Exploiting Orthogonality in Three Dimensional Graphics for Visualizing Abstract Data

James Wen

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-95-20
June 1995

Exploiting Orthogonality in Three Dimensional Graphics for Visualizing Abstract Data

James Wen

Department of Computer Science
Brown University
Providence, RI 02912

ABSTRACT

Using three dimensional graphics to visualize abstract data presents an interesting challenge because, by definition, there is no *physically intuitive* counterpart to abstract data. Relatively little work has been done in exploring how to best utilize the third dimension to increase visualization capabilities beyond simply increasing the volume of data presented. This paper presents a design principle that enriches the amount of semantic information that could be visualized by treating the third dimension not so much as one of three equivalent dimensions in three-space but as an independent channel of information.

KEYWORDS: information visualization, abstract data, scientific visualization, physically-based modeling, design principle, orthogonality, symmetry, three-space, 3D transformation

INTRODUCTION

With the ever increasing potential of computer hardware and specialized processors, technologies once considered too expensive computationally to be used as tools for interactive visualizations are becoming available for more general uses. One such technology is interactive three dimensional graphics. Even though it is more readily accessible now, its inclusion into the design stages of various visualization tools still needs to be more fully integrated. While some visualizations, like that of gaseous phenomena, almost require three dimensional graphics, others have no natural—with *natural* in this instance meaning *physical*—counterpart. By definition, there is no physically *intuitive* counterpart to abstract data; there is no natural way they *should* be depicted. The visualization of program structure, for example, is based upon the need to see things that are fundamentally abstract. Because most of these things were proposed through books and papers and because people are more apt to draw than to sculpt, we have become accustomed to visualizing such things as two dimensional entities.

The availability of three dimensional graphics should not, however, make this familiarity with two dimensional visualizations an irrelevant concept of interest only to an obsolete technology. The use of a two dimensional plane for visualizing information has a long and rich history that has produced a large body of work. By taking advantage of the existing work, the third dimension can be used solely as a vehicle to break the two dimensional plane in order to visualize a semantically richer set of data than current visualization techniques.

This paper proposes a philosophy that exploits the existing two dimensional methodologies for visualizaing abstract information, reserving the third dimension to be used in depicting new semantic information. The next section presents this philosophy as a design principle. The following section applies the principle to some fundamental operations in three dimensional graphics to come up with various forms of data visualization aids. Finally, using the techniques developed, a system for program visualization in an object-oriented enviornment is presented as a practical application.

DESIGN PRINCIPLE

The word *orthogonal* has two meanings. Geometrically, it refers to the situation where one object is placed at right angles to, or *perpendicular* to, another object. Analytically, it refers to the case where one notion is *entirely independent* of another. The approach presented here attempts to embody the former definition using three dimensional graphics in order to capture the latter definition when visualizing abstract information. Specifically, the analytic definition of independence can be captured and conveyed graphically by

0

the geometric presentation of perpendicularity. Three dimensional graphics offers an opportunity to visualize the orthogonality using the third dimension without encroaching upon the rich body of work that exists in two dimensional visualization techniques.

By keeping around the notion of planar techniques, a wealth of well studied solutions to the visualization of abstract data can be used. But the reader should keep in mind, however, that in this context, there is not just one two dimensional plane of information. Mathematically, there exists an infinite number of two dimensional planes in three-space and this notion should not be overlooked. The works of Piaget in the area of *object permanence* claims the existence of a mental grasp of an object as an entity, even if the object is not in its original view or in full view [3]. Such notions strongly support the observation that things constrained to a plane, even a free floating plane in three-space, are conducive to being grouped mentally, as sharing something in common—e.g. that they are related to each other under one relationship. The ease with which things can be grouped into macro structures becomes more important as the complexity or volume of data increases. Additionally, just as it is fairly easy to accept that things in a two dimensional plane all conform to some particular constraint, it is also easy to accept *without having to re-focus attention* that things off the plane may be semantically different.

The principle, then, is to attempt to *extend* rather than to replace two dimensional methodologies when exploring visualization techniques in three dimensions. The third dimension should be reserved to convey information that would otherwise serve to clutter or confuse if captured by a two dimensional layout. This is in contrast to systems that discard two dimensional visualization techniques in favor of a completely new system that exists in the more general three dimensional space. The reuse of two dimensional techniques in three space imposes an asymmetry that can be used to encode information whereby things that break the symmetry are seen to break it for a reason and so are easily accepted as being semantically different. Much work has concentrated on simply generalizing the techniques of two-space with a new and symmetrical direction—with *symmetrical* denoting the lack of a preferred direction—now available in three-space. But the third dimension can be seen as a potential to encode information of a *different* nature. The design principle argues for the use of the third dimension asymmetrically so that the notion of orthogonality in an analytic sense (independence) is captured by using orthogonality in a geometric—hence, visual—sense.

APPLYING 3D GRAPHICS TECHNIQUES

This section illustrates the design principle by applying it to various examples. For the purposes of this paper, the fundamental transformations of three dimensional graphics—namely, rotation, translation and scaling—are used to derive useful applications.

In what follows, the reader should bear in mind that the main idea is to present a two dimensional façade and to have additional information, information not easily captured by the traditional two dimensional techniques, conveyed by the third dimension. A user not aware of the three dimensionality of the system should not be at an disadvantage: the system should subsume its two dimensional counterpart. However, a user who does explore the third dimension will be shown much more additional information.

Rotation: Orthogonal Planes

The rotation transformation takes a point and an axis in three-space and rotates the point about the axis. Here, the rotation transformation is used to convey additional information by rotating it *out* of the two dimensional plane. The example that follows presents a way of capturing multiply defined semantic relationships over some given data set. While such information is difficult to show with traditional two dimensional techniques, a concise approach is found by applying the design principle.

Consider a binary relation, R_1, defined over a data set S_1 where $R_1 = \{< s, t > | s, t \epsilon S_1\}$. A traditional two dimensional layout may suffice to visualize such a structure where S_1 becomes the set of nodes and R_1 becomes the set of edges. Now consider another relation, R_2, defined over the set $S_1 \cup S_2$. The sharing of data makes it useful to view the two relations simultaneously but putting everything on one two dimensional graph can become too dense and confusing.

The world of abstract algebra provides a natural setting for the point to be made. First, a notion of a list of properties, call a *category* is defined. An entity is said to *belong to a category* if it has all the properties of that category. A *Ring*, for instance, is a category that requires its members to have the properties of closure under addition, closure under multiplication, and the notion of an additive inverse. Members of a *Ring*—or things that satisfy those properties—include *integers* and *matrices*. A *Field* is a category that requires its members to have all the properties of a *Ring* with the added requirement that there exists a multiplicative inverse. Notice that the *Field* category inherits its properties from the *Ring* category. The set of algebraic categories defines a hierarchy of such inheritance with each child requiring more properties to satisfy for its members [1]. The hierarchy that results is a directed acyclic graph. Figure 1 shows a small example.

The members of categories—those things that have all the required properties of the categories—are called *domains* and they correspond to the familiar domains of computation, e.g. *integers*, *reals*, etc. Domains themselves define hierarchies, but in a different man-

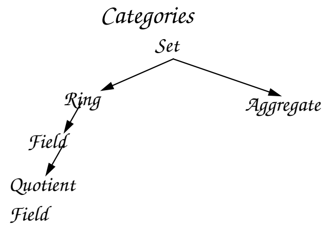

Figure 1: Category hierarchy

ner. *Positive Integers*, *Even Integers* and *Integers mod p* are all *Integers.* See Figure 2.

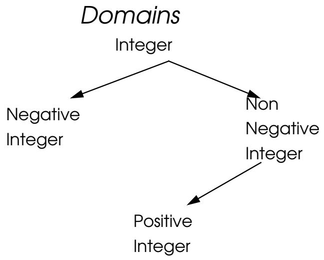

Figure 2: Domain hierarchy

What a user may typically like to do is to scan the graph of categories for a suitable one and then to proceed to examine the domains that belong to that category. Figure 3 shows a graph that tries to embody both classes of data. It requires a closer examination to notice that the relationship between *Quotient Field* and *Field* is different from the relationship between *Rational Number* and *Field.* In an example with just slightly more data, such a mixing of heterogeneous relationships on a two dimensional plane can become very confusing and difficult to read.

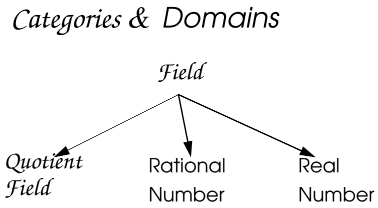

Figure 3: Heterogeneous relationships - both categories and domains

There are, in this example, two sets of relationships that overlap in data. It is desirable to somehow see them together because of the shared data, but without the possible confusion caused by the heterogeneity of data. Figure 4 shows how three-space allows a fairly compact and elegant way of visualizing both simultaneously. Information orthogonal to one abstraction is placed literally in an orthogonal plane. Figure 5 shows successive screen dumps of this example in an actual implemented 3D browser.

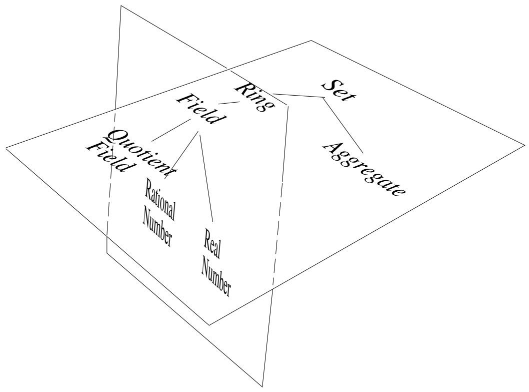

Figure 4: Using orthogonal planes to show multiple relationships.

Translation: Grouping by Height

The translation of an object in three-space refers to the displacement of the coordinates of the object—typically in cartesian space and with respect to either its previous position or to some pre-defined origin. This section examines how the translation transformation can be used to carry information.

Consider a traditional two dimensional layout of a given graph. In this context, the plane of the graph actually exists in three-space so the nodes of the graph can be allowed to slide up and down in a direction normal to the plane. This immediately provides a grouping effect based upon the height of the nodes from the plane. A more interesting effect could be had, however, if an interface existed that allowed a macroscopic control of the nodes in a more global fashion, as described in the example, below.

Define the viewer's position as some point $(0, 0, z)$ with the initial two dimensional layout constrained to the XY-plane. In this system, translational information is hidden in the node's position along the Z direction. A node of interest can be *pinched* so as to have its neighbors rise up with it, but to a lesser extent. The further away a neighbor is to the node of interest, the less it is dragged, as shown in figure 6.

In this way, a "mountain of locality" can be created, where the node of interest is on top and things further and further away are placed further and further down the sides of the mountain. A direct view would, with perspective, make the more interesting and higher nodes that are closer, bigger, and the less interesting nodes that are farther away, less prominent. Figure 7 illustrates how this appears in an implemented system.

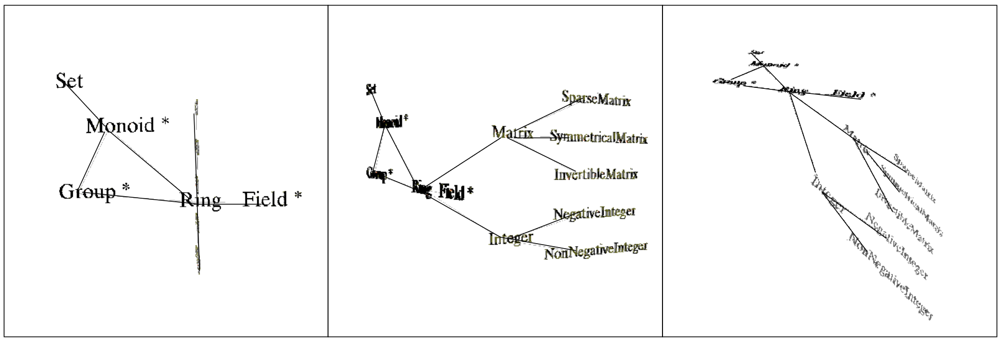

Figure 5: Screen dumps of *Orthogonal Planes*. Left to right: the orthogonal plane comes up perpendicular to main plane; the orthogonal plane is pivoted to allow simultaneous view of both planes; another viewpoint.

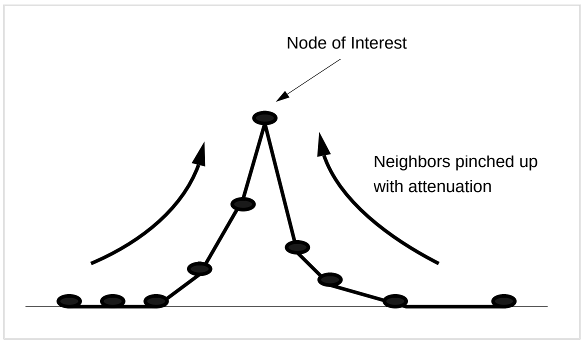

Figure 6: In *pinching*, the node of interest is lifted off the two dimensional plane dragging its neighbors up along with it.

This strategy of focusing on a particular node of interest and having its neighbors recede in prominence creates an effect not unlike that of the work done in information visualization with fisheye perspectives [2, 9]. However, it should be pointed out that this approach is, in fact, very different. The cited papers proposed altering the projected image so as to skew the perspective bringing some nodes forward and making other nodes recede. Such an approach does not allow the user to alter the viewpoint to afford another view of the data since the data does not truly exist in three-space but is only distorted with three dimensional projective techniques. The system presented here creates the focusing effect by actually moving the nodes in object space. By allowing the user to manipulate the actual data structures, the user may focus on multiple points by creating multiple mountains of localities—something not possible with fisheye lenses. Furthermore, the user is free to rotate the object freely in space. The ability to interact with the data object as a three dimensional entity, however, provides the user with an intuition of the data and its structure not possible by more passive means. The last two frames of figure 7 show the graph rotated to some orientation that yields a fuller view of the actual mountain of locality. Note that the second to last frame is shown with boxes instead of text. This is because the boxes are drawn much quicker in the implementation and, because the interactivity of the system is so important in giving the user a grasp of the three dimensionality of the data set, the system draws boxes when heavy interaction is involved to allow for smooth motion.

Scale: Depth Cues

The scale transformation takes all the points that define an object in three-space and multiples each by a given factor. Typically performed with the object at the origin, this operation changes the size of the object. In the context of the design principle, the interest is to look for ways to convey information that do not take away from pre-existing two dimensional technology but adds to it, in the orthogonal third dimension. An application of the scaling transformation is to convey a continuously changing scalar value that is hard to capture using traditional approaches of varying colors or using different shapes. This is examined below.

Flowview, part of the *FIELD* program visualization system [5], is used to visualize the flow of control of a program. Constrained to a two dimensional environment, it runs into some problems when handling recursive functions. Consider two mutually recursive functions, A and B. Figure 8 illustrates how simply visualizing the flow of control does not reveal whether the flow is descending a recursion or returning from one. The figure shows a situation where A invokes B which invokes A which, once again, invokes B. If, in the next step, the flow of control goes to A, it may be difficult to tell if B returned to A or whether B is invoking A. Having the layout fixed in two dimensions makes it hard to encode any information in the nodes themselves except with colors. As seen in the figure, colors—or shapes or fonts, for that matter—do not suffice as they do not vary over time in an intuitive way. What is needed is some indication of whether a routine is being pushed onto the run stack or it's being popped off the run stack.

Such information can be hidden in the third dimension. A straight on view of the system shows a two

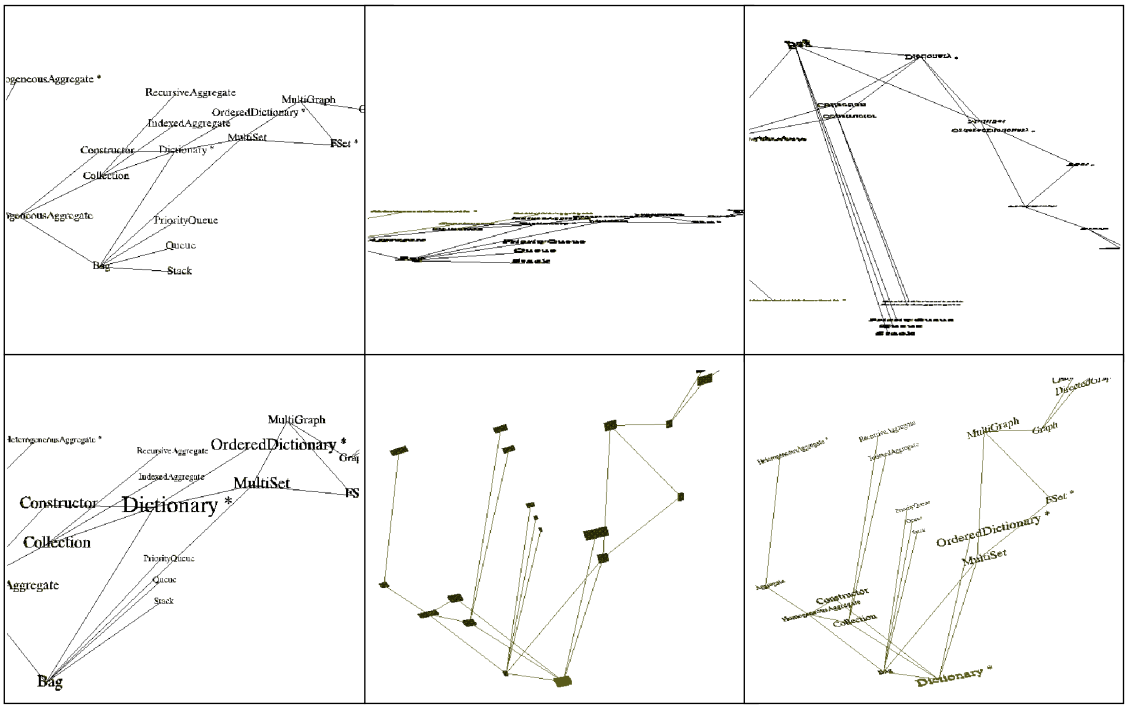

Figure 7: Screen dumps demostrating *Grouping by Height.* Top row, left to right: top view; side view; the **Dictionary** node is pinched off the plane. Bottom row, left to right: top view after pinch; new viewpoint shows mountain of locality as a three dimensional object and the boxes allow real-time interaction; same view with text.

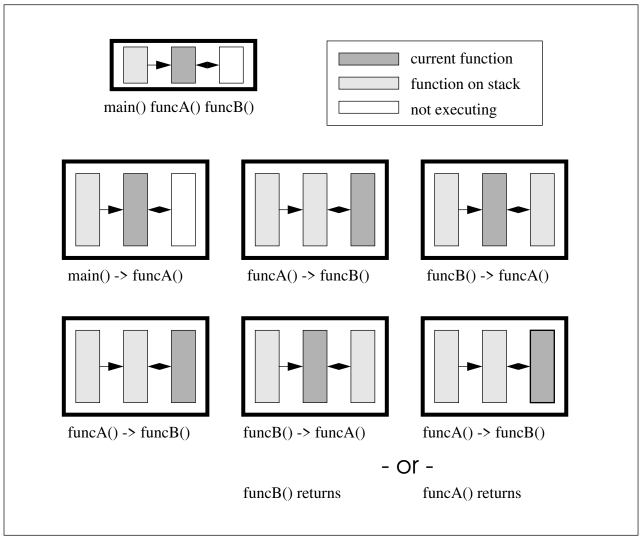

Figure 8: Recursive control flow in a simple call graph showing how it is impossible to tell, in the last two frames, whether the program is descending in recursive calls or returning from them.

dimensional graph. But the number of times a node is invoked can be encoded by changing the thickness, say, of a node. Again, the two dimensional appearance remains undisturbed but, if the graph is rotated, the user can get an instant profile of each node. The thickness of the node can encode any scalar function of the node and, in a programming environment, encoding the number of invocations is a useful notion. An instantaneous indication of a program descending recursions or returning from recursions can be had by simply rotating the graph so that the thickness of the nodes can be seen, as show in Figure 9. This idea is used in the implementation described in the next section and is illustrated in the Color Plates.

APPLICATION: AN OBJECT-ORIENTED BROWSER

Based upon the design principle and using the ideas forwarded in the previous sections, a system for program visualization in an object-oriented environment was built.

Object-oriented languages have various levels of abstractions that may be of interest to a developer. Most of these abstractions involve graphs of various sorts: class hierarchies, call-graphs, class-method relationships, etc. The different abstractions are difficult to show simultaneously due to the clutter that may result. While *flowview* allows the visualization of a call graph it requires another program from *FIELD*,

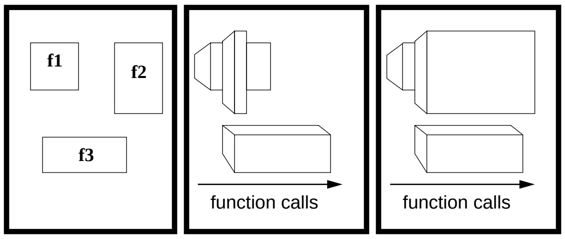

Figure 9: Encoding function calls in node thickness. The first frame shows a two dimensional representation of three functions. The second frame shows the viewpoint rotated to reveal depth in the third dimension. The final frame shows how the depth can changed as a way to profile the function.

cbrowse, to visualize the hierarchies that may exist in an object-oriented environment. These are shown in figures 10 and 11. Combining the two visualizations would clearly be desirable and an application of the design principle would make this possible.

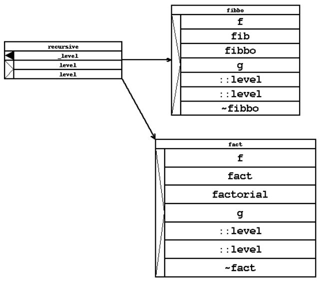

Figure 10: Simple C++ class hierarchy in *cbrowse*

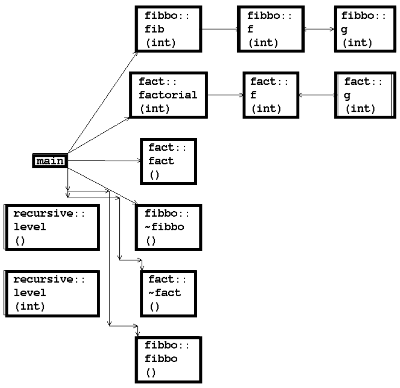

Figure 11: Call graph between methods in *flowview*

The rotated Orthogonal Planes, described above, are well suited for capturing the various levels of abstractions of object-oriented languages because they inherently embody multiple relationships over overlapping data. Using C++ as the prototypical object-oriented language, the application presented here (see Color Plates) shows the class hierarchy on one plane and the methods of the classes on orthogonal planes. In one view, the user can afford both the more abstract relationships between classes as well as the implementation level details of the call graph.

Using the ideas proposed in the section on scaling in the orthogonal direction, the nodes representing the methods are scaled to indicate function invocation. Each time a function is called, the node that represents it is made thicker by a preset factor. Upon returning, the node is scaled back to its original size. A function with ten copies of itself on the runstack is five times as thick as a function that has only two copies. If the program is terminated normally, the run stack should be empty and both nodes should return to their original sizes. The user may also switch over to a mode where a node is made thicker each time it is invoked but is not made thinner when it returns. This affords an instant profiling of the usage of functions with the most heavily used function being the thickest.

The system is implemented in a Sun Sparc 2 Workstation with a GT graphics accelerator card. It runs in real time allowing the user to use the system interactively and gain an intuition of the existence of the browser as a three dimensional object. Response to the application of the third dimension in this way has been very positive, both in usability and in usefulness.

PRIOR WORK

While the body of work in using three dimensional graphics in visualizing abstract data is relatively small, the approaches have been varied. If an intuitive counterpart can be found for an abstraction, the application of three dimensional graphics can be a much easier problem. The success of the 2D desk top metaphor, for example, has led to the development of the 3D office metaphor [4, 7].

Generalizing from a two dimensional abstract model to a three dimensional abstract model, the *cone tree* [6] takes the traditional triangular structure of the typical two dimensional tree and realizes it as a conical three dimensional tree structure. The extension into the third dimension, however, is treated symmetrically to the other two dimensions. That is to say, there is nothing special about the third dimension: it simply extends the storage capacity of the original collection of data and the pre-existing relationship as defined over a two dimensional plane. The limitations of restricting the graph to convey only one relationship is not addressed—only the quantity of data over that relationship is increased. The *perspective wall* [8] is a creative use of the third dimension in an asymmetrical fashion. However, unlike the orthogonal system where interactivity is desirable but not required, this system relies heavily upon real-time animation capabilities to convey the information gained by the third dimension. Such a dependence is both hardware intensive as well as taxing on the concentration and time of the user.

Rather than create three dimensional models, another approach is to use the technology from the mathematics of projective geometry necessary in three dimensional graphics. Ideas based upon the distortion of fisheye lenses have been investigated as ways of focusing on various interesting parts of a given data set [2, 9]. But, because they only work in image space, the data itself cannot be manipulated as a three dimensional object that allows the user to gain an intuition for the relationships as they exist spatially.

FUTURE WORK

Preliminary tests on large graphs have shown that the navigation through a spatial set of data is a very relevant area of study. With the data existing in three-space it can be very easy to get lost in a sea of data with some parts in front, others out of view, and still others "behind" the user.

The ability of the user to customize the placement of the nodes is also found to be highly desirable as the semantics of each particular session may be different and not easily described syntactically. User-defined node positions, however useful, can create a very sloppy system. An interesting and useful direction of studies would be to investigate some sort of constrained data movement in the interest of cleanliness. (Grouping by Height, for instance, is an example of using orthogonality both for input as well as output.)

On the design side, there exists much potential to expand the type of transformations that can be used to break the two dimensional plane when conveying orthogonal information. But, as a design principle, there is no upper bound as to how the ideas presented in the paper can be used to increase the possibilities of the visualization of abstract data using three dimensional graphics.

CONCLUSION

This paper presented a design strategy of using three dimensional graphics in a way that extends rather than replaces two dimensional technologies in information visualizations. By treating the orthogonal third dimension asymmetrically from the other two dimensions, it can be used to convey semantic information that would otherwise be difficult to convey clearly in two dimensions. Planar techniques developed for two dimensional visualizations are retained but the planes now reside in a more general three dimensional space. Information not captured by the planar layout can be encoded in a direction orthogonal to the plane of definition so as to, at once, preserve the original two dimensional layout as well as convey new information. By recognizing the advantages of, and exploiting the wealth of knowledge in utilizing two dimensional space for visualization, it may be possible to use three dimensional graphics to go beyond just visualizing more abstract data but to visualize more complex data and data that is much richer in semantics.

ACKNOWLEDGMENTS

The author would like to thank Steve Reiss and Tom Dean for their support, Scott Meyers for his constant encouragement, helpful advice and insightful suggestions, Jian Xu for his thoughtful questions and useful comments, and Vartan Gregorian for his generosity and inspiring vision.

REFERENCES

[1] John B. Fraleigh. *A First Course in Abstract Algebra.* Addison-Wesley Publishing Company, 1989.

[2] George W. Furnas. Generalized fisheye views. *CHI '86 Conference Proceedings*, pages 16–23, 1986.

[3] Ernest R. Hilgard, Richard C. Atkinson, and Rita L. Atkinson. *Introduction to Psychology.* Harcourt Brace Jovanovich, New York, 1971.

[4] D. A. Henderson Jr. and S. K. Card. Rooms: The use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. *ACM Transactions on Graphics*, 5:211–243, 1986.

[5] Steven P. Reiss. Interacting with the field environment. *Software Practice and Experience*, 20(1):89–115, 1990.

[6] George G. Robertson, Jock D. Mackinlay, and Stuart K. Card. Cone trees: Animated 3d visualizations of hierarchical information. *CHI '91 Conference Proceedings*, pages 198–194, 1991.

[7] George G. Robertson, Jock D. Mackinlay, and Stuart K. Card. The information visualizer, an information workspace. *CHI '91 Conference Proceedings*, pages 181–188, 1991.

[8] George G. Robertson, Jock D. Mackinlay, and Stuart K. Card. The perspective wall: Detail and context smoothly integrated. *CHI '91 Conference Proceedings*, pages 174–179, 1991.

[9] Manojit Sarkar and Marc H. Brown. Graphical fisheye views of graphs. *CHI '92 Conference Proceedings*, pages 83–91, 1992.