeley, CA 94720

† Montage Software, Incorporated
2000 Broadway, 20th Floor
Oakland, California 94612

ABSTRACT

In [STON93] we proposed a visual programming system called Tioga. The Tioga system applies a boxes and arrows programming notation to allow nonexpert users to graphically construct database applications. Users connect database procedures using a dataflow model. Browsers are used to visualize the resulting data.

This paper describes extensions to the Tioga browser protocol. These extensions allow sophisticated, flight-simulator navigation through a multidimensional data space. This design also incorporates wormholes to allow tunneling between different multidimensional spaces. Wormholes are shown to be substantial generalizations of hyperlinks in a hypertext system.

These powerful mechanisms for relating data provide users with great flexibility. For example, users can create magnifying glasses that provide an enhanced view of the underlying data.

1. INTRODUCTION

The design of user interfaces for database systems is an area in need of more attention [BERN89, STON93b]. Existing database user interfaces are often unfriendly and difficult for nonexperts to use. Most database interfaces take the form of textual programming languages or forms-based interfaces oriented towards business applications. In [STON93], we presented Tioga, a new paradigm for user interaction with a database management system (DBMS). Tioga is motivated by the needs of scientific DBMS users in the SEQUOIA 2000 project [DOZI92, STON92, STON93a]. Tioga uses the **boxes and arrows** notation popularized by scientific visualization systems such as AVS [UPSO89], Data Explorer [LUCA92], and Khoros [RASU92]. Tioga improves upon these systems by providing sophisticated data management using the POSTGRES DBMS [STON91]. In the Tioga programming model, boxes represent user-defined database queries, and edges between boxes represent flow of data. This paradigm allows nonexperts to build visual programs called **recipes** by interactively connecting boxes together using a graphical user interface. The underlying data manager is able to optimize and efficiently execute recipes. Figure 1 shows a typical recipe as constructed by a user. The flow of data in this figure ends at a browser which displays the data. A browser is simply

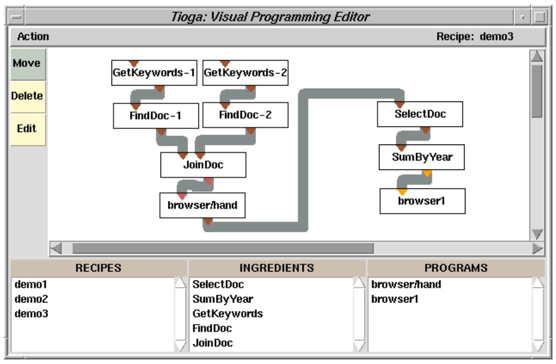

Figure 1
A Tioga Boxes and Arrows Diagram

another type of box that can be attached to a recipe wherever data needs to be visualized.

The Tioga browsing paradigm allows users to visualize data results in a multidimensional space. The browser protocol uses a **flight simulator** interface. A browser gives end users a **joystick** which they use to navigate through their data. Each browser expects data in some multidimensional coordinate system and fetches spatial subsets of this multidimensional space efficiently. The purpose of a recipe is to specify the data to be visualized, access the data through a database management system, and then locate the data in a multidimensional viewing space.

The existing Tioga implementation provides one type of browser. Additional browsers may be implemented by advanced users. In the current Tioga browser, the user chooses two dimensions to be displayed on the screen. Remaining dimensions appear as **sliders** which restrict the objects in the display to those which have values matching the constraints indicated by the sliders. Figure 2 shows a browser displaying objects in a latitude/longitude viewing space that contains California. The current navigational interface allows the user to pan over the two dimensions of the display or to zoom by enlarging a certain portion of the display. Clearly, more sophisticated navigation is desirable.

In [STON93], we explored the basic constructs of Tioga, and provided a query execution model. In [CHEN93], we expanded the Tioga model to interface to foreign systems and provided a notion of transactions for the Tioga environment. In this companion paper, we explore navigation in multidimensional space.

Our extensions to Tioga include:

- *enhanced detail*. Our system must be able to provide enhanced detail as a result of a zoom operation. For example, the Kodak photoCD representation for digital images supports a 2K by 3K by 8-bit color image format, and in addition provides four other images of lesser resolution culminating in a 128 by 192 by 8-bit **abstract** [EAST92]. A user would like the ability to see abstracts on the screen and then zoom in to view the images at a higher resolution. A similar feature was provided by SDMS [HERO80], but it was hard-coded into that execution engine. Hence, retargeting SDMS required a considerable amount of customization.

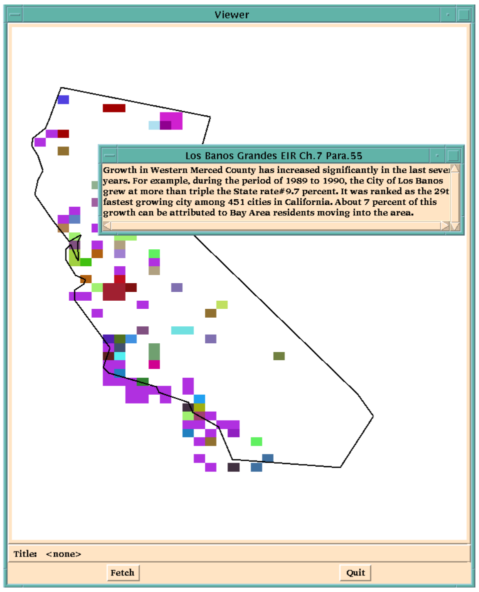

Figure 2
Data Displayed in the Tioga Browser

- *changing multidimensional spaces*. Enhanced detail implies a change in perspective within a multidimensional space. Users also want the ability to switch to a new multidimensional space. For example, a user could zoom in on a map of Berkeley to find the Computer Science building. Additional levels of detail could yield documents corresponding to Computer Science technical reports. These documents should be displayed in a different context than the latitude/longitude coordinates appropriate for the map of Berkeley. When a document is being viewed, a further zoom could yield the image of the author or the layout geometry of his or her office. Again, a different multidimensional space should be used.

- *multiple browsers*. Our system must support multiple levels of detail in the same display. For example, it should be possible to place a **magnifying glass** on a portion of the display and have a zoom operation performed only for the objects under the glass. The remaining objects in the display do not change and remain a context for the magnified data. Because the objects in the magnifying glass are shown with enhanced detail, this function is considerably more complex than simply changing the number of pixels used for display. Support for magnifying glasses requires that browsers be allowed to share windows.

In the rest of this paper, we explore our design in detail. Specifically, in Section 2 we define a zoom capability that allows enhanced detail. We proceed in Section 3 to define wormholes that allow users to change multidimensional spaces. We then turn in Sections 4 and 5 to our design for coordination of multiple browsers. In Section 6 we summarize our findings.

2. SHOWING ENHANCED DETAIL

To eliminate clutter in the display and to orient the user, data should have different representations when seen from different distances in multidimensional space. Intuitively, we wish to extend Tioga with the possibility of **zooming** into data to display more detail about screen objects. In our design, specific browser boxes within recipes are defined to be valid at certain distances from objects. These browser boxes are connected through an **elevation map**. The elevation map specifies all recipes containing browsers that show objects at different levels of abstraction. The map is used to control the invocation of different recipes as the user zooms in and out through the data space.

Specifically, we begin by associating with any browser in any recipe an **elevation range,** over which the browser displays data from this recipe. A browser is associated with a multidimensional coordinate system as noted above. In this presentation, we assume N dimensions which we denote A_1, ..., A_N. We add an N+1st dimension, designated **elevation**, which is used to indicate the user's perspective. This does not represent a physical elevation, but is rather a logical representation of a user's viewing distance from the N-dimensional space.

The current Tioga implementation displays two user-selected dimensions, A_X and A_Y, on the screen. In the original browser implementation, the user can change the range of these dimensions by resizing the window. The range is adjusted proportionally to the change in window size. Therefore, if $XWIN_I$ is the number of pixels in the X dimension in the initial window and $XWIN_W$ is the number of pixels in the X dimension after the

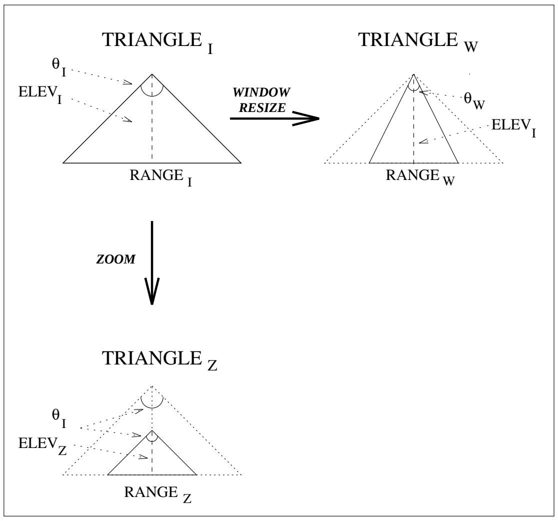

Figure 3
Changing Viewing Angle and Elevation

window has been resized, the range of dimension A_X may be recalculated as follows:

$$RANGE_W = RANGE_I \times \frac{XWIN_W}{XWIN_I}$$

Note that resizing the window has no effect on elevation, as Figure 3 illustrates. Assume that the user's initial position in a displayed dimension is $ELEV_I$ with viewing angle θ_I, as shown in $TRIANGLE_I$. Adjusting the window size while remaining at a constant elevation is analogous to changing the user's viewing angle, as shown in $TRIANGLE_W$.

In the new system, the user is also allowed to adjust the elevation. When the user zooms to $ELEV_Z$, θ_I remains constant, resulting in $TRIANGLE_Z$. Because $TRIANGLE_I$ and $TRIANGLE_Z$ are congruent, $RANGE_Z$ can be recalculated as follows:

$$RANGE_Z = RANGE_I \times \frac{ELEV_Z}{ELEV_I}$$

Adjusting the window size or zooming may select the same range for display. However, the two operations may have very different results. Adjusting the window size does not change the recipe providing input to the browser. Conversely, zooming may place the user in the elevation range of a different recipe. In this case, the recipe providing input to the browser is changed. The recipe valid at a specific elevation is chosen with an

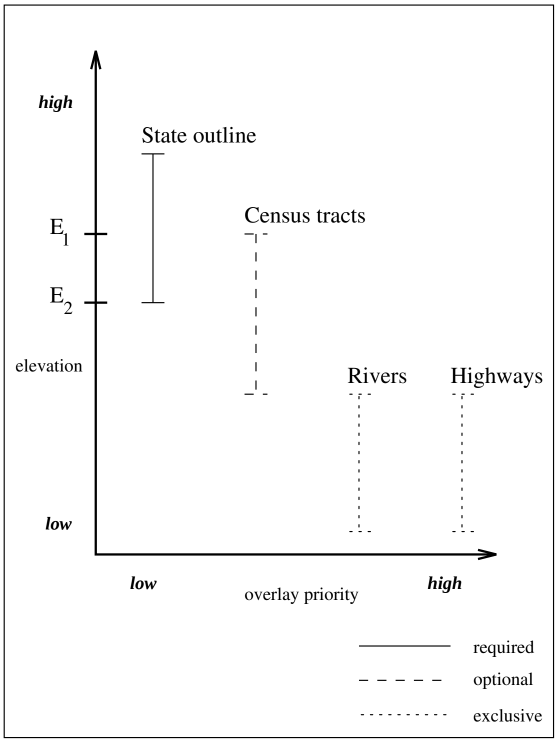

Figure 4
An Example Elevation Map

elevation map.

Figure 4 shows an elevation map containing four recipes, R1 ("State outline"), R2 ("Census tracts"), R3 ("Rivers"), and R4 ("Highways"). Each of these is valid in different elevation ranges. The intended semantics is that recipe R1 produces output for the browser at high elevations. Upon zooming in to elevation E_1, R2 starts being overlaid on R1 to produce a composite output. Further inward at E_2, R1 stops output and only R2 is visible. Further zooming will display output from recipes R3 and R4.

The recipes in an elevation map must have the following properties:

- A browser with the same name must occur in each recipe.
- Each browser expects instances in a common multidimensional coordinate system.

When more than one recipe at a given elevation may provide output to the browser, the elevation map also specifies the **overlay priority** of the recipes, which is shown on the horizontal axis. When conflicts occur in allocation of display space, objects from recipes with higher priority are visible on the screen in preference to those from recipes with lower priority.

The elevation map also contains a collection of semantic restrictions on the display of recipe output. Each recipe in an elevation map can be:

- *required*. In this case, recipe output must be displayed if the browser is at an elevation within the elevation range.
- *optional*. In this case, when the browser elevation first becomes contained in the elevation range of the recipe, the user is prompted as to whether he or she wishes to see output from this recipe. This behavior occurs if the user zooms into the elevation range from above or zooms out from below. At any given elevation range, a mechanism must allow users to turn on or off optional recipes valid at the current elevation. Using this construct, a user can change his or her mind about seeing (or not seeing) the objects from optional recipes.
- *exclusive*. A user can specify a **radio button** behavior for recipes that are valid at common elevations. With this behavior, then at most one of the recipes can be activated, and the user is presented with a menu of radio buttons to indicate which one.

Figure 4 illustrates these constructs in an example. First the user sees the outline of California. Upon zooming, he or she has the option to also see the census tracts in the state. At the next transition point, the outline of the state is no longer visible and the census tracts are optionally visible. Further zooming shows either the rivers or the highways of California, but not both.

Elevation maps allow a user to specify easily the semantics of the zoom operation, assuming that all recipes produce data in the same multidimensional space. We now turn to a mechanism for changing from one multidimensional space to another.

3. CHANGING MULTIDIMENSIONAL SPACES

Consider the following example in which Tioga presents information about the residents of Berkeley. Initially, the application displays a map of Berkeley. Zooming inward gives more detail about geographic objects, culminating with the outline of each individual residence. At this point, the user may invoke a new type of browser, defined for each residence, that displays an image of the people living there. Requesting detail on the residence therefore causes a different multidimensional space to be explored.

Intuitively, the user proceeds through a **wormhole** into a new multidimensional space. On the far side of the wormhole, objects have a spatial relationship that is unrelated to the relationship in effect on the near side. This behavior should be distinguished from a zoom operation where the same spatial relationship is present before and after the zoom. Therefore, we denote this operation of changing from one multidimensional space to another as **tunneling** through a wormhole, to differentiate it from zooming.

When the user tunnels through a wormhole, a new application is invoked taking in a run-time parameter, which is the identifier of the object or objects associated with the wormhole. In the example given above, the object identifier of the house would be passed to the new application to allow it to display only the people living there. It is also possible to define the wormhole over a collection of object identifiers. Specifically, we allow an arbitrary function to identify the objects for which the wormhole is defined. One such function could be:

```
retrieve (House.oid)
where House.architect = "Wright"
```

in which case the wormhole would be defined only for houses designed by Wright.

To construct a wormhole, the user must specify the following information:

- *the wormhole location*. We associate the wormhole with objects displayed by some recipe in some browser. Hence, the location is indicated by the three-tuple:

  (recipe name, browser name, query)

- *the new application that should be run on the other side of the wormhole*.
- *a* **tag** *associated with the wormhole*. Because there may be multiple wormholes for a given object in a given recipe, we require the tag field to allow a user to specify which wormhole should be followed.

In practice, a user-interface gesture indicates the user's desire to tunnel through a wormhole. The user then chooses a wormhole from the list of tags associated with the object(s) selected. At this point, a new application is invoked.

Using the concept of wormholes, a user can move from one multidimensional space to another. The reader can readily observe that wormholes are a substantial generalization of hyperlinks in a hypertext system [NIEL90, MARC88]. A hyperlink is a wormhole in which there is a prespecified object or collection of objects on each side of the hole. Wormholes allow a run-time specified set of objects to be on each side.

4. SLAVING AND CLONING BROWSERS

Two browsers in the same recipe can be **independent** of each other. In this case, when either browser moves in multidimensional space or zooms in elevation, then nothing automatically happens to the other browser. This behavior is appropriate when the browsers are displaying independent objects.

On the other hand, one browser can be **slaved** to a **master** browser. In this case, whenever the user changes the master's position in N-dimensional space, the slave's position in M-dimensional space automatically changes as well. More specifically, during the recipe definitions, the user defines a function, LOC_CALC, that translates requests in the master into requests to the slave:

```
LOC_CALC (N-space-region) --->
M-space-region
```

When the master is moved to $Region_1$, the slave will be instructed by Tioga to move to LOC_CALC ($Region_1$). Both browsers must recalculate their visible rectangles and execute commands to retrieve the objects which will be displayed.

Further, when the master changes elevation, the slave must automatically change elevation at the same time. Usually slaved browsers will be constrained to have the same elevation as their master; however, we allow the user to specify optionally a second function, ELEV_CALC:

```
ELEV_CALC (elevation) ---> elevation
```

In this way, the slaved browser can be constrained to operate at a second elevation that is a function, ELEV_CALC, of the elevation of the master.

There are four interesting ways in which slaved browsers may be constrained:

- Slaved browsers may display data from different regions in a multidimensional space. For example, a user may need to examine all areas 5 miles north of a pipeline to ensure that a toxin carried by the pipeline has not affected these areas. In Tioga, the pipeline could be viewed in one browser. A second browser could be slaved with a function LOC_CALC that maps an N-space-point to a new point 5 miles north.
- Slaved browsers may display the same data seen from different elevations. For example, the slave could show a detailed representation of what is seen in the display of the master. In effect, the slave would show the data of the first browser through a magnifying glass.
- Slaved browsers may display different data from the same points in a multidimensional space. For example, one browser could show color-coded precipitation data for an area while its slave displays temperature data for the same area.
- Slaved browsers may display related data in different multidimensional viewing spaces. For example, a master browser could show a map of California while a slaved browser displays a list of all seismographic sensor stations that are located in the area shown in the master browser. As the user looks at different parts of California, the slaved browser automatically adjusts the list of stations.

The four examples above assume the existence of two browsers in the same recipe. If only one browser exists, it may be more convenient to **clone** it rather than to place explicitly a new browser in the recipe graph. Cloned browsers can be slaved or independent. The examples above demonstrate possible uses for slaved clones. For instance, to display the same data at different levels of detail, the user could clone and slave a viewer to act as the magnifying glass.

Independent clones are appropriate in other situations. For example, it is easy for a user to "get lost" when moving around in a multidimensional space of varying elevation. Hence, the user would like to mark the position of something of interest and return to it at a later time. Cloning a browser allows one browser to remain stationary at the object of interest while the second one continues to browse the multidimensional space.

To support this functionality, we propose the following design. The original browser becomes the **originator** and the new browser becomes the **clone**. The originator and the clone have exactly the same elevation map. During the cloning operation, the user must specify if the clone is to be slaved to the originator, and if so must specify LOC_CALC and ELEV_CALC.

If the originator and the clone are independent (i.e. not slaved), then the second browser is initially assigned the same elevation and spatial location in multidimensional space as the first browser. The joystick of either browser can be moved arbitrarily, and the displays of the two browsers will typically diverge.

If the clone is slaved to the originator, then the clone is constrained to operate at an elevation and location offset relative to the originator; these offsets are determined by LOC_CALC and ELEV_CALC. In this case, the slaved browser operates at an elevation of

ELEV_CALC (elevation of originator)

and at a location given by

LOC_CALC (N-space-region of originator)

If LOC_CALC is the identity function, and ELEV_CALC specifies a zoom, then the clone will provide an automatic magnifying glass, without requiring the user to perform the zoom manually.

Using slaving and cloning, a variety of offset displays and magnifying glass effects can be constructed. The next section describes mechanisms that allow two browsers to share the same screen area, thereby permitting a true magnifying glass effect to be implemented.

5. SHARING WINDOWS

We would like to allow the user to place one browser inside another. For example, the user could place a magnifying glass browser inside another browser displaying a map of California. This is analogous to reading a map of California with the aid of a physical magnifying glass. To provide this functionality in Tioga, we require that browsers optionally be allowed to share their windows. This section explores this construct in detail.

If two browsers share the same multidimensional space, then it is permissible for them to share a browser window. Since a clone and its originator automatically share the same multidimensional space, they are automatically able to share a window.

When two browsers are declared to share a window, they are referred to as the **outer** browser and the **inner** browser. Each browser has separate slider bars that determine the content of its display. The outer browser uses the complete window to display its objects. The display of the inner browser is overlaid on the display of the outer one, thereby creating a single composite display which can be rendered on the screen by the window manager.

To create this composite, Tioga finds the center of the inner browser's current viewing region, and then locates this point in the outer browser's window. Next, Tioga positions the inner browser's viewing region at this location within the outer browser. There are three cases of interest:

- *Case 1:* the inner viewing region is completely inside the outer viewing region. In this case, the two regions are overlaid and displayed.
- *Case 2:* the inner viewing region is completely disjoint from the outer viewing region. In this case, the inner viewing region cannot be seen, and only the outer viewing region is visible.
- *Case 3:* the inner viewing region overlaps the outer viewing region. In this case, Tioga must clip the inner viewing region and then overlay the two displays as above.

These behaviors can be illustrated in the following examples. The user can clone a browser and run both the originator and the clone in a common window. A zoom operation on the inner browser will allow the user to observe a magnifying glass effect, whereby one browser is providing a detailed blowup of the region displayed by the other. Figure 5 shows a magnifying glass (the inner browser) positioned above a window (the outer browser) which is in turn positioned above a map (the multidimensional space).

Consider the case in which the inner and outer browsers are independent. In this situation, the position and elevation of each browser can be changed independently. Moving the inner browser will change its position above the map, and therefore, its contents. Because it moves independently, moving it will also change its position relative to

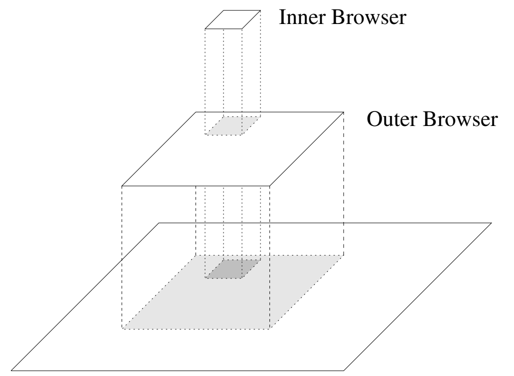

Figure 5
Viewing Regions of Nested Browsers

the outer browser. This will allow the user to magnify various areas of the map. If the user moves the magnifying glass completely out of the region displayed by the outer browser, then Case 2 above applies and the detail will not be shown. If the magnifying glass is partly outside this region, then Case 3 above applies and only a portion of the inner browser is displayed. Similarly, if the user moves the outer browser, the content of the inner browser does not automatically change, although its position relative to the outer browser changes.

To achieve different behavior, we can slave the clone to the originator. Since the inner browser is slaved to the outer browser, moving the outer browser will change the area displayed in both browsers. However, the position of the magnifying glass within the outer browser remains constant. Moreover, if the user zooms the outer browser to blow up the map of interest, say to a map of a specific county, then the magnifying glass also zooms.

The definitions presented above may be recursively extended so that an inner browser may in turn serve as an outer browser. This allows an arbitrary number of browsers to share a window, with a pairwise inner-outer relationship between them. Thus, a hierarchical collection of browsers can be defined. For example, a user may choose to have a magnifying glass on top of another magnifying glass, providing even more detail.

6. CONCLUSIONS

In this paper we have detailed mechanisms that allow navigation in multidimensional space. We first outlined a zoom capability that allows users to view data at different levels of detail. We next introduced the concept of wormholes. Tunneling through a wormhole allows users to view their data in the context of a new multidimensional viewing space. We then proposed mechanisms for linking browsers together as well as for creating new browsers. Finally, we detailed the behavior of multiple browsers sharing a portion of the screen. In combination, these constructs allow users to create effects that simulate magnifying glasses. We are currently extending the existing Tioga system to incorporate the features discussed in this paper.

ACKNOWLEDGEMENTS

We would like to thank Nobuko Nathan, Jeff Sidell, Alan Su, and Jiang Wu for their contributions to this work in both discussion and implementation.

REFERENCES

[BERN89] Bernstein, P.A., et al., "Future Directions in DBMS Research," SIGMOD Record, March 1989.

[CHEN93] Chen, J., et al., "Extending a Graphical Query Language to Support Updates, Foreign Systems, and Transactions," SEQUOIA 2000 Technical Report 93/38, University of California, Berkeley, December 1993.

[DOZI92] Dozier, J., "How SEQUOIA 2000 Addresses Issues in Data and Information Systems for Global Change," SEQUOIA 2000 Technical Report 92/14, University of California, Berkeley, August 1992.

[EAST92] Eastman Kodak Company, "Programmer's Guide for UNIX Systems," *Kodak PhotoCD Access Developer Toolkit*, 1992.

[HERO80] Herot, C., "Spatial Management of Data," ACM Transactions on Database Systems, December 1980.

[LUCA92] Lucas, B. et al., "An Architecture for a Scientific Visualization System," Proc. 1992 IEEE Visualization Conference, Boston, MA, October 1992.

[MARC88] Marchionini, G. and Schneiderman, B., "Finding Facts and Browsing Knowledge in Hypertext Systems," IEEE Computer, 1988.

[NIEL90] Nielson, J. "The Art of Navigating through Hypertext," Communications of the ACM, March 1990.

[RASU92] Rasure, J. and Young, M., "An Open Environment for Image Processing Software Development," Proceedings of 1992 SPIE Symposium on Electronic Image Processing, February 1992.

[STON91] Stonebraker, M. and Kemnitz, G., "The POSTGRES Next-Generation Database Management System," Communications of the ACM, October 1991.

[STON92] Stonebraker, M. and Dozier, J., "SEQUOIA 2000: Large Capacity Object Servers to Support Global Change Research," SEQUOIA 2000 Technical Report 91/1, Electronics Research Lab, University of California, Berkeley, March 1992.

[STON93] Stonebraker, M., et al., "Tioga: Providing Data Management for Scientific Visualization Applications," Proc. 1993 VLDB Conference, Dublin, Ireland, August 1993.

[STON93a] Stonebraker, M., et al., "The SEQUOIA 2000 Architecture and Implementation Strategy," SEQUOIA 2000 Technical Report 93/23, University of California, Berkeley, April 1993.

[STON93b] Stonebraker, M., et al., "DBMS Research at a Crossroads: The Vienna Update," Proc. 19th VLDB Conference, Dublin, Ireland, August 1993.

[UPSO89] Upson, C., et al., "The Application Visualization System," IEEE Computer Graphics and Applications, July 1989.