An Investigation of Issues and Techniques in Highly Interactive Computational Visualization

by

Michael John M^{C}Guffin

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

An Investigation of Issues and Techniques in
Highly Interactive Computational Visualization

Michael John M^{C}Guffin

Doctor of Philosophy

Graduate Department of Computer Science

University of Toronto

2007

Computers have changed the nature of data visualization in two important ways. First, computers allow increasingly large data sets to be automatically processed. Thus, much research effort has focused on increasing the speed and quality of graphical rendering, to accommodate data sets having greater cardinality, resolution, number of dimensions, etc.

Second, computers enable interactive exploration of different views and depictions of the same data over time. Although *interactive* visualization is now a norm, we argue that strategies for easing and improving this interaction are relatively underexploited, and show this in three case studies. Each study involves the design and implementation of novel techniques to aid visualization in some application domain. We assert the usefulness of a small set of guiding design goals for interactive visualization that are: to ease input, to augment output over space by increasing and improving output in any given state, and to augment output over time by using smooth transitions between states. The utility of these goals is demonstrated, firstly, through the results of each case study driven in part by the goals, and secondly, by using the experience gained from the case studies to generate a set of concrete design guidelines that specify how to better achieve the design goals, and that can be applied in future design work.

The first study considers visualization of 3D volumetric data, and uses spatial defor-

mations to reduce occlusion and increase the visibility of internal surfaces of the data. The second involves visualizing and browsing a rooted tree, and uses a space-filling algorithm to automatically increase the number of nodes visible to the user. The third examines visualization of genealogical graphs, and develops layout techniques to avoid crowding of nodes and edge crossings.

Each of the three studies pays attention to the meaningful subsets that exist within the data, and how to exploit these subsets during interaction. Each study also introduces an interaction technique enabling rapid and light-weight traversal of entire *sequences* of states, with visually continuous transitions across states. Following the case studies, the issues encountered are analyzed, a taxonomy of parameter manipulation is developed, and guidelines for future design are generated to help achieve the original design goals.

Taken individually, each case study contributes significantly to its domain in the specific aspects it investigates. Furthermore, the relative variety in our case studies results in a broad perspective with which to analyze the issues encountered. The unified contribution of the work lies in demonstrating multiple ways of pursuing the design goals, and in the design guidelines subsequently generated that can be applied to situations beyond our cases studies.

Acknowledgements

My doctoral studies have been a rich experience, at times fun, easy, exciting, and ego-boosting, and at other times trying, humbling, sleep-depriving, and exhausting. The biggest rewards I've enjoyed so far from the experience have been the opportunity to publish research, and the researchers and friends I would not have met otherwise.

I benefited immeasurably from having Ravin Balakrishnan as my supervisor. The amount of supervision and direction he gave me matched the amount I wanted: my frequent visits to his office early in my graduate studies were met with an open door, and my later decision to work more independently was encouraged with considerable freedom and trust from him. Much of his advice consisted of key tips or nuggets of insight, and in hindsight, almost every piece of advice he gave me was excellent and valuable. He also responded quickly to requests for help, e.g. to read over documents for comment. Ravin is a strong advocate for his students, and a motivating role model of efficiency, pragmatism, energy, and ambition.

I have also benefited greatly from an environment of many other excellent researchers who have provided me with direction, ideas, and expert knowledge. These include the members of my doctoral committee: Ravin, Gordon Kurtenbach (my former manager at Alias|wavefront when I interned there as an undergraduate, and the person who convinced me to pursue graduate studies), Ronald M. Baecker (who was formerly my professor in a course on CSCW), and Karan Singh (who was formerly my professor in a course on animation) — in preparing this dissertation, my committee's wisdom was to point me in a direction that I initially didn't have much faith in, but that turned out to be more fruitful than I expected — ; my co-authors; faculty and grad students in the Dynamic Graphics Project lab (including Joe Laszlo, Gonzalo Ramos, Maciej Kalisiak, Michael Tsang, Géry Casiez, Pierre Dragicevic, Anand Agarawala, Glenn Tsang, Michael Neff, and many others), Mark H. Chignell's Interactive Media Lab (including Shengdong Zhao and Sandra Jean "Sacha" "pimp my emacs" "it's insanely programmable" "I have too

much fun talking about emacs" Chua) and Ian Spence's Engineering Psychology Lab; the Department of Computer Science at University of Toronto (including Bowen Hui); and fellow researchers met at conferences.

Special thanks are due to my external examiner, Sheelagh Carpendale, whose expertise made her reading of the dissertation a valuable and much appreciated check in quality.

Part of my doctoral research was funded, facilitated, and stimulated by time spent at the IBM Toronto Laboratory, where I benefited from exchanges with Gord Davison, Rebecca Wong, David Budreau, Paul W. Smith, and Jen Hawkins.

Géry Casiez and Pierre Dragicevic helped me hone my ability to discuss research in French, as did the FSL2 ("Frites, salade, les deux?") lunch meetings (in Pierre's words: "Luttons pour l'égalité, la paix et la présence de frites et salade à parts égales dans nos assiettes").

Finally, thanks to the management and wait staff at 168 Bubble Tea Shop #36 (my green-tinted "other office"), especially Vivim; friends who kept me intellectually stimulated (including Eugene Kim and Brian Wong), and thanks most of all to Alicia, my children, and our extended families, for moral support, giving me time to work, and putting up with my frequent blank stares when my mind was on research.

Contents

List of Figures

Chapter 1

Introduction

Computers, and the exponential pace with which they have improved, have enabled the gathering, storage, and processing of ever increasingly plentiful data. At the same time, humans have biological limits on the bandwidth of their senses, the size of their memory, and their ability to process data, which to date have only improved at the glacial pace of natural evolution. These two trends may eventually change; however, for the foreseeable future, they create an interesting mismatch in abilities and a challenge when users wish to access and examine data through a computer. Moderately large data sets that are easily handled by computers can only be perceived and understood by users in the form of summaries, subsets, high-level features, or other reduced representations, and users may not know *a priori* which representation(s) are most useful for a given data set.

Fortunately, computers also allow real-time interactive generation, visualization, browsing, and exploration of many different *graphical* representations of a data set, or of different views of a single representation. Graphical representations are of particular interest because they can exploit the human visual system's relatively high-bandwidth capabilities, and can enable the user to discover patterns or salient features in the data that are difficult to extract automatically or that are not defined in advance. Graphical representations are also the most natural representation for data that is inherently spatial (i.e.

for which there is a preferred or natural embedding), such as geometric models or data physically sampled at different spatial locations.

Much of the research in visualization can be classified into three overlapping threads. The first and most basic of these focuses on developing new graphical representations and depictions of data. Examples include Chernoff faces [Chernoff, 1973] for multivariate data, Treemaps [Johnson and Shneiderman, 1991; Shneiderman, 1992b] for rooted trees, and videograms [Davis, 1995] for video. The design and discovery of new graphical representations is a fundamental pursuit, pre-dating *computational* visualization. Although computers make the generation and use of a given graphical representation more practical, computers are not *necessary* for any static depiction that could, after all, be drawn on paper.

The other two threads of research have arisen as a result of the use of computational devices. The second follows from computers allowing increasingly large data sets to be automatically processed. Within this thread, research has focused on increasing the speed and quality of graphical generation and rendering, to accommodate data sets having greater cardinality, resolution, number of dimensions, etc. Faster algorithms and numerical methods (e.g. [Engel *et al.*, 2001; Frick *et al.*, 1994]), bounds on space and time complexity (e.g. [Buchheim *et al.*, 2002]), and guaranteed frame rates (e.g. [Munzner, 1998]) are examples of the kinds of results of such work.

The third and final thread focuses on issues in *interaction*, and how to best support interactive visualization. Interactive visualization is now a norm, in part due to increasingly fast and inexpensive hardware. We now also have the capability for *highly interactive* visualization, where the user may rapidly rearrange and change their view of the data, even with rudimentary pointing devices for input and status quo video displays for output. Nevertheless, much potential remains unrealized. In our work, we have found examples of strategies for increasing interactivity that are well established in the human-computer interaction (HCI) community, such as popup interfaces that combine gestures

and menus (e.g. marking menus [Kurtenbach and Buxton, 1993], control menus [Pook *et al.*, 2000], flow menus [Guimbretière and Winograd, 2000]), but that have not been applied to benefit visualization. We have also found opportunities to improve output, by making fuller or better use of screen space.

Given the myriad different representations and views that a user might explore during interaction, there is value in trying to maximize the content of any given view, and in facilitating the transition between useful views, to ease browsing and reduce the total exploration time necessary for a user. To address these concerns, we formulate a small set of design goals for interactive visualization, detailed in section 2.2, that serve as a launching point for our work. These goals are to ease the input required for effecting changes in state, and to enhance output both in each state of the visualization (i.e. by increasing or improving the output) and over time across state transitions (by displaying visually "smooth", continuous feedback). We demonstrate the results and benefits of pursuing these goals in three case studies. Each study involves the design and implementation of novel interaction techniques to aid visualization in some application domain. Each case study also investigates a different way of exploiting meaningful subsets within the data during interactive transitioning between views of the subsets.

The first study considers visualization of 3D volumetric data. The subsets involved, corresponding to different regions of space, are of two kinds: *semantic layers*, such as layers of tissue in a medical data set; and regions of space enclosed by geometric primitives (such as planes, boxes, spheres) that can be positioned by the user. Displaying many or all of these subsets in 3D creates occlusion problems. To address this, we allow spatial deformations to be applied in novel ways to the subsets, which can increase the visibility of many internal surfaces of the data. The application of these deformations enable screen space to be better used, which improves output, and the use of marking menus and 3D widgets enables light-weight input.

The second study involves visualizing and browsing a rooted tree, where the subsets

are subtrees. Output is increased by filling screen space through automatic revelation (i.e. expansion) of subtrees. As a side benefit, this can also reduce the number of selections required of the user to expand nodes, which reduces the input required of the user. A controlled experiment demonstrates that, although our design does not always benefit the user for the task studied, it does sometimes result in significantly improved performance.

The third study examines visualization of genealogical graphs, i.e. networks of individuals connected by family relationships. We show that attempts to draw entire genealogical graphs lead to severe crowding of nodes, and thus propose visualization and browsing based on meaningful subsets that don't suffer from the same crowding, namely, trees of ancestors and trees of descendants. Although this restricts which nodes are visible to the user at any given time, the visible subsets scale better in terms of edge crossings, demonstrating a different way of improving output. Input is also improved, as transitions between subsets are facilitated by the use of marking menus and a "subtree-drag-out" popup widget.

All three case studies also involve the use of smoothly animated transitions in response to user actions that change the view. Although the use of animated transitions is not novel, each of the case studies also introduces an interaction technique enabling rapid and light-weight traversal of an entire *sequence* of states, with smooth transitioning between each state.

The next chapter provides background on visualization in general by surveying much of the most important literature, and also outlines the design goals of our work in section 2.2. Chapters 3 through 5 then present the three case studies, which are also described in separate publications [McGuffin *et al.*, 2003; 2004; McGuffin and Balakrishnan, 2005b]. Following these, Chapter 6 analyses issues encountered in the case studies, develops a taxonomy of how parameters in a visualization can be interactively manipulated, and proposes a set of general guidelines for future design work. Finally, Chapter 7 summarizes our conclusions, contributions, and potential future directions.

Chapter 2

Background

2.1 Overview of Visualization

The word *visualization* has traditionally referred to an internal mental process: the act of "[forming] a mental image of (something not present to the sight)" [Guralnik, 1984]. We use the word in a more modern sense, to refer to either (a) the rendering[1] of data or information into graphical forms or representations that can be visually perceived by humans, or (b) the study of algorithms and techniques for such rendering. One might say that visualization, in the modern sense of the word, is intended to aid visualization, in the older sense of the word. More concretely, the goal of visualization is to aid examination of data, and thus accelerate discovery and insight.

Some overviews [Card *et al.*, 1999, chapter 1] [Thomas and Cook, 2005, Table 2.2] give over a dozen distinct ways that visualization can amplify cognition. One interesting example, of several possible ones, is how visualization can support *epistemic actions* [Kirsh and Maglio, 1994; Neth and Payne, 2002]. Epistemic actions are external, physical actions performed to change ones internal computational state and to make "mental

[1]Since "visualization" derives from the same root as "vision", the word suggests the formation of perceptual input for a (human) observer, or the act of perceiving that input. Perhaps a more observer-neutral term would be "graphicalization", emphasizing rendered output from the computer.

computations easier, faster, or more relaible" [Kirsh and Maglio, 1994], and are contrasted with pragmatic actions that are performed to achieve some goal in the physical world. For example, the rotation of shapes in the game Tetris [Kirsh and Maglio, 1994] is sometimes performed by players to aid cognition about the game rather than to achieve a target orientation to advance in the game. Although computational visualizations are typically more abstract than the physical world, they also typically allow users to "play with" data and look at it in many different ways, which can have similar benefits for cognition.

Of course, human vision is not the only sense through which data can be communicated, however it is the sense with the greatest bandwidth, and other modalities such as audio or haptic output are beyond the scope of our consideration. Thus, the *output* from a visualization algorithm is limited to at most 3 spatial dimensions (which could be projected onto a 2D output device), and 1 temporal dimension (which is used, for example, in animation, video playback, and implicitly during interaction). In contrast, the data *input* to a visualization can have arbitrary structure and dimensionality (Figure 2.1).

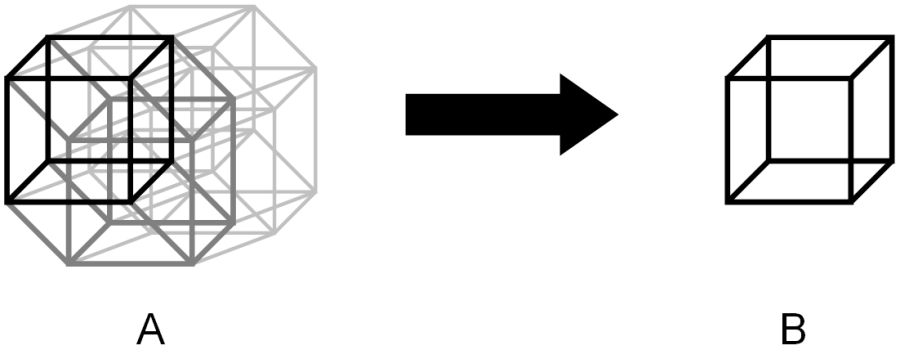

Figure 2.1: Visualization as a mapping. *A:* The input data may have arbitrary cardinality, dimensionality, connectivity, or other structural properties. (Here, a hypercube is used to symbolize such data, to suggest potentially high dimensionality and/or complexity.) *B:* The output graphical representation, regardless of the output device or medium, is limited to at most 3 spatial dimensions and 1 temporal dimension. Due to limitations of the human visual system, the space of colours used in the output is further limited to 3 dimensions, and even the spatial resolution of the output is bounded by the angular resolution of the human retina (on the order of 1 minute of arc per "pixel" in the fovea [Ware, 2000]).

The design of good visualizations depends, at the lowest level, on the capabilities of

the human visual system, and is thus informed in part by knowledge of human perception. High-level design principles for static (e.g. printed) visualizations have grown out of fields such as cartography, statistics, and graphic design. The advent of computers with real-time interactive graphics [Sutherland, 1963] has spawned the study of new issues and design principles surrounding *interactive visualization*. Collections of design principles, overviews and taxonomies of the field of visualization, and other key references include [Bertin, 1967; 1977; Tufte, 1983; 1990; 1997; Shneiderman, 1996; Card and Mackinlay, 1997; Chi, 2000; Tory and Möller, 2004; Wilkinson, 1999; Card *et al.*, 1999; Ware, 2000; Spence, 2001; Card, 2003; Thomas and Cook, 2005]. Some of the most significant recent research in visualization is published in the annual proceedings of the IEEE Visualization (VIS) conference, held since 1990, and in the co-held IEEE Symposium on Information Visualization (InfoVis), held since 1995. There are also dozens, if not hundreds, of other conferences and journals related to human-computer interaction, computer graphics, and human perception where research in visualization is reported.

The following sections consider subfields within visualization, divided according to the type of data involved, and then divided according to the commonly used terms *scientific visualization* and *information visualization*.

2.1.1 Visualization of Multi-dimensional Multi-variate (mdmv) Data

The term *mdmv* [Wong and Bergeron, 1997] refers to a major class of data studied in visualization. Formally, the data are points or tuples, each having a number of coordinates or attributes, and a data set is a subset of a cartesian product. A useful distinction is made between *dimensions* and *variables*: the former are domain sets (independent quantities), the latter are co-domain sets (dependent variables). Consider, for example, the flow of a fluid past an object such as a wing. A simulation of the flow might involve a regular grid of points, for each of which values are stored for pressure, temperature,

and velocity, updated at each time step. The data set from such a simulation would be a 4-dimensional 5-variate (4d5v) set: 3d for space, 1d for time, 1v for pressure, 1v for temperature, and 3v for velocity.

Unfortunately, the visualization literature does not consistently distinguish between dimensions and variables in this manner — they remain overloaded and ambiguous terms. Nevertheless, the distinction made within mdmv allows for the inclusion and comparison of many kinds of data. Mathematical functions and relations involving any number of continuous and/or discrete quantities can all be considered mdmv data sets. The tuples in a relational database table are a kind of multi-variate data set, with each column corresponding to a variable (some authors refer to this as cases × variables, or objects and attributes [Card and Mackinlay, 1997] [Card *et al.*, 1999, p. 18] [Card, 2003, p. 554]). Measurements recorded in a controlled experiment, with their associated statistical distributions, are also mdmv data sets. Volumetric data arranged on a grid, such as CT, MRI, or ultrasound scans, are 2- or 3-dimensional multi-variate data sets. Video footage is usually 3d3v data (2 spatial dimensions, 1 temporal dimension, and 3 variables for the primary colour components).

Many visualization techniques for mdmv data grew out of efforts by statisticians to effectively present data. Overviews of visualization techniques from this statistical perspective are given by Tukey[2] [1977] and by subsequent authors [Chambers *et al.*, 1983; Cleveland, 1985; Cleveland and McGill, 1988; Cleveland, 1993]. Further back in history, William Playfair (1759–1823)'s work [Playfair, 1786; 1801] was the first to make extensive use of statistical graphics of empirical data, and contains the first instances of, or inventive uses of, line graphs, bar charts, pie charts, circle charts, a diagram similar to a Venn diagram (predating John Venn's birth) and glyphs [Spence, 2005; Spence and Wainer, 2005]. Underlying most of these is the essential notion of a coordinate system, the origin

[2]Known for having coined the words *software* in 1958 and *bit* in 1946. Tukey also invented the box plot and other graphical techniques.

of which is attributed to René Descartes (1596–1650), who introduced coordinate systems in 1637 in *La Géométrie*, the 3rd appendix to his *Discourse on the Method* [3]. It is for this that the *cartesian* plane is named: Cartesius is Descartes' Latin name. This marked the birth of analytic geometry, a combination of geometry and algebra, and was described by 19th century philosopher John Stuart Mill as "The greatest single step ever made in the progress of the exact sciences." [Anton, 1988, p. 3].

Wong and Bergeron [1997] give an overview of modern research in mdmv visualization, with emphasis on issues such as data sets involving a large number of variables, projection onto lower dimensional spaces, interaction, and the discovery of patterns (such as correlations) in the data.

Although mdmv data often involve many dimensions and variables, it should be kept in mind that we are not limited to only mapping these to the at-most 3 spatial dimensions and 1 temporal dimension in a graphical representation. Attributes of data can also be mapped to other visual features, such as the colour of individual pixels, or to the colour, size, orientation, shape, and motion of glyphs or icons (see [Bertin, 1967; Chernoff, 1973; Kleiner and Hartigan, 1981; Pickett and Grinstein, 1988; Beddow, 1990] for early examples in 2D, or see [Ward, 2002] for a survey). In addition, *interactive* visualization allows for dynamic, transient mappings and remappings of attributes, allowing any number of attributes to be explored by a user.

2.1.2 Visualization of Graphs

Another major category of data sets are connective structures: a set of elements that are linked together in some fashion. In other words, graphs and variations on them, such as trees, multitrees [Furnas and Zacks, 1994], polyarchies [Robertson *et al.*, 2002], directed acyclic graphs (DAGs), hypergraphs and multigraphs [Cormen *et al.*, 1990], Ted

[3] Famous for containing the phrase *cogito ergo sum*, "I think therefore I am". The full name of the work in French is *Discours de la méthode pour bien conduire sa raison et chercher la vérité dans les sciences.*

Nelson's zzstructures (see [McGuffin and schraefel, 2004] for a description of these in graph-theoretic terms), and hierarchical graphs or compound graphs. Graphs are used to model many situations, such as social networks, communications networks (e.g. the internet), hypertext and hypermedia structures (e.g. the World Wide Web), components of software source code (e.g. call-graphs), genealogical relationships, electrical circuits, relationships between words in a thesaurus, and biochemical reactions in cells relevant to genomics and proteomics. The nodes and edges of graphs may be weighted, coloured, typed, labelled, or contain some other additional data, allowing for many aggregate and hybrid forms (the *scene graphs* used in computer graphics are an example of this). Herman et al. [2000] give a survey of visualization techniques for graphs.

Unlike the case with mdmv data, the information in a graph is usually thought of as essentially topological, not geometric. Representing a graph *graphically* requires choosing a mapping from nodes to spatial locations, i.e. *embedding* the graph in space, because nodes don't generally have any preferred spatial coordinates. Thus, a highly relevant field is that of graph drawing, which might be described as an outgrowth of graph theory and computational geometry, and which studies the algorithmic embedding of graphs in 2- and 3-dimensional space. To date, the graph drawing community has focused largely on the efficiency of graph drawing algorithms, in terms of time and memory required, and also in terms of the output (e.g. How much area is covered by the embedding? How long are the edges? How many edge crossings are there?) There is also growing interest, however, in perceptual, higher-level aesthetic, and interaction issues. An overview of the field is given in [Di Battista *et al.*, 1999]. A major venue for research in this area are the annual proceedings of the International Symposium on Graph Drawing (GD), held since 1992 and published by Springer-Verlag. Examples of historically important algorithms to emerge from this field are force-directed layout for general graphs (e.g. the spring embedder [Eades, 1984]), Sugiyama et al.'s [1981] layout algorithm for DAGs, and the Reingold-Tilford algorithm for drawing rooted trees [Reingold and Tilford, 1981], and

there is also significantly earlier computer-based work [Baecker, 1967].

Rooted trees are a very important and widely-used structure, and dozens of distinct representations have been proposed for them. These all boil down to variations, extensions, or combinations of the 4 basic representations in Figure 2.2. Identification of some of these representations (but not all four together) as basic forms is found in [Bertin, 1967; Knuth, 1968; Johnson and Shneiderman, 1991; Eades *et al.*, 1993].

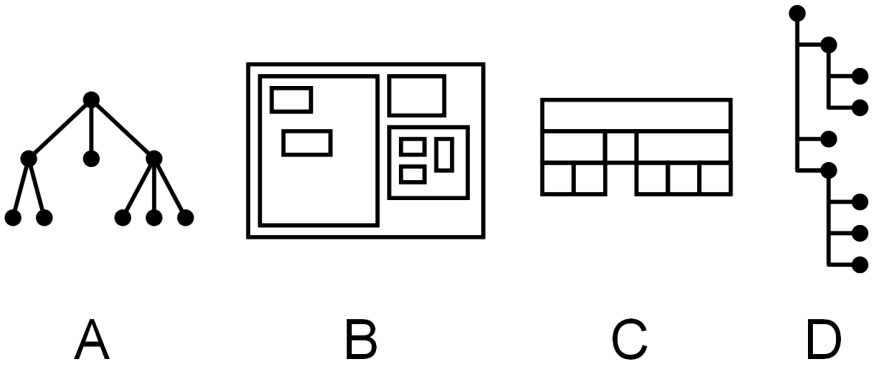

Figure 2.2: Different graphical representations of the same rooted tree. *A:* node-link (see [Wetherell and Shannon, 1979; Reingold and Tilford, 1981; Walker II, 1990; Buchheim *et al.*, 2002] for classical drawing styles of this). *B:* nested containment, or enclosure (used, for example, in Treemaps [Johnson and Shneiderman, 1991; Shneiderman, 1992b]). *C:* a layered "icicle" diagram (e.g. [Barlow and Neville, 2001]) that uses adjacency and alignment to imply the tree structure. *D:* an indented outline view (see [Venolia and Neustaedter, 2003] for an interesting variation on this).

In comparison, general graphs are less constrained than trees, and therefore do not afford as many orderly drawing conventions. Graphs are usually drawn using a node-link representation (Figure 2.2, A). It is possible, however, to represent graphs in ways analogous to Figure 2.2, B, such as with Venn diagrams, or hybrid representations [Harel, 1988; Sindre *et al.*, 1993].

It is also possible to represent a graph as an adjacency matrix. For example, if each node has a corresponding row and column of pixels, then each edge can be represented by filling in the pixel at the intersection of a row and column. Such a graphical representation has the advantage of eliminating all edge crossings, however the appearance of

the representation depends heavily on the ordering chosen for the nodes.

A final observation, which relates this section to the previous one, is that graphs can be thought of as a kind of relation. For example, a directed graph $G = (V, E)$ can be viewed as essentially the subset $E \subseteq V \times V$, which is a relation. If the nodes in V are numbered $1, \ldots, \|V\|$, then a graphical "plot" of E corresponds directly to the adjacency matrix of G. Thus, graphs can be thought of as mdmv data sets rather than as topological entities, however because there is no preferred ordering for the nodes, the variables (node sets) of the mdmv set are nominal (also called categorical) rather than ordinal or continuous.

2.1.3 Other Data Types

Some data sets involve aspects of both of the previous categories. For example, in a relational database, the tables in the whole database are interrelated through their foreign keys, establishing a directed graph. At the same time, within any given table, each row is a tuple, and the entire table is an mdmv data set. As another example, a group of cities may be interconnected by a network of roads, and also have attributes for each city, such as latitude and longitude (establishing a preferred, geographical embedding in the plane), population, area, precipitation, etc. As a 3rd example, a graph whose structure changes over time involves both connective information and a temporal dimension usually associated with mdmv data.

There are also basic varieties of data not explicitly mentioned in the previous sections. The most important is text (which potentially includes source code, hypertext, etc.), which could be classified as a 1d1v kind of data, but which has such particular properties and conventions for presentation and use that it is perhaps best considered apart from other mdmv data, as Card et al. [1999] do.

Another data type not yet mentioned are geometric models, such as geometric configurations in the plane, or 3D polyhedral surface representations of physical objects,

or higher-dimensional geometric objects such as hypercubes. These can be considered mdmv data, where each variable (in the mdmv sense) corresponds to a geometric dimension (in the common, spatial sense). Mathematicians also have an interest in visualizing general manifolds, which may have arbitrary dimensionality and are not necessarily Euclidean. Although every manifold has an inherent spatial dimensionality, it is sometimes the topology (or graph-theoretic "connectedness") of the manifold that is of interest. Visualizing such spatially high-dimensional objects, such as 4D polyhedra or manifolds, is usually done by projecting, (re-)embedding, or slicing out a lower-dimensional version of the manifold.

2.1.4 Scientific Visualization *versus* Information Visualization

Visualization as a discipline is often divided into *scientific visualization* and *information visualization*, however this division seems to be at least partly a historical accident.

According to [Rosenblum, 1994, p. 61], the birth of scientific visualization as a recognized field is associated with a report [McCormick *et al.*, 1987] by the United States' National Science Foundation's (NSF) Advisory Panel on Graphics, Image Processing, and Workstations. Scientific visualization has grown out of applications of computer science to the natural sciences, especially physics, such as in the simulation and measurement of physical quantities. Many of the researchers participating in this have come from the computer graphics community, and have paid much attention to filtering, image processing, and other data treatment techniques.

According to [Card *et al.*, 1999, p. 8], the term "information visualization" was introduced in [Robertson *et al.*, 1989], making it only slightly more recent. As a field, it seems to have grown more out of, and been influenced more by, the human-computer interaction community, rather than the computer graphics community.

Various definitions have been proposed for these two terms. Scientific visualization and information visualization have been said to correspond to (1) data that is inherently

spatial, or not, respectively; (2) data that represents something physical, or not, respectively; (3) situations where the embedding is given, or chosen, respectively; (4) data that is continuous, or discrete, respectively; (5) representations that are geometric, or symbolic, respectively. It has also been suggested that, rather than being disjoint fields, scientific visualization overlaps with information visualization, or even that the former is a subset of the latter.

Whatever distinctions we may define between the two fields, there are now many examples of research which cannot be strictly classified in only one field or the other. For a recent discussion of how scientific visualization and information visualization can be fit within a larger and more satisfying taxonomy of visualization work, see Tory and Möller [2004].

2.2 Design Goals for Highly Interactive Visualization

This section sets three broad design goals for improving user interfaces and enhancing interactivity in visualization. Briefly, they are to increase the amount and/or quality of output, to ease input, and to display smooth visual feedback during state transitions. These goals motivate much of the design in the case studies described in Chapters 3 through 5. After the case studies, Chapter 6 retrospectively analyses the issues encountered, and proposes a set of design guidelines to help achieve the goals in future work. In particular, the design guidelines in Chapter 6 concern how to design input and output for parameter manipulation, which is a core concern for interactive visualization.

Our goals all aim to improve communication between the user and machine and enable faster interaction, hence the name *highly interactive visualization* for this research. Figure 2.3 sketches the concept as aiming to reduce the amount of input required and also augmenting the output given in response to input. Augmenting output can be done

over both space (i.e., increasing and improving output in any given state) and over time (i.e., displaying smooth transitions between states).

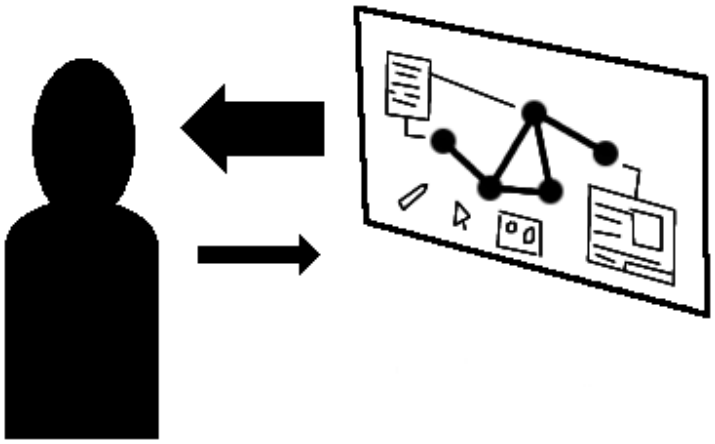

Figure 2.3: A user interacting with a computational visualization. Loosely speaking, we seek to increase the ratio of output-to-input.

Increasing output takes advantage of the computer's ability to process and display large amounts of data; the system's output channel is limited mostly by human abilities. Easing input allows the user to concentrate more on the interpretation and browsing of data rather than the operation of the interface, and also allows the user to browse more data in less time. Ideally, the system should display a maximum useful amount of information in response to the minimum amount of input necessary to convey the user's intentions and interests.

In increasing output, we should not simply pack the most bits possible into every pixel without regard to human perceptual abilities. The information that is output should be useful to the user and should lend itself to easy interpretation so the user can quickly scan it. Too much raw information may be as much of an obstacle to performance as too little. In some cases, increasing the *useful* output to the user may involve improving the presentation of output, or even reducing the raw number of bits presented, rather than increasing the number of bits. In some sense, we want the output of the visualization to fit or match the size of the display, and the human's abilities, as best as possible. Thus, our first goal is to *increase and improve output.*

The second goal is to *ease input*, meaning make input easier, faster, lighter-weight, requiring fewer operations. For example, to transition from one view of the data to another, the command(s) or action(s) invoked by the user should be as simple as possible.

In addition to considering output for each view of the data, or in each state of the visualization, it is important to consider the output *over time* during a transition between views or states, as this is an essential component of interactive visualization. We should seek to design visual depictions of state transitions to be as useful and informative as possible. Typically, each transition is very brief, and the user's main interest during a transition is in arriving at the new state. So we do not necessarily want to maximize the information displayed during a transition. However, it is reasonable to have the system display output during a transition that shows or clarifies relationships between the starting and ending states. Our third subgoal is thus to *use smooth transition*, i.e. to make the transitions between states appear continuous.

As will be seen, in trying to achieve these goals, rather than there being tradeoffs between them, we instead find many techniques which help achieve two or all three of the goals simultaneously.

2.2.1 Increasing and Improving Output

The amount of useful output from a visualization is often significantly lower than the upper bound on what could be output (e.g. making use of every pixel). Increasing useful output can not only make better use of the user's visual system, but also indirectly reduce the need for input, since the user may need to traverse fewer states of the visualization to gain whatever information or insight they seek from the data, and may allow the user to reach a target state faster by selecting and navigating through data elements further along a desired path, making greater progress with each selection.

At times, underutilizing available display space can be desirable if the user has deliberately focused down to some small amount of information, and the user doesn't need

or want to see any more. However, in exploratory browsing, often neither the system nor the user knows precisely what information is of interest, which motivates trying to make the fullest and best use of available space. Certainly, this must be balanced with the need for *some* whitespace to enable an orderly arrangement and help the user scan and interpret information, however it nevertheless seems reasonable to try to keep such blank space to the minimum necessary and maximize the information shown. Space-filling techniques are one way of trying to make full use of available space. These are best applied in situations where the data is embedded in 2D rather than 3D, since filling 3D space tends to aggravate occlusion problems rather than make more data visible. Automatic space-filling reduces the amount of input required by making selection of data more implicit: whatever the user selects as a focus is automatically extended as far as possible to fill the view. A natural application of space-filling techniques is in the visualization of trees. For example, Treemaps [Johnson and Shneiderman, 1991; Shneiderman, 1992b], Pad++'s directory browser [Bederson and Hollan, 1994], Nguyen and Huang's [2002] space-optimized trees, RINGS [Teoh and Ma, 2002], and SpaceTree [Plaisant *et al.*, 2002] all fill space with the descendants of a user-selected node. In our work on Expand-Ahead (Chapter 4), we develop a variation on these techniques that, like SpaceTree, maintains a constant node size, but which can fill space more aggressively than SpaceTree. Unfortunately, as with any space-filling technique, there is a risk that filling the user's view with too much information may hinder the user's performance, if the information is difficult to visually scan. We investigate this issue experimentally in the case of Expand-Ahead.

In contrast to underutilizing space, the complementary situation could be described as "overutilization" of space, where many graphical elements are projected onto the same region. Occlusion is a common problem in 3D visualization, and perhaps the most severe kind of occlusion occurs with volumetric data, where voxels densely occupy a volume of space. Many techniques exist to allow a user to see the inner regions of volume data, but

these usually come at the cost of removing surrounding contextual data. Our research on browsing volumetric data (Chapter 3) uses deformations to pull, peel, or spread apart subsets of the data, allowing the user to see inside a data set while retaining surrounding context in the same view.

Finally, even when given a reasonable amount of information to display, there are many questions regarding how to arrange it on the display, to represent and clarify abstract relationships, or to make structures easier to perceive. Our work on genealogical graphs (Chapter 5) examines issues pertaining to this, such as the choice of output (in our case, choosing a subgraph to display), the design of layouts, achieving properties of alignment and avoiding edge crossings, and scalability.

In summary, each case study presents a different, novel approach for the enhancement of output, investigating different aspects of this shared goal.

2.2.2 Easing Input

Pointing devices, such as mice, are a popular, flexible, and general means for inputting information. The HCI literature has many lessons and schemes related to pointing devices which are still not exploited as fully as they could be in interactive visualization systems.

To gain the maximum use of a display space, we can give it a dual purpose, making it available for input (e.g. pointing or gesturing) as well as output. Many software systems place output views and input controls in separate windows, but appropriate design often allows the spaces for output and input to overlap. This not only increases the area available for output, but can also ease selection: if the user's motor space is not *tiled* (completely covered) with Fitts' targets, then the targets are not as easy to point at as they could be [McGuffin and Balakrishnan, 2005a]. Another common benefit from using display spaces for both input and output is a reduction in potential problems from division of attention.

One way the display space can afford input is for the data elements displayed to be

selectable, and for selection of them to cause a useful change of state. For example, if each element corresponds to some subset of the data (e.g. a subtree, or a neighbourhood), then selecting an element may cause the corresponding subset to be displayed, hidden, or rendered differently.

Another general way to use display spaces for both input and output is with popup user interface elements and/or gestural input. Popup widgets allow for transient, kinaesthetically held modes which can reduce error rates [Sellen *et al.*, 1992]. Popup linear menus are common in commercial software, but are relatively primitive compared to radial menus [Callahan *et al.*, 1988], marking menus [Kurtenbach and Buxton, 1993], control menus [Pook *et al.*, 2000], flow menus [Guimbretière and Winograd, 2000], FaST Sliders [McGuffin *et al.*, 2002], and the hotbox [Kurtenbach *et al.*, 1999]. Popup widgets and gestural input can also be invoked or initiated over a data element in the view, combining the selection of a "noun" (the data element or its corresponding subset) with the selection of a "verb" (e.g. a command in a menu that modifies the visualization) and possibly parameters for the verb (such as with control menus). In previous HCI literature, nouns and verbs have referred to objects and operations for editing those objects, respectively, with a focus on finding the best way to integrate the selection of nouns and verbs [Bier *et al.*, 1993; Buxton, 1986], or to make one or the other implicit while avoiding modes and the possibility of modal errors. In visualization, the verbs that we are interested in are changes to the way the data is viewed rather than changes to the data itself, however the same ideas from HCI can be used to ease selection and interaction. The integration of noun and verb selection can afford faster input, and impose a structure on input that better matches the user's mental chucking [Buxton, 1986].

2.2.3 Using Smooth Transitions Between States

Animation is one way of creating a "smooth" (i.e. visually continuous) transition from one state to another. Previous authors [Woods, 1984; Baecker and Small, 1990; Bartram,

1997] have described the potential advantages of animation, and animated transitions are a feature of many interactive systems [Robertson *et al.*, 1991; 1993; Bartram *et al.*, 1995; Lamping *et al.*, 1995; Grossman *et al.*, 2001; Plaisant *et al.*, 2002]. (Issues related to designing animated transitions that are understandable are also discussed by Misue et al. [1995], concerned with how to design incremental adjustments to layouts of graphs, and Carpendale et al. [1997b], who discuss how to design fisheye-style distortions that are comprehensible.) Tversky and Morrison's [2002] survey of research found that animations are not necessarily as effective as static graphics at portraying changes over time, however they focused on the case where the objective is to communicate information about changes such as "weather patterns", "the circulatory system", or "the spread of cultural inventions like writing" or other temporal processes. In contrast, we focus here on situations where the visualization must transition from one state to another, and rather than visually "snapping" to the new state, we advocate displaying a smooth transition, through animation or other means. Our own anecdotal experience, and the designs of many previous interactive systems, strongly suggest that smooth transitions can allow the user to continuously track representations of data, maintaining their mental model of the system, retaining awareness of the context of the current state, and spend less time assimilating new states. If an animation is too fast or complicated or involves too many changing elements, at worst the user may ignore or fail to completely understand the transition. However, as long as the animation is brief (e.g. a fraction of a second), a smooth transition probably incurs essentially no cost to the user over a discontinuous transition, and at the same time can often benefit the user.

Taking this idea further, there is the potential for even greater benefit in supporting not just smooth transitioning from one state to another, but also across entire sets of states (e.g. sequences of states), and to allow the user to quickly and easily traverse these sets, with a continuous response from the visualization system. This could allow the user to explore more states in less time and with less effort, and could help the user better

understand how the states relate to each other.

As will be seen in section 6.1.3, animation is actually just one of three closely related ways of creating smooth state transitions.

2.2.4 Closing Remarks

Rather than competing or requiring tradeoffs, the above goals can be pursued simultaneously. Decreasing the input required for state transitions allows the user to explore more states of the visualization, which increases the output over time. In addition, increasing the output in each state reduces the number of states the user needs to traverse, which reduces the total input required.

The next three chapters present three case studies that each explore a different domain and a different approach to pursuing the above design goals. Each case study contributes significantly to its particular domain [McGuffin *et al.*, 2003; 2004; McGuffin and Balakrishnan, 2005b]. Furthermore, by sampling these different domains, we encounter a variety of problems and gain relatively broad experience in issues related to the design goals shared across the case studies. These issues are summarized and analyzed in Chapter 6.

Chapter 3

Using Deformations for Browsing Volumetric Data

Many traditional techniques for "looking inside" volumetric data involve removing portions of the data, for example using various cutting tools, to reveal the interior. This allows the user to see hidden parts of the data, but has the disadvantage of removing potentially important surrounding contextual information. This chapter explores an alternate strategy for browsing that uses deformations, where the user can cut into and open up, spread apart, or peel away parts of the volume in real time, making the interior visible while still retaining surrounding context. We consider various deformation strategies and present a number of interaction techniques based on different metaphors. As will be seen, our designs pay special attention to the *semantic layers* that might compose a volume (e.g. the skin, muscle, bone in a scan of a human). Users can apply deformations to only selected layers, or apply a given deformation to a different degree to each layer, making browsing more flexible and facilitating the visualization of relationships between layers.

3.1 Introduction

Volumetric data can contain an enormous amount of densely packed data points, or *voxels* (from VOlume ELements). Visualizing such data is challenging. While in the real world, we typically only perceive the surfaces of objects, computational visualization of volumetric data should ideally not have such a restriction. Where possible, we should be able to see the data *throughout* the volume simultaneously. However, this is especially difficult to achieve on a flat 2D display surface, where the user is, in some sense, forced to pick one point of view at a time, and where the number of voxels in the data can easily exceed the number of available pixels. Even with display hardware that generates true 3D output, the image projected onto the user's retina's remains 2D, so occlusion of data remains a problem.

Consider these three general strategies[1] for "peering inside" volumetric data:

1. Cutting away or removing portions of the data, to eliminate occlusion of inner regions

2. Making some or all of the volume semi-transparent, allowing the user to see inside or through layers of data

3. Spatially transforming or deforming the volume, to displace, project, break apart or separate outer portions and reveal inner portions

Strategies 1 and 2 have been widely used in volume visualization systems. Strategy 1 includes all the common boolean masks used to "carve away" parts of a volume, such as cutting planes, cutting boxes, cutting spheres, etc., and more generally includes any technique that selects and displays a subset of the data, such as showing the voxels bounded by an isosurface. With strategy 2, the control over transparency is often achieved

[1] Appendix A considers in detail these and other strategies for dealing with occlusion in volumetric data, as well as in other kinds of data, and points out, as we do here, how deformations are a promising topic for research in volume visualization when compared with these other strategies.

by adjusting a *transfer function* which maps voxel values to colour, opacity, and other properties used in rendering.

As pointed out by Carpendale et al. [1997a], although transparency and removal of outer data both make inner data more visible, they also result in loss of context. This can make it difficult for users to form an integrated mental picture of the entire volume.

Strategy 3 is based on deforming the data in some manner. We use the term "deformation" somewhat loosely here, to refer not only to smooth, non-rigid transformations, but also piece-wise rigid transformations, discontinuous transformations, and combinations of these. Thus, strategy 3 includes techniques such as "exploded views" (often used in assembly manuals for mechanical devices) that simply translate parts away from each other, as well as more exotic transformations, which, apart from some recent research [Carpendale *et al.*, 1999; 1997a; Kurzion and Yagel, 1997; LaMar *et al.*, 2001], have remained largely unexplored for the purpose of browsing volumetric data.

Our goal in using deformations is to increase the visibility of the inner portions of the volume, without completely removing the surrounding data that normally occludes the inside. This is akin to focus+context schemes that allow a user to "zoom in" on data of interest, while using remaining screen space to show the surrounding context. Appropriately chosen deformations could, for example, split open a volume, showing displaced structures side-by-side, making it easy for the user to see how they connect, and allowing the user to mentally stitch them together into a whole. Deformations with familiar real-world analogues (e.g. cutting and peeling the skin off a fruit, or the layers off an onion) are also likely to be readily understood by users.

In this chapter, we describe a prototype system that implements different metaphors for deformation-based browsing of volumetric data. As will be seen, a key element of our approach is to support differential treatment of the various *semantic layers* in a data set. By "semantic layers", we mean subsets of the data that are useful or meaningful to the

user. These layers could be defined geometrically, for example as sections created with parallel planar cuts. More typically, layers would depend on the voxel data values, for example boundaries that are found during segmentation or isosurface extraction. In the context of medical visualization, there is at least anecdotal evidence that anatomists, for example, prefer to remove tissue layer by layer [Höhne *et al.*, 1992], rather than making arbitrary planar cuts.

3.1.1 Approach to Pursuing Design Goals

Whenever configurations of geometric elements are embedded in 3D space, occlusion is a concern, and sometimes the major challenge, for visualization. In the context of the design goals in section 2.2, when faced with occlusion problems, improving the output is much more about judiciously reducing or rearranging the information displayed rather than trying to pack more information into a display. As briefly described in the preceding Introduction, and detailed in the Background section that follows, there are already various existing techniques for dealing with occlusion and with the visualization of volumetric data. Nevertheless, there are three aspects of the case study in this chapter that make it worthwhile and interesting with respect to the design goal of increasing and improving output. First, volumetric data is an extreme case of data embedded in 3D, where the very nature of the data causes occlusion problems to be at their worst, and yet volumetric data is also very simple and widely applicable data representation. Second, unlike most other techniques, our use of deformations will enable de-occlusion of the interior of a volume while also retaining surrounding contextual data on the screen — an excellent way of addressing the goal of increasing the useful amount of output. Third, the use of deformations for *browsing*, especially of volume data, hasn't received much prior attention from researchers, inviting new ground to be broken.

To address the 2nd design goal, of easing input, we apply techniques familiar in input research in HCI, using direct, "in place" manipulation, and pop-up marking menus and

3D widgets, to avoid the divided attention and awkwardness that would come with panels of traditional widgets.

The importance of the 3rd design goal, of having smooth state transitions, is evident when we consider the risk that deformations might sometimes render data unrecognizable, or change the spatial arrangement of voxels in ways that are unfamiliar or difficult to understand. To counter this, we use smooth animations to show changes or transitions in the shape of the data. For example, if the user invokes a tool that peels back a layer, rather than suddenly "snapping" the layer into a fully peeled state, the layer is continuously peeled in real time, to show the user what is happening.

Finally, regarding the 2nd and 3rd design goals, we leverage the rapid operability of marking menus by allowing repeated flicks in certain directions to perform layer-by-layer traversal of a sequence of states, e.g. to progressively peel or unpeel layers, with smoothly animated transitions throughout. Such a simple use of repeated flicking for input proves quite effective, and we are unaware of other examples of its use in visualization.

3.2 Background

Here, we first provide general information on volumetric data. Next, we survey previous research related to the case study that uses deformations for visualizing 3D (especially volumetric) data, pointing out how our work differs. (Appendix A provides a more extensive survey of techniques for dealing with occlusion, both with and without deformations, and with volumetric data as well as other kinds of data.)

3.2.1 Volumetric Data

Volumetric data, or volume data, refers to data or geometric models defined over a volume of space. These are contrasted with surface representations, such as polygonal surface meshes and spline surfaces, which are typically less memory-intensive, faster to render,

and have been more commonly used in computer graphics.

A common form of volumetric data has voxels (data points) arranged on a regular grid. Such data is used in finite-element simulation (of fluids or other physical media) as well in various kinds of reconstructed medical data (from ultrasound scans, computerized axial x-ray tomography (CT), magnetic resonance imaging (MRI), positron emission tomography (PET), or cryosection photography) — see Figure 3.1. Because of the simplicity and widespread use of such data, we focus on volumetric data on regular grids.

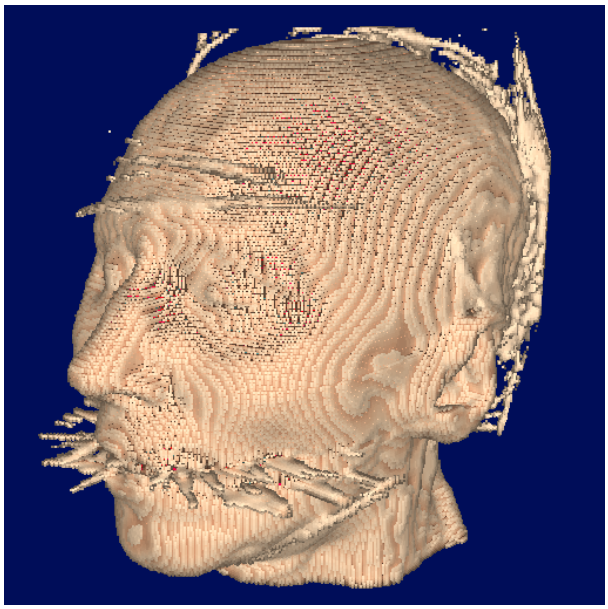

Figure 3.1: A volumetric data set derived from a CT scan of a head. The head has been segmented from the surrounding "empty" space. Voxels are arranged on a regular grid, and each voxel contains a scalar value that is mapped to a colour.

The value associated with each voxel may be a scalar, vector, tensor, or other mathematical object. If values must be defined at all points in the volume, and not just at voxel centres, then some form of reconstruction of values (such as trilinear interpolation) can be performed using the voxel values.

Much previous work with volumetric data in computer science has centred on two issues. First is how to correctly and efficiently render such data, especially when voxels have variable opacity. Second is how to perform *segmentation* of the data into disjoint objects;

and *classification* of voxels into (possibly overlapping) subsets or according to properties deduced from the data. Classification may also be described as soft-segmentation, and is often used to determine the *transfer function* that maps voxel values to colour and opacity values.

Although advances in hardware and algorithms have produced steady progress in solving these problems, the issue of occlusion in volumetric data remains a challenge inviting novel solutions. Note that, unlike 3D embeddings of abstract information, occlusion in volumetric data is an unavoidable aspect of the data, because the data has an inherent spatial embedding that fills a volume. The occlusion is also particularly severe, because voxels are essentially adjacent to one another, and may be very large in number. Our case study uses deformations to address the occlusion problem in new ways; the next section surveys previous related research.

3.2.2 Related Work

Many of our deformation techniques are inspired by surgical metaphors, where the user cuts into and opens up data. There have been attempts to create high-fidelity simulations of surgical procedures [Pflesser *et al.*, 1995, for example] for education, training, and rehearsal. These often involve the use of virtual reality, haptic feedback, and the simulation of the physical properties, such as elasticity and hardness, of the tissues being operated on. Bruyns and Montgomery [2002] describe virtual tools that look and behave like scalpels, scissors, and forceps, allowing a 2- or 3-dimensional mesh to be cut and peeled open. While this approach has the advantage of easily understood, very literal, metaphors, it imposes an interaction style limited to what is possible in the physical world, without fully exploiting the additional capabilities of the computational medium. Our present work, in contrast, allows the user to explore and visualize volumetric data in ways that would be physically impossible. For example, in our system, users can peel away bone just as easily as skin, or change the location of an incision after the cut has

been made by "swimming" the location of the cut through the volume. Because we are not concerned with simulating a physical process such as surgery, the user can browse data in a more light-weight and free-form style. We can also build intelligence into our browsing tools, so that they, for example, automatically detect boundaries between layers of data and do not cut across these boundaries.

Another difference between medical applications in general, and our work, is that medical specialists usually have a good idea of the underlying anatomy of a volume and thus can estimate where to look to find features of interest. In contrast, our techniques are designed to support general purpose exploratory browsing, and could be used with volumetric data of unknown content. This makes focus+context techniques all the more appropriate, since showing more of the data on the screen can make it easier and faster for users to find interesting data.

Looking beyond surgical applications, the deformation of volumetric data for general visualization has also been explored. For example, Laidlaw [1995] segmented scans of a banana and a human hand, and created animations of their skin peeling off. Our work, however, uses deformations for real-time, interactive browsing.

Kurzion and Yagel [1997] describe a "discontinuous ray deflector" that gives the appearance of cutting into a volume and spreading open the voxels. This is similar to the "book" metaphor used by Carpendale et al. [1999] where data are spread open like pages of a book. The same book metaphor has been used in traditional anatomical diagrams, where organs are shown cut in half and spread apart on consecutive pages of a book [Agur and Lee, 1999, for example pp. 622–623, 718, 719]. In the next section, we describe our own Hinge Spreader tool which also uses this metaphor. Our work, however, also extends the existing repertoire of deformations with other tools.

Focus+context techniques have also been proposed for volumetric data. Carpendale et al. [1997a] describe a *visual access distortion* technique that clears a path of visibility to a point of interest by pushing occluding data away from the line of sight. This "cleared

path" remains on the line of sight as the scene is rotated, giving the appearance of a constantly shifting deformation. Thus, the rotation and deformation of the data are coupled. In our system, however, the user deforms the data in a desired way, and can then view the deformed data from any angle; i.e. rotation and deformation are separate actions. Although this introduces a risk that the user may have to rotate the scene more deliberately to gain a clear line of sight, the user also has more freedom and control to look at the deformed data in different ways.

LaMar et al. [2001] describe a focus+context technique that magnifies a region inside a volume. This magnified region is visible to the user if the surrounding data is semi-transparent, or if a cutting plane is used to reveal the inside. As will be seen in the next section, our own Sphere Expander tool may seem similar in that it expands regions of data. However, the Sphere Expander pushes voxels away from a central point rather than magnifying data. Thus, it can be used not only to enlarge an existing cavity or hole in the data, but also to create a hole where none previously existed. Furthermore, our Sphere Expander can be applied differentially to the layers in a data set, yielding new interaction possibilities.

3.3 Design Approach

At the outset of our research, we imagined three main actions that might be supported by a volume browsing system: (i) selecting a region of the volume, (ii) changing the appearance (i.e. transfer function, including opacity) of the selected region, and (iii) spatially transforming or deforming the selected region. Together, these three actions could be composed to achieve the same effect of potentially any existing browsing technique. Although we have not yet implemented the full vision of these three composable actions, we have explored issues surrounding region selection and deformation of a region.

Users commonly select regions of a volume using geometric primitives, such as half-

spaces (planes), spheres, or boxes. The deformation tools in our system similarly have simple geometric shapes, and are constrained to act only on voxels within these shapes. However, the features of interest within a volume may have irregular shapes. Höhne et al.'s [1992] Anatomical Atlas supported "selective cutting" tools, that were sensitive to the layers in the data, and could be made to only act on certain layers. Thus, a cutting plane could be used to first remove skin, then bone, etc. The tools in our system are also sensitive to the layers in a data set, and can act differentially on them. Users can treat each layer separately, making selection of related voxels simpler and more implicit.

Unfortunately, the subsets of a volume data set are not always best thought of as *layers* — take for example the internal organs in a human body, or even vascular structures. Furthermore, volumetric data is often noisy and difficult to cleanly segment into distinctive subsets. Nevertheless, we chose to assume the existence of layers, and focus on how a user might manipulate such layers, because this inspired unique interaction techniques.

It is informative to compare the layers in a volumetric data set to a stack of cards or papers, and to consider the various ways in which these could be browsed. Mander et al. [1992] describe different ways of browsing virtual piles of documents, emulating the effect of riffling or thumbing through a physical pile of paper. Beaudouin-Lafon [2001] designed novel interaction techniques for overlapping windows. One of these allowed a user to peel back the corner of one or more windows, to take a peek at occluded windows. Taking inspiration from these previous examples, and through our own brainstorming, we developed Figure 3.2, which identifies a few other methods for manipulating layers. Section A.4.2 gives a few additional examples of previous work uses of layers that can be flipped through or peeled to browse data, however there are no similar examples of this for volumetric data. As already argued, deformable, peelable layers can reveal interior regions without sacrificing context; additionally, affording *rapid* flipping through layers of volumetric data could allow the user to quickly scan a large amount of visual information,

which could have advantages similar to *rapid serial visual presentation* (RSVP) [Spence, 2002].

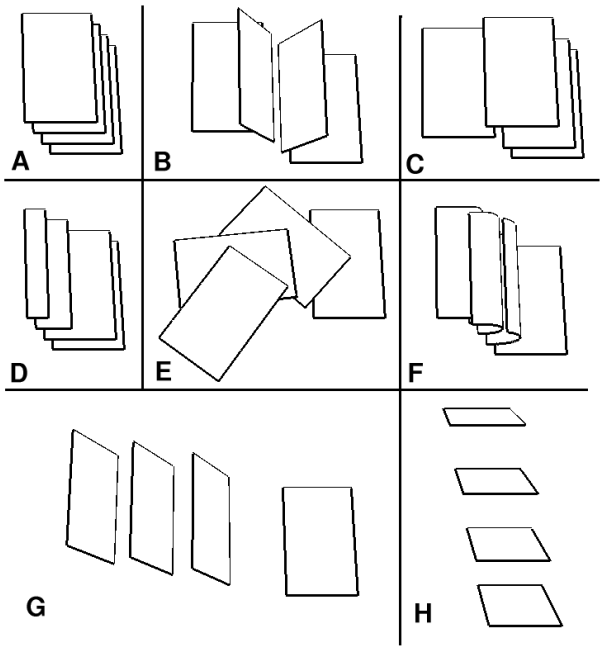

Figure 3.2: Techniques for browsing the layers depicted in A. These techniques were identified as a result of examining previous work and by brainstorming other possibilities. B: *leafing* through the layers like pages of a book. C: pulling out an individual layer. D: compressing upper layers to reveal lower layers. E: *fanning* layers open like a hand of cards or like a Chinese fan. F: *peeling* layers back. G: *flipping* layers over — here the user is flipping the 3rd layer from the left, and the other layers to the left are pushed along like dominoes. H: an exploded view of the layers.

In keeping with the layers-as-a-stack-of-cards analogy, there are three different points of view one might want of a given layer: first, a *dorsal* view of the back/top/outer surface of the layer; second, a *ventral* view of the underneath/bottom/inner surface of the layer (this is visible when the layer is flipped or turned over in some way); and third, a *cross-sectional* view, showing the thickness of the layer from the side, in which case it is often useful to see neighbouring layers stacked above and below the current layer.

Interactive browsing techniques that enable the various manipulations of Figure 3.2, and that also support dorsal, ventral, and cross-sectional views — possibly simultaneously — of one or more layers, are more likely to afford the user with flexible and useful

vantages. We have tried to incorporate these elements in our designs.

Figure 3.2 already hints at some interesting ways in which layers could be deformed for visualization. Other deformations of interest can be inspired by surgical metaphors, whereby data might be cut into and spread open in different ways. Two questions to consider when choosing a deformation are: should the deformation be rigid or non-rigid, and what kind of continuity conditions should be satisfied by the deformation?

Rigid deformations, i.e. rotating and/or translating out a piece of a layer, have the advantage of preserving lengths and angles and volume, which could be important for performing measurements, or simply for assurance that the data being visualized has not been distorted. On the other hand, non-rigid deformations, such as curvilinear "peeling", encompass a much broader range of possibilities, and may be more realistic in medical contexts for giving an impression, if only approximate, of how tissue would deform if it were physically peeled.

Regarding continuity, one issue is how a deformed region of data should remain "connected", if at all, with the rest of the volume. We return to this question in section 3.4.7. At this point, however, we can list some general, desirable mathematical properties. We want the deformation, a mapping $f: \mathbb{R}^3 \mapsto \mathbb{R}^3$, to have the following properties:

- f should be piecewise continuous: we want the deformation to be "mostly" continuous, with at most a "small" number of discontinuities.
- f should be piecewise invertible: we want the resulting volume to exhibit little or no self-penetration, since this would increase occlusion.
- f should be orientation-preserving: we don't want any region of the volume to undergo inversion (scaling by a negative factor) or mirror reflection.

At the same time, f needn't necessarily be rigid (distance- and angle-preserving), volume-preserving, or affine (preserving collinearity (straight lines) and ratios of distances). Each "piece" of the domain may undergo any combination of translation,

rotation, scaling (with a positive factor), shearing, as well as non-affine non-uniform transformations that might be described as "bending", "stretching", "squishing", and "twisting".

3.4 Prototype Implementation

Our prototype volume browser was implemented in C++ using the OpenGL and GLUT libraries, and runs under the Linux and Microsoft Windows operating systems.

The individual voxels of the data can be rendered with two different modes: either as *points* (tiny squares, actually) whose size in screen space is chosen to give the appearance that adjacent voxels are just touching[2]; or as *splats* (these can be thought of as fuzzy dots; in computer graphics terminology, they are billboards with texture maps whose transparency increases toward the edges). The first mode is useful for rapid rendering and interaction, and the second can be used for higher-quality rendering with transparency[3]. Although techniques exist for higher quality volume rendering, for example using hardware texturing and trilinearly interpolating voxel values, we chose to render individual points or splats to keep our prototype simple and maximally flexible. Any deformation that remaps the voxel locations can be supported by our system, since each voxel is rendered on its own. This gives us the freedom to focus on exploring interaction techniques, rather than optimized, high quality rendering.

On a 1.7 GHz laptop with an *nVidia GeForce4 Go* graphics card, 32 MB of video memory, and 512 megabytes of RAM, our system can render over 500000 voxels (as `GL_POINTS`) at 13 full screen frames per second, or over 4000000 voxels at 2 full screen frames per second. Since real time interaction is critical, our system downsamples large data sets and renders them at a lower resolution during interaction. Full resolution

[2]This is accomplished with the OpenGL primitive `GL_POINTS`, adjusted in size with `glPointSize()`.

[3]DVR-style transparency, as well as additive (i.e. commutative, order-independent) alpha blending, are both supported, with a global transparency factor that can be varied. See section A.3.1 for more explanation of these terms.

rendering is performed after the system has been idle for a given timeout (e.g. one second), or whenever the user explicitly requests it. An even better implementation might render different parts of the volume at different resolutions. For example, only the portion of the volume currently in the user's focus could be rendered at full resolution, without precluding real-time interaction.

To support arbitrary transformations of voxel positions, we explicitly store the position of each voxel, rather than storing a 3D bitmap. This is also more efficient for sparse data sets. Voxel positions are stored in an octree, where each node of the octree has a bounding box as well as a colour attribute that can take on the values *black*, *white*, or *both*. A dividing plane can be applied to the octree, and voxels can be quickly categorized by colouring them black or white, according to the side of the plane they lie on. Intersections or unions of halfspaces can be coloured by applying multiple planes (Figure 3.3). Operations on the octree, such as rendering voxels, deforming voxel positions, or copying voxels into a second octree, can be applied to the whole octree, or to only a subset of a given colour.

As a minor optimization, each voxel position is not stored in its own leaf node. Instead, each leaf node stores a small number (e.g. 8) of voxels in an array that can be traversed more quickly than an equivalent subtree with one voxel per leaf. Rather than storing a colour flag for each voxel, we save memory by storing black voxels in the first n elements of the array, and white voxels in the remaining elements. Changing the colour of a voxel requires swapping a single pair of elements and adjusting the value of n.

To support operations that treat each layer of data differently, each layer of voxels is stored in a separate octree (Figures 3.4). Thus, having N layers requires N octrees. This does not, however, imply using N times more memory than a single octree for all the layers would. Each layer typically exhibits some spatial coherence, and can be stored efficiently in an octree.

Figure 3.5 shows the computational steps taken to deform and render a volume.

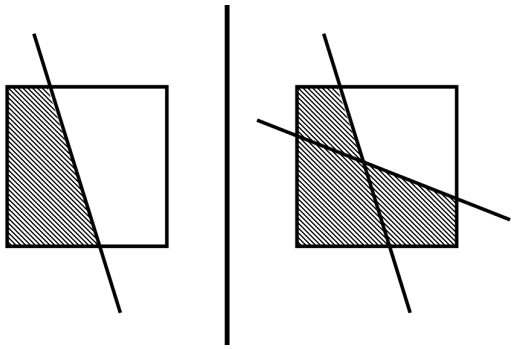

Figure 3.3: *Left:* Given an octree and a plane, we can efficiently determine which nodes in the octree lie on each side of the plane and which nodes straddle the plane. In our case, the leaf nodes of the octree are voxels, and each node of the octree has a colour attribute that is given a value of *black* if the node lies on one side of the plane, or *white* if it lies on the other side, or a value of *both* if it straddles the plane. Thus, the colour attributes in the octree nodes store the result of this "planar categorization" operation performed with the given plane. *Right:* Applying a second planar categorization operation to only one colour of nodes allows us to compute the intersection or union of two halfspaces. This can be repeated to compute, for example, the colour of nodes that are inside or outside a box-shaped region or other convex polyhedron.

Figure 3.5 *A:* The original N layers are stored in a set of N octrees (shown on the left), and in addition we have a second set of another N octrees (shown on the right) that are initially empty. *B:* The voxels in the 1st set of octrees are categorized (i.e. coloured) according to whether they are to be deformed or not. For example, if the deformation tool is shaped like a box with an open top, then 5 planar categorization operations (3 are seen here) might be performed to compute which voxels are inside the box, and these voxels might be coloured white while voxels outside the box are coloured black. *C:* The voxels to be deformed (e.g. those coloured white) are copied into the second set of octrees. *D:* the voxels in the second set of octrees are deformed. This involves changing the coordinates stored in the leaf nodes of the octrees. In this example, the deformation consists of a rigid rotation, and the angle of rotation for the last layer is not the same as for the other layers, so we see how different deformations can be applied to each layer. *E:* Finally, we render all voxels highlighted in red, i.e. all black voxels in the first set of octrees (those outside the box, that were not deformed) as well as all the voxels in the

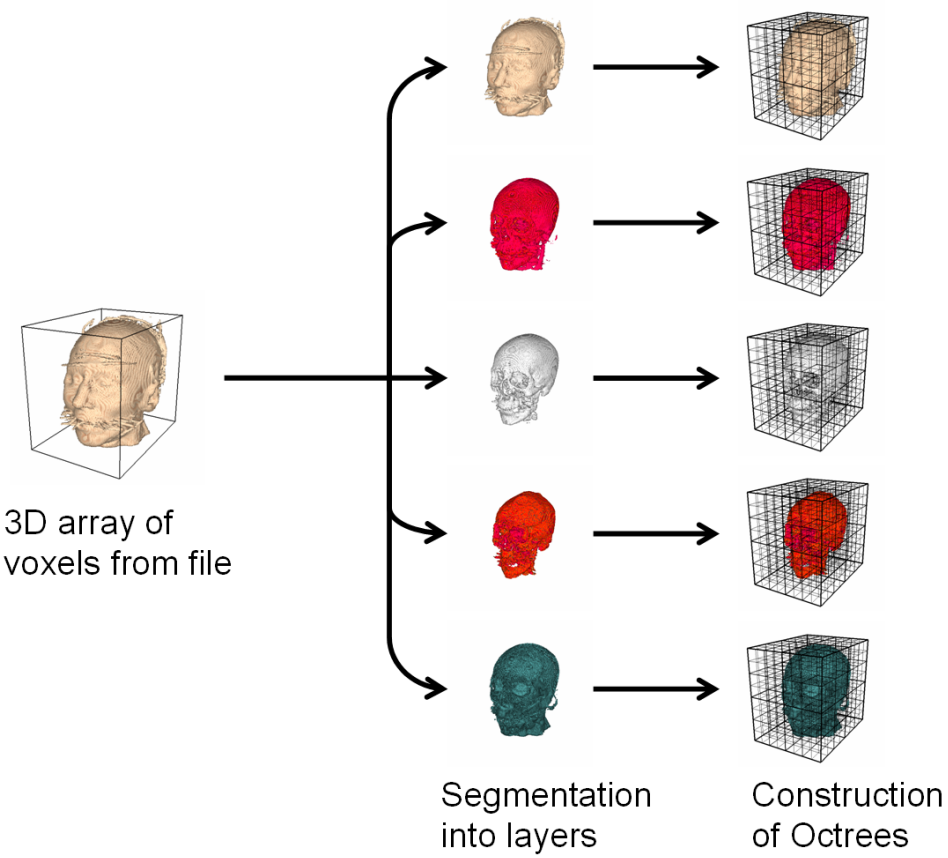

Figure 3.4: When a data set is read in by our prototype, it is segmented into layers (this step is non-interactive and is performed according to information associated with the data set), and then each layer is stored in a separate octree.

second set of octrees. (Note that the categorization of voxels in the first set of octrees, and the deformed positions of voxels in the second set of octrees, are updated lazily — e.g., changing the angle of rigid rotation in this example would only cause the positions in the second set of octrees to be recomputed, but changing the position of the box-shaped tool would cause the categorization in the first set of octrees to also be redone.)

Each voxel has an associated normal, and is rendered with lighting to provide the user with shading cues. Voxels near a surface of the data set have a normal computed from their neighbourhood — this computation is slow, but need only be done once, at load time. Voxels in the interior of the volume are initially not visible, and have a zero normal. However, deformations can cut or split open the volume and reveal these

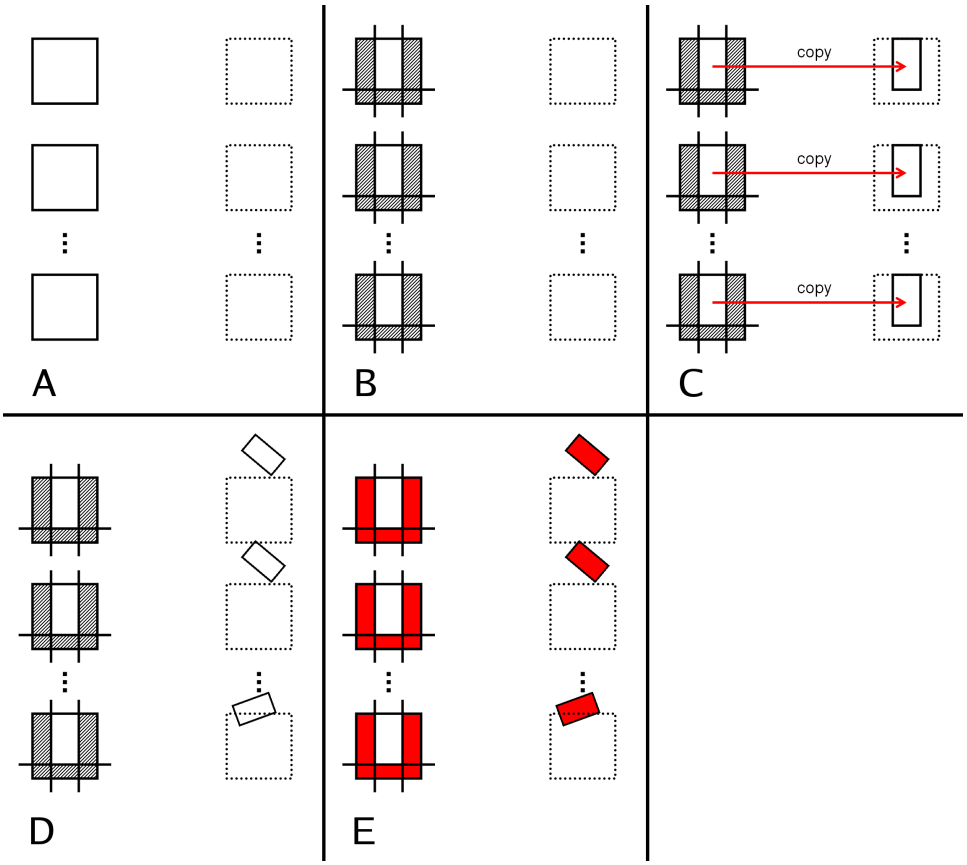

Figure 3.5: Steps in deformation and rendering. (Refer to the text for details.)

interior voxels. Thus, the normals of interior voxels are dynamically recomputed, in a fast but approximate way, based on the current deformation, and based on a guess of the orientation of the closest surface. Although the result is only approximate, the depth cues resulting from lighting the scene were found to be preferable over having no lighting.

The browsing tools in our system are positioned, oriented, and resized using 3D widgets [Conner *et al.*, 1992], i.e. objects embedded in the 3D scene that can be clicked and dragged. 3D widgets are also used to control the parameters of the different deformations. Each draggable component of our 3D widgets is highlighted when the mouse cursor passes over them, to hint to the user which elements can be dragged. The shape of the widgets also suggests how to use them: arrow widgets are for translation or adjustment of a linear quantity such as the radius of a sphere; circle and arc widgets are for rotation or adjusting angles. In an early version of our prototype, we noticed perceptual problems

with the visual design of our 3D widgets. We thus improved them by adding more depth cues (Figure 3.6).

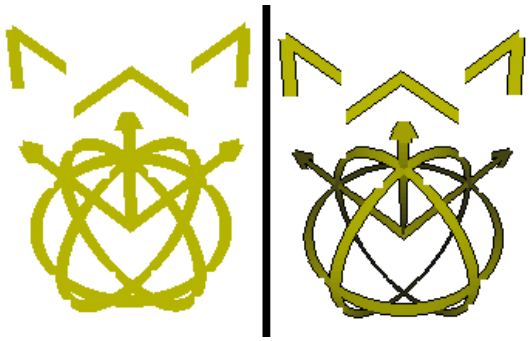

Figure 3.6: Example 3D widgets before and after design changes that enhanced depth cues. On the right, shading and variation in thickness are used, and intentionally exaggerated, to suggest depth. A thin halo of black pixels is also drawn to ensure contrast with whatever data may be in the background. The arrow widgets are used for translation along an axis, the circle widgets are used for rotation around an axis, and the 'L'-shaped widgets are for translation within a plane.

Another challenge encountered was that some deformations have many adjustable parameters, and if each of these is controlled with a separate 3D widget that is always visible, the screen becomes cluttered with widgets. We therefore identified situations where certain widgets were not likely to be used, and changed our prototype to only display a given 3D widget if the current state warrants it. For example, Figure 3.14 shows a set of layers that are fanned open. Only after they are fanned open do additional 3D widgets appear, attached to each layer.

Figures 3.7 through 3.19 show our system browsing a scan of a human head that was pre-segmented into 5 layers. At any given time, only one browsing tool is active, which the user chooses from a popup radial menu [Callahan *et al.*, 1988] or marking menu [Kurtenbach and Buxton, 1993] (our menu is only one level deep, and so could be described as either a radial menu or marking menu). The active tool only affects the currently selected layers of the data, leaving unselected layers unchanged. When a layer is selected or unselected, an animation shows the layer's transition from one state to the other, for example, from a deformed state to an undeformed state.

Two mechanisms were implemented for selecting/unselecting layers. First, the selection state of each layer can be toggled individually through a set of hotkeys assigned to each layer — but these could just as easily be virtual check boxes in a menu. Second, there are two special items in the popup radial menu, the right and left items, that also control layer selection. These items can be invoked with quick flick gestures to the right or left, and have the effect of (a) selecting the outermost unselected layer, or (b) unselecting the innermost selected layer, respectively. Thus, the user can, for example, make 5 flick gestures to the right, causing each layer, from the outermost to the innermost, to be successively selected. If the currently selected tool peels layers away, this would have the effect of successively peeling each layer in the natural ordering.

To support this behaviour, the system needs some notion of which layers are inside or outside other layers. We manually assigned a global, fixed ordering, from outside to inside, of the layers in our head data set. This ordering is important not just for selection, but also for deformations that automatically deform layers to different degrees, such as the fanning out in Figure 3.14, where inner layers are rotated by a larger angle than outer layers. Although the fixed ordering is acceptable for our head data set, in general this would not be a viable solution. It is possible for the semantic layers in a data set to not have any single ordering from outside to inside. An improved prototype would compute a locally acceptable ordering of layers on the fly, given the current location and orientation of the deformation tool. Such an ordering might be computed by sampling the relevant data along parallel rays, and finding the "average ordering" of layers encountered along the rays.

No collision detection is performed between layers. Hence, layers can interpenetrate as they are manipulated or animated. However, as long as the user is manipulating a region of data where the the inside-to-outside ordering of layers is reasonably accurate, the interpenetration of layers is minimal.

The deformation tools in our system will be described in the following subsections.

We also implemented more traditional cutting tools, specifically: a cutting plane, cutting hinge, cutting sphere, and cutting box. Figure 3.7 shows two of these in action. Just as in Höhne et al.'s Atlas [Höhne *et al.*, 1992], our cutting tools are sensitive to the layers in the data, and only remove voxels from the currently selected layers. Thus, they can be thought of as intelligent scalpels that cut no deeper than the innermost selected layer. An alternative way of thinking of these tools, especially the cutting box, is that they behave like 3D magic lenses [Viega *et al.*, 1996], in that they make the currently selected layers fully transparent.

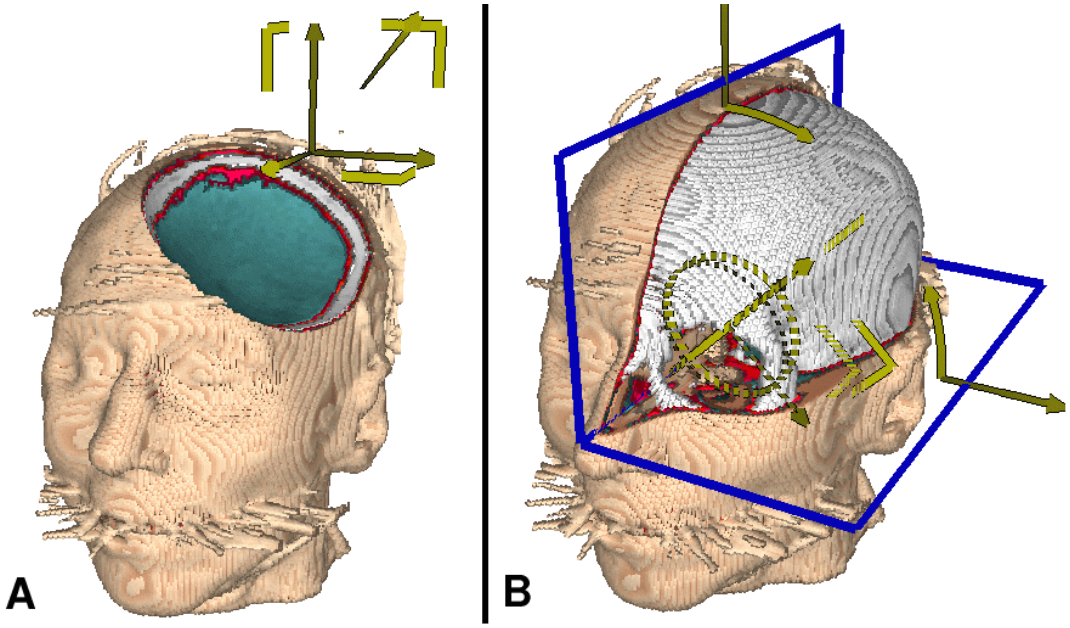

Figure 3.7: Examples of cutting tools. A: a cutting sphere. B: a cutting hinge. Each cutting tool can be made to cut away all layers (as in A) or only a subset of layers (as in B).

3.4.1 Hinge Spreader Tool

The Hinge Spreader (Figure 3.8) tool is a dihedral shaped object that pushes all the voxels between the hinge to either side. As mentioned in the Background section, this deformation can be used to create views of data that resemble anatomical dissections spread open like a "book" [Agur and Lee, 1999].

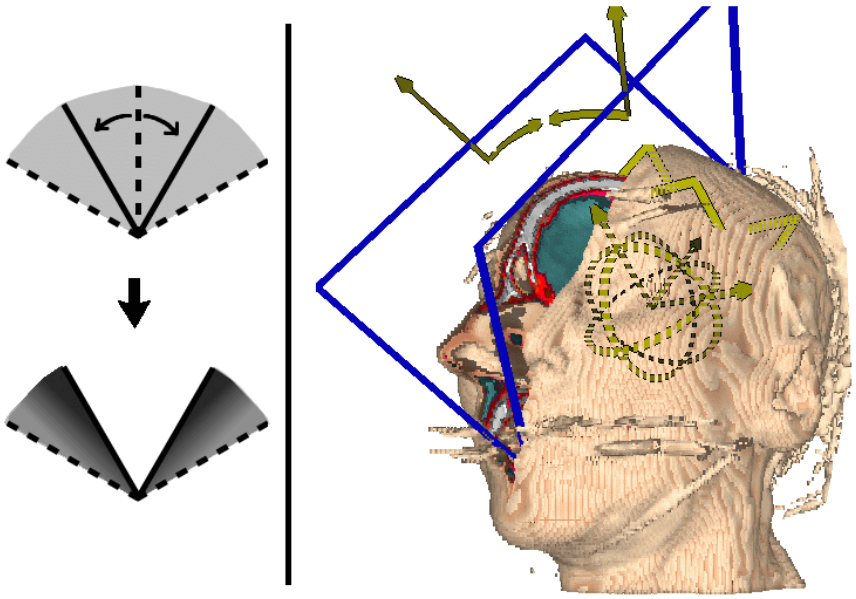

Figure 3.8: The Hinge Spreader. Left: a sketch of the voxels before and after deformation. Solid black lines show the hinge seen by the user. Dashed lines delimit the voxels affected by the deformation. Voxels are pushed away from the bisector of the hinge, compressing surrounding voxels that are within twice the hinge's angle. Right: a face is split down a line through the nose. Note that both halves of the nose are still present – no voxels have been removed or cut away, they have simply been pushed aside.

3D widgets enable positioning and orientation of the tool, and also allow the angle of the hinge to be adjusted. Note that the cutting hinge in Figure 3.7 B has the same dihedral shape as the Hinge Spreader, but *removes* the voxels within the hinge, rather than displacing them.

As with all our tools, the Hinge Spreader only deforms voxels of selected layers. When layers are selected or unselected, an animation shows the voxels of that layer transition from a deformed state to a resting state, or vice versa. Interestingly, the Hinge Spreader can be used to create views that look like exploded diagrams (Figure 3.9) when applied to only a subset of the layers.

The hinge form factor of this tool, and of the cutting hinge, have interesting properties with respect to interface design. First, the positioning, orientation, and angle of a hinge could be easily controlled with a hand-held prop, much like Hinckley et al.'s [1998] use of a cutting plane prop in their "doll's head" interface. Users have a strong mental model

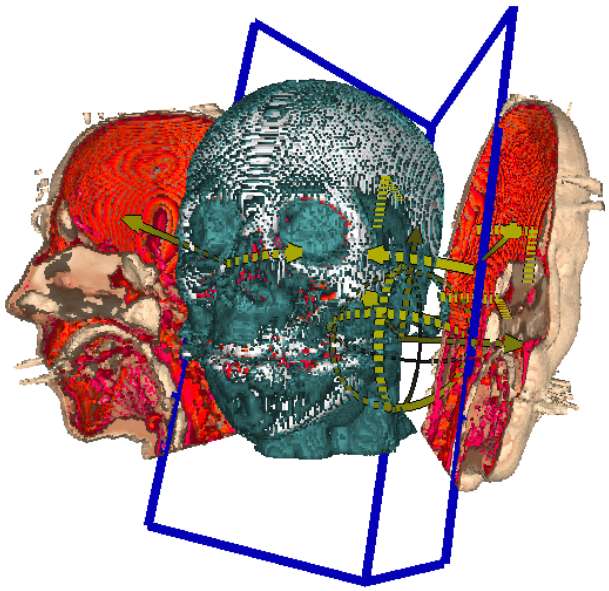

Figure 3.9: The Hinge Spreader, acting only on layers outside the skull, has been pushed all the way through the head, and therefore spreads both halves of the skin off the skull. This provides a kind of "exploded view".

of the shape and function of a hinge, and would probably be able to use a hinge prop as successfully as Hinckley et al.'s cutting plane. In addition, the use of a hinge has certain advantages over a plane. A hinge opened up to 180 degrees reduces to a plane as a special case, and so is more general than a plane. Acute hinge angles allow for more context to be maintained close to a focal point. Finally, two-handed techniques are possible where a user holds two hinge props, and could make fast, compound cuts or spreads of the data.

3.4.2 Sphere Expander Tool

The Sphere Expander tool (Figure 3.10) pushes voxels away from a central point. The centre and radius of the sphere are controlled with 3D arrow widgets. Placing this tool outside and near the surface of a volume creates a dent in the volume, which in itself may not be useful for browsing. However, when placed inside a volume, the Sphere Expander can be used to inflate the voxels of the volume outward. Since we render each voxel as a point, sufficient inflation eventually makes the voxels sparse enough to see through —

a kind of cheap transparency. The Sphere Expander can also be used to create a hole in a layer, by selecting only that layer, and by placing the centre of the sphere on the layer (Figure 3.10, right hand side). Interestingly, we did not initially know whether the Sphere Expander tool would turn out to be useful. After implementing it, however, we discovered that our layer-based architecture allowed for situations like that in Figure 3.10.

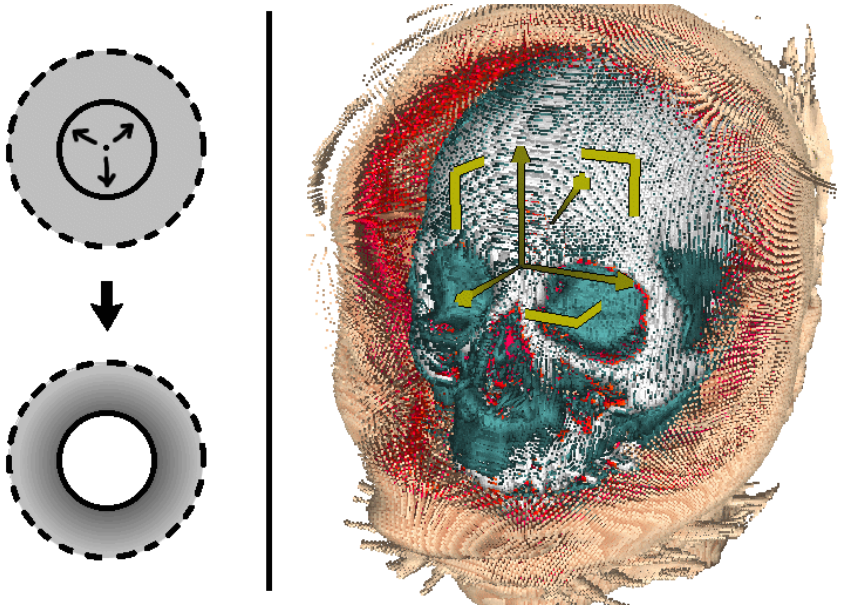

Figure 3.10: The Sphere Expander. Left: voxels before and after deformation. All voxels contained in the solid sphere are pushed outside, compressing surrounding voxels that are within twice the sphere's radius. Right: the Sphere Expander, acting only on layers outside the skull, is centred on a point on the face above the nose. This opens up a hole in the face and lifts much of the skin off the skull, creating a kind of "window" through which we can see the skull and surrounding skin.

3.4.3 Box Spreader Tool

In the same spirit as the previous two deformation tools, the Box Spreader (Figure 3.11) pushes all voxels outside the shape of the tool, in this case a box. This tool was inspired by *rib spreaders*, instruments used in chest cavity surgery. The Box Spreader could be used to cut in to the virtual chest of a human data set, and spread open the outer layers of the chest, revealing internal organs. When the box is made wide enough, however, the

deformation can eventually lift outer layers off a data set, as in Figure 3.11.

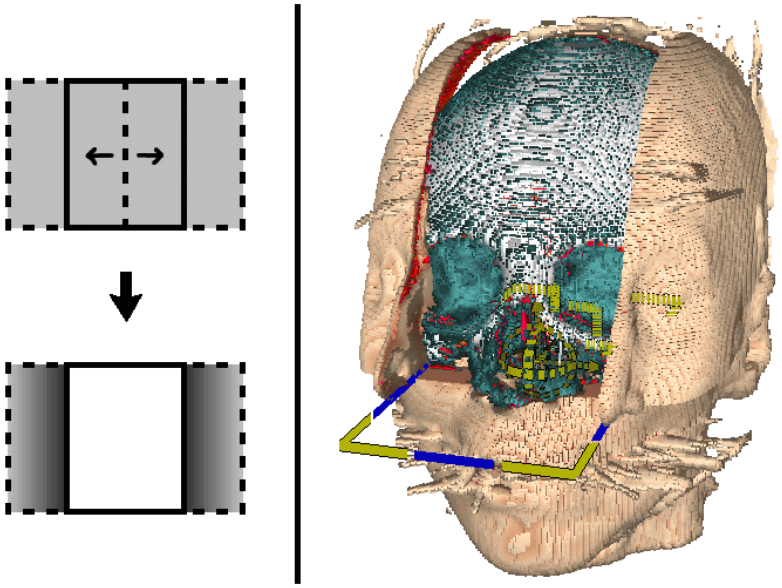

Figure 3.11: The Box Spreader. Left: voxels before and after deformation. All voxels contained in the box are pushed sideways, compressing surrounding voxels that are within twice the box's width. Right: the Box Spreader, acting only on layers outside the skull, cuts the upper face in half and pushes each half off the skull.

3.4.4 Leafer Tool

The Leafer is shaped like a tray (Figure 3.12). The voxels above each half of the tray can be hinged open using 3D widgets. Selecting or unselecting layers causes them to smoothly rotate out or back in to place. A rapid succession of selections initiates an animation with layers temporarily spaced out, or "leaved" (Figure 3.13), affording the user a brief glimpse of the shape of individual layers. This style of browsing inspired the name of the Leafer tool. Note that once the animation is complete, however, all selected layers are rotated open with the same angle.

The three views of layers mentioned in section 3.3 are all made available, simultaneously, with the Leafer. Figures 3.12 shows the areas where layers are seen from a dorsal, ventral, or cross-sectional view. Selection or unselection of a single layer causes that layer to transition from one view to the other.

After hinging open the halves of the Leafer's tray, the layers that make up each half can be fanned open (Figure 3.14). Fanned out layers can then be pulled out and/or flipped over, showing the components of the "dissected" voxels in context with the rest of the data set.

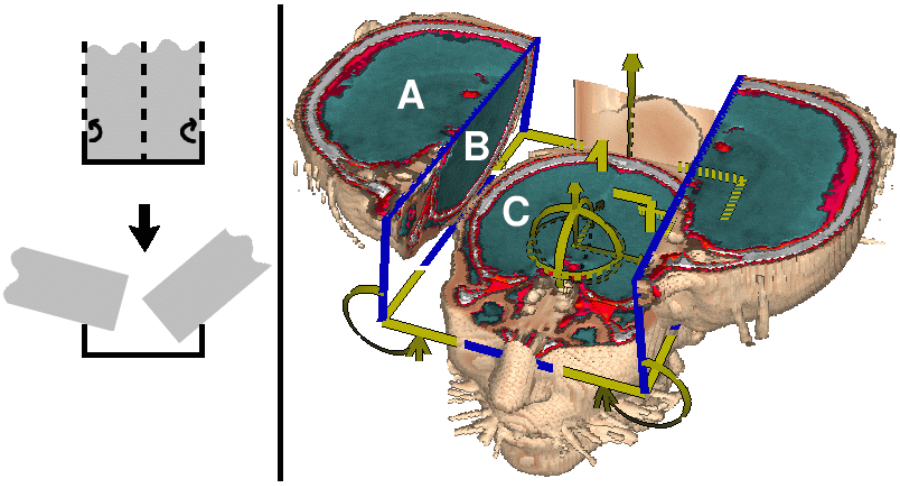

Figure 3.12: The Leafer. Left: voxels before and after deformation. Voxels above each half of a "tray" are (rigidly) rotated away from the centre of the tray. The top edges of the tray are the axes of rotation. Right: the Leafer is used to hinge open parts of a head. Here, the depth of the tray has been set to zero, i.e. the axes of rotation coincide with the bottom edges of the tray, whereas in the figure on the left, the depth of the tray is non-zero. Three areas are labelled A, B, and C, to show how the Leafer provides cross-sectional, ventral, and dorsal views of layers, simultaneously.

3.4.5 Peeler Tool

The Peeler, like the Leafer, consists of a tray that can be positioned and oriented to encompass a region of interest, and allows each half of the tray to be opened up. Unlike the Leafer, however, the Peeler uses a non-rigid, curvilinear deformation to open up the layers (Figures 3.15 and 3.16).

As with the Leafer, selecting layers when the Peeler is active initiates an animated transition, during which the user can see in between the moving layers. However, unlike the Leafer, the Peeler also affords control over the degree of peeling for each layer independently, through arrow widgets attached to each layer. This gives the user an extra

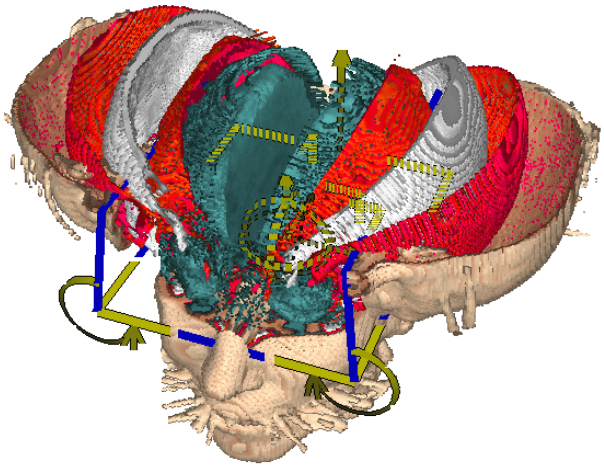

Figure 3.13: When the user selects/unselects a layer, the Leafer animates the rotation of the layer open or back in to place. Here, the user has selected each layer in rapid succession, and the leafer is midway through an animation opening them up. In this way, the user can "leaf" through the layers, as if they were pages of a book.

level of control, allowing the user to create spaces between the layers and keep them in this state for further browsing. Figure 3.17 shows this with the data set of a head, and Figure 3.18 shows it with a different data set, when using *splatting* (i.e. rendering with splats) to render with transparency.

3.4.6 Radial Peeler Tool

We also created a variation on the Peeler called the Radial Peeler (Figure 3.19). Each voxel is peeled radially away from the axis of the tool, as if the tool were poking a hole in the volume and turning the layers inside out, somewhat like a flower opening up.

Figure 3.16 sketches the deformation for the left half of the regular Peeler, where voxels are peeled to the left. If this sketch is revolved around the vertical axis $x = \rho$, where ρ is the radius of the Radial Peeler, the resulting form corresponds to how the Radial Peeler deforms voxels: they are peeled away from the axis $x = \rho$.

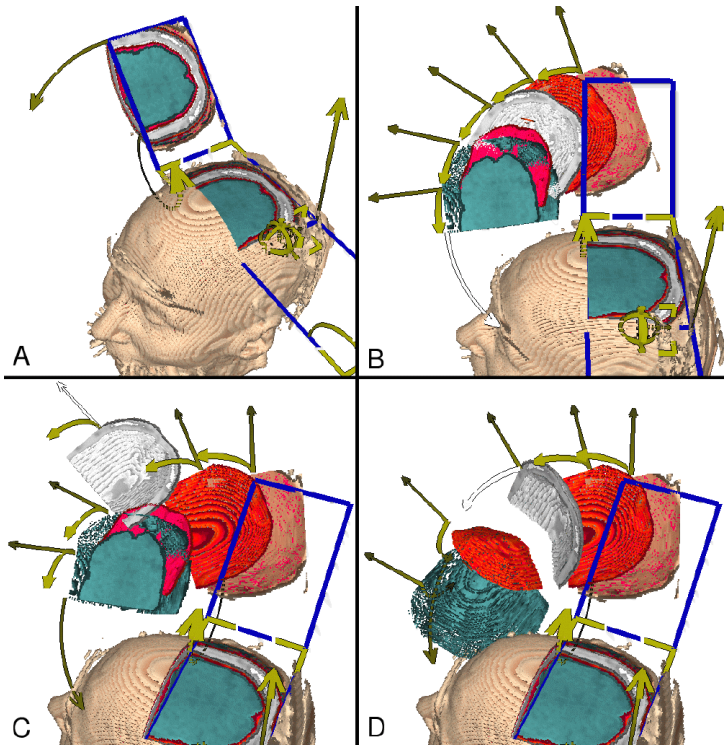

Figure 3.14: A: The left half of the leafer hinges open the top of the head. B: The layers that make up the hinged-open region are fanned open. New widgets appear attached to each layer. C: A translation widget is used to pull out an individual layer. D: Rotation widgets are used to flip over layers.

3.4.7 Observations

Similarities can be seen between our set of tools and the layer browsing techniques of Figure 3.2. The Hinge Spreader, Sphere Expander, and Box Spreader all non-rigidly compress and push layers to reveal data, just like Figure 3.2 D. The Leafer combines the techniques of Figure 3.2 B, E, C and G. And the Peeler, of course, corresponds to Figure 3.2 F. Our tools are not the only possible combinations of techniques, and extensions are possible (such as the exploded view of Figure 3.2 H), but we have demonstrated the applicability of layer-based techniques for browsing volumetric data.

An interesting tradeoff to consider is how much control to give the user. The Leafer allows users to "leaf" through layers (Figure 3.13) using animation. This shows how animation is useful not only for helping the user maintain their mental model of the data

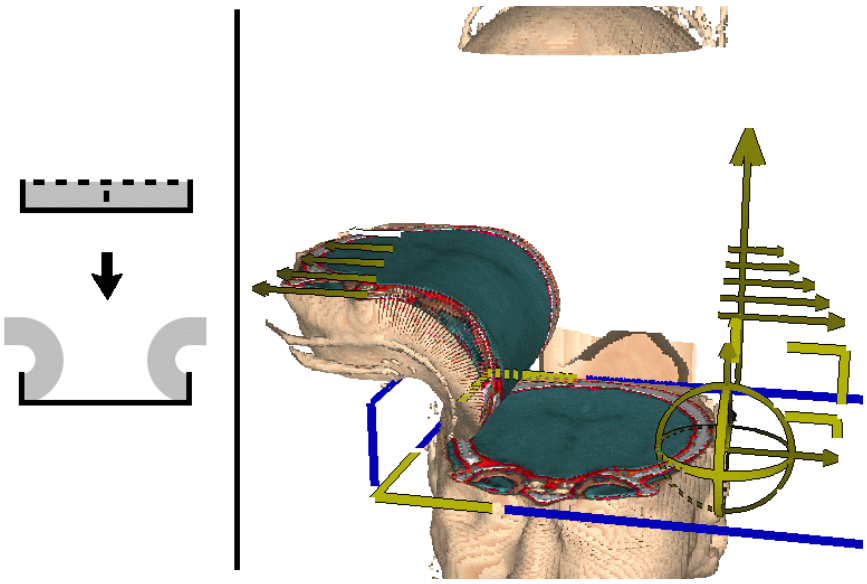

Figure 3.15: The Peeler. Left: voxels before and after deformation. Voxels in each half of a "tray" are peeled off the bottom of the tray. Right: the left half of the Peeler's tray is positioned to encompass the brow of the head, and this is peeled off. Notice that, above the Peeler, a thin section of the top of the head has been automatically translated upward and out of the way.

as it deforms, but can be a browsing technique in itself. In contrast, the Peeler also allows the user to individually peel each layer by different amounts (Figure 3.17), giving the user more control, but this comes at the cost of more 3D widgets that clutter the screen. In general, we reduced clutter from widgets by only displaying them when the state of the deformation warrants their presence. However, additional techniques for reducing the number of widgets shown at any time, without limiting the user's power, would be valuable. One possibility is to develop a kind of popup 3D widget, that is shown only when requested by the user.

The Leafer and Peeler also shed some light on the question of how to connect a deformed set of voxels to the rest of the volume. The Leafer rigidly deforms voxels, creating a sharp "seam" at the axis of rotation. By hinging open the Leafer far enough, the user can easily see this seam where the voxels connect. However, such rigid rotation can lead to interpenetration of the deformed voxels and the rest of the volume. The Peeler, on the other hand, is more continuous in the sense that the deformed voxels

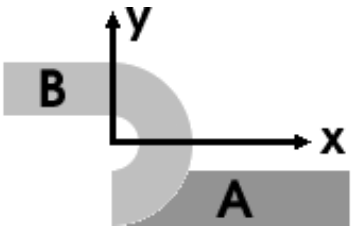

Figure 3.16: The deformation on the left half of the Peeler's tray. Each point $(x, y) \in A$ is mapped to a new point $(x', y') \in B$. Let R be the radius at which length is preserved by the deformation. Points where $x < R\pi$ are mapped to the curved region via $(x', y') = (-y \sin(x/R), y \cos(x/R))$. Other points are shifted and rotated with $(x', y') = (-(x - R\pi), -y)$.

connect smoothly with the rest of the volume. Interpenetration of voxels is less likely, and reduced in severity if it does occur. However, in the case of the Peeler, the seam along which peeled regions connect with the volume is much harder to see, since it is usually occluded by the peeled layers. It may or may not be important for the user to be able to see these seams or contact edges, however, it is a tradeoff to consider when choosing a deformation.

3.4.8 Initial User Feedback

Informal trials with members of our lab led to some improvements in the visual design of the tools and 3D widgets. Furthermore, an earlier prototype of our system didn't employ animations, i.e. deformations would cause the volume to suddenly "snap" to a transformed state. During demonstrations of this prototype, people often had trouble understanding how exactly the tools were deforming voxels. Hence our incorporation of smoothly animated transitions.

More recently, we had a professional anatomist, having little experience using 3D software, try out our system during an informal, one hour session. The session consisted of a mix of designer-driven demonstration and user-driven exploration of the tools.

The anatomist found that the direct manipulation 3D widgets afforded flexible control and were easy to understand. Deforming or pulling out portions of the volume *in context*,

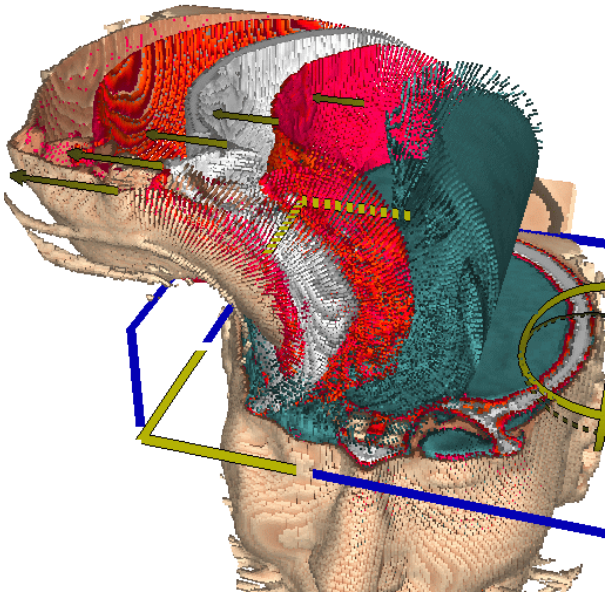

Figure 3.17: A close up of the peeler in action. Here, each layer has been peeled to a different degree (using the small arrow widgets attached to each layer). The spacing between layers makes the interfaces between them visible.

with the rest of the volume still displayed, was found to be very valuable, for keeping track of "where you are in the whole". Animated transitions were also found to be valuable. The anatomist suggested that they would be very appropriate in educational settings, e.g. to show layers actually peeling back, rather than showing a sudden change of state.

The anatomist also suggested that, in some situations, after a layer has been peeled away, it may not be important to continue displaying the layer, since the user's goal may be to simply see the tissues revealed underneath. However, there were other situations were the anatomist found it important to keep all data present, for example when showing the two halves of the Hinge Spreader.

We have subsequently shown our system to another professional anatomist, whose feedback indicates that the most important improvement to make for enabling practical use is to support higher resolution data.

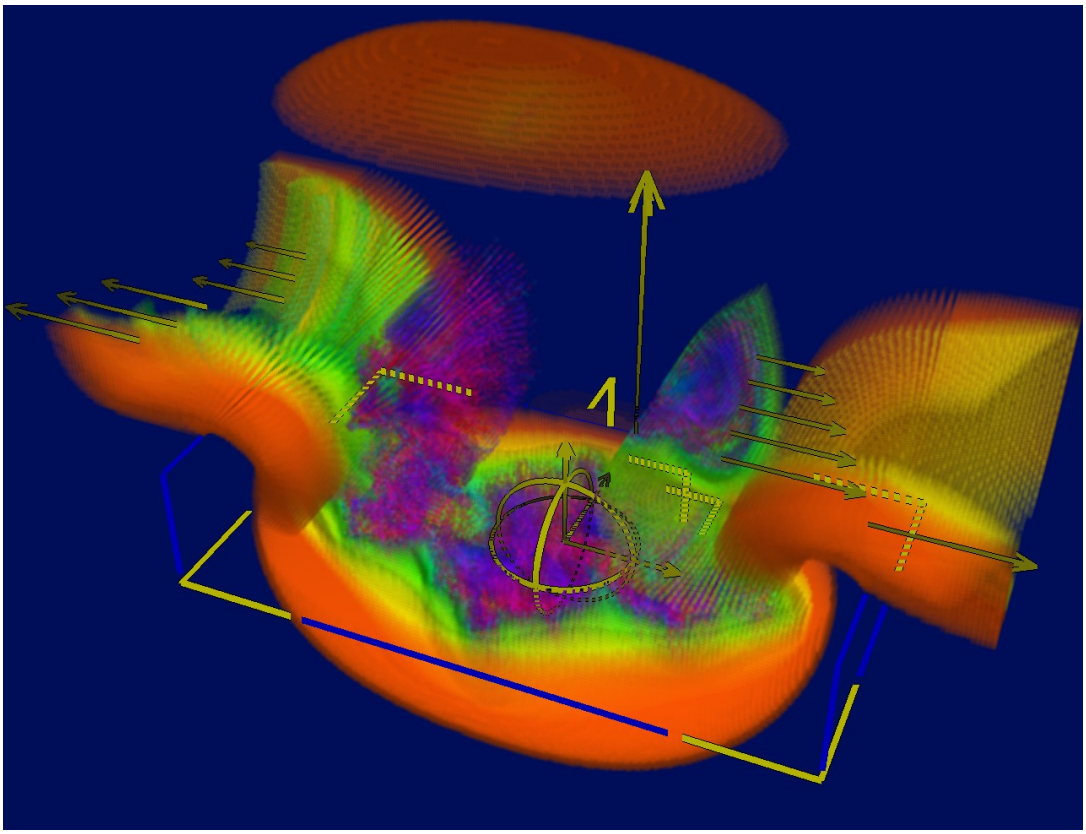

Figure 3.18: The peeler used to browse a 3D slice of a 4D quaternion Julia set, with simultaneous use of transparency.

3.5 Closing Remarks

This case study extended the range of deformations used for exploratory browsing of volumetric data. Our prototype demonstrates one way of integrating these deformations with differential treatment of the layers in a data set, as well as with 3D widgets and use of animation. We have identified various tradeoffs and design issues brought to light by our work. Initial user feedback suggests that our techniques are useful for helping a user understand and maintain context while exploring different regions of a data set.

More broadly, a general issue that this case study dealt with is occlusion, which in a sense arises from having too much data embedded in 3D that must be projected down to 2D. The case study in the next chapter deals with a different problem, where not enough data is projected onto the 2D display surface, but the design goal with respect to output

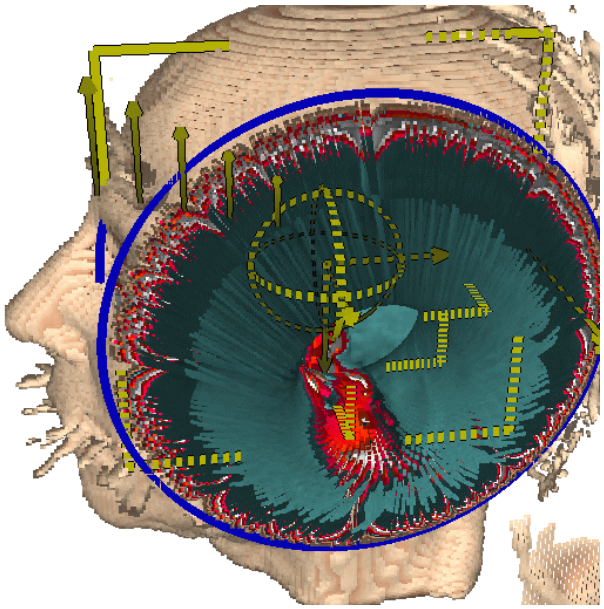

Figure 3.19: The Radial Peeler. Rather than peeling away two halves of a tray as in Figure 3.15, this tool peels away all the voxels in a cylinder. Two circles (only one of which is visible here) delimit the cylinder. Voxels are pulled through the top of the cylinder and then stretched away from the cylinder's centre. A hole along the cylinder's axis is thus opened up, allowing the user to peer inside.

remains the same: to make the amount of useful output as large as possible.

Chapter 4

Expand-Ahead:

A Space-Filling Strategy for Trees

Many tree browsers allow subtrees under a node to be collapsed or expanded, enabling the user to control screen space usage and selectively drill-down. However, explicit expansion of nodes can be tedious. This chapter introduces *Expand-ahead*, a space-filling strategy by which some nodes are automatically expanded to fill available screen space, without expanding so far that nodes are shown at a reduced size or outside the viewport. This often allows a user exploring the tree to see further down the tree without the effort required in a traditional browser. It also means the user can sometimes drill-down a path faster, by skipping over levels of the tree that are automatically expanded for them. Expand-ahead differs from many detail-in-context techniques in that there is no scaling or distortion involved. We present 1D and 2D prototype implementations of expand-ahead, report results of an experimental evaluation of performance with expand-ahead, and identify various design issues and possible enhancements to our designs.

4.1 Introduction

Large tree structures can be difficult to view, navigate, and manage. To help mitigate this, users are often given the ability to view only a subset of a tree at a time. The subset might be specified through selective hiding (collapsing) and revealing (expansion) of subtrees (e.g. Figure 4.1(top left,bottom left)) or might be limited to the "contents" (i.e. children) of one node at a time (Figure 4.1(top right)). Small subsets are more likely to fit on the user's available screen space, and thus have less need for scrolling- or zooming-based navigation.

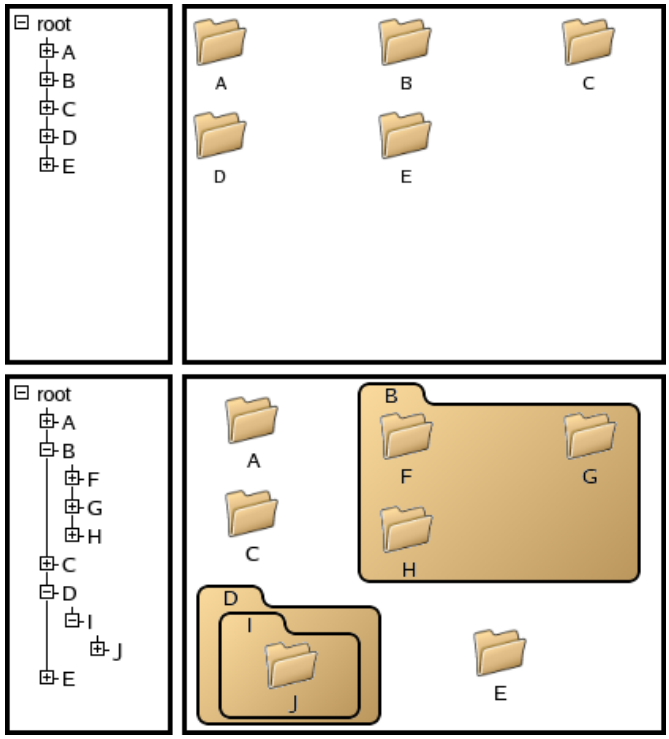

Figure 4.1: Top left and top right: an outline view, and 2D view, respectively, of a node's children, with much space left unused in both cases. Bottom left and bottom right: concept sketches showing that much of the available space can be filled by expanding some children.

At the same time, working with subsets of a tree can be inconvenient. Many subsets will only partially fill screen space. Although the user may move from subset to subset during browsing and exploration, showing the user small subsets can leave their visual

system's capacity underutilized and prolong navigation tasks. For example, in the specific task of travelling down a path to a leaf, the user is often required to explicitly expand, or "click through", each node along the way. If the user does not know or forgets what is located under a node, they must explicitly expand, or travel into, the node to find out, and then backtrack if they discover they took a wrong turn. Some interfaces show previews of node contents in the form of summaries or thumbnails; however, these only help direct the user's navigation — the user must still explicitly expand each node along a desired path.

Often, during navigation, a user may be momentarily viewing a relatively small or narrow portion of the tree, with unused screen space left over (Figure 4.1(top left,top right)). In these cases, we suspect it might often be beneficial for the system to *automatically* expand some nodes, to fill up the available screen space (perhaps resulting in something like Figure 4.1(bottom left,bottom right)). If the space consumed by such expansion does not exceed the limits of the user's viewport, then no scrolling or zooming will be required as a result of the automatic expansion. The intention is that, of the nodes revealed by such expansion, those that are *not* of interest to the user can be safely ignored, and those that *are* of interest are now visible to the user for free, with no extra input from the user.

We call this scheme *expand-ahead*, because the system automatically expands pathways downward and *in advance of* explicit expansion by the user. Expand-ahead allows the user to see the contents of more than one folder at a time. It is only performed when there is unused screen space, and is only done to the extent allowed by such space. The automatic expansion never proceeds so far that it would exceed the available screen space, because this would impose a penalty on the user due to the scrolling or zooming necessary to see the resulting information. To make the fullest possible use of screen space, expand-ahead is not, generally, carried out to the same depth along all possible pathways. Instead, usually some nodes are chosen for expansion over others at the same

depth. To determine which of the alternative nodes to expand, a heuristic or policy is required, which is given as a parameter to the expand-ahead algorithm. The heuristic can be designed to give preference to nodes that are more likely to interest the user.

Because expand-ahead reveals more information without requiring additional input, we suspect it may benefit general browsing and navigation tasks. In particular, it may allow users to drill-down a path faster, by allowing them to skip over nodes that are automatically expanded. Users may also tend to take fewer wrong turns down paths, because they can see further ahead. On the other hand, in a more free-form browsing scenario, expand-ahead may benefit users by allowing them to incidentally notice interesting nodes that they had no intention of drilling down toward, and that would have otherwise remained undiscovered without expand-ahead.

There are also potential drawbacks to using expand-ahead. Like other adaptive user interface techniques, automatic reconfiguration of interface elements can sometimes confuse users, or give them an impression of not being in control. Inappropriately designed adaptation can even hinder rather than help the user. In light of this, we conducted an experiment to evaluate the performance of users with expand-ahead under controlled conditions.

4.1.1 Approach to Pursuing Design Goals

Consider the 1st design goal: to increase and improve the output. In the last case study, occlusion was the problem with the output, caused by having *too much* data projected onto the display. Traditional approaches remove some of the data to reduce occlusion, whereas our use of deformations was intended to only rearrange the data to make it more useful and informative. Either way, the emphasis is on improving the output, and not increasing the (raw) amount of data projected onto the display.

In contrast, for the present case study, the problem to address is *under*utilization of screen space, thus our approach naturally enough is to *add* to the data displayed to make

fuller use of the output channel's capacity. However, in the visualizations of trees we'll consider and design, the output is not purely pictographic — it contains many text labels, and these cannot be read in parallel by the user. So, increasing the amount of output shown to the user may incur a cost if the user finds it necessary to read many more text labels that turn out to be irrelevant. Our hope is that this will not be a major problem if the graphical layout of the output enables the user to quickly direct their attention to the most interesting information. However, the possibility that the user may be hindered by having to process more information is a general risk that designers must be aware of when increasing the amount of output displayed. We measure the effect of taking this risk, with respect to one task, with a controlled experimental evaluation.

The task used in the experiment is drilling-down a path to a leaf node. Unlike perhaps other space-filling techniques for visualizing trees, the idea for Expand-Ahead was conceived not just to provide a more complete view of the selected subtree, but also very much to facilitate this drill-down task. The unused screen space that we wish to reduce is not just a waste of display space for output, but a waste of *input* space where selectable targets could be. Thus the 2nd design goal, of easing input, is addressed by increasing the number of selectable nodes displayed to the user at any given time, so they may navigate further away from their current state with each selection, thereby traversing the space of states with greater speed.

As with the previous case study, addressing the 3rd design goal, of having smooth state transitions, is again important due to the changes that occur to individual nodes (collapasing, expanding, changing position) when the user changes the state of the visualization. In this case study, in addition to using animated transitions, we also implement a novel technique for the user to traverse a sequence of previously visited states with a single drag of the mouse, further pursuing the 2nd and 3rd design goals.

4.2 Background

Many schemes exist for browsing large spaces in which a tree, or other information, is embedded. Carpendale [1999, chapter 2] surveys these techniques, including scrolling, zooming, fisheye views, and various other detail-in-context views. Conceptually, these schemes can be thought of as changing the *presentation* [Carpendale, 1999, chapter 1] of the space (e.g. by smoothly deforming it), without changing the information's *representation* or embedding in the space. The collapsing and expanding of tree nodes, however, is probably more naturally thought of as a change in the tree's representation or embedding. Despite this, the effect of expand-ahead is somewhat similar to that of focus+context techniques, in that the automatic expansion of descendants under a node of interest can be thought of as revealing more of the neighbourhood around the user's focus.

Some tree representations, like Treemaps [Johnson and Shneiderman, 1991; Shneiderman, 1992b], Pad++'s directory browser [Bederson and Hollan, 1994], Nguyen and Huang's [2002] space-optimized trees, and RINGS [Teoh and Ma, 2002], pack nodes into the available screen space by scaling down the size of nodes, allotting progressively less space for nodes further down on the tree. Although this allows a large number of nodes to be fit on the screen, any labels or information displayed with the nodes becomes increasingly illegible in the lower levels of the tree. Depending on the user's goals, it may be preferable to see fewer nodes, but have labels and other information all equally legible. For example, in the context of their work, Plaisant et al. [2002] quote one user saying "Make it readable or don't bother showing the nodes".

In expand-ahead, the size of text labels is held constant, so that automatic expansion never reduces the legibility of text. Furthermore, and unlike Treemaps for example, the space allocated to a given subtree is not based on the "size" of the subtree, but rather is a function of whatever expansion heuristic has been chosen.

SpaceTrees [Plaisant *et al.*, 2002] show preview icons of collapsed subtrees, and also

perform a form of intelligent, automatic expansion for the user. From our perspective, SpaceTrees implement a special case of expand-ahead which we will later call *uniform expand-ahead* (see section 4.9). When users select a focal node in a SpaceTree, the number of levels opened under that node is maximized, as allowed by available screen space. Each level, however, is only expanded if all the descendants on that level can be revealed. Our more general notion of expand-ahead allows certain nodes on a given level to be expanded, while their siblings may not be. This yields representations that are not as orderly and regular as SpaceTrees, but allows us to fill space more aggressively than SpaceTrees do. Another difference is that SpaceTrees expand nodes based solely on available space, whereas in expand-ahead, the decision of which nodes to expand is influenced by the expansion heuristic, making it somewhat more flexible. A final, less critical difference, is that SpaceTrees use a traditional node-link representation for the tree, whereas we have explored automatic expansion within outline (Figure 4.2) and nested containment (Figure 4.3(bottom,top right)) representations.

Many adaptive interfaces automatically reconfigure or rearrange interface elements to try and help the user by reducing the effort or amount of input required (see [Sears and Shneiderman, 1994] for discussion of this in the context of menus). Unfortunately, such adaptation can also confuse and frustrate the user, especially if the actions taken automatically are inappropriate and/or the user does not understand how the system determines which actions to take. There is a danger that expand-ahead could cause related problems, particularly if the expansion heuristic is poorly chosen. Expanding nodes that don't interest the user would only increase the amount of noise on the display that must be filtered out by the user, making it harder to find nodes that do interest the user. To try and alleviate this problem, expand-ahead never changes the *ordering* of nodes, as some adaptive menus do. Although automatic expansion may introduce irregular spacing between siblings, the user may still be able to employ a subdivision strategy when searching for a node, since the ordering of neighbouring nodes does not

change. Furthermore, when a node is expanded, *all* its children are displayed, rather than, for example, just the most frequently accessed subset, as is done in Microsoft Office's adaptive menus.

In partial support of our design, browsers that display a 2D row-column arrangement of icons, such as in Figure 4.1(top right), typically reflow the icons when the browser window is resized, changing the number of columns and rows. This behaviour is familiar to users, and seems to be far less disturbing than a re-ordering of icons would be.

In section 4.9, we speculate on ways to make expand-ahead more consistent in the way it presents information, to further reduce the drawbacks of its adaptive behaviour.

4.3 The Expand-Ahead Algorithm

Let F be a node in the tree T that the user has selected as the node of interest, or *focal node*. Our current implementation of expand-ahead works as follows: (1) expand F, and allocate space on the screen for F and its children; if there is any space left over, then (2) try expanding each of the children of F in turn, such that the available screen space is never exceeded; if any of them were successfully expanded, and there is still space left over, then (3) try expanding each of the children of the children of F that were successfully expanded, such that the available screen space is never exceeded; etc. Stop when there is no longer enough screen space to allow any more expansion, or when we have reached the leaf nodes.

Notice that the order in which the algorithm attempts to expand nodes is breadth first, or level-by-level.

Often, there may be sufficient space to expand one or another child, but not both. In this case, some means is necessary to determine which child to expand, or in which order to attempt expansion of children. Let $w(n)$ be a weight associated with node n. The expand-ahead algorithm prefers expansion of nodes with a greater weight over those with

less weight. Thus, the $w(n)$ function can encode various heuristics for node expansion, which may be based, for example, on the likelihood that a given node will interest the user.

More formally, the expand-ahead algorithm is given as Algorithm 1. In the pseudocode, curly braces enclose comments.

Algorithm 1 ExpandAhead(T, F)

```
{initialize all nodes to be collapsed}
CollapseAllNodesInTree( T )
{expand F and all its ancestors}
n ← F
while n ≠ NIL do
  n.isExpanded ← true
  n ← n.parent
{check if there's screen space left over}
ComputeLayout( T, F )
if there is unused screen space then
  {expand as many nodes under F as possible}
  d ← 1
  repeat
    noNodesSuccessfullyExpanded ← true
    S ← set of all visible nodes at depth d under F
    sort S by weighting function w
    {try expanding each node in S}
    for all n in S, in decreasing order of w(n), do
      if n has children then
        n.isExpanded ← true
        ComputeLayout( T, F )
        if available screen space is exceeded then
          {backtrack}
          n.isExpanded ← false
        else
          noNodesSuccessfullyExpanded ← false
    d ← d + 1
  until noNodesSuccessfullyExpanded
  ComputeLayout( T, F )
```

The ComputeLayout subroutine called in the pseudocode is responsible for computing the embedding of the tree T, i.e. allocating space for all visible nodes and positioning them on the screen with respect to the focal node F. ComputeLayout can be chosen

to generate any tree layout style that is desired, be it of a node-link style, a nested containment layout, or otherwise.

The $w(n)$ weighting function encodes the heuristic for choosing which nodes to expand. For example, setting $w(n) = 1/n.\text{numChildren}$ causes nodes with a small number of children to be preferred over nodes with more children. Such a weighting tends to maximize the number of nodes that are expanded automatically, since more nodes can be expanded if each has few children. Another possible weighting is $w(n) = n.frequency$, i.e. nodes that were visited more frequently by the user in the past are given a greater weight, since they are more likely to be visited again by the user.

The ExpandAhead algorithm described by the pseudocode is invoked every time the user selects a new focal node F, which might be done by simply clicking on a visible node, or travelling upward to the current focal node's parent, or selecting a previously visited node from the browser's history.

The foregoing description assumes the user is only interested in one focal node at a time, as is supported by our current implementation. In section 4.9, however, we describe how expand-ahead might be extended to support multiple focal nodes.

4.4 Comparison with Furnas' DOI

Furnas [Furnas, 1981; 1986] introduced a "degree-of-interest" (DOI) function that can be used to generate fisheye-style presentations of information. Given a set S of elements that could be displayed, in general if S is large we must decide which elements of S to display and which to elide. For each $x \in S$ we define the intrinsic, *a priori* importance associated with x as $LOD(x)$ ("level-of-detail"), i.e. LOD is greater if x is more important, more intrinsically interesting, higher-level, or of a coarser granularity. In addition, given a focal element $f \in S$, we define the distance from f to x as $D(f, x)$. Then, the *a posteriori* interest associated with x is $DOI(x) = F(LOD(x), D(f, x))$, and might typically be

$DOI(x) = LOD(x) - D(f, x)$. So, elements that are more intrinsically important, or that are closer to the focal element f, have a greater DOI. Then, given a threshold k, we can generate fisheye-style presentations by only displaying elements whose $DOI > k$.

How is Furnas' DOI function related to the Expand-Ahead algorithm and its weighting function $w(n)$? Certainly, the LOD function and weighting function $w(n)$ are analogous, however there are other differences between the two approaches. Is there a DOI function and threshold that would result in the same behaviour of the Expand-Ahead algorithm? Not without a significant reformulation of the DOI, which even if performed would be no easier to compute than the ExpandAhead algorithm. One reason for this is that, in Expand-Ahead, whether a descendant of the focal node is displayed or not depends on whether enough space is available to display it, which in turns depends on the total space available and on which other nodes have already been allotted space. In contrast, in the definition of the DOI function, the only part that depends on x is the distance $D(f, x)$ to the focal element x, which does not naturally capture information about total available space or presence of other nodes. Furthermore, note that in Expand-Ahead, a node with a lower weight but also at a lower depth will *always* be preferred over a node with a higher weight but at a greater depth, because of the breadth-first nature of the algorithm. This aspect is also not easily captured in a definition of a DOI function, where typically for any given distance (analogous to depth) the DOI can be varied arbitrarily by varying the LOD (analogous to weight).

Neither the DOI nor the Expand-Ahead approach is more general than the other, though one could imagine a generalization of both approaches that would capture all their associated behaviour, but that would also be more complicated and have more parameters to be tuned.

4.5 1D Prototype

Our prototype expand-ahead browsers were implemented in C++ using the OpenGL and GLUT libraries, and run under the Linux and Microsoft Windows operating systems.

The tree browsed by our prototypes can be either read in lazily from the file system, allowing the user to browse their directories and files; or can be extracted as a breadth-first tree (BFT) of a digraph described in an input file. In the future, we plan to modify our prototypes to allow dynamically changing the root of the BFT, and investigate the use of our browsers for exploring graph structures.

The 1D prototype displays nodes in the form of an outline view — it is 1-dimensional in the sense that nodes are arranged as a list, with horizontal indentation showing the tree structure. Unlike many other outline browsers, our 1D browser does not allow the user to independently toggle the expansion of individual nodes as can be done with the "+" and "−" icons in Figure 4.1(top left,bottom left). Support for this might be added eventually (see section 4.9), but we chose a simpler design for our first prototype. Instead, expansion is controlled only by selecting the focal node F. Clicking on a node makes it the new focal node, which is moved to the top of the viewport, with its descendants displayed below it, and expanded according to the expand-ahead algorithm.

Figure 4.2 shows the 1D browser with two different focal nodes. The expansion heuristic used here, as well as in our later prototype, is $w(n) = 1/n.\text{numChildren}$.

4.5.1 A Rough Model of User Performance

One of the potential advantages of expand-ahead is that it may allow a user to drill-down a path faster, by skipping over levels that are expanded automatically. If the tree is thought of as a hierarchical menu, then expand-ahead is one way of making the tree, or menu, broader and more shallow: fewer levels need be explicitly traversed by the user, and at each step, the user has more nodes to choose from than without expand-ahead.

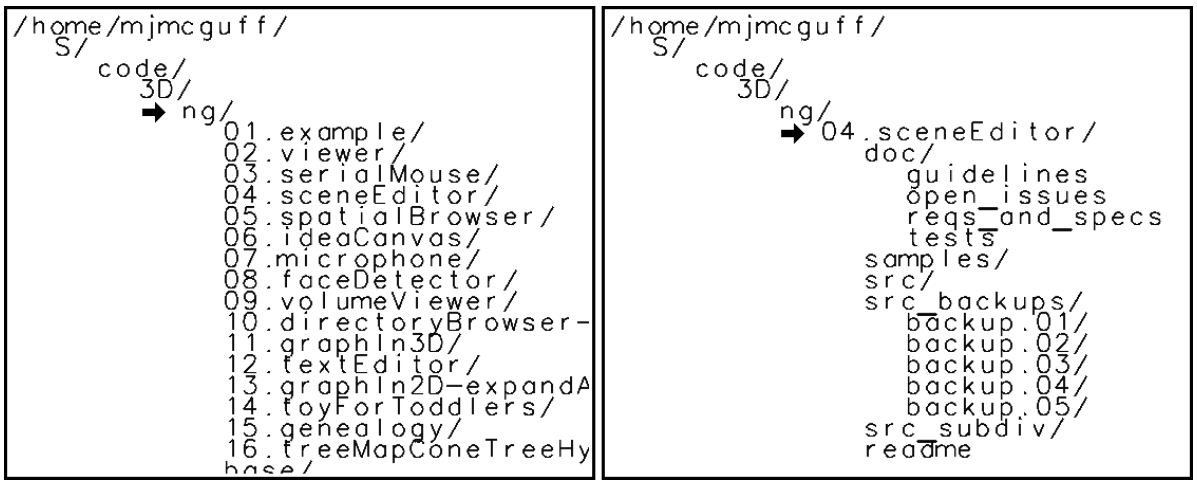

Figure 4.2: In these views of files, slashes at the end of a label indicate that the file is a folder and may contain children. Left: The user has selected "ng" as the focal node (indicated with the arrow). This node contains too many children to fit in the viewport — viewing all the children requires scrolling. Because of this, the expand-ahead algorithm has not expanded any of the children. Right: The user has selected "04.sceneEditor", a child of "ng", as the new focal node. Since the children ("doc", "samples", "src", etc.) of the focal node consume only some of the vertical space, the expand-ahead algorithm has filled the rest of this space by expanding two of the children, namely "doc" and "src_backups".

The question of breadth vs depth in menus has been studied before [Shneiderman, 1992a, chapter 3] [Landauer and Nachbar, 1985] and it has generally been found that reducing depth by increasing breadth allows selection of leaf items to be made faster overall.

A quantitative estimate of the advantage, if any, of expand-ahead would be valuable. In the particular case of a 1D outline tree view, we can model the task of drilling down in terms of another well understood model: Fitts' law [Fitts, 1954; MacKenzie, 1992]. Fitts' law predicts that the average time T to acquire, or click, an on-screen target of size W at a distance D from the cursor is

$$T = a + b \log_2(D/W + 1) \tag{4.1}$$

where W is the target width measured along the direction of motion, and a and b are experimentally determined constants that depend on factors such as the particular input

device used for pointing. From equation 4.1, we see Fitts' law predicts that decreasing the size W of a target, or increasing the distance D to a target, both increase the time required to acquire the target.

If drilling down a path involves clicking on each of a sequence of nodes, this can be modelled as a sequence of Fitts' target acquisitions.

Consider an approximately balanced tree with N leaf nodes and a constant branching factor B. Assume that the tree is displayed as a 1D outline, such as in Figure 4.1(top left), and that the height of each node in the outline view is W. If the user only sees one expanded node at a time, the total height of the outline view is BW. Furthermore, if the user's cursor starts at a random vertical position, and must travel to a random node, the average distance $D_{average}$ to travel will be $BW/3$ (since the mean distance between two points randomly selected on a unit segment is 1/3).

Without expand-ahead, travelling down a path from the root to a leaf requires one click per level in the tree, or $C = \log_B N = \log_2 N / \log_2 B$ clicks. The time required for each click can be broken down into a sum of the time T_F to *find* the desired node to click on (including any time to visually process information on the screen), and the time T to acquire the target, as given by Fitts' law. The total time to drill-down is then

$$\begin{aligned}
& C(T_F + T) \\
&= C(T_F + a + b\log_2(D_{average}/W + 1)) \\
&= (\log_B N)(T_F + a + b\log_2(BW/3W + 1)) \\
&\approx (\log_B N)(T_F + a + b\log_2(B/3)) \\
&= (\log_B N)(b\log_2 B + a - b\log_2 3 + T_F) \\
&= b\log_2 N + (\log_B N)(a - b\log_2 3) + \frac{\log_2 N}{\log_2 B} T_F
\end{aligned} \tag{4.2}$$

As stated earlier, the effect of expand-ahead in a 1D outline is to visually flatten and broaden the tree being navigated. Although the tree's topological structure does not

change, expand-ahead reveals more nodes to the user, increasing the number of nodes the user may click on at each step, and decreasing the number of levels the user must explicitly click through. Thus, expand-ahead can be thought of as increasing the "visual" branching factor B of the tree, which reduces the necessary number $C = \log_B N$ of clicks. However, because B is increased, so is the average distance $D_{average} = BW/3$ the user must travel for each click, and so therefore is the time T required for each click.

Interpreting B as the visual, or effective, branching factor allows expression 4.2 to describe both the cases with and without expand-ahead. Keeping in mind that our goal is to minimize the total time to drill-down, we examine each of the terms in expression 4.2. The first term $b \log_2 N$ is the time required for the user to "express" (via their pointing device) the $\log_2 N$ bits of information associated with the leaf node. This does not depend on B, and hence is not affected by use of expand-ahead. The second term $(\log_B N)(a - b \log_2 3)$ is the number of clicks multiplied by a constant time penalty associated with each click. Assuming this term is positive (a has been found to be considerably larger than b in many Fitts tasks), the term is minimized when B is maximized, which *favours* expand-ahead. It is unclear how the last term $(\log_2 N / \log_2 B)T_F$ may change as B changes. This depends critically on the nature of the time T_F to find the next node to click on. T_F most likely increases with B, because an increased B means the user will have more nodes to visually scan. If T_F increases linearly with B, as would be expected in a scan-and-match visual search, then the last term of expression 4.2 will also increase with B, which would argue *against* using expand-ahead. However, if T_F only increases logarithmically with B, as may be expected if the nodes are ordered alphabetically and the user employs a subdividing visual search strategy (see [Landauer and Nachbar, 1985] for discussion of this with respect to the Hick-Hyman law), then the last term of expression 4.2 should remain approximately constant.

In summary, if T_F is at most a logarithmic function of B, then expand-ahead should decrease the total time to drill-down a path. However, if T_F increases faster than loga-

rithmically, it is unclear whether expand-ahead would yield a net increase or decrease of the total time.

The above is only a first attempt to model performance with expand-ahead. Unfortunately, there seems to be no way of directly measuring T_F, and hence no easy way of testing the model against an experiment. The main value of the model is its indication that the possibility of a net benefit from expand-ahead hinges on how quickly users will be able to find the desired target for each click, and thus may depend critically on the graphic design chosen for the presentation of information. In any case, experimental investigation is needed to measure actual performance.

4.6 2D Prototype

As with all 1D outline tree browsers, our 1D prototype arranges nodes along one direction (the vertical), and only uses the 2nd direction for indentation, rather than for showing additional nodes of the tree. Our 2D prototype attempts to make full use of both directions by tiling nodes along rows and columns (Figure 4.3). Expanded nodes are represented using nested containment, and drawn as folders with a tab for their label. Unexpanded nodes can be optionally shown as either simple text labels (Figure 4.3(top left,bottom)), or with icons (Figure 4.3(top right)).

Recall the ComputeLayout subroutine, called in the ExpandAhead algorithm, which computes the layout or embedding of the tree. In our 1D prototype, ComputeLayout is a simple and fast subroutine, because the layout of nodes is very regular. However, in our 2D prototype, the ComputeLayout subroutine involves a recursive, bottom-up computation of the layout of the nodes, performed by some 400 lines of C++ code, and done once for each node that the ExpandAhead algorithm tries to expand. Thus, while the ExpandAhead algorithm proceeds *down* from the focal node in a breadth-first manner, each invocation of the ComputeLayout subroutine traverses the visible nodes from the

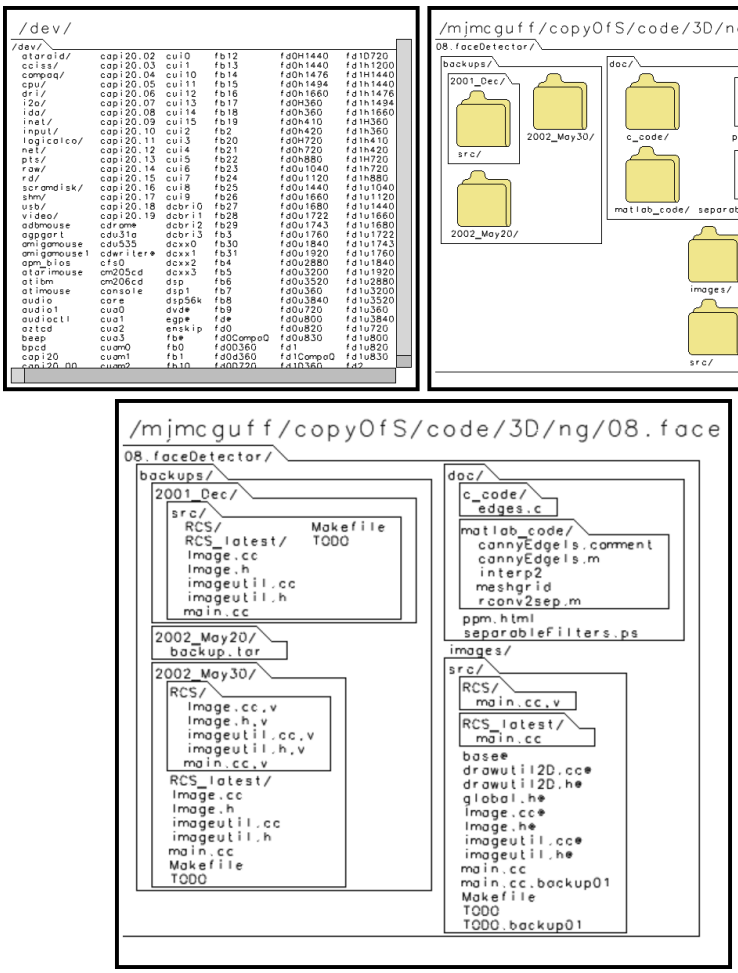

Figure 4.3: Top left: the children of the focal node are arranged in rows and columns, but are too numerous to fit in the viewport. Hence, scrollbars are provided to pan the view, and no automatic expansion of nodes is performed. Bottom: a different focal node, with fewer children, allows expand-ahead to be performed. Top Right: viewing the same focal node as bottom, with icons enabled.

deepest nodes *upward*, computing the space required by each node as a function of the space required by its children. Fortunately, on a 1.7 GHz laptop, all these computations only create a noticeable delay if the user is looking at over 500 nodes simultaneously. In addition, we have identified some possible optimizations that could be made to our particular ComputeLayout subroutine which remain to be implemented.

The layout done by the 2D ComputeLayout subroutine arranges each set of children within rows and/or columns. The flow of the layout can be optionally changed between either (a) filling each column, from top-to-bottom, in an inner loop, and creating whole

columns left-to-right in an outer loop (this flow is used in Figure 4.3), or (b) filling each row, from left-to-right, in an inner loop, and creating whole rows top-to-bottom in an outer loop (as per Figure 4.1(top right)). A second independent option controls whether nodes are centred within cells of a "grid" with rows and columns that cut across the entire grid; or whether nodes are packed along one direction in the manner of a greedy line-breaking algorithm [Achugbue, 1981], resulting in the brick-like arrangement of Figure 4.3(bottom,top right).

When computing the layout of children within an expanded node, a choice must be made as to the number of rows or columns to use. For example, 12 equally sized children could be arranged in a grid of 3×4, or 4×3, or 2×6, etc. We use an approximate rule of thumb that tries to arrange children such that the parent node has an aspect ratio close to 1.

As with our 1D prototype, the focal node in the 2D prototype is selected by clicking on the desired node. A change in the focal node can cause a large change in the arrangement of nodes, which is especially noticeable in our 2D prototype because it can display many more nodes than the 1D prototype. Early testing of our initial 2D prototype quickly convinced us that some kind of animated transition was critically needed, to help the user maintain their mental model of the tree's layout, and see which nodes are hidden, revealed, or repositioned/resized during a change of focus. Figure 4.4 shows the 2D view of nodes before, during, and after such an animated transition.

Inspired by the design of the 3-stage animations in SpaceTrees [Plaisant *et al.*, 2002], we implemented animated transitions consisting of 5 distinct phases: (1) fading out visible nodes that will be hidden after the transition, (2) collapsing the outline of expanded nodes that must be collapsed by the end of the transition, (3) moving and resizing nodes using linear interpolation, (4) expanding the outline of nodes that were initially collapsed but that must be expanded by the end of the transition, (5) fading in nodes that are newly visible. We adjusted the animation to last a maximum of 1 second in total, and to skip

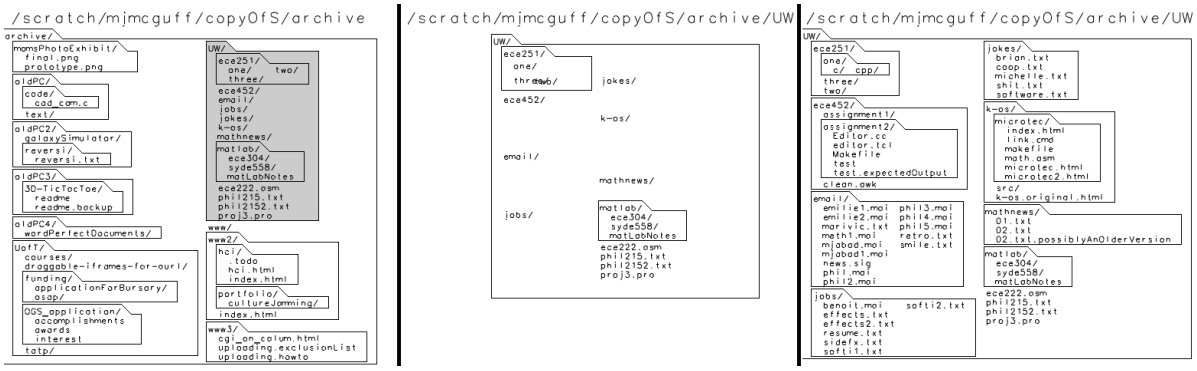

Figure 4.4: Left: the user selects a new focal node (hilited), causing an animated transition to begin. Middle: midway through the transition, the subset of nodes that are visible both before and after the transition are being moved and enlarged on their way to their final configuration. Right: after the transition, the user sees the contents of the new focal node.

over a stage if it does not involve any nodes in the given transition.

Our prototype maintains a history of focal nodes visited. As in a web browser, this history can be navigated using *Back* and *Forward* buttons. Hitting either button invokes a reverse (or forward) animation to the previous (or next) focal node in the history. In addition, the user may hold down the right mouse button to pop up a dial widget (Figure 4.5) that can be rotated to scrub over the animations. Rotating the widget clockwise or anticlockwise moves forward or backward through the history, at a rate of one focal node per cycle. The user may scrub at any speed, or stop and linger, allowing for careful examination of complicated transitions if desired.

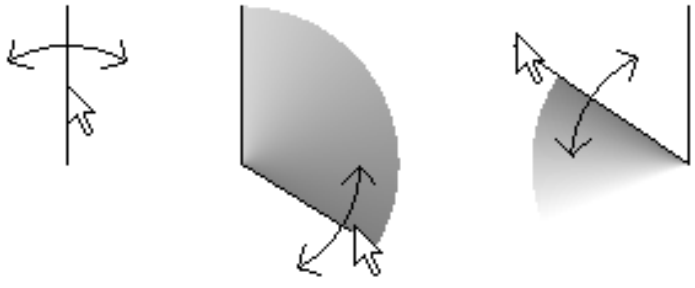

Figure 4.5: A popup dial widget. Dragging rotates the dial, which is used to scrub back or forward over animated transitions.

In addition to showing changes in focal nodes, animated transitions are also used to show changes in layout resulting from user-requested changes to the font size used for text labels. A decreased font size means each unexpanded node requires less space, allowing more nodes to be expanded, which sometimes changes the layout significantly. The user can incrementally decrease the font size one pixel at a time, by hitting a hotkey repeatedly, invoking a sequence of animations showing the successive changes in layout. Visually, this is comparable to zooming in, in that gradually more detail (i.e. lower levels of the tree) is revealed. However, unlike literal zooming, the focal node, and hence the surrounding context, never changes. Decreasing the font size in effect allows the user to drill-down everywhere in the tree simultaneously, yielding an increasingly detailed "birds-eye" view of the tree (Figure 4.6). We call this *zooming down*. Of course, sufficient reduction of the font size eventually makes the text labels illegible. The reverse action, of incrementally increasing the font size, is similarly a variation on zooming out and rolling up, which we call *zooming up*.

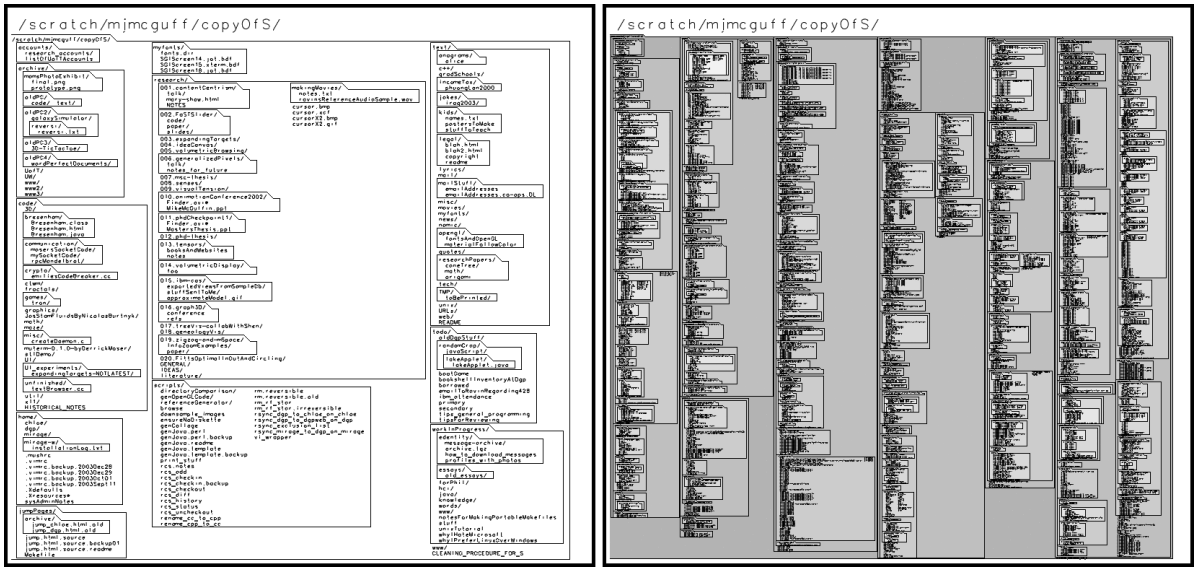

Figure 4.6: *Zooming down*: a variation on zooming in and drilling down. Left: the font size has been reduced so that 250 nodes are visible. Right: the font size has been further reduced, so that now 2400 nodes are revealed. To make the tree structure more apparent, nodes are filled with a shade of grey dependent on each node's depth (see section 4.9).

4.7 Pros and Cons of Expand-Ahead

The intended benefits of expand-ahead include revealing more information to the user by exploiting available screen space, and enabling faster drill-down due to fewer clicks being required of the user.

At the same time, there are various potential drawbacks to using expand-ahead. Having more targets on the screen implies a higher average distance to travel to acquire a target, which, by Fitts' law (equation 4.1), increases acquisition time. Having more information on the screen also means the user will probably spend more time visually scanning and parsing the information, and may be distracted by irrelevant information. These factors were modelled in section 4.5.1, without coming to a definite conclusion on their cost. Other potential drawbacks of expand-ahead are that, by not expanding nodes to the same depth uniformly, expand-ahead can give the user a lopsided view of the tree, since nodes on the same level can be treated differently — this can be either good or bad. Finally, the arrangement and expansion of nodes shown to the user can change not only when the focal node changes, but also if the font size or window size changes, or if the tree's structure changes (e.g. due to insertion or deletion of nodes). Such rearrangement can cause confusion, and if frequent enough, would inhibit habituation and make it impossible for the user to memorize the spatial location of nodes.

Despite this, rearrangement may not be a severe problem in many practical cases. Expand-ahead never changes the ordering of nodes, so users may still learn to find nodes quickly by using their neighbours as relative landmarks. The relocation and reflowing of nodes in our 2D prototype is comparable to the reflow of rows and columns in interfaces such as in Figure 4.1(top right), which are already familiar to many users. Changes in font size might be infrequent for many users, and changes in tree structure may not be disturbing to the user if it is the *user* who performs, and is thus aware of, any change to the tree. Animated transitions can also help the user keep their mental map of nodes intact during changes.

Since it is unclear how the potential benefits and drawbacks of expand-ahead compare, we performed a controlled experiment involving a drill-down task, and measured user performance under various conditions. The drill-down task was chosen because we consider it a fundamental task in which performance is easy to measure. However, we also suspect that expand-ahead could benefit performance in other tasks that require browsing or acquiring an overview of the data, and of course these tasks would have to be evaluated separately to fully evaluate expand-ahead as a technique.

4.8 Controlled Experiment

4.8.1 Goals

To measure the net effect of expand-ahead on user performance, a controlled experiment was performed in which users completed a task using expand-ahead and without using expand-ahead. In particular, we wanted to determine if users are able to drill-down (i.e. travel down from the root to a leaf) faster with expand-ahead than with purely manual expansion.

4.8.2 Apparatus

The experiment was run on 3 computers (enabling us to run 3 users in parallel), each located in an isolated, soundproofed room, and each running Microsoft Windows. The screens were 15 inches in size, set to a resolution of 1024×768 pixels. The experiment program was run in full screen mode, with a 16 pixel high font used for text. The input device was a mouse held in the user's dominant hand, with the keyboard used only to start each trial by hitting a key with the user's non-dominant hand.

4.8.3 Participants

Users were solicited from a pool of external users through the User Centred Design Department of the IBM Toronto Software Development Lab. 12 users participated in our study, 8 women and 4 men, all right handed, whose usual computer use ranges between 1 and 12 hours per day, 5 to 7 days per week. The users were aged 23–57 years (mean 38.25, standard deviation 11.4).

4.8.4 Task

Users completed a number of trials, within each of which the user had to drill-down a target path and select a leaf node of a tree. Before the beginning of each trial, the screen first showed the user the path of the target leaf for the next trial, as a slash-delimited string of nodes, e.g. "abc/def/...". To start the trial, the user had to place the mouse cursor in a 10×10 pixel start box at the upper left corner of the screen, and hit the spacebar with their non-dominant hand. The screen then displayed the target path at the top of the screen in red, the path of the user's current focal node (initially set to the root node at the start of each trial) immediately below in black, and the tree representation in the remaining screen space, using either expand-ahead or not (Figure 4.7). Users then clicked on nodes to travel down the desired path until they reached the target leaf, which ended the trial.

The reason the target path was shown to users before the start of each trial was to give users a chance to read the path and better retain it in short-term memory during the trial. This should reduce whatever variance there might be in the recorded times due to re-reading the target path during the trial. Forcing users to place their cursor in the start box also reduced variance, ensuring that users always started in the same initial position.

Errors were not allowed during trials. If the user clicked on a node not along the

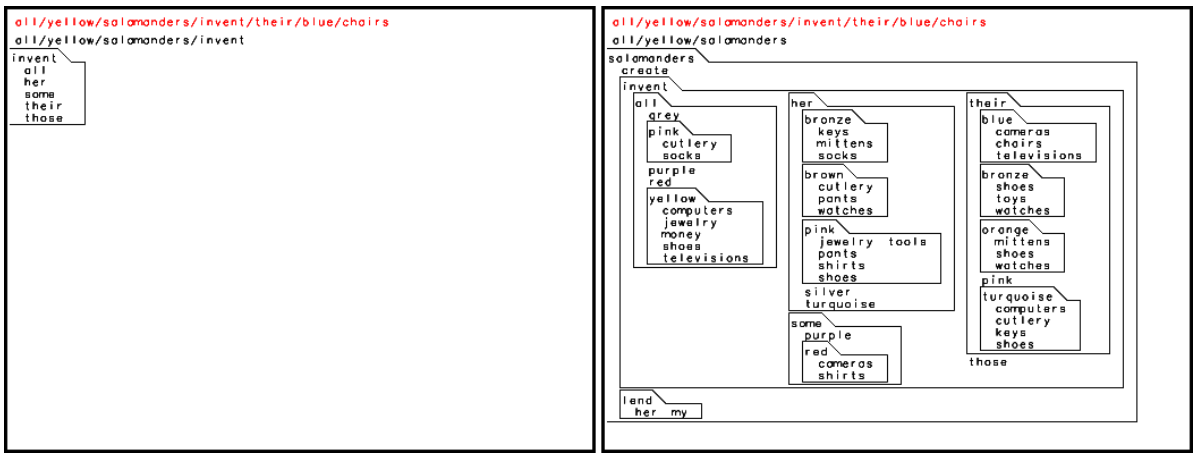

Figure 4.7: The information displayed in the middle of a trial, after the user has clicked a few times, without expand-ahead (Left) and with 2D expand-ahead (Right). The target path is shown in red at the top of the screen, with the current path shown immediately below.

target path, the computer emitted an audible beep without changing the focal node, and the user was forced to continue the trial until successful completion. Thus, in some sense, the total time for the trial incorporates the cost of errors. Forcing the user to successfully complete each trial, even after an error, has the advantage that there is no incentive (even subconsciously) for the user to go faster by committing errors and terminating trials early.

4.8.5 Conditions

Trials were performed under 3 main conditions: no expand-ahead, 1D expand-ahead, and 2D expand-ahead. Within each main condition, two different trees were used during trials, to test two different ranges of branching factors. Finally, within each main condition, and within each of the two trees, users performed 3 different kinds of drill-down tasks. These were: traversing a different, random path for each trial; traversing the same path repeatedly over many trials; and traversing the same path repeatedly over many trials, but perturbing the tree slightly before each trial. These 3 drill-down tasks were chosen to test performance with unpracticed paths, practiced paths, and practiced paths with

perturbation, respectively.

The order of presentation of the 3 main conditions was counterbalanced with a latin square design, and the order of presentation of the two trees was also balanced over the 12 users.

Both trees were of depth 7, and were structured such that the path of a leaf would spell out a coherent 7-word sentence of the form *quantifier colour animal verb possessive-pronoun colour noun*, such as "all yellow salamanders invent their blue chairs". Structuring the levels of the trees this way made it easy to generate the trees offline, programmatically, with a given desired branching factor, and also with some random variety in the children of each node. Although many of the paths form fanciful sounding sentences, these are easier for users to remember during trials than a random string of characters would be. Note that the children of each node were always ordered alphabetically, to better enable users to use a subdividing visual search strategy during trials.

The first tree had internal nodes whose branching factor varied uniformly between 2 and 5. The second tree was bushier, with a branching factor varying uniformly between 8 and 11. These values were chosen because they are close to the two extreme branching factors for our conditions, without being too extreme. A constant branching factor of just 1 would give too great an advantage to expand-ahead (which would be able to expand the tree all the way to its single leaf), and a branching factor much greater than 10 would mean that, for the font and screen size used, expand-ahead would usually revert back to the status quo of no automatic expansion.

In the first drill-down task, users were given a different random path for each of 10 trials. In the second drill-down task, users were given the same path for 5 trials, and then a second path for another 5 trials. In the third drill-down task, users were again given 2 paths for 5 trials each, however in this case the tree was perturbed slightly before each trial, by swapping random subtrees at various levels, causing a corresponding change in the computed layout of the tree, and a change in which nodes would be expanded by the

ExpandAhead algorithm.

In summary, the whole experiment involved

12 participants

$\times$ 3 main conditions (no expand-ahead, 1D expand-ahead, 2D expand-ahead)

$\times$ 2 trees

$\times$ 3 drill-down tasks

$\times$ 10 trials

= 2160 trials in total

4.8.6 Results and Discussion

We broke down the measured data into 3 subsets, corresponding to each of the 3 drill-down tasks, and examined the effect of various factors on the recorded times in each subset.

Analysis of variance (ANOVA) showed that the participant had a significant ($F > 30$, $p < 0.0001$ for each of the 3 tasks) effect on the time to complete each trial. The average time for each participant varied roughly evenly between 10.1 seconds for the fastest user, and 19.8 seconds for the slowest user. This large variance could have been in part due to the range of ages of users, and the apparently different levels of fatigue under which each user performed the experiment. For example, the slowest user reported feeling sleepy during the experiment.

The two trees used also had an effect on performance. Within each of the 3 tasks, the relatively skinny tree, with branching factor 2–5, afforded significantly ($F > 70$, $p < 0.0001$) faster performance than did the bushier tree with branching factor 8–11. This makes sense for the non-expand-ahead condition, since a larger branching factor requires the user to travel farther on average for each click, and also makes sense in the expand-ahead conditions, since expand-ahead can expand skinny trees more deeply, on average.

Within each task, the main condition had a significant effect on time ($F = 7.5$, $p < 0.0006$; $F = 14.7$, $p < 0.0001$; and $F = 3.3$, $p < 0.0362$ for the 3 tasks, respectively). Following are the average times for each of the tasks, broken down by main condition. Stars appear beside times significantly different from the other times in the same task, as determined by a multiple means comparison.

For unpracticed, random paths:

Main Condition	Time (seconds)
no expand-ahead	14.541
1D expand-ahead	15.930 * (significantly worse)
2D expand-ahead	14.912

For practiced, repeated paths:

Main Condition	Time (seconds)
no expand-ahead	13.006
1D expand-ahead	13.085
2D expand-ahead	11.361 * (significantly better)

For practiced, repeated paths, with perturbation:

Main Condition	Time (seconds)
no expand-ahead	13.149 * (significantly better)
1D expand-ahead	13.883
2D expand-ahead	14.005

As seen by the above tables, in the 1st task, with unpracticed, random paths, performance with 2D expand-ahead was not significantly different from that with no expand-ahead. In the 2nd task, with practiced paths and no perturbation, 2D expand-ahead was significantly faster than having no expand-ahead, by approximately 12.7%. In the 3rd task, with perturbation, 2D expand-ahead was significantly slower than having no expand-ahead, by approximately 6.5%.

These results suggest that, for practiced paths in absence of perturbation or rearrangement of nodes, users are able to quickly target the desired nodes along the path, probably by memorizing their location, and reach the leaf node faster with expand-ahead than without, by skipping over the levels expanded for them. With perturbation, however, users were slower in the 2D expand-ahead case than without expand-ahead, implying that the time T_F to find each next node increased enough to outweigh the benefit of having fewer clicks to perform.

These results are not so surprising in light of the expected tradeoffs that usually accompany adaptive user interfaces: they can help performance in some situations, but also hinder it if the user does not find items in their expected place. It is encouraging to note, however, that although 2D expand-ahead was 6.5% slower than no expand-ahead in the perturbed tree case, it was faster by 12.7%, or about twice as much, in the un-perturbed case.

Furthermore, a few aspects of our experiment may have artificially biased the results against expand-ahead. In real situations, changes made to the tree's structure are often made by the user themself, e.g. adding or deleting portions of their own file structure, rather than imposed by the system through a randomized perturbation. At least one user remarked after the experiment that she felt expand-ahead would have been easier to use if she had built up the tree herself and been familiar with its contents, rather than browsing a tree never seen before. Also, in practice, changes to a tree such as a user's file system are not as frequent as the perturbations in our experiment were. Infrequent changes to the tree's structure would be more conducive to habituation by the user.

Finally, our experiment only tested performance at drilling-down a path, and leaves open the question of whether expand-ahead benefits more general browsing tasks. For example, expand-ahead not only allows a user to skip over levels that have been automatically expanded, it also allows the user to see deeper down a tree. This means the user may notice more information and discover nodes that they would not have otherwise

travelled down to.

4.9 Design Issues and Potential Enhancements

This section describes various enhancements that could be explored in future design work.

4.9.1 Sticky or Hard Expansion, versus Soft Expansion

In our prototypes, the user only selects the current focal node F, and the ExpandAhead algorithm determines which nodes to expand under F. This behaviour could be made more general by instead allowing for two types of expansion: sticky, or hard expansion, that is controlled by the user; and soft expansion, that is set by the ExpandAhead algorithm. The user would be able to explicitly expand one or more nodes, leaving them in a forced expanded state. The ExpandAhead algorithm would then allocate screen space for these nodes, and only expand other nodes if there remains more screen space. Such behaviour would allow the user to effectively create multiple points of focus, by hard-expanding each node of interest, after which the ExpandAhead algorithm would fill up any remaining screen space with automatic, soft expansion.

4.9.2 Locking Node Positions for Persistent Layout

One of the participants in our experiment said she would like the ability to customize which levels of the tree she sees expanded together, and to always see the levels that way. Features that allow the user to manually position or "lock down" the relative placement of nodes would help alleviate the detrimental effects of rearrangement and allow for better landmarking and more consistent displays, thus reducing the time necessary to visually scan for nodes. The system could, for example, allow the user to lock down certain nodes of particular interest, while other nodes are free to flow around them.

4.9.3 Uniform Expand-Ahead and Partial Expansion

As mentioned in section 4.2, SpaceTrees [Plaisant *et al.*, 2002] implement a kind of expand-ahead, but where levels are only expanded if they can be expanded completely. This has the disadvantage that screen space cannot be filled as completely, but also means that node expansion occurs in a much more regular and uniform way. Such *uniform expand-ahead*, which only expands entire levels under the focal node, reveals each possible path down from the focal node to the same depth. This may result in displays that are easier for the user to understand.

Another possibility not yet explored in our prototypes is that of partial expansion of nodes, whereby a node might be expanded to show some of its children, giving the user a partial preview of its contents, but also show some indication of elision (perhaps similar to Lee and Bederson's [2003b; 2003a] ellipsis nodes) if there are other children not shown. If f is the fraction of children that are shown in a partial expansion, f could be chosen to be proportional to $w(n)$, again allowing for heuristics to guide the expansion.

Combining uniform expand-ahead with partial expansion (i.e. whereby all nodes on a level would be each partially expanded) might enable more efficient filling of screen space without sacrificing the regular and uniform treatment of nodes on the same level.

4.9.4 Improvements in Graphic Design

Following our experimental evaluation, we also investigated various changes to the graphic design of our tree representations, in an attempt to make them easier to visually interpret (Figure 4.8). This direction could be further pursued.

4.10 Closing Remarks

This chapter has presented a general model for automatically expanding nodes to fill screen space. The expand-ahead model can be applied to many different representa-

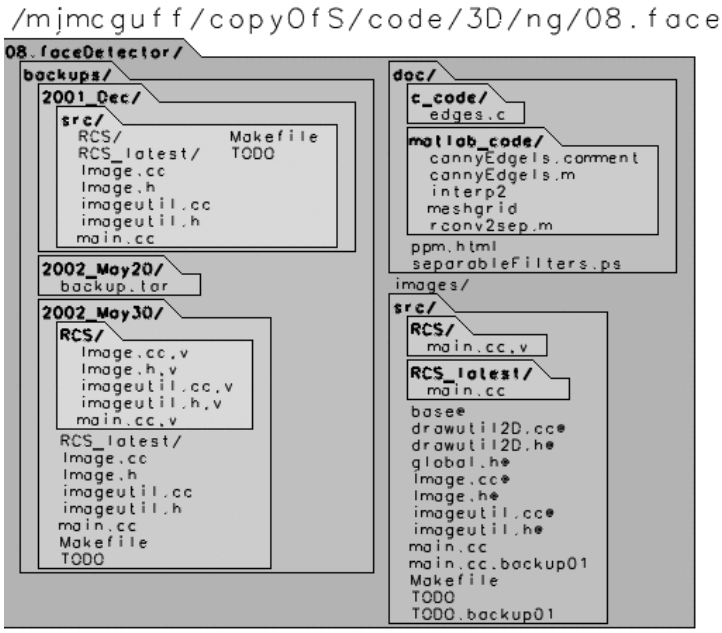

Figure 4.8: Experimental enhancements to the graphic design of our 2D browser. To make information easier to parse quickly, nodes are filled with a shade of grey indicating depth, and labels of expanded nodes are shown in bold.

tions of trees, including node-link representations and nested containment representations. A pseudocode algorithm for implementing this model was given, which takes a heuristic weighting function $w(n)$ as a parameter to guide the expansion according to a client-chosen policy. We have also given an approximate model of user performance with expand-ahead, presented two prototype implementations, and reported experimental evidence that expand-ahead can improve performance during a drill-down task under appropriate conditions.

The case study in the next chapter contains yet another approach to our design goals, after which all three studies are compared and analyzed.

Chapter 5

Interactive Visualization of Genealogical Graphs

This chapter considers the general problem of visualizing "family trees", or genealogical graphs, in 2D. A graph theoretic analysis is given, which identifies why genealogical graphs can be difficult to draw. This motivates some novel graphical representations, including one based on a *dual-tree*, a subgraph formed by the union of two trees. Dual-trees can be drawn in various styles, including an indented outline style, and allow users to browse general multitrees in addition to genealogical graphs, by transitioning between different dual-tree views. A software prototype for such browsing is described, that supports smoothly animated transitions, automatic camera framing, rotation of subtrees, and a novel interaction technique for expanding or collapsing subtrees to any depth with a single mouse drag.

5.1 Introduction

Genealogy, the study of "family trees", plays a significant role in history (e.g. of royal families, and of human migration), genetics, evolutionary biology, and in some cases, religion. It also shows no sign of waning as a hobby of the public, especially given new

software tools, databases, and means of communication and sharing made available by the internet.

Unfortunately, the depiction of relationships in a large family is challenging, as is generally the case with large graphs. The diagram in Figure 5.1, for example, contains many long edges, and doesn't clearly show which nodes are all in the same generation. Although there are a few hundred nodes in the diagram, these are organized around just a few lineages and nuclear families — many lines of ancestry and descent have been omitted. In addition, family trees (or genealogical graphs, as we will call them) are not arbitrary or unconstrained graphs — they have special structural properties that can be exploited for the purposes of drawing and interactive visualization. Interestingly, other than Furnas and Zacks [1994], we have been unable to find previous work in the mathematical, graph theory, or graph drawing communities that analyzes the graph theoretic properties of genealogical graphs.

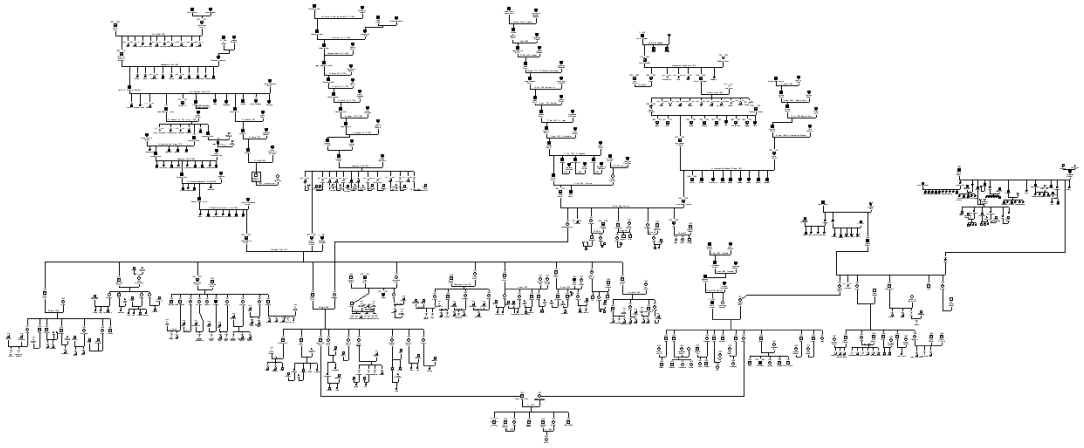

Figure 5.1: Portion of a genealogical graph for an actual family, laid out manually, containing well over 600 individuals and spanning almost 400 years. (Sample data set supplied with GenoPro [GenoPro Inc., 2005]).

Although genealogical graphs are often referred to as family *trees*, this is misleading. Every individual has a tree of ancestors (sometimes called a *pedigree*), as well as a tree of descendants (Figure 5.2, left), each of which can be drawn in familiar and easily understood ways. A drawing of both of these trees is sometimes called an *hourglass* chart

in the genealogical community, and has been called a *centrifugal view* [Furnas and Zacks, 1994] in the literature. (It is also similar to [Wesson *et al.*, 2004].) Hourglass charts only show some information, however. Each ancestor has themself a tree of descendants, and each descendant has a tree of ancestors (each of whom has a tree of descendants, etc.). It is not uncommon for users to experience frustration with diagramming software, where the user must repeatedly and manually move increasingly large subsets of nodes to create room for new data. It is also not obvious that the underlying structure is best described as a topological tree. Finally, trying to automatically draw such graphs leads to problems and design tradeoffs.

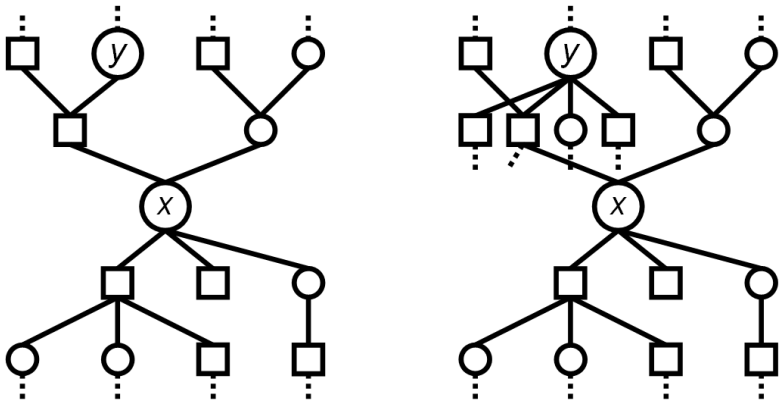

Figure 5.2: *Left:* Node x has a tree of ancestors (parents, grandparents, etc.) and a tree of descendants (children, grandchildren, etc.), both of which may be drawn with conventional tree-drawing techniques. *Right:* It is more challenging, however, to also show the descendants of y, or worse still, to show the descendants of *every* ancestor of x, and the ancestors of *every* descendant of x. Note: in this and other figures, squares represent males, circles females.

We present a brief analysis of genealogical graphs and identify how and why it is difficult to draw them. This motivates an investigation of alternative graphical depictions, leading to the development of a *dual-tree* scheme that generalizes hourglass charts, and that may be used for visualizing any multitree [Furnas and Zacks, 1994]. We describe a software prototype that implements this scheme, that supports smoothly-animated rotations and transitions between dual-trees, and that uses a novel interaction technique for expanding or collapsing subtrees to any depth with a single mouse drag. Although

this work is geared toward genealogy, some of the design principles and techniques used are also applicable in other domains.

5.1.1 Approach to Pursuing Design Goals

In the previous case studies, our approach to achieving the 1st design goal, of increasing and improving output, were to create techniques to address fairly general problems with output: overutilization of display space in the case of occlusion in 3D, and underutilization in the case of visualizing a structure laid out in 2D. The approaches were informed by the particular kind of data involved (volumetric data and rooted trees, respectively), but could conceivably be applied to other data embedded in 3D or 2D, respectively.

In this chapter, our approach starts in a much more data-centric manner. Rather than trying to conceive of a general solution to the problem of, say, edge-crossings in output, we perform a careful analysis of the properties of the data to visualize, and then design visualization techniques that take into account these properties, to avoid problems in the output such as edge crossings, wasted space, and poor aspect ratio; i.e. to improve the output. This is arguable a more applied way of designing a visualization, however we feel that in the end, there are still some general lessons to take away from the case study, related to choice of subset to display, and use of constraints in structure to facilitate display and interaction.

Similar to the previous case studies, the 2nd design goal of easing input is addressed by using popup widgets. To make use of the display space for input as well as output, each node in the output can be selected and have either a marking menu or a new "subtree-drag-out" widget invoked over it (see section 5.6.1). Because many nodes can change position during state transitions, we again implement animations to address the 3rd goal of using smooth transitions. Furthermore, the subtree-drag-out widget can be used to traverse many states in a single drag, somewhat like the popup dial widget of Chapter 4, however the present case study pushes this idea further by allowing the user

to traverse a different sequence of states for each node, rather than just traversing the single sequence defined by history.

5.2 Background

Genealogical relationships have been recorded and depicted for centuries, however the traditional charts appearing in books tend to be simple, usually showing at most a few dozen individuals, and are often organized around simple patterns such as lineages (e.g. one's father, paternal grandfather, etc.), or a single tree of ancestors, or a single tree of descendants. Commercial software packages enable the compilation of datasets with hundreds to thousands of individuals, but are not designed to automatically visualize such large data sets. They either require the user to arrange data manually, or have automatic layout algorithms that only operate on a subset of the data or that don't work well in all cases.

Yet, there is a significant demand for automatic visualization of data. The documentation for [GenoPro Inc., 2005] states "GenoPro wrote the AutoArrange routine to import Gedcom files, but noticed many are using the AutoArrange to layout their genealogy tree. This routine took several months to write, debug and test, yet generated more emails than all the other features combined. About 95% of all the genealogy trees GenoPro received by email were AutoArranged."

In addition, whether automatically generated or not, conventional charts of large, extended families inevitably contain at least some long edges or nodes displaced far away from their close relatives, to make room for other nodes (e.g. Figures 5.1 and 5.5). Thus, even given a robust automatic layout algorithm, it is not clear that displaying *entire* genealogical graphs of thousands of nodes would be ideal, since numerous long edges or edge crossings would make navigation and interpretation difficult.[1] A better solution may

[1] One anecdote concerning a family reunion recounts how participants exceeded the area of four picnic tables in trying to layout their genealogical information. Another story reports the existence of a single

be to display subgraphs that are automatically laid out, and allow the user to flexibly transition between subgraphs.

Bertin [1967] mentions an elegant way of drawing genealogical graphs, where each individual is a single line segment (thick for men, thin for women) and where nuclear families are points. Each line segment may connect two nuclear families: one in which the individual is a parent, and one in which they are a child (this is similar to p-graphs [White and Jorion, 1992]). Although such diagrams are much simpler looking than traditional ones, they ultimately suffer from the same exponential crowding (see § 5.3.4).

Ted Nelson has proposed zzstructures (the generic name for ZigZag®) as a general structure for storing information. It has been shown [McGuffin and schraefel, 2004] that zzstructures are equivalent to a kind of directed graph. Nelson has demonstrated that genealogical graphs can be encoded within zzstructures, using the scheme in Figure 5.4, D. The choice of this scheme, however, is due more to its compatibility with typical zzstructure visualizations, rather than due to an inherent appropriateness for genealogical graphs. For example, many visualizations of zzstructures are based on a 2D cursor centric view (described in [McGuffin and schraefel, 2004]), which can show one nuclear family at a focal point (parents and children arranged along perpendicular directions), surrounded by some extended family nodes. Unfortunately, such visualizations make it difficult to see which nodes are all within the same generation.

Multitrees [Furnas and Zacks, 1994] are a kind of directed acyclic graph (DAG) where any two nodes are either connected by zero or 1 directed paths. In other words, multitrees are diamond-free DAGs, where a *diamond* is a pair of distinct directed paths from one node to another node. As a consequence, every node x in a multitree has a tree $D(x)$ of descendants and a tree $A(x)$ of ancestors (Figure 5.3). Furthermore, the trees in a multitree can overlap: given nodes x and y in a multitree, $D(x)$ and $D(y)$ may share one or more subtrees, and if not, then $A(x)$ and $A(y)$ may share one or more subtrees. Furnas

data set containing 30000 interconnected individuals.

and Zacks [1994] explain how genealogical graphs constructed according to Figure 5.4, C can correspond to multitrees, if there is no intermarriage (i.e. diamonds). They also propose two visualization techniques for multitrees: a centrifugal view (essentially Figure 5.2, left) and a view of a directed path ("lineage") between two nodes along with children and parents of the path [Furnas and Zacks, 1994].

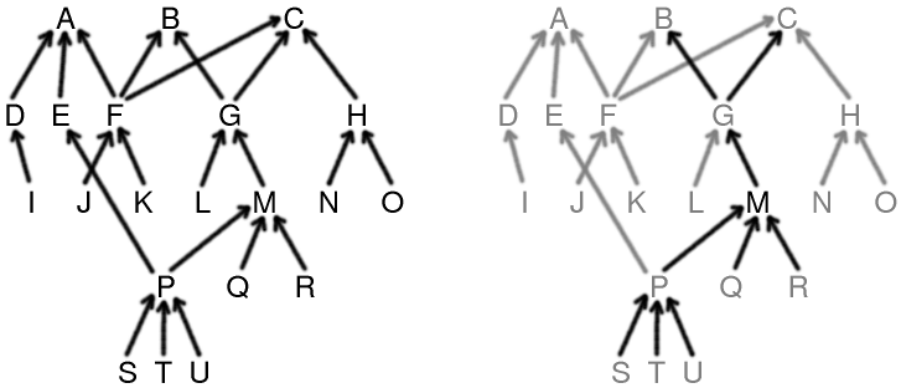

Figure 5.3: *Left:* an example multitree. Observe that the two trees of descendants rooted at nodes A and C, respectively, share two subtrees, rooted at nodes F and P, respectively. *Right:* Node M is highlighted, along with its tree of ancestors and tree of descendants.

Anthropologists have studied systems of kinship, examining, for example, how family structures and terminology for describing one's kin vary across cultures, and how these relate to genealogy (e.g. [Read, 2000]). The current work focuses instead on issues relevant to graph drawing and visualization.

Our research differs from the previous work by analyzing in more detail some of the properties specific to genealogical graphs, and by proposing some novel graphical depictions of them. In particular, our dual-tree scheme generalizes the Furnas-Zacks centrifugal view/hourglass chart, and also generalizes the "lineage" view of the same authors [Furnas and Zacks, 1994]. We investigate novel ways of displaying and interacting with dual-trees.

5.3 Analysis of Genealogical Graphs

In the following, some of the observations and concepts generalize to various non-traditional family arrangements, such as individuals having multiple spouses, or having more than two parents (e.g. adoptive in addition to biological). However, a traditional family model is a useful one to keep in mind, at least initially. Also, for convenience, the word "marriage" is used in a loose sense, to refer to the relationship between the parents of one or more children.

Some in the genealogical community [Hoffman, 1999] have called for the ability to encode richer information and more kinds of relationships, e.g. foster children, family friends, etc. Increased freedom in a genealogical system would make it approach a general hypermedia system, with a correspondingly general interface. However, we have found that the constraints imposed by first following a traditional family model inspire interesting design and visualization possibilities. Future work may possibly extend or adapt our designs to include more kinds of family relationships.

5.3.1 Preliminaries

We first establish some terminology to describe relationships between individuals. Beyond the familiar relationships of *parent*, *child*, *ancestor*, and *descendant*, we also consider *consanguine* relatives, i.e. individuals with a common ancestor (also called "blood relatives") such as siblings and cousins. In addition, we define *conjugal* relatives as individuals connected by an undirected path through one or more marriages. For example, brothers-in-law are conjugal relatives, as would be x and any of x's spouse's consanguine relatives.

Cousins are consanguine relatives whose most recent common ancestor occurs at n generations prior to the cousins, and in which case the cousins are $(n-1)$th cousins (i.e. 1st cousins if they share a grandparent, 2nd cousins if they share a great-grandparent,

etc.). Note that the cousin relationship is not transitive: individual x may have a cousin y on x's maternal side, and another cousin z on x's paternal side, however y and z are not, generally, cousins, though they are related conjugally through the marriage of x's parents. More generally, consanguine relationships are not transitive, but conjugal relationships are, since our definition allows them to pass through multiple marriages.

Finally, we use the term *nuclear family* to refer to (normally two) parents and their children.

5.3.2 Intermarriage and Pedigree Collapse

Intermarriage corresponds to an *undirected cycle* (i.e. a cycle in the underlying undirected graph) in a genealogical graph. We distinguish between two kinds of intermarriage: *Type 1* intermarriage is between consanguine spouses, e.g. spouses who are also (possibly distant) cousins. *Type 2* intermarriage is between spouses who are conjugal relatives via a path going through one or more marriages other than their own marriage. Examples of type 2 intermarriage include two sisters (or cousins) from one family marrying two brothers (or cousins) from another family not initially related to the first family. In the graphs we consider, all marriages are modelled — even those that are eventually dissolved. Thus, if a woman divorces a man x and marries his brother y, this constitutes type 2 intermarriage, because the woman was already conjugally related to y through her first marriage to x.

Assuming that the ancestry of an individual x is free of type 1 intermarriage, then x has 2^n ancestors at the nth generation prior to x. At a conservative 30 years per generation, this exponential number of ancestors exceeds the physical capacity of the earth at less than 2000 years into the past. We can therefore conclude that the ancestry of x must contain type 1 intermarriage. The phenomenon of encountering type 1 intermarriage in *every* individual's ancestry, when traced back far enough, is called *pedigree collapse* [Shoumatoff, 1985].

In addition, statistical modelling suggests that all humans alive today share a (not necessarily unique) common ancestor who lived just a few thousand years ago [Rohde *et al.*, 2004], implying that all living humans are "blood relatives".

Pedigree collapse guarantees that type 1 intermarriage occurs in every real-life genealogical graph, if extended back far enough in time. The presence of such diamonds in one's "tree" of ancestors obviously creates problems for drawing such a graph. Fortunately, many genealogical data sets are free of intermarriage because they do not extend back far enough in time, and in any case are usually locally free of intermarriage. Furthermore, algorithms and visualization techniques designed for acyclic graphs may be adapted to genealogical graphs containing intermarriage, by creating virtual duplicates of individuals to "hide" the cycles.

5.3.3 Conditions Resulting in Trees, Multitrees, and DAGs

When are genealogical graphs really trees, or multitrees, or neither? This depends on the presence of type 1 and type 2 intermarriage, and on which scheme is used to construct the genealogical graph.

Let G be a genealogical directed graph (digraph) constructed according to one of the schemes B–E in Figure 5.4. If scheme B or C or E is used, then edges are always incident from younger to older nodes, thus G is a DAG. If scheme B or C or E is used, and there is no type 1 intermarriage (which would correspond to a diamond in G), then G is a multitree. If scheme B or D or E is used, and there is no type 1 or type 2 intermarriage, then the underlying undirected graph G' is a free tree (*free tree* is another term for *topological tree*, i.e. an unrooted tree).

In many cases, then, a genealogical graph may be a free tree, or at least a DAG. Trees are planar, and many techniques exist for drawing them with no edge crossings. However, it is often desirable to see the nodes in a genealogical graph ordered by time, to make the generations in the graph apparent. Such an ordering is impossible to achieve

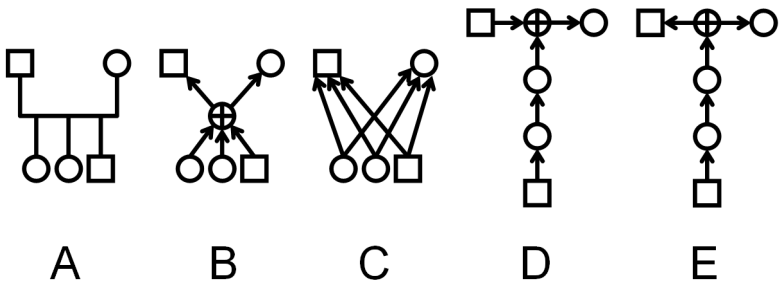

Figure 5.4: *A:* conventional notation for a nuclear family: squares are male, circles female, and children extend downward from an edge connecting the parents. *B–E* show different ways of modelling such a family within a directed graph. *B:* the $\oplus$ symbol denotes a "spousal union" node. *C:* alternative scheme that avoids any special, intermediate node, but requires more edges when there are 3 or more children. *D:* Nelson's scheme for encoding families within zzstructures. Each child links to its next older sibling, and the eldest child links to the "spousal union" node. *E:* a variation on *D* that prevents cycles in the directed graph.

in general without edge crossings. Partially relaxing the ordering by generation, so that each node is only "locally ordered"[2] with respect to its parents and children, allows edge crossings to be eliminated in a free tree. However, long edges are still generally unavoidable (Figure 5.5).

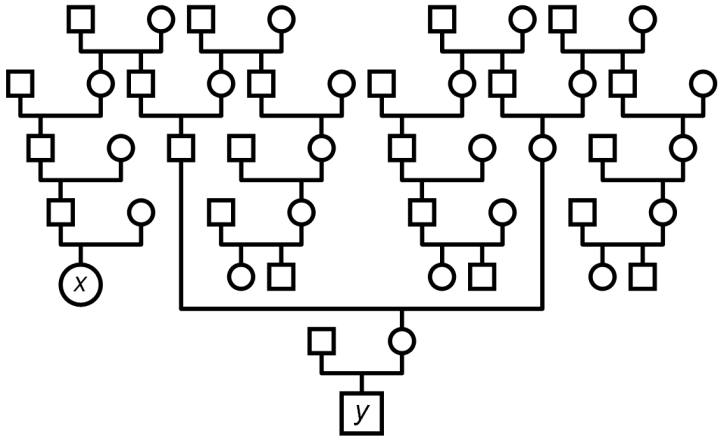

Figure 5.5: Example situation where a long edge cannot be avoided, even if some branches are rotated. Also, the vertical ordering of nodes by generation is broken: it is not immediately apparent that nodes x and y are of the same generation — they are 3rd cousins. The ordering by generation could be restored by introducing edge-crossings, but at least one edge would still be long.

[2]In graph drawing terminology, locally ordered means *upward*, and (globally) ordered by generation means *upward* and *layered* by generation.

DAGs can be drawn automatically using standard algorithms, such as Sugiyama et al.'s [1981]. In this case, however, edge crossings and long edges are both unavoidable, and as with any automated graph drawing technique, the output from a 2D DAG embedder is increasingly difficult to use and understand as the size of the graph becomes very large. It is also possible that new algorithms designed with the specific properties of genealogical graphs in mind may scale better than generic DAG embedders.

The "bushiness" apparent in Figure 5.5 illustrates a core problem in genealogical graphs, of nodes quickly becoming crowded as the graph is extended in various directions. The next section examines and quantifies this problem in more detail.

5.3.4 Crowding Within Genealogical Graphs

We now consider an idealized, simplified genealogical graph G^*, and show that problems arise in trying to draw even this idealized graph. This motivates some non-traditional visual representations.

Let G^* be a genealogical graph, constructed according to Figure 5.4, B, where every node has two parents, one sibling of the opposite gender, one spouse of the opposite gender, and where every marriage produces one child of each gender. Also assume that generations are well-defined, e.g. births are synchronized within each generation. Furthermore, G^* contains no intermarriage, hence the underlying undirected graph is a free tree, and thus G^* is planar.

Assume we want to draw a connected subset of G^* such that nodes are all allocated the same size, and nodes in the same generation have the same vertical coordinate, so that each generation corresponds to a single row of nodes.

Figure 5.6 shows such a drawing, for 9 nuclear families spanning 4 generations. Ellipses indicate the directions in which G^* extends. Intuitively, extending the portion of G^* shown in all directions would require not only crossing edges (to maintain alignment of generations), but also lengthening certain edges to make room for expansion, causing

certain spouses and/or siblings to become distant from each other.

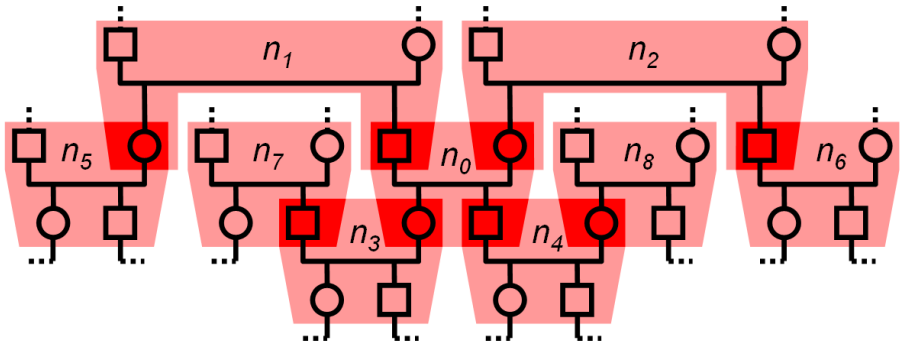

Figure 5.6: A portion of an idealized genealogical graph, G^*. Nine nuclear families are shown (each outlined in pink), labelled $n_0, \ldots, n_8$. Ellipses indicate the many directions in which this diagram could be extended, suggesting that nodes would rapidly become crowded.

To reinforce this intuition, consider the set S of nuclear families at the same generational level as n_0. Figure 5.6 shows $S = \{n_0, n_5, n_6, n_7, n_8, \ldots\}$. Notice that n_0 is connected (via the intermediary families n_1, n_2, n_3, n_4) to 4 other families n_5, n_6, n_7, n_8 in S. Following the ellipses, each of n_5, n_6, n_7, n_8 is connected (again through intermediaries) to 3 other nuclear families in S, each of which is in turn connected to another 3, etc. Even though S corresponds to a single generation of nuclear families, the paths connecting families in S correspond to a free tree, and the number of nuclear families in S that are r edges away from n_0 grows exponentially with r. Similarly, if we consider connections through increasingly distant ancestors, each node has 1 sibling, 4 first cousins, 16 second cousins, and 4^n nth cousins. Unfortunately, these nodes must be fit within a 1-dimensional row, where the space available only grows linearly with the geometric distance from the centre of the diagram. The consequence is that the edge-length-to-node-size ratio becomes arbitrarily high. Figure 5.7 further illustrates.

This is reminiscent of Munzner's [1997] observation that, when embedding a tree in a Euclidean space of any dimensionality, the number of nodes grows exponentially with the level, but the space available only grows geometrically. The case in Figure 5.6 is qualitatively worse, however, because the "exponential crowding" occurs within *each and*

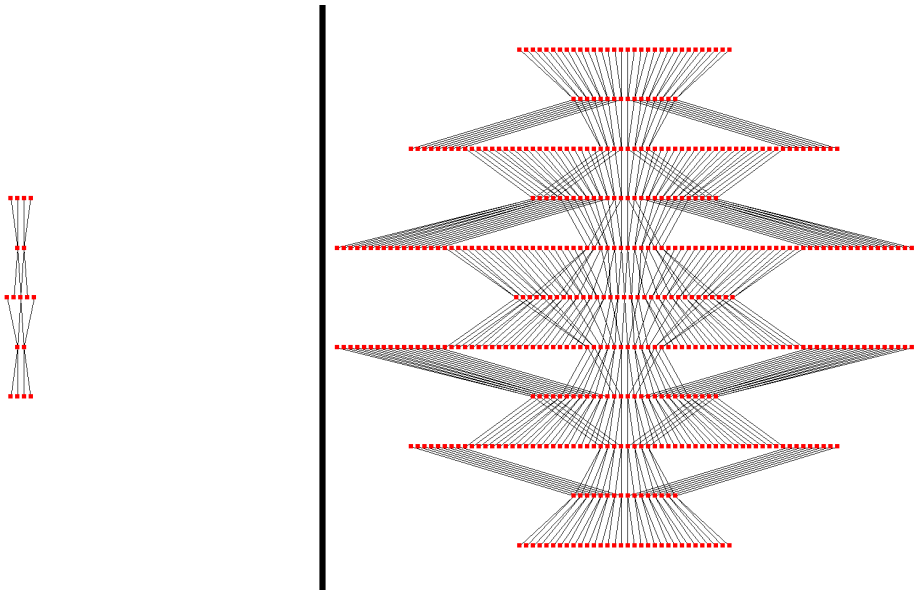

Figure 5.7: Embeddings of neighbourhoods from G^*, of radius 2 (left) and 5 (right). In these diagrams, the nodes are nuclear families, and edges are individuals who participate in two nuclear families (as a parent in one, and a child in the other). Notice the rapid crowding along the horizontal direction, along which nodes are much more closely packed than along the vertical. As the radius of the neigbourhood increases, edges become arbitrarily long, with many crossings, and are slanted at extreme angles. Also, nodes far from the centre of the diagram are embedded increasingly far from their immediate neighbours.

every generation as more and more of G^* is displayed, rather than worsening progressively with deeper levels.

As an aside, consider a more general idealized family modelled with a graph $G_{k,j}$, where every nuclear "family" contains k children and j parents. We have already asserted that, in $G^* = G_{2,2}$, every node has 4^n nth cousins. We leave as an exercise to interested readers to confirm that in $G_{k,j}$, every node has $(k-1)(jk)^n$ nth cousins.

5.4 Some Alternative Graphical Representations

The rapid crowding of nodes that occurs in genealogical graphs inspired us to explore graphical depictions that show different parts of the graph at different scales. By allocating progressively smaller areas to nodes, we might usefully pack more information into a

single representation.

Figure 5.8 shows a fractal layout for G^*. (More generally, such a fractal layout could also be used to depict any free tree.) There is no limit to the extent of the graph that could be drawn this way, however nodes eventually become imperceptibly small. Also notice that this depiction trades away an ordering of nodes by generation to gain non-crossing edges of bounded length.

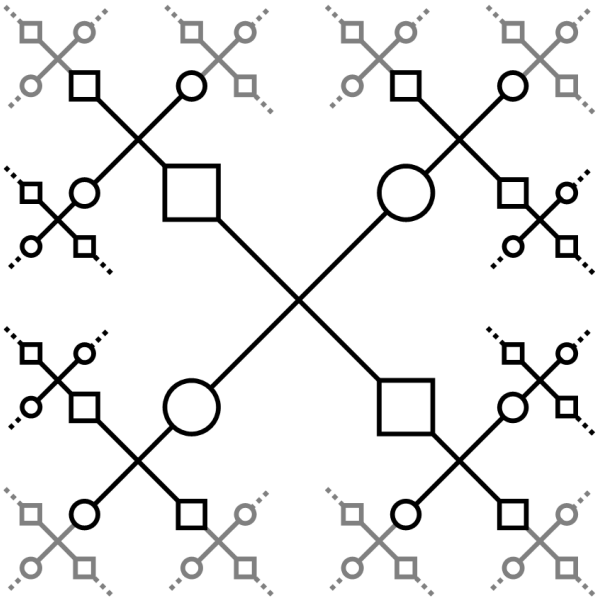

Figure 5.8: A fractal layout for G^*, showing the same 9 nuclear families as in Figure 5.6, along with some additional nodes in grey.

Interactive browsing of the tree in Figure 5.8 could be done by zooming and panning, or by having the user dynamically select the "focal" region that is shown largest in the centre. In the latter case, the resulting interactive visualization might be similar to fisheye graph browsers (e.g. [Munzner, 1997]), though it would differ in the details of how nodes surrounding the focal region are shifted and scaled.

In the process of exploring graphical depictions for genealogical graphs, we found it useful to consider the different ways in which rooted trees are represented. Figure 2.2 shows what we consider to be the most basic styles for drawing rooted trees, 3 of which are identified in [Bertin, 1967; Knuth, 1968]. A familiar example of nested containment

(Figure 2.2, B) are Treemaps [Johnson and Shneiderman, 1991; Shneiderman, 1992b]. The indented outline (Figure 2.2, D) representation may appear to simply be a variation on the node-link (Figure 2.2, A) representation, but in fact the indented outline style would still be unambiguous without any edges drawn: its essential feature is the use of indentation to imply structure. Many variations on the styles in Figure 2.2 have been described in the literature, based, for example, on polar coordinate systems, or on embeddings in 3D rather than 2D, or on combinations of existing styles.

The majority of new tree representations, however, have been applied to rooted trees, whereas free trees are drawn almost exclusively using the node-link style (Figure 2.2, A). Nevertheless, representations based on rooted trees could be applied to free trees, if the user had a way of dynamically choosing a node to serve as a temporary "visual" root. The user would then be able to see the tree from different perspectives, by transitioning from using one node as a root to another. Such interaction might be useful for temporarily and visually highlighting various regions of the free tree.

This idea allowed us to adapt the nested containment style (Figure 2.2, B) to genealogical free trees resulting in a novel representation (Figure 5.9). In general, nested containment representations could be used with any free tree, and thus with any genealogical graph where there is no intermarriage of type 1 or type 2. However, the representation can be simplified if we assume that, in addition to there being no intermarriage, every node participates in at most two nuclear families: one in which they are a child, and one in which they are a parent (in other words, nodes cannot have multiple spouses in different nuclear families). This assumption allows us to omit the "spousal union" $\oplus$ nodes (Figure 5.4, B) and leave these implicit, as we have done in Figure 5.9. In Figure 5.9, lower left and lower right, each individual corresponds to a rectangle, and each rectangle may have one nuclear family nested within it, and also be part of another nuclear family containing the rectangle. Parents appear in the upper half of a rectangle, and children in the lower half. Note that this representation would easily accommodate

the case of nuclear families containing more than 2 parents, by simply subdividing the upper half of rectangles into more than 2 sub-rectangles.

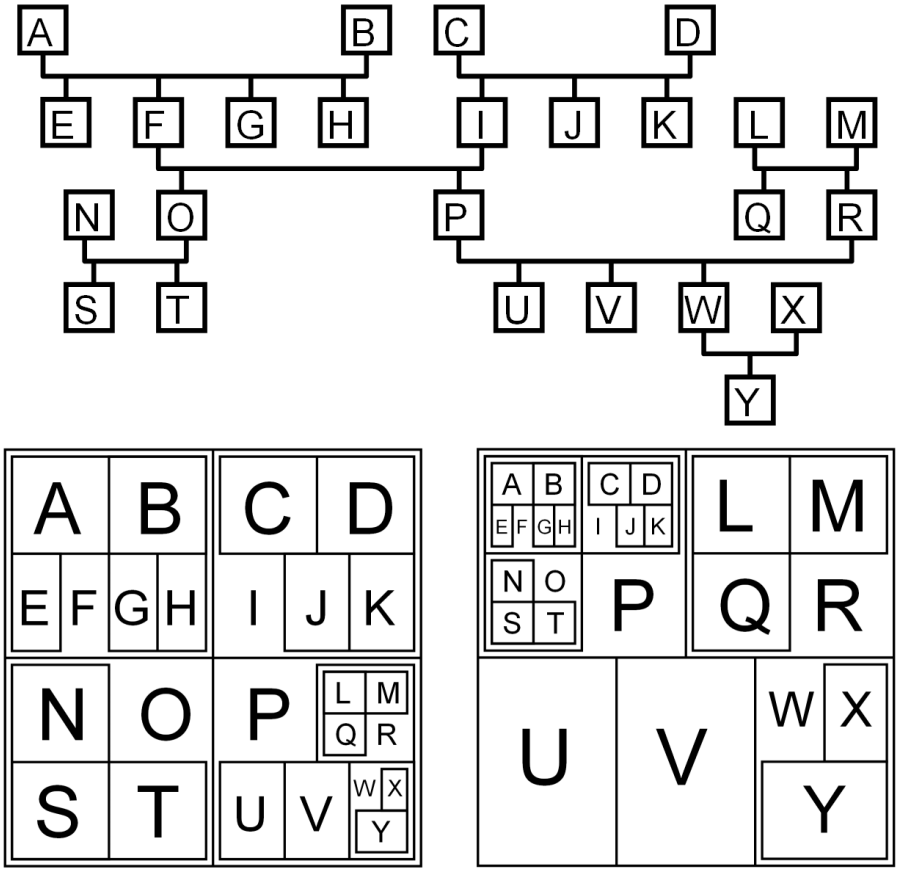

Figure 5.9: A free tree can be drawn using the nested containment style of Figure 2.2, B, if the user's current "focus" is used as a temporary root. Note that in nested containment, the root is the outermost node, that encloses all other nodes. *Top:* a genealogical graph, drawn using conventional notation. For simplicity, squares are used for all individuals, not just males. *Lower left:* the same graph, drawn using nested containment, with the nuclear family $\{F, I, O, P\}$ as the root. This is analogous to the representation in Figure 5.8, with larger nodes *containing* smaller nodes rather than being connected to them with line segments. *Lower right:* now, the nuclear family $\{P, R, U, V, W\}$ is the root.

5.5 Dual-Trees

Although the novel representations in Figures 5.8 and 5.9 are interesting, they do not order nodes by generation. Their unfamiliarity might also make them difficult to interpret

for many users. We now describe a scheme that is closer to traditional diagrams.

The general problem of scaling a visualization to graphs of thousands of nodes, and the added problem of dense crowding in genealogical graphs, convinced us to focus on visualizing only a subset of the graph at a time, and therefore to identify which subset might be best. Some general questions to ask in such a situation are: What are the *canonical*, or standard, subsets of the data that would be familiar to users? Which of these canonical subsets, or *combinations* of them, can be shown at once in a manner than is *easy to interpret* and that *scales well*?

In the case of genealogical graphs, two obvious canonical subsets are trees of descendants and trees of ancestors. As already mentioned, showing both of these at once (Figure 5.2, left; Figure 5.10, A) results in an hourglass chart. To show more information, we propose offsetting the roots of the trees with respect to each other, as in Figure 5.10, B. The result, which we call a *dual-tree*, is a more general kind of union of two rooted trees. (The result can also be thought of as a single free tree, or a "doubly rooted tree", following the observation in [Furnas and Zacks, 1994] that the ancestors and descendants of a directed path in a multitree form a free tree.)

The dual-tree $A(x) \cup D(y)$ contains a superset of the information in an hourglass chart, because $A(x) \supset A(y)$ and $D(y) \supset D(x)$. In an hourglass diagram of $A(x) \cup D(x)$, the choice of x is a tradeoff between the number of ancestors and number of descendants revealed: choosing x in an older generation reveals a larger tree of descendants, but reduces the number of ancestors shown. In contrast, with dual-trees, we can always choose x and y to be in the most recent and oldest generations, respectively, to maximize the coverage of the subset displayed.

Because a dual-tree diagram consists of only 2 trees, it can be drawn in a straightforward manner, and may prove to be easy to understand and interpret. It can be drawn with no edge crossings, with nodes ordered by generation, and it scales relatively well, since the crowding of nodes within it is no worse than the crowding that occurs in

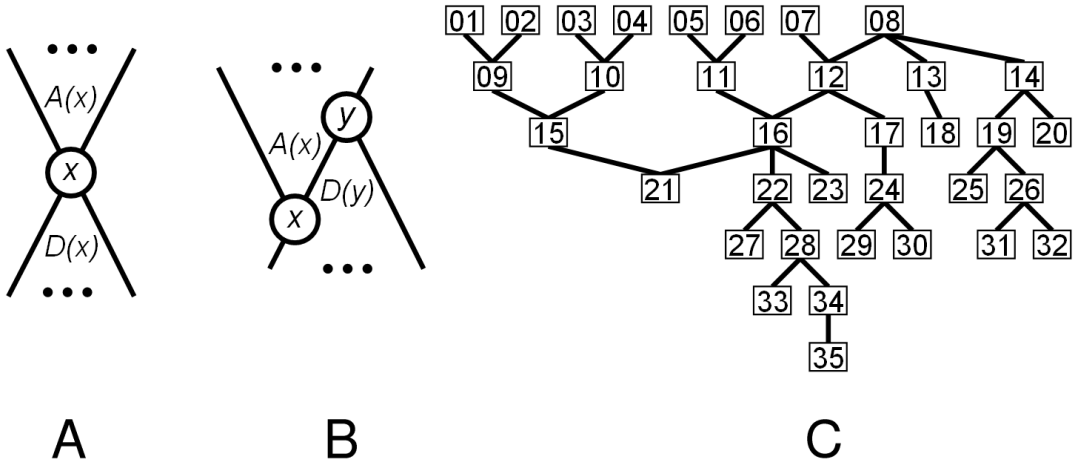

Figure 5.10: Combinations of canonical subsets of genealogical graphs. *A:* The tree $A(x)$ of ancestors and tree $D(x)$ of descendants of x form an hourglass diagram. *B:* This dual-tree scheme shows more information, by showing $D(y) \supset D(x)$. *C:* An example dual-tree, where the tree of descendants is rooted at 08, and the tree of ancestors is rooted at 21. The parents of 21 are 15 and 16, and the children of 16 are 21, 22, and 23; implying that 21 is at least a half-sibling of 22 and of 23.

individual trees.

To combine two trees in the style of Figure 5.10, B and C, the root y of the tree of descendants must be a right-most node in the tree $A(x)$ of ancestors. Likewise, x must be a left-most node of $D(y)$. Thus, changing x or y generally requires rotating subtrees to make the new roots right- and left-most. One scenario in which the dual-tree might be particularly useful is in families where surnames are passed down from the paternal side. In such a family, if y is chosen to be the oldest paternal ancestor of x, then the dual-tree would simultaneously contain every ancestor of x (in $A(x)$), as well as every individual having the same surname as x (in $D(y)$), or alternatively every individual having the same surname as any chosen ancestor of x. We are not aware of any other traditional and scalable depiction of families that can show this. For example, Figure 5.13 shows Tom Smith, his ancestors, and other Smiths in single dual-tree.

Figure 5.10, C is based on the node-link style of drawing trees (Figure 2.2, A). The indented outline style (Figure 2.2, D), however, is often more space-efficient, especially when nodes have long text labels, so we tried to adapt it to dual-trees. Figure 5.11 shows

the steps involved in this. The key to combine the two trees was to use an alternative convention for drawing edges taken from Venolia and Neustaedter [2003], and analogous to the left-child, right-sibling pointer implementation of tree data structures [Cormen *et al.*, 1990]. The result in Figure 5.11, C accommodates long text labels to the right or left of nodes without requiring new whitespace to be introduced between nodes, as would be the case in Figure 5.10, C.

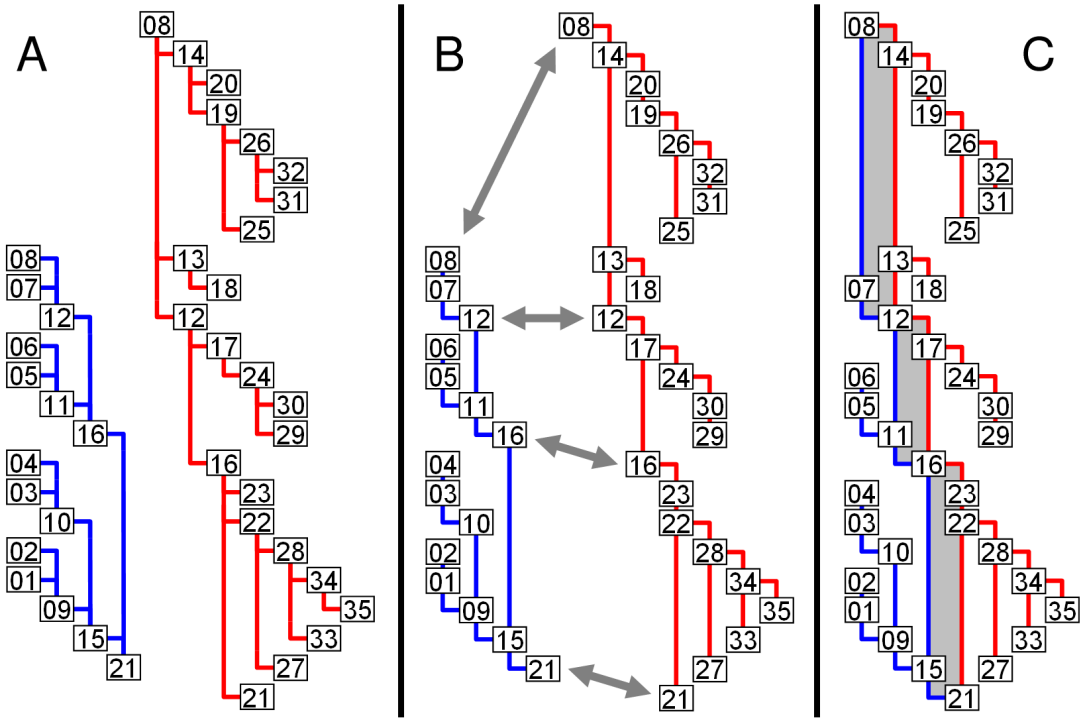

Figure 5.11: Three stages in adapting the indented outline style to dual-trees. The nodes and labels are the same as in Figure 5.10, C. The tree of descendants is rooted at 08, and the tree of ancestors is rooted at 21. *A:* Each tree is drawn in indented outline style. *B:* Edges are drawn in an alternative way, to clear the space between the trees. (Consider for example the tree of descendants drawn in red. In *A*, edges are drawn downward from each parent and then extend horizontally to each child. In *B*, however, edges extend horizontally from each parent and then down *through* children.) Arrows show matching nodes in both trees. *C:* The two trees combined.

Note that dual-trees can be used to browse and visualize any multitree, even if some nodes have multiple spouses, or there is type 2 intermarriage, or some nodes have more than 2 parents. Note also that, in Figures 5.10, C and 5.11, C, nodes in the same generation are clearly shown as such, as they correspond to a single row or column,

respectively.

5.6 Software Prototype for Dual-Trees

To experiment with browsing based on dual-trees, a software prototype was developed, written in C++ using the OpenGL and GLUT libraries, allowing it to run under the Linux and Microsoft Windows operating systems. The prototype reads in a GEDCOM file as input, from which a directed graph is constructed according to Figure 5.4, C.

The digraph is then pre-processed to remove directed cycles and diamonds to obtain a valid multitree. To do this, a breadth-first traversal identifies all undirected cycles in the underlying undirected graph. For each cycle, we count the number of times the edges change direction along the cycle, yielding a non-negative even integer. If the result is zero, we have a directed cycle in the digraph; if the result is 2, we have a diamond; if the result is 4 or more, this may or may not correspond to type 2 intermarriage but in any case is allowed in a multitree. So, if the result is zero or 2, we mark one of the edges involved to be skipped in the embedding algorithm. The display routine, however, can draw these special edges in an alternative colour, to highlight them.

The prototype only displays one dual-tree subset of the graph at a time, but allows the user to interactively transition from subset to subset and browse the entire graph, which might be very large. Each time a new dual-tree subset is chosen, the embedding routine is invoked to determine its layout. Two styles of layout are supported: classical node-link style, and indented outline style. Regardless of the style used, the embedding involves two stages: first, computing two preliminary embeddings E_A and E_D of the tree of ancestors and the tree of descendants, respectively, and second, combining E_A and E_D into a final embedding E_F of the dual-tree.

In the case of classical node-link layout, E_A and E_D are computed with an adaptation of the Reingold-Tilford algorithm [Reingold and Tilford, 1981], though a slightly better

implementation would use Buchheim et al.'s [2002] improvements. To combine E_A and E_D and produce E_F, the embedding routine shifts E_D so that it is beside E_A, such that nodes in the same generation are aligned. Next, consider the set of nodes that appear in both trees, which we call the *axis* of the dual-tree, i.e. the path between the two roots. Each node n in the axis has a position p_A given by E_A and a position p_D given by E_D. The final position of n is computed as the weighted average $p_F = (ap_A + dp_D)/(a + d)$ where a and d are the number of ancestors and descendants, respectively, of n. Our rationale for this weighting is that we don't want the change in n's position to result in many edges having an extreme slope; thus, the more edges n has in one of the trees, the closer its final position should be to its position in the preliminary embedding of that tree.

In the case of the indented outline layout, E_A and E_D are computed in a simple recursive bottom-up pass. Next, pairs of consecutive nodes on the axis are "stretched out" so that E_A and E_D match up along the axis, and finally E_F is produced (cf. Figure 5.11, B, C).

In both cases, the time required for the entire embedding process is linear in the number of nodes embedded.

Figure 5.12 shows screenshots of output. The classical node-link layout can be done along two different orientations (Figure 5.12, top left and bottom left) yielding different total areas and aspect ratios. The area of the bounding rectangle for the indented outline layout (Figure 5.12, right) tends to be smaller than that of the other two layout styles, however its aspect ratio also tends to be far from 1. Such an aspect ratio could be an advantage, however, as it could simplify navigation, requiring the user to scroll mainly along just one direction in a zoomed-in 2D view. Figure 5.13 shows the visual design of nodes in more detail.

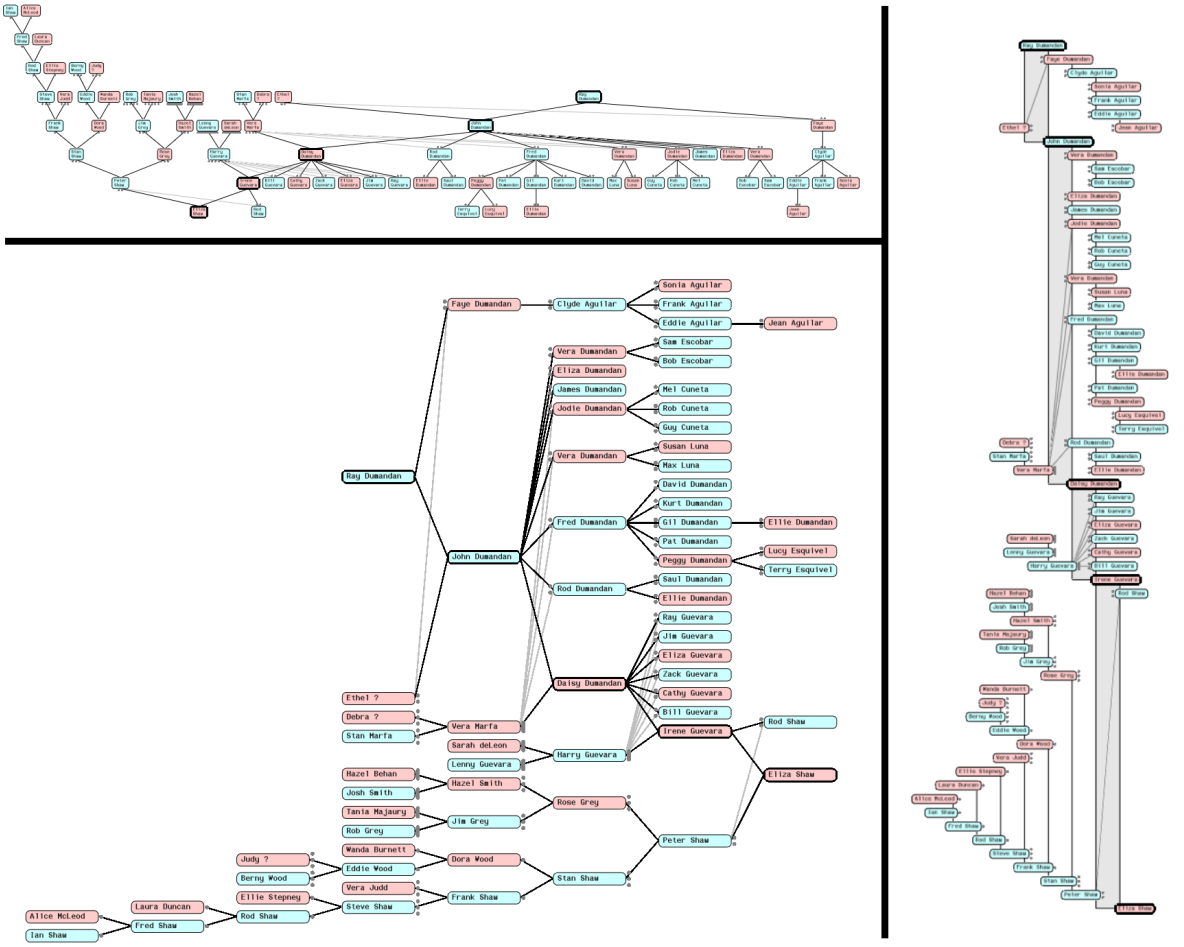

Figure 5.12: A dual-tree laid out 3 different ways by the prototype. Nodes are coloured by gender. *Upper Left:* Classical node-link, similar to Figure 5.10, C, with generations progressing top-to-bottom. *Lower Left:* Classical node-link, with generations progressing left-to-right. *Right:* Indented outline style, similar to Figure 5.11, C.

5.6.1 Interaction Techniques

To transition between different dual-tree subsets, the basic operations performed by the user are: expanding/collapsing parents of a given node, and expanding/collapsing children of a node. These actions can be invoked through a 2-item marking menu [Kurtenbach and Buxton, 1993] affording ballistic "flick" gestures, in the direction of parents or children, to toggle their expansion state. Expanding a node can also cause automatic rotation. For example, if node n is in the tree of descendants, expanding upwards toward its parents requires that n first be rotated onto the axis. Such rotations generally require

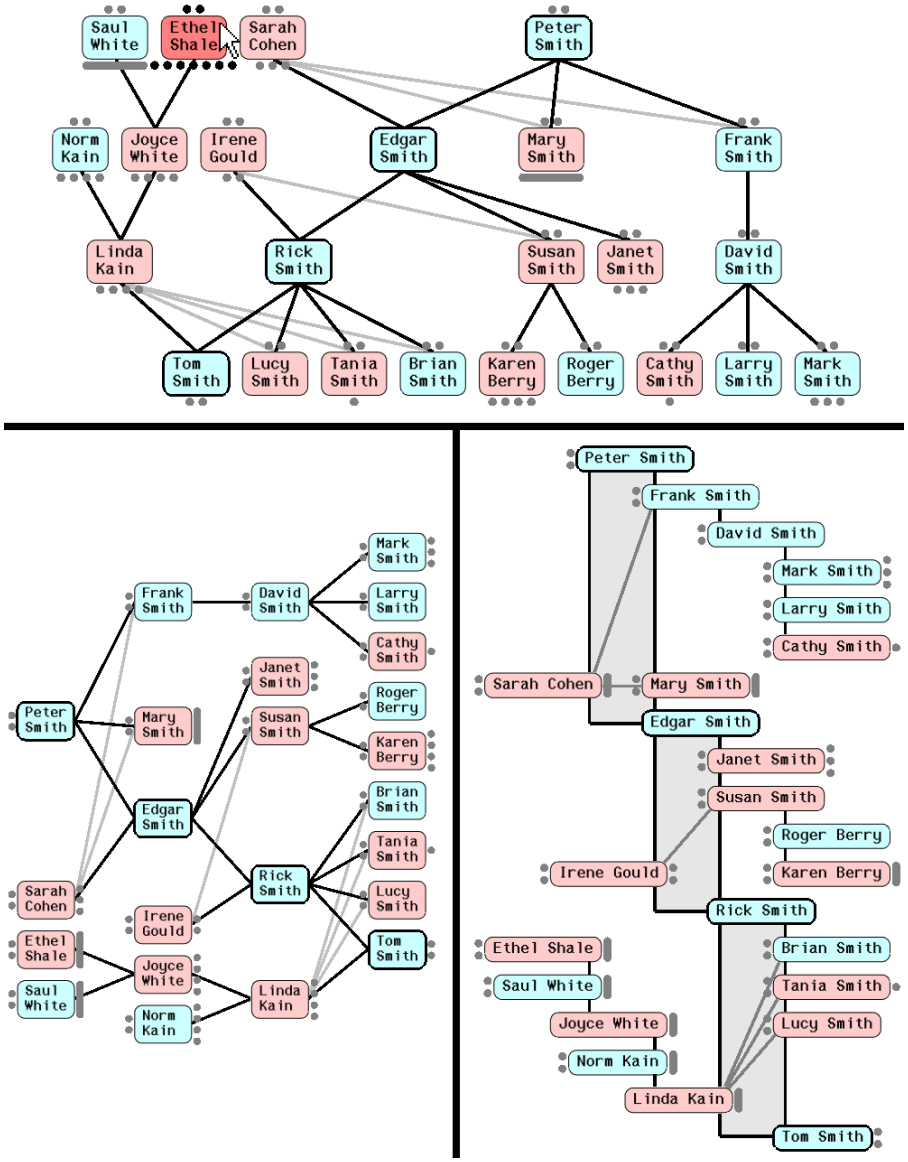

Figure 5.13: Another dual-tree under 3 layouts: classical node-link top-to-bottom (*Top*) and left-to-right (*Lower Left*), and indented outline style (*Lower Right*). Black edges are part of the dual-tree, and nodes with bold borders lie on the *axis*, or path between the two roots "Peter Smith" and "Tom Smith". Additional edges from children to parents are shown in grey, to make nuclear families more apparent. For example, Rick and Susan are siblings, sharing the same parents Irene and Edgar, however Janet is a half-sibling with a different mother, as shown by the lack of a grey edge from her to Irene. Grey dots on either side of a node provide previews of the number of parents or children that the node has. Preview dots that are too numerous are collapsed into oblong shapes (e.g. under Saul and Mary), and shown in full when the cursor rolls over the node, as shown under Ethel (*Top*). Once over a node, the user can reveal hidden parents and children by flicking in the appropriate direction to expand the node.

that certain other nodes be collapsed, to maintain the dual-tree scheme.

Expansion, collapsing, and rotation of nodes is shown with smooth, 1-second animations to help the user maintain context. As in [Plaisant *et al.*, 2002], our animation has 3 stages: fading out nodes which need to be hidden, moving nodes to their new positions, and fading in newly visible nodes. Although many nodes may need to move in different directions during a transition, the user may benefit from tracking even just a few nodes that serve as visual anchors or landmarks. We feel that even a complicated animation is better than no animation at all, and could always be slowed down if the user wishes with a technique such as the dial widget in [McGuffin *et al.*, 2004].

The user may zoom and pan the 2D view of the graph with the mouse, and optionally activate automatic camera framing that is animated during transitions.

In browsing genealogical graphs, we have found it is often desirable to expand downward from an individual to their most recent descendants, or to expand upward to their oldest ancestors. This can be done with the marking menus using a sequence of flicks, with one or more flicks for each generation. However, an even faster method is available through a *subtree-drag-out widget* for "dragging out" subtrees to any depth. To use this widget, the user first clicks down (with a secondary mouse button) on a node (Figure 5.14, A), and then drags either up or down (i.e., toward ancestors or descendants) to select the subtree on which they want the widget to operate. After this initial drag, the length and colouring of the widget (Figure 5.14, B) indicate both the maximum depth of the subtree, and also the depth to which the subtree is currently expanded. The user may drag towards the subtree, to expand it further one level at a time, or away from the subtree, to collapse it one level at a time. In keeping with a metaphor of relative adjustment, the user may also release over the centre of the widget, to dismiss it with no effect, which is useful for cancelling.

After popping up this widget and performing the initial drag to select the subtree to operate on, the user may then drag ballistically to quickly open or close the entire

subtree. Although in general subtrees may be quite large after just a few levels, the trees of descendants and ancestors in typical genealogical data tend to be fairly shallow, seldom spanning more than a few hundred years. Furthermore, even though the user may ballistically expand multiple subtrees upward and downward in quick succession, the automatic rotations that result from expansion often cause other nodes to disappear, thus the user is much less likely to experience an "explosion" in the number of expanded nodes.

5.6.2 Initial User Feedback

As a first step toward evaluating our prototype and informing design changes, an informal session was held to solicit feedback from a practicing genealogist who has also lectured on genealogy. The user reported using computers an average of 2 hours/day, and is familiar with two common genealogy software packages. The session lasted 1 hour, and consisted of a mixture of free-form exploration by the user, demonstration and explanation by the first author, and semi-structured navigation tasks given to and performed by the user.

After some time interacting with the prototype, the user reported finding it "very clear" and "very easy", and was "impressed with its manoeuvrability". (Note that, at the time the session was held, the subtree-drag-out widget had not yet been implemented. The user did, however, discover and successfully operate the marking menus with no help.)

The user also commented that the "unfamiliarity" of the depiction of family relationships "takes getting used to". The user mentioned the lack of a symbol explicitly linking spouses, which is shown in conventional diagrams.

The user successfully completed all navigation tasks, even though these required expanding upward and downward multiple times, and even when using the indented outline style dual-tree. The user was also able to correctly interpret indented outline depictions, pointing out the parents and children in nuclear families.

The user was also shown printouts of sample output from a commercial genealogy software package, and asked for opinions, comments, or personal preferences in comparing the different diagrams and the output of the prototype. The user seemed rather neutral, and so was given an explanation of some potential positive and negative differences between the dual-tree scheme and other representations. The user remained neutral, however, saying "I can understand [each of the depictions]. [...] I don't know that there are any pros or cons."

Of course, more sessions with other users would be necessary to gain a fuller comparative picture, however we are encouraged by the fact that the user was able to interact with and interpret the output of our prototype.

5.7 Closing Remarks

In this case study, we have analyzed the nature of genealogical graphs, characterized how they are difficult to draw, and presented novel graphical representations for them. In particular, our dual-tree scheme scales as well as a single tree, orders nodes by generation with no edge crossings, is easy to interpret, can be used for browsing any multitree, and generalizes both the hourglass chart/centrifugal view of Furnas and Zacks and the "lineage" view of the same authors [Furnas and Zacks, 1994]. Furthermore, our interaction technique for expanding or collapsing subtrees to any depth with a single mouse drag could be used in other domains for general tree browsing, and might possibly be adapted for general graph browsing.

The next chapter looks back on the three case studies to compare them and analyze some of the common issues encountered.

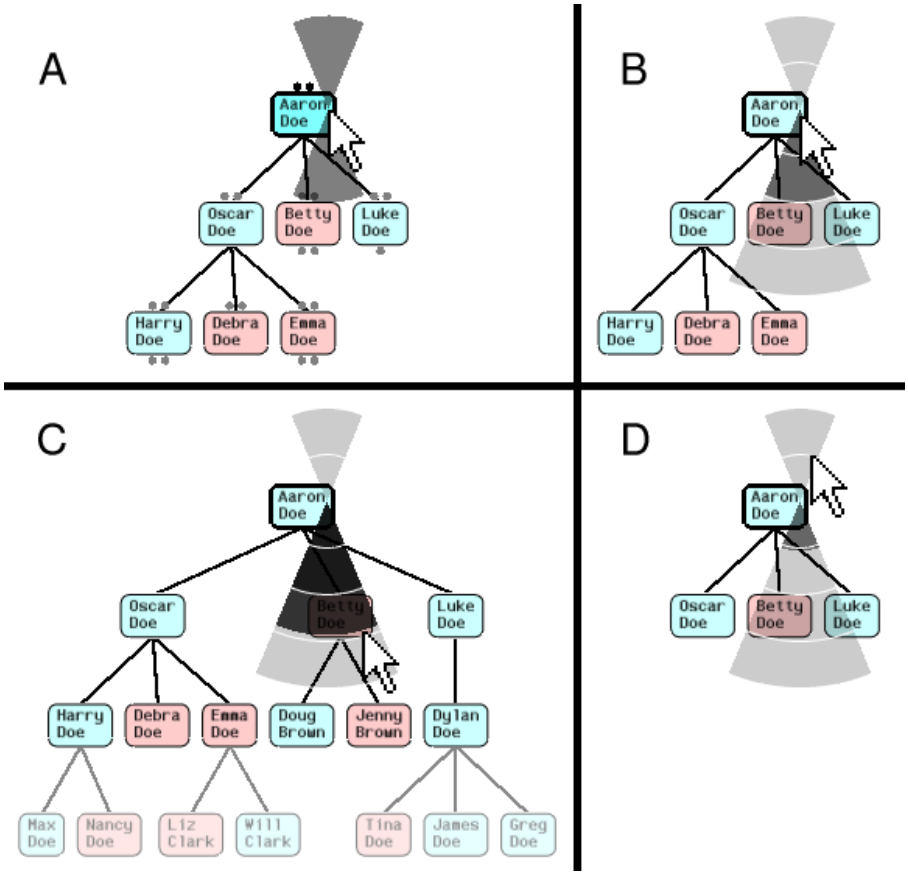

Figure 5.14: A semi-transparent popup widget for expanding or collapsing subtrees in a single drag. The user pops up the widget over a node (A), and may now drag up or down to select whether to operate on the tree of ancestors or descendants. After dragging down slightly, the tree of descendants has been selected (B), and now the widget displays the number of levels the user could drag to change this tree: at most 4 levels down to expand to the full depth, or at most 2 levels up to collapse. Furthermore, the first 2 levels down are shaded in to indicate that they are already partially expanded. C and D show the subsequent feedback after dragging down almost 3 levels, or up 1 level, respectively. Releasing the mouse button completes the operation. The node's tree of ancestors could similarly be expanded or collapsed in a separate invocation of the widget.

Chapter 6

Comparison and Analysis of Case Studies

The preceding three chapters each presented a case study in designing an interactive visualization, developing approaches to easing input (through popup widgets, gestural input, and 3D widgets), increasing or improving output (through manual management of occlusion, through automatic filling of space, or by automatically constraining the subset of data shown to one that can be drawn neatly), and making state transitions visually smooth (through animation or other means), as well as examining additional issues specific to each case study.

The first part of this chapter reflects back on these case studies, by comparing and contrasting them in their approach to the design goals. This prompts the development of a simple taxonomy of the kinds of parameters that a user may interactively change in a visualization.

This motivates the second part of the chapter, which examines in more detail the kinds of parameters that describe the state of any visualization, and how these parameters can be interactively manipulated. The taxonomy is further developed with the addition of more dimensions, and is populated with many point examples, both from the case studies

and from other work. This maps out a general design space, describing in some detail the "knobs and levers" that can be connected to a visualization for interactive manipulation.

Finally, in the third part of this chapter, we use the preceding comparisons and taxonomy to propose a set of guidelines for designing highly interactive visualizations in future work. Whereas the design goals of section 2.2 largely propose *what* to try to achieve in interactive visualization, the guidelines at the end of this chapter propose more precisely *how* to achieve important aspects of two of the goals, namely easing input and designing smooth transitions. These two goals are the ones most intimately tied to interaction, because at the lowest level (i.e. smallest time scale), all interactions with a visualization involve input that changes a parameter, followed by a transition to a new state.

6.1 Comparison of the Case Studies

For brevity, throughout this chapter we refer to each case study with a single initial: **V** for **V**olume browsing with deformations (Chapter 3), **E** for **E**xpand-Ahead (Chapter 4), and **G** for **G**enealogical visualization (Chapter 5).

In the following, for each of the design goals, we compare the case studies in their approach to the goal.

6.1.1 Approach to Increasing or Improving Output

Each of the case studies deals with a different kind of data and takes a very different approach to increasing or improving the output in the visualization. At a high level, output is improved by either managing occlusion (**V**), by filling space (**E**), or by choosing a graphical representation with improved scalability (**G**). Within the details, there are many additional differences.

With **V**, the data is a kind of mdmv data (section 2.1.1), that has a natural embedding

in 3D, which creates occlusion problems. With **E** and **G**, the data is a kind graph in both cases, whose embedding is free to be chosen by the designer, and occlusion problems are avoided by choosing an embedding in 2D.

To improve output in **V**, the 3D occlusion problems are addressed by using deformations. We feel that these deformations are appropriate for volumetric data, especially anatomical data, because the deformations are often self-illustrative (especially when both the data and the deformation are continuous, and when the user sees the deformation evolve over time as an animated process), and the metaphor of surgical deformation eases understanding even further. For other kinds of data embedded in 3D though, such as architectural models of buildings, a different set of deformations, or different occlusion-management techniques, may be more appropriate. Alternative techniques for managing occlusion are surveyed in Appendix A.

In **E** and **G**, although the embedding is done in 2D and occlusion problems can be completely avoided, there is still the question of how to make the best use of the available 2D space. Because there is no natural embedding or prescribed arrangement of nodes, as designers we have a considerable amount of freedom here. We would like to maximize the use of 2D space, while also keeping the layout orderly so the user may quickly scan and find what interests them in the visualization. Showing all the data at once is not generally feasible, so the user is provided with the means to collapse and expand subsets of the data (in **E** and **G**, these subsets are subtrees). The difference between **E** and **G** in how they attempt to improve output, then, is that **E** automatically *expands* subsets for the user to fill space, to reduce the work required of the user; whereas **G** imposes a "template", so to speak, on the data displayed, and automatically *collapses* nodes (during rotations) so that this orderly template — the dual-tree — is always maintained.

As covered in section 4.2, the prior work most similar to **E**'s automatic filling of space are SpaceTrees [Plaisant *et al.*, 2002]. With SpaceTrees, automatic expansion of a level of nodes is an all-or-nothing act, whereas in **E** some nodes may be expanded while others on

the same level are not, resulting in greater usage of space but less uniform presentation of information. The mixed results of our experimental evaluation demonstrate that trading off between space usage and uniformity is risky. Thus, until a better understanding of how to best implement Expand-Ahead is developed (see section 4.9), the conservative course for designers using these space-filling techniques is to make the presentation uniform (as with SpaceTrees), even at the expense of some automatic expansion.

The prior work most similar to **G**'s use of a template of nodes is probably Ted Nelson's I-view and H-view for zzstructures [McGuffin and schraefel, 2004, Figure 5]. As with the dual-trees in **G**, the idea behind I- and H-views is to extract a subgraph that can be flattened in 2D and drawn neatly, without edge-crossings. The tradeoff in considering these kinds of templates is that, although the user is always guaranteed an orderly layout of information, the user may not be able to see two or three nodes simultaneously that they would be able to see if they had finer-grain control over what is collapsed or expanded. Whether or not to use a template-based approach, let alone which template to use, is highly dependent on the application and task(s).

6.1.2 Approach to Easing Input

Input is eased in the case studies through in-place manipulation, popup widgets, marking menus, gestural input, and combinations of these. These are applied in traditional ways, but are also extended in novel ways that we now point out.

In-place manipulation is used in **E** and **G** where nodes can be clicked on directly (either to select a focal node, or to popup a menu, respectively); and is also used in **V** and **G** in the mouse-based camera controls (where translating in 2D or 3D, and orbiting in 3D, operate with a metaphor of the user dragging space with the mouse). In-place manipulation is also an important aspect of the 3D widgets in **V**, used to adjust deformation parameters. Marking menus, which are one kind of popup widget, are used in **V** to select browsing tools and to select or deselect layers, and in **G** to collapse or

expand parents and children.

Two novel popup widgets were also designed: the popup dial widget in **E**, and the subtree-drag-out widget in **G**. Both allow a gestural-style input that can be either fast and approximate, or slow and precise. Both are also interesting in that they use *continuous* mouse input to control parameters that are *discrete* (position in history in **E**, and depth of subtree in **G**), with visually smooth feedback that interpolates between the discrete states during dragging. This was done to allow the user to control the speed at which a smooth transition occurs, but also had the consequence of allowing the user to quickly traverse a sequence of states in a single invocation of the popup widget and single drag of the mouse. In contrast, for example, the 3D widgets of **V** use *continuous* mouse input to control *continuous* parameters, and the marking menu flicks use *discrete* input to control *discrete* parameters (in which case rapid traversal of a sequence of states is still possible, but requires repeated ballistic flicks). Figure 6.1 shows a 3-category taxonomy, populated by example interaction techniques from the case studies, according to whether the parameter being controlled, and the input controlling it, are discrete or continuous.

Another interesting difference between the two novel popup widgets is that the popup dial widget uses *circular* motion for input (this was chosen because the user can dial many cycles without running out of space, allowing them to traverse a long history); whereas the subtree-drag-out widget uses linear motion (this was chosen so the user would be able to quickly drag to the end of the widget to fully expand or collapse a subtree). This difference is revisited later in this chapter, when the design guidelines are developed.

6.1.3 Approach to Smooth Transitions

The importance of smooth transitions in our case studies was not apparent to us until after prototyping systems without them. In both of the first two cases studies (**V** and **E**), the initial prototypes did not use smooth transitions, and were found to be unacceptably disorienting (by us, before even showing users), forcing us to modify our designs. By

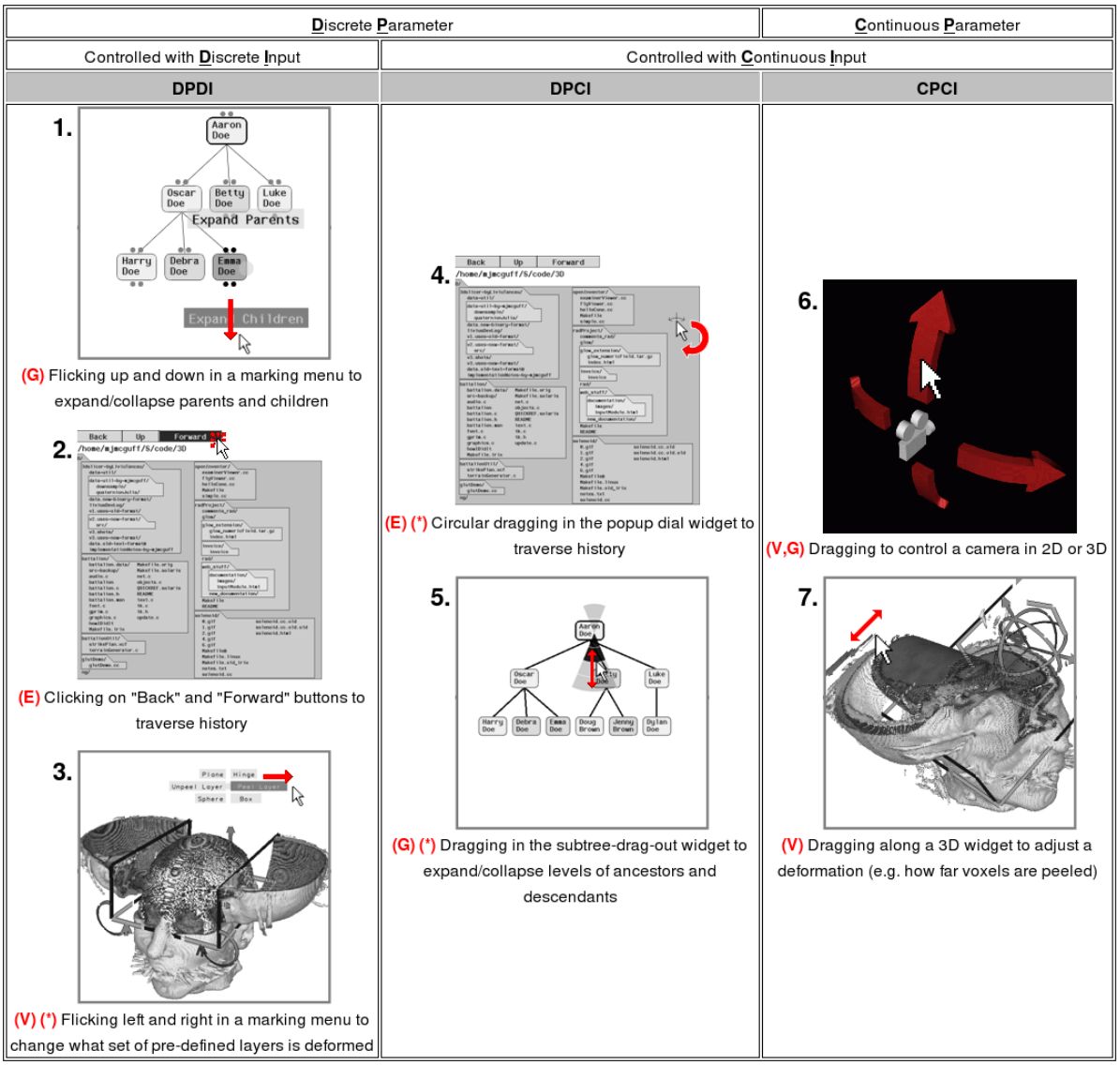

Figure 6.1: A simple taxonomy of parameter manipulation, populated with examples of interaction techniques for manipulating individual parameters. Red letters indicate the case study from which each example is taken, and red asterisks (*) indicate the more novel parameter manipulation techniques.

the third case study (G), smooth transitions were included in the earliest plans for a prototype, as we were convinced that the added complexity of implementing them would not only be worthwhile, but necessary.

All three case studies make use of *animations* to produce smooth transitions over discrete changes, for example: when a layer is selected or deselected in V, when the "Back" or "Forward" buttons are clicked in E, and when the parents or children of a

node are expanded or collapsed in **G** using a marking menu flick. Such animations, once initiated, are driven by the system and play out at a fixed rate. They are initiated by discrete input events (such as clicks or button presses), and correspond to a change in a discrete parameter of the visualization. In our terminology, however, animation is just one of three ways for a transition to be made smooth. Animations were implemented for all of the parameter manipulations in the **DPDI** column of Figure 6.1.

After our experience with the first case study, we designed certain transitions in the other two case studies to progress under continuous user control, by dragging the mouse (with the popup dial widget and the subtree-drag-out widget — column **DPCI** of Figure 6.1). This was done to allow the user to optionally slow down a transition, which might be useful if there are many simultaneous graphical changes that the user wants to examine in more detail. Under this kind of control, if the user has "dragged midway" between discrete states k and $k+1$, the system renders output that appears to be halfway between those two states. The user controls the progress of the transition, and may slow it down, reverse it, or even drag past state $k+1$ to state $k+2$. With animations, because the speed of the transition cannot be changed during the transition, there is a fixed compromise between having the transition fast enough to not delay the user yet slow enough to be understandable. However, under continuous control, the user may adjust this tradeoff dynamically. Since these continuously-controlled transitions do not have a fixed speed or duration, but they nevertheless involve visual continuity between two discrete states, we call them *interpolations* rather than animations.

Finally, for completeness we consider instances in the case studies where continuous parameters are controlled with continuous input (column **CPCI** of Figure 6.1). This includes dragging the mouse to translate or zoom a camera in 2D (as done in **G**) or translate and rotate a camera in 3D (done in **V**), as well as dragging the mouse along the 3D widgets in **V** to manipulate deformation tools. (There are no examples of such interaction in **E**, because no continuous parameters are made available to the user).

When adjusting such continuous parameters, simply providing *real-time feedback* creates visually smooth transitions. At this point, it is worth recalling the ideas behind *direct manipulation* [Shneiderman, 1982; 1983] and *dynamic queries* [Shneiderman, 1994]. Direct manipulation involves representing virtual objects that can be manipulated with real-world metaphors, with incremental and reversible operations (e.g. dragging), where feedback is immediately and continuously updated (i.e. in real-time). An example of direct manipulation is moving a window frame by dragging it with the mouse. Dynamic queries extend these ideas to the manipulation of more abstract entities — namely, ranges of attributes of data that are to be displayed. The early work [Shneiderman, 1994] in dynamic queries used sliders to manipulate ranges of attributes of mdmv data displayed in 2D (such as a scatter plot), and later work [Hochheiser and Shneiderman, 2002] has used widgets overlaid on the data. Regardless of how direct or concrete the interaction is, though, the requirements for continuous manipulability and real-time feedback are naturally applied to *all* continuous parameters of a visualization, yielding a third way of achieving smooth transitions, alongside animation and interpolation.

An interesting feature common to the 3 case studies, which *was not deliberately designed in* but found to be in all 3 after the fact, is their allowance for easy and *rapid* traversal of entire *sequences* of discrete states, with smoothly updated visual feedback. Figure 6.2 illustrates one such sequence from each case study: selecting/unselecting layers in **V** with right/left marking menu flicks, history traversal in **E** with the popup dial widget, and expanding/collapsing levels of nodes with the subtree-drag-out widget in **G**. Of course, almost any visualization system allows some kind of traversal of sequences of states by successively changing one or more parameters, but in our examples this traversal is *rapid* because it can be done either within a single mouse drag (i.e. a single invocation of a widget) or by rapid, successive, ballistic flicks that do not require any retargeting of the mouse or any other time consuming actions between flicks. Another example of rapid traversal of states, in a traditional user interface, would be a word

processor, where the user can rapidly and repeatedly click on the "scroll down" arrow button of a scroll bar to scroll through a document — such a traversal of states is rapid because it involves no retargeting between actions, and can be made smooth with a brief animation for each click. In the case of the scroll bar, however, the parameter being varied is continuous. The examples in Figure 6.2 are novel for demonstrating how to vary a discrete variable, using either continuous input (dragging) or ballistic flicks, all with visually smooth transitions.

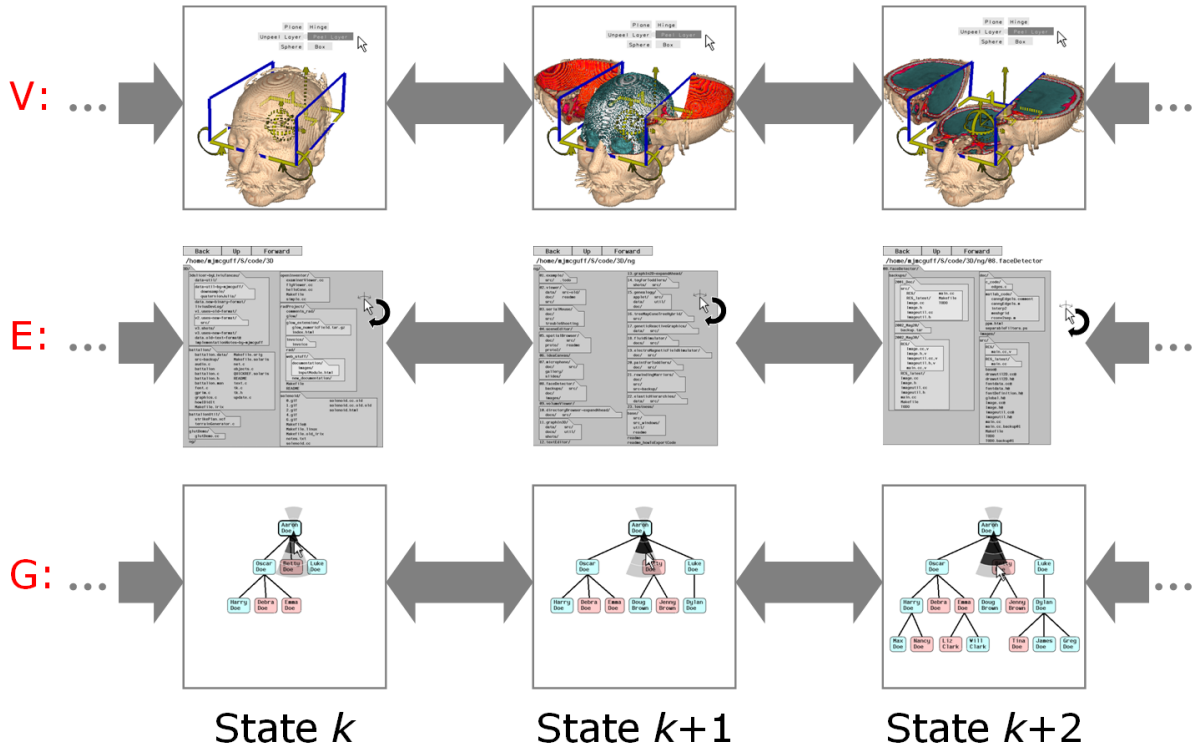

Figure 6.2: Rapid traversal of sequences of states in each case study, with a marking menu, the popup dial widget, and the subtree-drag-out widget.

Figure 6.3 summarizes many of our comparisons so far with respect to input and transitions. It refines the taxonomy in Figure 6.1 by distinguishing between nominal parameters and ordinal parameters, splitting the **DPDI** column of Figure 6.1 into two columns. It also indicates the kinds of parameters for which animation, interpolation, or real-time feedback can be used (or *was* used, in the case studies) to make transitions smooth.

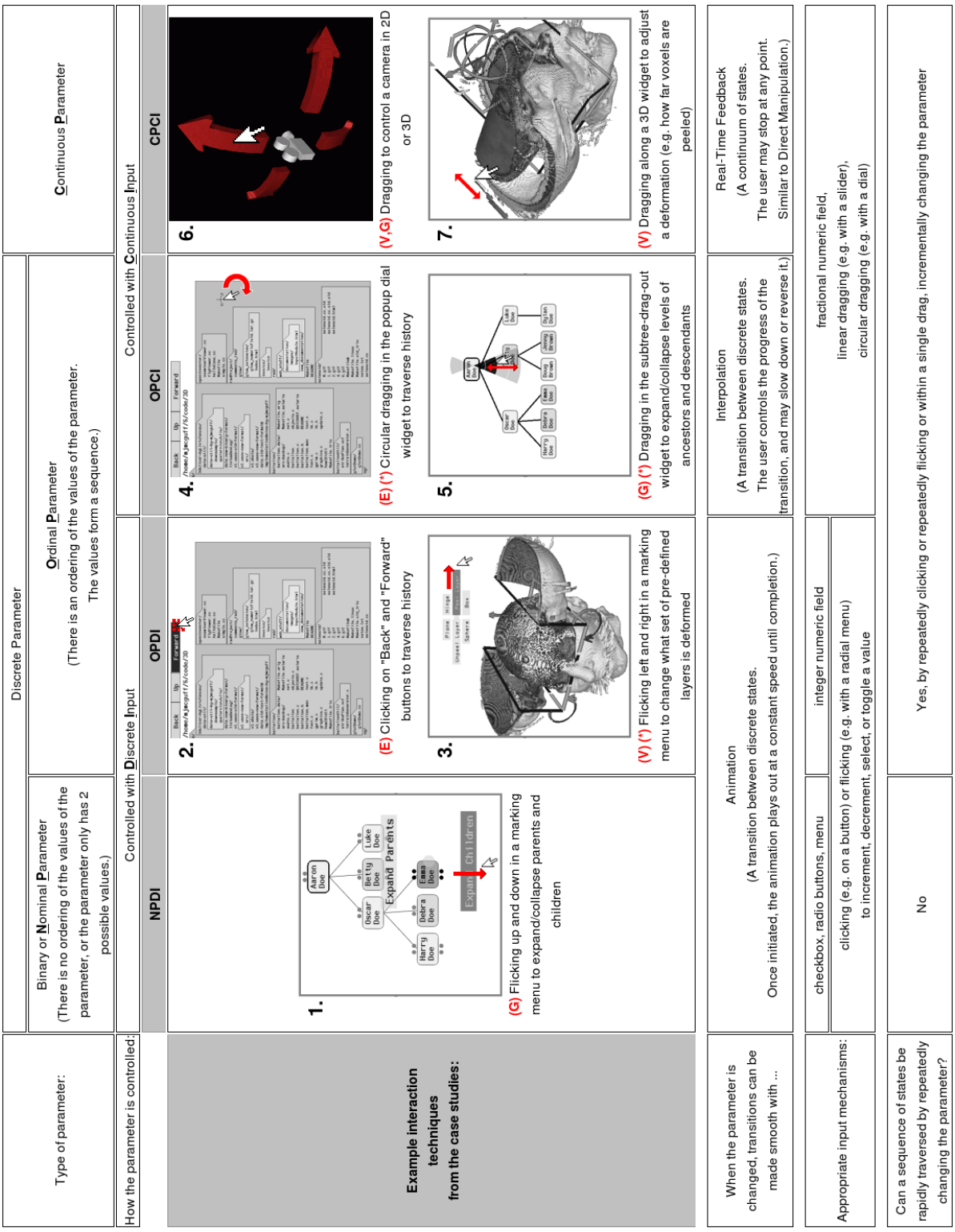

Figure 6.3: A second version of the taxonomy of parameter manipulation. Again, red asterisks (*) indicate the more novel parameter manipulation techniques from the case studies.

The distinction between nominal and ordinal variables *per se* dates back to Stevens [1946], who used these terms to describe statistical data. In the context of interactive manipulation of parameters of a visualization, this distinction is necessary for indicating when rapid traversal of sequences of states is possible. An ordinal parameter is one whose possible values are ordered, forming some natural sequence, therefore allowing the user to repeatedly increment or decrement the parameter by repeating some simple action. For example, in **V**, the k outermost layers of the volumetric data set are peeled, and the user can increment or decrement k with marking menu flicks. k is an ordinal parameter, but requires that the layers be ordered from inside to outside so that the phrase "k outermost layers" is well defined. Were there no well-defined ordering of layers, the user might have to perform a slightly more complicated action to peel each layer (e.g., selecting a layer by pointing at it and *then* flicking), precluding the possibility of *rapid* traversal of a sequence of states.

6.1.4 Observations

The design goals are not achieved independently of each other. The effects of achieving them combine synergistically: smooth transitions reduce the time required to understand different views of the data, which can reduce the total input necessary or increase the total useful output over time; improved output in each state reduces the need to supply input for transitions; and eased input reduces the effort required to view the data in many different ways to obtain useful output.

Two of the design goals — easing input and using smooth transitions — are involved most in interaction, and our analysis of the case studies' approaches to these goals has focused on arguably the *lowest level* (i.e. smallest time scale) aspect of interactive visualization: the manipulation of individual parameters of the visualization, to interactively move from one state to another, on the time scale of individual input events. Each case study involves several interaction techniques for manipulating parameters, many of which

are classified in Figure 6.3. The most novel of these allow entire sequences of discrete states to be rapidly traversed (Figure 6.2). However, we have only characterized parameters by whether they are nominal, ordinal, or continuous, and there is more we could say about the kinds of parameters encountered in visualizations in general. The next section of this chapter elucidates more details about the nature of these parameters, by considering the components of the state of a visualization, i.e., the state of the *visualization pipeline*, or mapping from data to output. We will further develop the taxonomy by adding more dimensions to it, and populate it with more examples, both from the case studies and from other work.

6.2 Parameters and Controls in Interactive Visualization

To gain a deeper understanding of how the user may vary parameters of a visualization's state, we will consider visualization as an activity in a more general and abstract sense than we have so far. In the following, we define more precisely our scope of consideration, and build on a simple model of the visualization pipeline, to further develop our taxonomy of the parameters of a visualization's state that the user may vary.

6.2.1 Interactive Visualization as a Subset of HCI

Much of human-computer interaction (HCI) is concerned with the use of computers for manipulating and rendering data, by and for human users. The data might be a multimedia document, a hypertext, a collection of images and videos, etc. Manipulation of data corresponds to activities such as authoring and editing; and rendering is done visually, aurally, or through other sensory channels.

Most interactive systems involving a single user can be modelled according to Krasner and Pope's Model-View-Controller paradigm [1988], which we present here using slightly

different terminology. As shown in Figure 6.4, A and B, we assume the user is interested in manipulating and/or viewing some data. The output to the user is a rendering of data through a viewer of some kind. Input to the system occurs through controls (i.e. user interface elements). We can extend the Model-View-Controller paradigm by distinguishing between 3 kinds of controls: first, controls that change the data (Figure 6.4, C); second, controls that change the state of the viewer, and hence the view of the data (Figure 6.4, D); third, controls that change the state of the controls, which might be called meta-user interface elements (Figure 6.4, E) [McGuffin, 2002]. Chi and Riedl [1998] similarly distinguish between the first and second kinds of controls (referring to "value operations" versus "view operations"), and Beaudouin-Lafon and Mackay [2000] touch on the notion of meta-interfaces.

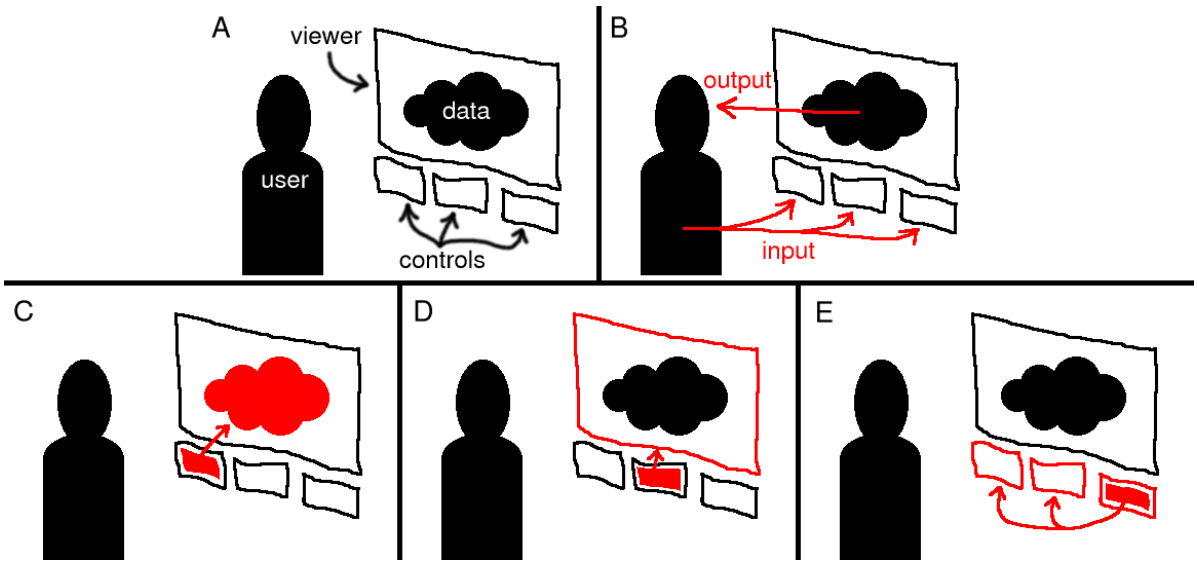

Figure 6.4: Basic elements of an interactive system. *A and B:* a user interested in some data sends input through controls and receives output through a viewer. *C:* some controls modify the data. *D:* other controls modify the view of the data. *E:* meta-controls modify the mechanisms used for input.

Interactive visualization can be defined as an activity involving a subset of this model. Specifically, we consider visualization involving *visual* rendering of data through a single viewer, and involving input through controls of the second kind (Figure 6.4, D). Within this scope, interaction is for the purpose of changing the rendered view of the data, and

does not include activities such as editing or annotation.

We now examine and decompose the computational process involved in visual rendering of data.

6.2.2 The Visualization Pipeline, and its Active Subsets and Operators

The mapping from data to graphical representation in a visualization (Figure 2.1) can be broken down into the stages of a *visualization pipeline*. Different versions of this pipeline have been presented by previous authors [Chi and Riedl, 1998] [Card *et al.*, 1999, chapter 1] [Carpendale, 1999, chapter 1]. Here, we consider a version (Figure 6.5) adapted to the discussion that follows, intentionally omitting or treating in more detail various aspects as appropriate.

Figure 6.5: The visualization pipeline.

The pipeline starts at a *data* set, which may be an mdmv data set, or a graph, or some other form of data.

Next, some or all of the data is *embedded* (i.e. assigned locations, coordinates, a layout) in a space suitable for viewing by a human user; this space may have at most 3 spatial dimensions and 1 temporal dimension (as in looping animations or video playback). Interactive changes to the embedding correspond to rearrangement [Spence, 2001] of the visualization, and include operations such as re-sorting entries in a table by different attributes, or laying out a graph on a plane according to different spatial criteria.

After spatial locations have been assigned to the data, the *graphical embellishment* stage assigns properties such as colour, opacity, geometric form or shape, graphical decorations, and drawing style to data elements. Examples of interactive changes to this

stage include temporarily hiding or showing data elements, or modifying the transfer function used to determine optical properties in volume rendering.

Finally, a *view transform* stage performs operations such as global translation, rotation, scaling, projecting, compositing, and clipping. This stage corresponds to the affine camera projection used in traditional computer graphics.

The stages in the pipeline can be thought of as functions, or *operators* as we will refer to them, but only approximately. Figure 6.5 suggests that the input to each operator is limited to the output of the preceding operator, however there is no strict hiding of any information from any operator. For example, the graphical embellishment stage generally depends on the data, as well as on the embedding of that data. The operators can thus be thought of as stages of processing on a globally accessible set of information, or alternatively as operators that each pass along their original input with their newly generated output to the next operator. Furthermore, these stages need not operate in isolation from each other. Even the order in which the operators are applied may sometimes seem contrary to that suggested by Figure 6.5. For example, an embedding in 3D may depend automatically on the current viewing direction (e.g. [Carpendale *et al.*, 1997a]), as may the opacity (e.g. [Viola *et al.*, 2004]) or other graphical properties (e.g. semantic zooming changes the rendering style of data elements and subsets of data based on scale [Bederson and Hollan, 1994]). What is most important for our discussion are the separate roles played by each operator.

Often, the operators in the pipeline that perform embedding or embellishment can be specified in a piece-wise manner. For example, in the embedding stage, we might specify that one subset of data be embedded a certain way, another subset be embedded in a different region of space or in a different manner (e.g. translated away or stretched out), and the remainder of the data not be embedded at all. Similarly, in the graphical embellishment stage, we might specify that different subsets of the data be assigned different colours or shapes or opacity values, or that one subset in particular be made

invisible (with opacity zero). We refer to these different subsets as *active subsets*, and imagine the pipeline with separate streams for each active subset (Figure 6.6). So, rather than thinking of singular embedding and embellishment operators defined piece-wise over all the data, we shift our point of view to one where each active subset (or "piece") has its own (non-piece-wise) operators. Note that, as shown in the figure, there is only one view transform used for all these active subsets, because we assume the embedding is done in a common world space. (There is research on multiple projection methods [Agrawala *et al.*, 2000; Singh, 2002], where output from multiple cameras is combined into a single image, however, to keep our model simple, we adopt the perspective the such combining of multiple projections occurs as part of the embedding step.)

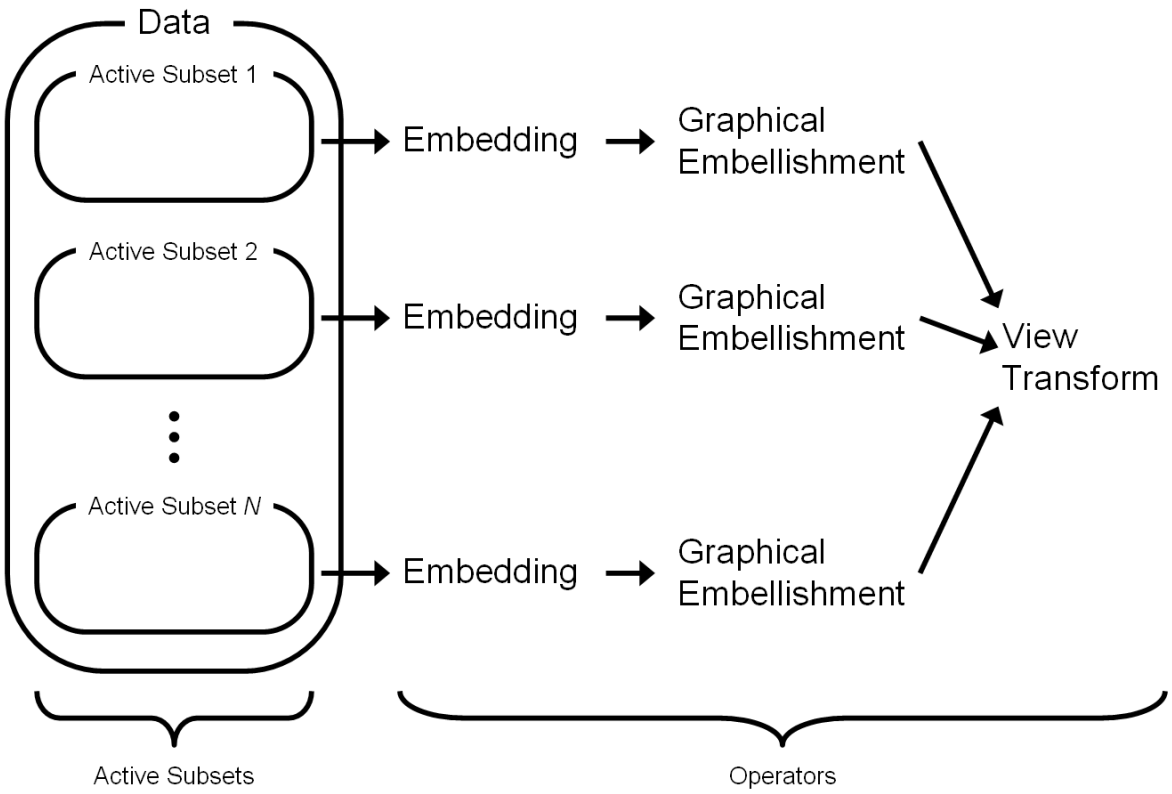

Figure 6.6: A more detailed model of the visualization pipeline. The "active subsets" of the data are the data currently being visualized, and each active subset has its own associated embedding and embellishment operators.

We use the term *state* to refer to the state of the visualization pipeline, i.e. the state

of all the parameters specifying all the active subsets and operators in the pipeline. Input from the user is used to specify, or modify, this state. Examples of techniques for specifying an *active subset* include: specifying a range of values of some attribute (e.g. all data points with a temperature between 20 and 25 degrees), specifying a region of space containing embedded data points (e.g. all data elements enclosed in an on-screen region selected with the mouse), or selecting a single data element that has a corresponding subset (e.g. the subtree rooted at a node, or the neighbourhood surrounding a point). On the other hand, the specification of an *operator* may involve choosing one of two modes for rendering (e.g. embedded or not embedded; opaque or fully transparent; detailed or overview), or choosing between a few discrete modes (e.g. different rendering styles), or choosing from a continuous set of possibilities (e.g. spatial displacement, opacity, viewing direction).

Users typically do not specify a visualization by fully and explicitly specifying each active subset and its operators. Much more commonly, the visualization begins with some initial state of active subsets and operators, and the user modifies that state, either by modifying the membership of one or more active subsets, or by modifying an operator. Such repeated modification of the state is the essence of interactive visualization.

In some cases, the user's actions may mostly involve just one kind of modification to the state. For example, in a 3D object viewer, there may be a single fixed active subset corresponding to the entire data set, and interaction may be limited to changing the camera view and adjusting a global opacity value. Conversely, in other visualizations, an entire session may involve only changing the membership of the active subset(s) while leaving operators fixed. Examples include dynamic queries [Shneiderman, 1994] where the user may use sliders to specify the members of a single visible active subset; web browsers where the active subset is a single node (web page) and its outbound edges (embedded links), and the user clicks on a link to replace the active subset with a new one; tree (e.g. file system) visualizations where the single active subset of nodes can have

nodes added or removed by clicking on nodes to expand or collapse their children.

We focus our attention on *parameters* that define the active subsets and operators, and how these might be modified by the user. The basic interactions the user can perform to change a visualization's state involve changing these parameters. Figure 6.7, A-B illustrate a change in parameter(s) defining the active subsets. Figure 6.7, B-C illustrate a change in parameter(s) defining an operator. By associating appropriate user interface controls with various parameters, we can enable the user to traverse sequences of states, possibly rapidly and with smooth feedback as in Figure 6.2. We will next add more dimensions to the taxonomy of Figure 6.3 to classify in more detail these parameters.

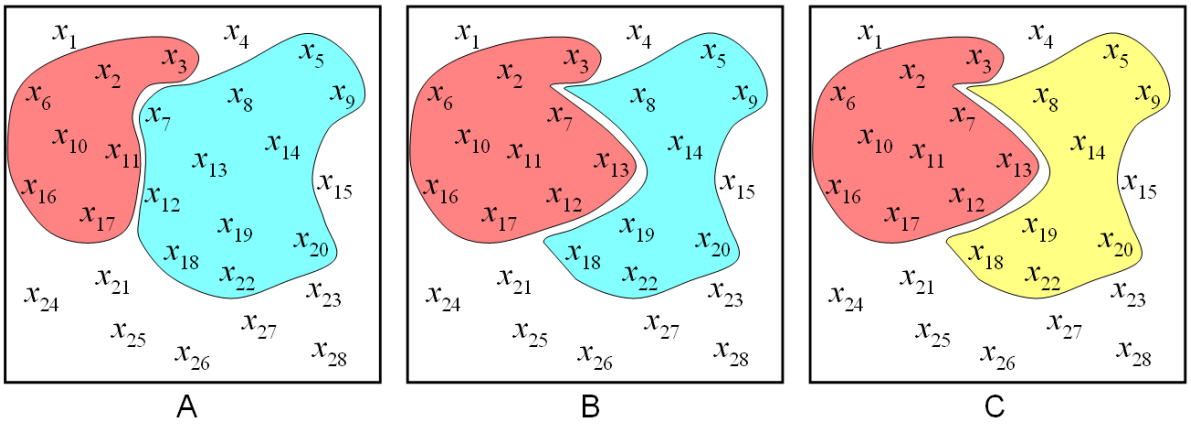

Figure 6.7: Active subsets of a data set $U = \{x_1, x_2, \ldots\}$ being visualized by the user. The appearance of the visualization to the user is not shown; this figure only shows the subsets of elements involved in the visualization, with no particular embedding. The elements of U may be nodes of a graph, voxels of a volume, etc. A: Two active subsets S_1 (in red) and S_2 (in blue), each with associated operators in the visualization pipeline. For example, S_2 might be rendered with an embedding (deformation), colour, or opacity different from S_1. Active subsets are disjoint, and data $U \setminus (S_1 \cup S_2)$ outside them is not rendered. B: A change in the boundaries (membership) of the active subsets, by changing some parameter(s) defining the active subsets, is one way of changing the visualization (corresponding to the **AS** column in Figure 6.8). Elements x_7, x_{12}, x_{13} are now rendered with S_1's operators rather than S_2's. C: A change in an active subset's operators, by changing some parameter(s) defining the operators, is a second way of changing the visualization (corresponding to the **OP** column in Figure 6.8). For example, S_2 might now be rendered with an embedding, colour, or opacity different from S_2 in (B).

6.2.3 Additional Classifications of Parameter Manipulation

This section presents several dimensions or criteria that can be used to classify parameters of a visualization and their manipulation, and also crosses selected dimensions to produce two tables (Figures 6.9 and 6.11) that show more detailed views into our taxonomy of parameter manipulation techniques. The tables are populated with examples from the case studies as well as from other work.

Active Subsets versus Operators, and Pipeline Stages

Figure 6.1 uses two dimensions to organize its columns:

- **D1:** Whether the parameter is continuous or discrete.
- **D2:** Whether the input used to change the parameter is continuous or discrete.

Figure 6.3 refines this by replacing **D1** with a dimension that distinguishes between three kinds of parameters:

- **D1′:** Whether the parameter is continuous or discrete, and if discrete, whether its values form an ordered sequence (i.e. the parameter is ordinal) or not (i.e. the parameter is either binary or nominal).

To these dimensions, we now add

- **D3:** Whether changes to the parameter change an active subset (**AS**) or change an operator (**OP**).
- **D4:** The stage of the pipeline at which the effect of changing the parameter is seen: embedding, graphical embellishment, or view transform.

D3 distinguishes between the two kinds of parameter changes illustrated in Figure 6.7 that we have already discussed.

D4 we explain here with an example. For parameters associated with an operator (**D3=OP**), **D4** is simply the stage of the operator. However, for parameters associated with an active subset (**D3=AS**), e.g. parameters that define the boundaries of the active subsets, **D4** is the stage at which the pipeline is affected by a change in these boundaries. For example, imagine a 3D magic lens [Viega *et al.*, 1996], e.g. a 3D box that can be positioned by the user within a 3D scene, and imagine that this lens or box causes the elements of the scene within it to be rendered semi-transparent. There are two active subsets: the set of elements outside the box (rendered with 100% opacity), and the set inside the box (rendered with, say, 50% opacity), and each of these subsets has its own stream of operators in Figure 6.6. If the user translates the box's position, this would cause the active subsets to change, or equivalently cause the boundary (i.e. the box) between them to change (**D3=AS**), and as a consequence the opacity of some elements would change (**D4=graphical embellishment**) as new elements enter the box or old ones leave. On the other hand, if the user adjusts the opacity of the box's contents, this is a parameter of an operator (**D3=OP**, **D4=graphical embellishment**) associated with the box's active subset. Now, imagine instead that the box causes elements within it to be deformed rather than rendered semi-transparent. Then, changes to the box's position would be **D3=AS**, **D4=embedding** changes, and changes to the deformation would be **D3=OP**, **D4=embedding** changes.

We combine **D1**, **D2**, **D3** and **D4** into a single taxonomy in Figure 6.9. However, the reader may first examine Figure 6.8, which only involves **D3** and **D4**. Figure 6.8 can be used as an overview of the major parts of Figure 6.9, and contains explanations of each cell within the **D3** $\times$ **D4** crossing.

In Figure 6.9, **D3** $\times$ **D1** $\times$ **D2** define the columns, **D4** defines the rows, and the table is populated with every parameter manipulation present in the case studies (indicated with red letters), as well as several additional parameter manipulations from other example systems.

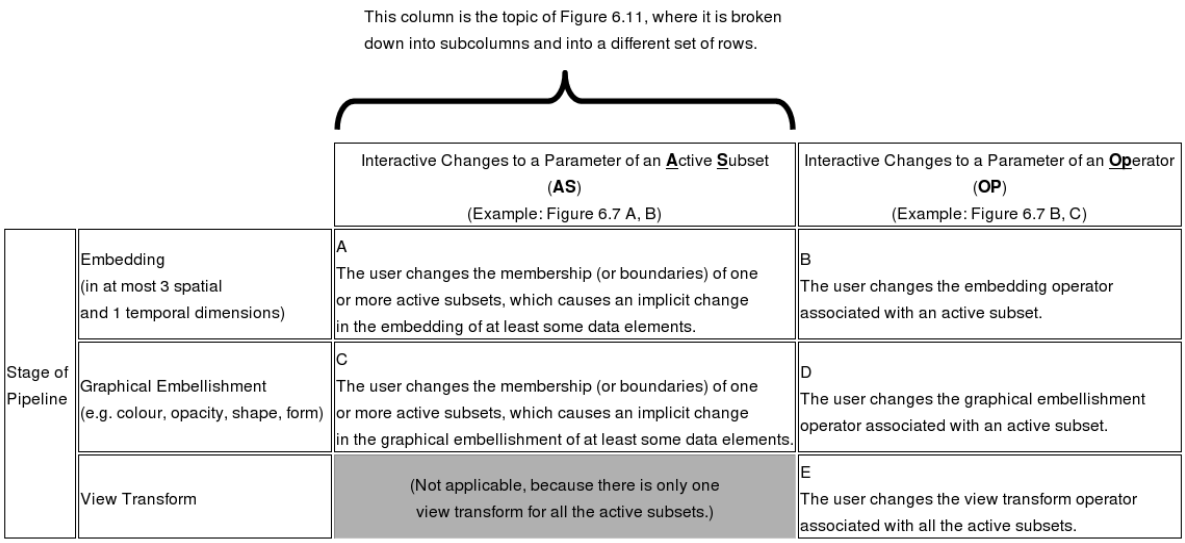

Figure 6.8: A high-level overview of the taxonomy in Figure 6.9. Columns indicate whether an active subset or an operator is changed, and rows indicate *where* the effect of the change is seen by the user. In Figure 6.9, the **AS** and **OP** columns of this table are broken down into subcolumns, and the cells are populated with examples. In addition, in Figure 6.11, the **AS** column of this table is broken down into subcolumns, and into a different set of rows.

Figure 6.9 makes clearer how the popup dial widget of **E** and subtree-drag-out widget of **G** are novel, as they are the only examples we know of in the **AS-DPCI** column. The **OP-DPCI** column is also relatively unexplored in previous work, containing only a technique from Burtnyk et al.'s StyleCam work [Burtnyk *et al.*, 2002], as well as a technique for interpolating between visualizations that was described (but not implemented) by Tory et al. [Tory *et al.*, 2005], and two novel techniques (marked with red asterisks) that we thought of in trying to populate the taxonomy. In both of the novel techniques, the user may "slide" (by dragging with their mouse) to transition between different layouts of data, either quickly or slowly as desired. These techniques were directly inspired by studying the taxonomy and asking what kind of existing or imagined techniques should go in each cell.

Stage of Pipeline	Interactive Changes to a Parameter of an Active Subset				Interactive Changes to a Parameter of an Operator			
	Discrete Parameter (tends to involve graph data)		Continuous Parameter (tends to involve mdmv data)		Discrete Parameter		Continuous Parameter	
	Controlled with Discrete Input		Controlled with Continuous Input		Controlled with Discrete Input		Controlled with Continuous Input	
	AS-DPDI		AS-DPDI	AS-CPCI	OP-DPDI		OP-DPCI	OP-CPCI
Embedding (in at most 3 spatial and 1 temporal dimensions)	Collapsing/Expanding nodes in browsers such as SpaceTree [Plaisant et al., 2002] Toggling pre-defined subsets of nodes of a graph or tree for filtering, e.g. viewing only files in a file system that have certain extensions (.jpg, .png, ...) and toggling those extensions **(V)** Hotkeys that toggle whether each pre-defined layer is deformed or not **(V) (*)** Left and right marking menu flicks to change what set of pre-defined layers is deformed **(E)** Selecting a new focal node causing drill-down **(E)** "Back" and "Forward" buttons to traverse history **(G)** Up and down marking menu flicks to expand/collapse parents and children		**(E) (*)** Popup dial widget to traverse history **(G) (*)** Subtree-drag-out widget to expand/collapse levels of ancestors and descendants	Changing a filter on nodes of a graph or tree, e.g. viewing only files modified within the last *x* hours in a file system and changing *x* **(V)** Adjusting the size of a subset of voxels that is deformed (e.g. the thickness of a slab that is peeled) using 3D widgets	Selecting nodes in Cone Trees [Robertson et al., 1991], causing a rotation of the selected node to the front **(E,G)** Selecting different layout modes (e.g. left-to-right, top-down, etc.)	**(*)** Sliding between different layout modes for a graph or tree **(*)** Sliding between different orderings of tabular data (e.g. sorting by name, by date, etc.)	Dragging the focal point in Hyperbolic Browser [Lamping et al., 1995], causing a "fisheye" deformation to update **(V)** Adjusting a deformation (e.g. how far voxels are peeled) using 3D widgets
Graphical Embellishment (e.g. colour, opacity, shape, form)	(Could be useful in a 3D embedding of a graph, for controlling which nodes have different opacity levels)			Dynamic queries **(V)** Translating a 2D/3D magic lens [Bier et al., 1993; Viega et al., 1996] or translating a cutting plane **(V)** Resizing regions of space, such as cutting boxes or spheres, to be rendered with a given transfer function (e.g. opacity), using 3D widgets Adjusting the domain (region of space) of a transfer function for volume rendering [Kniss et al., 2001]	Selecting different rendering modes (e.g. colour schemes, rendering styles, etc.)	Sliding between different rendering modes or transfer functions (discussed, but not implemented, by [Tory et al., 2005])	Adjusting the codomain (opacity, colour, etc.) of a transfer function for volume rendering
View Transform	(Not applicable)				**(G)** Transitions between different, automatically framed camera views	Sliding from one pre-selected view to another in StyleCam [Burtnyk et al., 2002]	**(V,G)** Traditional techniques for controlling cameras in 2D and 3D	
Transitions made smooth with ...	Animation	Interpolation	Real-Time Feedback (similar to Direct Manipulation)	Animation	Interpolation	Real-Time Feedback (similar to Direct Manipulation)		

Figure 6.9: A third version of the taxonomy of parameter manipulation. Examples drawn from the case studies are indicated as such, and red asterisks (*) indicate the more novel techniques. A simpler overview of this table is in Figure 6.8.

Changes to Active Subsets: More Details

It is possible to examine in further detail the nature of parameters that can be adjusted in an interactive visualization. Within the **D3=OP** half of Figures 6.8 and 6.9, it is not obvious what more can be said in general terms, since embeddings and their parameters vary greatly between different kinds of data, and there is already previous work that examines the possible mappings involved in graphical embellishment (e.g. [Bertin, 1967]). However, within the **D3=AS** half of these figures, there are other general dimensions to be considered, and we feel the **D3=AS** half is worthy of attention from the perspective of possibilities for interaction.

We have already discussed active subsets, however there are other kinds of subsets of data encountered in the case studies, such as the layers in **V** (which are disjoint subsets), and the subtrees n **E** and **G** (which are not disjoint). Because these subsets can play a role in how active subsets are interactively changed, a classification of them would be useful. By considering many different examples, we observed that these different subsets all seems to be either implied by the data, or defined transiently by the user in terms of some selected data element(s) during interaction. This brings us to Figure 6.10, which illustrates the three different kinds of subsets we will consider.

Furthermore, when a parameter is being manipulated, although the manipulation may result in a change to an active subset, the parameter itself may be more directly associated with some other kind of subset. For example, in the subtree-drag-out widget of **G**, the user may adjust a *depth* parameter to add or remove nodes of a subtree to the current active subset A of visible nodes. However, this *depth* parameter is really a parameter of the selected subtree (a kind of *neighbourhood* in Figure 6.10), not a parameter of A. Thus, a new dimension is

- **D5:** What kind of subset is the parameter most directly associated with? (Figure 6.10 shows the three possibilities.) Is it associated with an active subset (e.g.

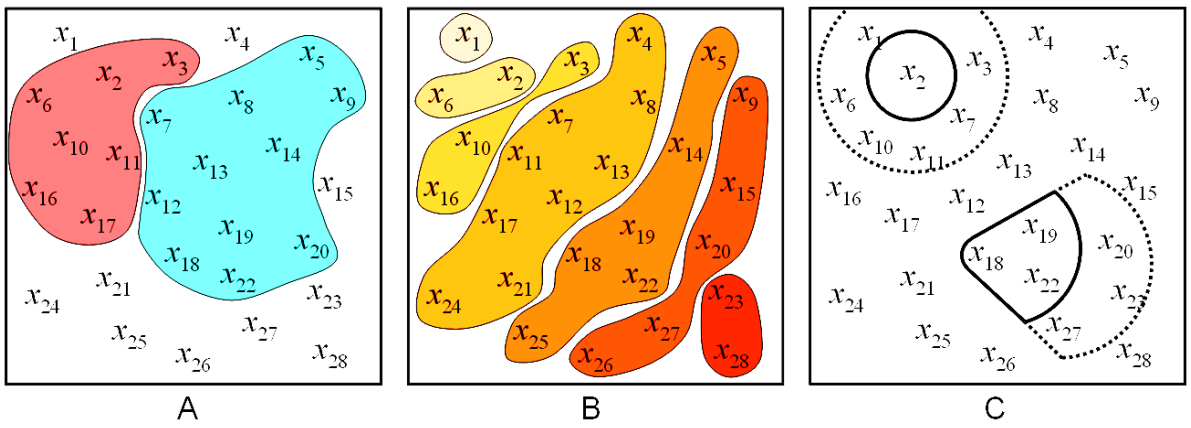

Figure 6.10: Three kinds of subsets of the data set $U = \{x_1, x_2, \ldots\}$. A: *active subsets* are the currently visualized subsets. B: *pre-defined subsets* may be "naturally in" the data or implied by the data's attributes or structure. Examples include the layers used in volumetric data in **V**. Note that, in general, these subsets needn't be disjoint. C: *neighbourhoods* are each defined by some central element (x_2 and x_{18} in the figure) and possibly other parameters such as a radius. Examples include the subtree to some depth of a selected root node (used in **E** and **G**), or a plane through some selected point. Proximity to neighbours may be defined by structure (e.g. distance in edges) or in terms of some embedding.

such as the min and max parameters defining an interval of data in a dynamic query)? Or, is it associated with a pre-defined subset of data (such as an integer identifying one of the ordered layers in **V**)? Or, is it associated with a neighbourhood of some selected element (such as the radius or depth of a group of nodes)?

There are additional subtleties in how active subsets are updated. In the example of the subtree-drag-out widget of **G**, the user may *add* or *remove* nodes from the existing active subset of visible nodes. However, in other systems, a single input action may *replace* an active subset with a new one, such as when the user clicks on a node in **E** to drill down to a new subtree. This yields one more dimension:

- **D6:** How are the active subset(s) modified by the change in the parameter? Are the active subset(s) enlarged by some selected elements, or reduced by some selected elements, or replaced by a newly selected subset of elements?

Note that **D5** and **D6** are specific to the **D3=AS** half of Figures 6.8 and 6.9.

A tabular taxonomy made with too many dimensions can be unwieldy and unenlightening. Thus, rather than combine all of **D3** through **D6** into a single taxonomy, we create a second taxonomy that emphasizes the new dimensions to consider, and that complements the information in Figure 6.9. We modify the **AS** portion of Figure 6.9, replacing **D1** with **D1′** (splitting one column in two), replacing **D4** with **D6**, and adding colour for **D5**, to produce Figure 6.11.

The choice of using **D6** for rows and **D5** for colour is not arbitrary. If instead **D5** is used for rows and **D6** for colour, the resulting table has 4 empty cells. Although empty cells in tabular taxonomies can be useful for inspiring new ideas, the empty cells in this case did not seem particularly interesting, and the primary purpose of the taxonomy should be to map out and summarize a design space and aid comparison. Thus, we chose the mapping that results in a more evenly populated arrangement. Interestingly, when first sketching this taxonomy, the alternative arrangement originally involved 5 empty cells, not 4. One of the 5 inspired the addition of the first example ("Painting, or drag-selecting, regions...") listed in the **AS-CPCI** column in Figure 6.11, providing another example of how a taxonomy can point to overlooked ideas.

Observe that parameters in the **AS-NPDI** column of Figure 6.11, like the parameters in the **NPDI** column of Figure 6.3, are not conducive to rapid traversal of sequences of states, since those parameters do not have any natural ordering for their values. The other columns, however, do allow for this.

The user input required to change an **AS** parameter in the taxonomy can be broken down into a sequence of steps, specifying the information required by the system. We describe the generic pattern followed in the form of an algorithm:

- ▷ Select an active subset to modify, i.e. specify i where $A_i \subset U$.

- ▷ Select a parametrized subset $S(p) \subset U$ that is a function of some parameter p. $S(p)$ may be a modified form $A'_i(p)$ of A_i, or may be a predefined subset $P(p)$, or

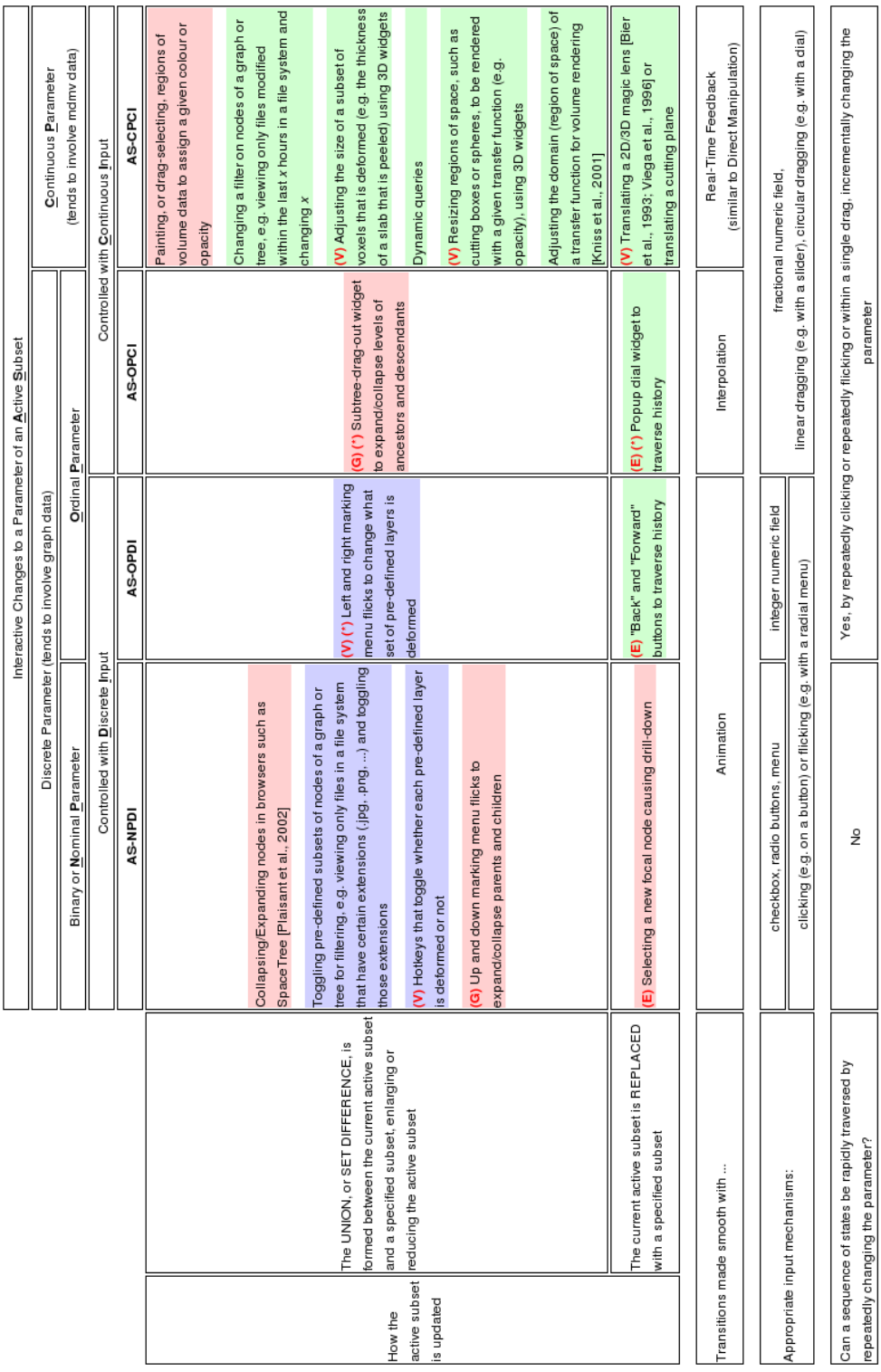

Figure 6.11: A fourth version of the taxonomy of parameter manipulation, focused on the **AS** portion of Figure 6.9.

may be a neighbourhood $N(e, p)$ of some selected element $e \in U$. Note that these 3 possibilities correspond to **D5**.

▷ Select how A_i is to be modified by $S(p)$: by assignment with $A_i \leftarrow S(p)$, or by set union with $A_i \leftarrow A_i \cup S(p)$, or by set difference with $A_i \leftarrow A_i \setminus S(p)$. Note that these 3 possibilities correspond to **D6**.

▷ Loop {

  ▷ Select, or modify, the value for the parameter p.

  ▷ Observe visual feedback showing the new A_i.

} until done

In practice, depending on the interaction technique employed, often at least one of the above steps is implicit, and input can be made even lighter weight if selection and adjustment steps are coupled, as can often be done with a popup or gestural technique. Furthermore, the final loop in the algorithm above may be controlled with dragging or flicking or repeated clicking on a target or button, allowing for a high degree of interactivity.

6.2.4 Related Work on Taxonomies and Models of Visualization

To appreciate how our preceding taxonomy of parameter manipulation relates to existing literature, we briefly survey the most relevant models, taxonomies, and theoretical frameworks that have been previously described for classifying and describing visualizations. One aspect often considered in such work is the type of data being visualized (in fact, data type is the basis for the split between sections 2.1.1 and 2.1.2). For example, Shneiderman [1996] gives 7 data types: 1D, 2D, 3D, temporal, multi-dimensional, tree, and network. Tweedie [1997, Figure 2] lists 5 types of "structural representations": rectilinear, circular, ordered, unordered, and stereograms (i.e. 3D). Card and Mackinlay [1997]

list the following possibilities for the "data type" of each data dimension: nominal, ordinal, quantitative, intrinsically spatial, geographical, or a "set mapped to itself — graphs". Chi [2000] lists the following categories of visualizations based on "data type": scientific, geographical, 2D, multi-dimensional, landscapes and spaces, trees, network, text, web, and spreadsheets. These classifications all have their use, however if we dissect the details of data type, there are subtleties that may not have received full attention in much of the previous work. In particular, when speaking of dimensionality, it is useful to distinguish between (a) the number of independent quantities in the data, (b) the number of dependent quantities in the data, and (c) the dimensionality of the space that the data is embedded in, and recognize that *these are orthogonal aspects of a visualization* and hence should probably be *crossed* in the interest of producing a complete taxonomy. The term "mdmv" (section 2.1.1) makes the difference between (a) and (b) explicit. Some data sets, such as trees and graphs, have no intrinsic dimensionality, but can be embedded in a space of 1 or 2 or 3 dimensions, whose dimensionality is quite different from the data's dimensionality (or lack thereof, in the case of trees and graphs). Hence, categories with the names "2D", "3D", and "networks" that are not further qualified can be ambiguous and are also not mutually exclusive.

Another aspect often considered in models or taxonomies of visualization is the mapping from data to graphics, filling out the details of Figures 2.1 and 6.5. As discussed earlier in this chapter, this mapping is often broken into stages of a dataflow pipeline [Chi and Riedl, 1998] [Card *et al.*, 1999, chapter 1] [Carpendale, 1999, chapter 1] [Card, 2003]. Guidelines exist for the design of visual encodings [Bertin, 1967; 1977; Mackinlay, 1986], and Card and Mackinlay [1997] develop a compact tabular notation for describing the mapping used in a given visualization, with limited consideration of interaction (they distinguish between radio buttons and sliders for discrete and continuous input, respectively). Tory and Möller [2004] give a taxonomy of mappings, at two levels of abstraction, based on whether they are continuous or discrete, how constrained

the mapping is, and the dimensionality of the data variables involved.

With respect to interaction, current taxonomies have identified the mid- to high-level operations and tasks that users perform with interactive visualizations. For example, Shneiderman [1996] gives 7 tasks: overview, zoom, filter, details-on-demand, relate, history, and extract. Other lists of tasks or operations have also been developed [Chuah and Roth, 1996; Amar and Stasko, 2004; Amar *et al.*, 2005]. Beyond these, Tweedie [1997] discusses other issues related to interaction, including the role of direct *versus* indirect manipulation, and the concept of a continuum of control from manual to automated. Chi and Riedl [1998] also discuss direct manipulation and state that the "amount of interactivity" varies along the visualization pipeline, increasing as we get closer to the final view. However, many of the relatively low-level (i.e. small time scale) details of interaction have not yet been worked out, and Figures 6.3, 6.9, 6.11, and section 6.2.3 contribute new ideas here, by considering in detail how the user may interactively manipulate individual parameters controlling a visualization.

Some important topics that have not been treated in this dissertation are the generation of derived data (e.g. abstractions, aggregates, summaries, processed and filtered data, edited or user-annotated data) [Chuah and Roth, 1996; Tweedie, 1997; Chi and Riedl, 1998] and visualization involving multiple coordinated views [Chuah and Roth, 1996; North, 2000]. Instead, as specified in section 6.2.1, we have focused on interaction with a *single* view of data, where input is used to modify the *view* of the data and does not modify the data or cause new data to be generated.

6.2.5 Contribution of our Taxonomy of Parameter Manipulation

As discussed earlier, the manipulation of parameters of a visualization is the core low-level issue of interactive visualization. How to control such manipulations, and what output to display during such manipulations, are the concern of two of our design goals,

to ease input and to use smooth transitions.

The previous literature on models or taxonomies of visualization does not focus on interactive manipulation of parameters at this low level, and our taxonomy elucidates many details regarding such manipulation. Previous taxonomies of visualization have distinguished between nominal, ordinal, and continuous types of *data* (e.g. [Card and Mackinlay, 1997]), but we are unaware of the same distinction being made for parameters of subsets or operators. With regard to types of subsets, other kinds of subsets are defined by Chuah and Roth [1996], who identify interactive operations performed on subsets ("control objects" and "focus sets"), which are relevant in data analysis where the user may process data to generate new data and create multiple coordinated views. However, our distinction between active subsets, pre-defined subsets, and neighbourhoods (Figure 6.10) appears to have no equivalent in the previous literature. We have also clarified the commonality between direct manipulation and smooth transitions: the basic principle underlying both is that any parameter manipulation by the user should cause smooth, visually continuous feedback, whether the parameter is continuous or not.

In addition to being novel, our taxonomy exhibits some notable desirable properties. Taxonomies in general sometimes consist of a single dimension, or perhaps a hierarchical tree of categories, or perhaps a list of dimensions. Our taxonomy not only identifies 6 different dimensions, but also *crosses* them to produce tables. Such tabular views into a taxonomy are highly desirable, for they effectively summarize and make implicit many patterns of relationships and comparisons. As evidence of elegance in the arrangement in Figures 6.3 and 6.11, there is something meaningful to be said about every contiguous group of the columns **NPDI**, **OPDI**, **OPCI**, **CPCI**: the two left-most columns involve discrete input, the two middle columns involve ordinal parameters, the two right-most involve continuous input, the three left-most columns are for discrete parameters, and the three right-most are for parameters that can be varied to rapidly traverse sequences of states.

Our taxonomy is useful for comparing interaction techniques from the case studies, and comparing these with techniques from other work. In fact, Figures 6.9 and 6.11 contain almost all the parameter manipulation techniques we could think of. (The only exceptions we have found — visual access distortion [Carpendale *et al.*, 1997a], importance-driven rendering [Viola *et al.*, 2004], and semantic zooming [Bederson and Hollan, 1994] — involve an unusual dependency between stages of the pipeline, where the embedding or embellishment depends on the camera view.) Part of the reason we were able to identify 6 dimensions is that the case studies differ from each other in so many ways, suggesting many different comparisons.

Ideally, a taxonomy should not only aid in understanding existing examples, but also point to novelty. Two examples of this that were produced in developing the taxonomy are the novel techniques in the **OP-DPCI** column of Figure 6.9 that were inspired by asking what techniques would fit in each cell. In addition, our taxonomy clarifies how the popup dial widget of **E** and the subtree-drag-out widget of **G** are novel: we know of no other examples that fit in the **AS-DPCI** column of Figure 6.9. It is now clear, however, how to populate this cell with more techniques of the same kind: give the user continuous control over discrete parameters, with visually smooth feedback that interpolates between the discrete states.

Finally, the most practical result of developing this taxonomy has been to provide the stimulus to generate several design guidelines for future design work. These guidelines are described in the next section.

6.3 Proposed Design Guidelines

The preceding analysis of parameter manipulation clarifies the kind of input required to control parameters, how to make state transitions smooth, and when rapid transitioning over sequences of states is possible. We now present a set of design guidelines (**DG**) for

interactive visualization, centred on the manipulation of parameters and the depiction of transitions. These guidelines were developed by reflecting on the experience gained from the case studies, and on the preceding analysis and taxonomy. They augment the design goals of section 2.2 by providing concrete suggestions for design, especially with respect to the two goals to ease input and to use smooth transitions. The manipulation of parameters, and the output to display during such manipulation, are arguably the most central issue in *interactive* visualization, since it is parameter manipulation that causes state changes.

In the below list, design guidelines **DG1**–**DG7** concern parameter manipulation, and **DG8**–**DG12** concern the depiction of smooth transitions. For some of the guidelines, we also point out improvements they suggest to the work done in the case studies.

DG1. **Identify the parameters.** As a design for an interactive visualization is being developed, identify the parameters of the visualization that will be available to the user for manipulation. Find where these parameters lie in Figures 6.3, 6.9 and 6.11. Check if these are the only parameters the user could need or want to manipulate. Try to identify parameters of the visualization that won't be available to the user, in part by examining portions of the taxonomy not covered by the design, and confirm that these parameters should be left out of the interface.

DG2. **Use smooth transitions.** Design the state transitions caused by parameter manipulation to be smooth, using either animation, interpolation, or real-time feedback.

DG3. **Check if nominal parameters can be made ordinal.** If a parameter that will be available is nominal, check if there might be a way to order its values and make it ordinal, even if the ordering is only temporary. Ordinal parameters allow for sequences of light-weight changes to the parameter, where the user can repeatedly increment or decrement the parameter through some simple input. For example, in

V, the pre-defined layers of data have an out-to-in ordering which allows repeated marking menu flicks to select or deselect layers.

Suggested improvement to **V***:* Given a volume data set where the pre-defined subsets have no fixed ordering, a temporary ordering might be imposed according to the z-depth of subsets with respect to the current camera view or current browsing tool. This would allow the user interface in **V** to handle data composed of subsets other than layers.

DG4. **Consider using history as a parameter.** The history of an object's state provides a potentially useful ordinal parameter to manipulate, especially if the cost of navigating between states may be high. For each subset (such as an active subset) or parameter in the visualization that will change or be changed over time, check if it would be useful to allow the user to visit previous values or states of this object. Such an object x can be viewed as a function $x(t)$ of time t, and varying the ordinal parameter t traverses the history $x(T), x(T-1), \ldots, x(0)$. This was done in **E** where x was the active subset of visualized nodes, however the notion of history makes sense in any interactive visualization.

Suggested improvement to **V** *and* **G***:* Modify the prototypes to record a history of states of the visualization, and allow the user to navigate this history with a technique such as the popup dial widget of **E**.

DG5. **With ordinal parameters, use discrete input for frequent, small changes; use continuous input to allow slowing down complex transitions.** If a parameter is ordinal, or can be made ordinal, then a choice must be made whether to allow manipulation of that parameter through discrete input, continuous input, or both. If the user will often want to make a single change to the parameter resulting in a single state transition (as opposed to a sequence of transitions), then discrete input is appropriate, due to the speed and ease of input it affords. On the

other hand, if the visual changes involved in state transitions are complex, and the user may want to slow down a transition to examine the changes, then continuous input is appropriate, for the control over speed that it affords. Notice that **E** allows for both of these ("Back" and "Forward" buttons as well as the popup dial widget for traversing history), as does **G** (marking menu flicks and the subtree-drag-out widget for ancestors and descendants), whereas **V** only allows discrete input to be used (to control which layers are selected).

Suggested improvement to **V***:* Allow layer selection in **V** to be controlled with continuous input, in addition to the current support for discrete input. Such continuous input could be faster than discrete input if the user wants to select several layers, e.g. causing them to peel away. Continuous input would also allow the user to optionally slow down the gradual deformation of layers as they are selected and deselected, allowing us to partially circumvent the tradeoff described in section 3.4.7 between having more control over the deformation applied to each layer, on the one hand, and less clutter from 3D widgets, on the other. Continuous control over selection would give the user a generic way of slowly deforming each layer in sequence without extra 3D widgets for each layer.

DG6. **Support rapid traversal of sequences of states.** If a parameter is ordinal or continuous, or can be made ordinal, then the interface should support rapid, incremental manipulation of that parameter, enabling traversal of sequences of states. As indicated in Figures 6.3, this can be done through discrete input via clicking or flicking (or repeatedly hitting a hotkey), or continuous input via linear dragging or circular dragging.

DG7. **With continuous input, use circular dragging for arbitrarily long dragging; use linear dragging for limited range drags and rapid dragging to endstates.** When traversing sequences of states with continuous input, if the se-

quences may be arbitrarily long, then circular dragging is appropriate, since only a small amount of space is required for the drag. (Circular dragging also has the advantage of allowing the C:D ratio to be varied continuously by changing the radius of the drag.) On the other hand, if the sequences have a limited length, and the user may often want to transition to a state at the end of such a sequence, then linear dragging may be best, to allow the user to quickly drag to the end of the sequence (this is done in **G** when dragging out all descendants or ancestors of a node), or alternatively a special mechanism involving discrete input (such as a virtual button or menu item) could be provided to "fast forward" to the end of a sequence.

Suggested improvement to **V***:* If continuous control over layer selection is provided in **V**, as suggested by **DG5**, and if we assume the number of layers in a volume data set to be small (e.g. less than 10), then *linear* dragging rather than circular dragging would be a good choice, to enable the user to quickly select or deselect *all* layers by dragging to endpoints.

DG8. **Perform transitions in stages, with one for each kind of graphical change.** If a transition involves multiple kinds of graphical changes, break the transition down into serial stages with one stage for each type of change.

DG9. **Use stages to reduce occlusion during a transition.** If the transition involves elements moving or changing together in such a way that there is occlusion between elements, consider breaking the transition down into serial stages to reduce occlusion. For example, in **V**, if a set of contiguous layers are peeled together, much of the data in these layers will be occluded by the front most layer(s). However, peeling the layers one-by-one allows the user to briefly see surfaces between the layers during the transition. Note that in **V**, strictly serial deformation of layers is not enforced (e.g. one layer may begin peeling before the adjacent layer has finished

peeling), however the user may cause the deformation of layers to be more parallel or more serial by flicking quickly or slowly, respectively, with the marking menu.

DG10. **Order stages to reduce complexity.** If some of the graphical elements are present in both the start and end states of the transition, then these elements should be represented in a continuous fashion throughout the transition, rather than disappearing and then reappearing. Furthermore, if the transition has multiple stages (due to **DG8** or otherwise), the stages should be ordered to reduce the complexity of output during as much of the transition as possible, by performing simplifications early, and by deferring the addition of complexity until later. A useful template for the ordering of stages is: first, stages that completely remove elements; second, those that simplify, collapse, or shrink elements; third, those that do not change the number of elements or the complexity of the instantaneous output, e.g. translating or rotating elements; fourth, those that elaborate, expand, or enlarge elements; and fifth, those that add whole new elements. This ordering eliminates as many distractors as possible during the middle stage(s) involving translation and rotation, freeing up the maximum of attentional resources to interpret the changes in the elements present throughout the transition. Applying this guideline results in the 5-stage animation in **E**, the 3-stage animation in **G**, and the 3-stage "trim, translate, and grow" animation in SpaceTree [Plaisant *et al.*, 2002].

DG11. **Design transitions to show relationships between start and end states.** If none of the graphical elements are present in both the start and end states of the transition, it may be less obvious how to smoothly transition from one to the other, or it may seem like there is no way to smoothly transition. Nevertheless, as a guideline, consider the possibility of displaying an intermediate state during the transition that would show elements from the start and end, to illustrate some relationship between the two end states. For example, when performing a large

pan across a 2D image, where the start and end views do not overlap and are distant from each other, rather than panning at high speed, it is usually preferable to zoom out to a point where both views and visible, and then zoom in to the new view [Furnas and Bederson, 1995; van Wijk and Nuij, 2003]. As another example, if the transition is from one subgraph to a disjoint subgraph, it may be good to expand and reveal nodes until both subgraphs are visible, and then collapse nodes until only the second subgraph is visible. Notice this requires an ordering of stages very different from that recommended in **DG10**, where at least some elements are present throughout the entire transition.

If it is not appropriate or possible to have an intermediate state showing the start and end states together, then consider using one of a few different kind of transitions to convey some information about the state change. For example, in a web browser, when the user clicks on a link to transition from one webpage to another that completely replaces the first, a smooth transition may seem useless or ill-defined. However, the pathnames in URL addresses of webpages induce a partial ordering corresponding to a tree, with well-defined notions of moving up, down, or sideways through the tree. So, one possibility would be to show brief animations of a webpage sliding out of the window and a new one sliding in, with the direction of these sliding motions indicating the user's movement within the tree. Another example is the use of "Page Up" and "Page Down" keys to move by a screenfull when viewing a document. As long as the user is aware of the key they have pressed, they may never be confused about the direction of scrolling, whether the scrolling is animated or not. However, without animation, such scrolling is disorienting for a 3rd party observing the user's screen. So again, although the start and final states may not overlap or show any information in common, the use of two different animated transitions (smooth scroll up, smooth scroll down) can impart useful information. Schlienger et al. [2006] describe other ways of imparting information with different

kinds of transitions.

DG12. Hilite important elements during transitions. Hilite important elements or focal nodes during the transition to make them pop out preattentively during the transition. For example, if the user initiates a state change by selecting a focal element, it should be easy to track changes in the position of this node during the transition, and hiliting the node is one way to facilitate this.

Suggested improvement to **V**, **E**, *and* **G**: Hiliting of elements could be implemented or improved in the prototypes of all three case studies. In **V**, if the user flicks rapidly enough, multiple layers will animate simultaneously, and in this case the system could hilite the most recently selected or deselected layer during its animation. In **E**, there is currently no hiliting of nodes during transitions. In **G**, during a subtree rotation, the selected node is hilited, but there is little contrast between the hilited colour and colour of other nodes, so this could be improved.

Although the taxonomy and guidelines clarify when to use input such as clicking, flicking, linear dragging, or circular dragging, we leave it up to the designer to decide exactly how to incorporate these input methods into a complete interaction technique, which may involve a popup widget, context sensitive invocation, additional gestures, and appropriate visual feedback. The interaction techniques used or developed in the case studies (marking menus, popup dial widget, subtree-drag-out widget) provide useful examples. Additional examples that designers might use or modify include control menus [Pook *et al.*, 2000], flow menus [Guimbretière and Winograd, 2000], FaST Sliders [McGuffin *et al.*, 2002], which can all be invoked over a particular on-screen object and popped up to choose a parameter (of the object) to be manipulated. Another simpler example is a class of specialized numeric field widgets (such as those common in software products from Discreet Logic, circa 2000) where the user can click down on a field and drag left or right out of the field to increase or decrease the number in the field; the entire drag

thus combines selection of the parameter to adjust and the adjustment. A final example for consideration is the bimanual toolglass [Bier *et al.*, 1993], enabling a user to select a noun (object or data) and verb (operation) simultaneously. Such a toolglass might be used in **G**, where the noun is a node in the graph, and the verbs in the toolglass could include buttons for expanding or collapsing children or parents, buttons for expanding or collapsing all descendants or ancestors of a node, and buttons for initiating drags that expand or collapse subtrees of descendants or ancestors, replacing the subtree-drag-out widget.

In thinking about **DG5**, an interesting idea arose for enabling rapid discrete input combined with optional control over the speed of a single state transition, through a unified interaction technique. In **V** and **G**, flicks within a normal radial menu are used to change a discrete parameter. Now, rather than a radial menu, imagine a modified control menu [Pook *et al.*, 2000] with two critical radii r_1 and $r_2 > r_1$. The user pops up the menu, drags radially outward to select a parameter, and if the user continues to drag (ballistically) past r_1 and r_2, then the menu item is selected as in a normal radial menu and the state transition ensues. On the other hand, if the user drags into $[r_1, r_2]$, the user can then drag back and forth between r_1 and r_2, to scrub over the transition between the two states. Thus, the user may choose to invoke a rapid state transition, or slow and stop the transition at any point. The disadvantage of this technique is that it is not as well suited for rapid traversal of sequences of states, since each transition requires a flick longer than r_2. However, this problem could be circumvented by making r_2 small and close to r_1, and dynamically expanding $[r_1, r_2]$ if the user dwells within the interval, to ease continuous control when it is desired. The $[r_1, r_2]$ interval would then be a kind of expanding widget that expands radially to ease manipulation of the parameter, but not to ease its initial acquisition as has been done in previous work [McGuffin and Balakrishnan, 2005a].

Another idea for future work came from thinking about **DG4**. A common issue with

history in general is that, by revisiting former states and branching off in new directions, users effectively visit a tree of states. The popup dial widget in **E** only allows a path of states to be revisited, from the most recent state back to the root (initial state). An interesting problem is to design a more sophisticated interaction technique for revisiting any node in the tree of history, perhaps with a single drag or gesture, whose shape indicates which branch(es) to follow. Among other issues, the design of such a technique would involve balancing the amount of visual feedback given on the tree of history, and the visualization whose history is being traversed!

One notable counterexample to the approach suggested by **DG11** is the system described by [Burtnyk *et al.*, 2006], which uses "cut/fade transitions" rather than continuous camera motion to move between places of interest. This is partly justified by the desire to achieve emotional, cinematic effects, and also because the object or scene being viewed is familiar enough to the user that a change of shot is unlikely to be disorienting.

Chapter 7

Conclusions and Future Directions

7.1 Summary

In our pursuit of achieving and understanding highly interactive visualizations, we have presented three guiding design goals (section 2.2) and demonstrated a variety of approaches to them in three case studies. Each case study involved designing and implementing novel interaction techniques for visualization.

In the first case study, deformations were used to manage occlusion in 3D, to reveal the interior of volumetric data while retaining surrounding contextual data on screen. Six new browsing tools were developed using deformations, layer-based interaction, 3D widgets, and marking menus. Tradeoffs and design issues specific to the use of deformations for browsing were identified, and initial user feedback from domain experts is positive, and indicates that the aspect most in need of improvement is the resolution and quality of the rendering.

In the second case study, a general algorithm called Expand-Ahead was developed for filling space with nodes of a tree. Many different graphical representations of a tree could be used with this algorithm, and two were implemented in prototype form: one with a 1D indented outline representation, the other with a 2D nested containment representation.

A novel popup dial widget for scrubbing over state transitions was also implemented. A theoretical model of user performance in the 1D case was derived from Fitts' law. This model, while not empirically confirmed, indicates that it is critically the speed with which the user can visually find a target node that determines whether Expand-Ahead is worthwhile or not in drill-down tasks. A controlled experimental evaluation showed that the Expand-Ahead technique can significantly improve user performance in some cases, but can also significantly hinder it in other cases.

The third case study examined how to improve the presentation and layout of a class of graphs that model genealogical relationships. Graph-theoretic and layout-related properties of genealogical graphs were characterized in more detail than we have found in previous work. Four new graphical representations were proposed, two of which were prototyped. Initial user feedback collected so far from a domain expert is inconclusive, but indicates that one aspect to improve is the depiction of nuclear families. Analytically, the representations that were prototyped generalize previous representations, and do so with no edges crossings, with crowding no worse than that in ordinary node-link representations of trees, and while maintaining alignment of nodes by generation. A novel subtree-drag-out widget was also developed for expanding or collapsing subsets of nodes in a single drag.

Finally, the case studies were compared and analyzed, which led to the development of a taxonomy that classifies ways of manipulating the parameters of a visualization and that identifies many low-level details of interaction. Twelve concrete, prescriptive design guidelines concerning parameter manipulation and the depiction of smooth transitions were proposed (section 6.3).

7.2 Contributions

The case studies demonstrate that highly interactive and novel visualizations can be designed in part by considering our design goals of increasing or improving output, easing input, and using smooth transitions. Although each case study involved a different approach to achieving the design goals, our work does not exhaustively explore or even enumerate all possible approaches to the goals, suggesting that many novel designs for highly interactive visualization remain to be discovered. At the same time, we feel that the case studies sample a wide enough variety of approaches to serve as useful examples for many future designs.

Each case study introduces a novel interaction technique enabling rapid traversal of sequences of discrete states with visually continuous feedback. The two more novel of these techniques use continuous input to transition across discrete states and give the user control over the speed of the transition, placing them in column **AS-DPCI** in Figure 6.9 and column **AS-OPCI** in Figure 6.11. We are not aware of any previous interaction techniques that fall in these columns, however there is no reason that simliar techniques could not be used in many future visualizations.

Two of the case studies also provided novel examples of how to exploit subsets in the data to facilitate interaction: the ordered layers in volumetric data, and the trees of ancestors and descendants in genealogical graphs, were both instrumental in enabling rapid interaction. Recent work by other researchers [Beaudouin-Lafon, 2001; Dragicevic, 2004; Agarawala and Balakrishnan, 2006] show that layers and piles in other kinds of data continue to enable promising interaction techniques, and the subtree-drag-out widget could be adapted for selection of neighbourhoods in more general kinds of graphs.

Of the three case studies, perhaps the one with the most impact to date is the first. Occlusion in 3D is a general problem with many visualizations, and the use of deformations for managing or reducing occlusion is still not widespread at the time of writing. Our original paper reporting the work [McGuffin *et al.*, 2003] appears to have helped fuel

an interest among researchers in deformations and related techniques for visualization — an online search found over 20 citations of our paper in the academic literature published in 2004–2005. An additional contribution of the first case study in this dissertation, not included in our original paper, is the survey of techniques for managing occlusion in Appendix A, which includes a simple taxonomy of techniques in section A.1, positioning the work of the first case study within a wider context.

The taxonomy in Chapter 6 allowed us to articulate a conceptual connection between direct manipulation and smooth transitions, and also made clear in what way the popup dial widget and subtree-drag-out widget are novel, and also pointed to additional novel interaction techniques (in the **OP-DPCI** column of Figure 6.9). Compared to previous taxonomies for visualization (section 6.2.4), our taxonomy of parameter manipulation examines in more detail low-level aspects, i.e. it focuses on the temporally fine grain interaction involved with adjustment of individual parameters. Such a low-level focus has limited scope, but is clearly important in the context of highly interactive visualization where there is a tight feedback loop between the user and the computer system.

Lastly, the guidelines proposed in section 6.3 suggested a number of improvements to the case studies (see **DG3**, **DG4**, **DG5**, **DG7**, and **DG12**), demonstrating that the guidelines stimulate design insight. These guidelines would be useful in future design work, and could also eventually contribute to the development of a systematic methodology for designing interactive visualizations.

7.3 Future Directions

7.3.1 Using Deformations for Browsing Volumetric Data

The current software prototype only allows one deformation tool to be used at a time, and can only combine deformations in fixed, hard-coded ways. For example, the Leafer tool combines many techniques of Figure 3.2 in one particular order, but many other orderings

or combinations are possible. A useful extension would be to reduce deformations to their simplest orthogonal components, and allow users to combine them in a piecewise fashion over different regions of space, and/or in a hierarchical fashion on a single region of space. More flexible tools could be designed that allow the user to dynamically combine primitive operations or widgets, to construct their own custom tools that compose the effect of multiple deformations.

The layers in the current prototype are defined offline. Interaction techniques could be added for dynamically changing or redefining the layers or subsets of the data, perhaps using soft-segmentation with an interface similar to [Kniss *et al.*, 2001].

Some aspects of the current design make use of the layers in the data to simplify interaction by creating implicit constraints. We have so far assumed a fixed ordering for these layers, however, many data sets contain subsets (e.g. internal organs or vascular structures) that don't have a fixed inner-to-outer ordering or that don't even resemble layers. One possibility to handle such data with similar interaction techniques is to have a temporary ordering of subsets computed by raycasting from the current camera view and finding the order in which subsets are encountered. With such an ordering, the user could peel or otherwise deform layers from front-to-back, to work their way toward portions of data further from their point of view.

In general, deformations that have many parameters (e.g. parameters for each component of the deformation, or for each layer) can clutter the screen with 3D widgets. Ideally, a widget should only be visible when the user wants to interact with it. Popup 3D widgets, or use of gestures instead of widgets, might allow the user to summon and invoke interface elements when needed to eliminate this problem. Another way to partially reduce clutter, and to ease selection of widgets, would be to create expanding 3D widgets that are easier to select when the user needs them but that occlude less data when not in use [McGuffin and Balakrishnan, 2005a].

In the case study, we only considered scalar volumetric data, however working with

other kinds of data could lead to new innovations. These could include non-scalar volume data such as vector fields (in which case it might be important to show how curves such as fluxlines or streamlines are connected across cuts that split open the volume), or time-dependent data, or even non-volumetric data such as architectural models.

Higher resolution data sets could be supported using incremental front-buffer rendering that progressively refines the visual quality of the output without sacrificing responsiveness. Even better quality output would be produced with hardware accelerated volume rendering (e.g. [Rezk-Salama *et al.*, 2001]). We did not implement volume rendering in our prototype due to the complications that would arise from deforming the data, however ongoing advances in graphics hardware and in research into new rendering techniques with such hardware suggests that our interaction techniques could be combined with true volume rendering at interactive frame rates.

Finally, a promising application for our browsing techniques is in the teaching of anatomy. Refining our software for a pedagogical setting and observing its use by teachers and/or students would provide a real-world setting to drive improvements and perform evaluation.

7.3.2 Expand-Ahead

Section 4.9 listed a few possibilities for future work: supporting notions of sticky or hard expansion as well as soft expansion; allowing the user to lock the positions of nodes for a persistent layout; using uniform expansion and/or partial expansion with elision; and improvements in graphic design to better support the user's perceptual interpretation of output.

In addition, new heuristics for expansion could be developed and tested, perhaps based on frequency, or on search strings or wildcard filters provided by the user to indicate the type of node they are interested in.

More controlled experiments could be conducted to confirm and/or refine our model

of user performance based on Fitts' law. Tasks other than drill-down could also be tested, to see if expand-ahead can facilitate more general navigation and browsing tasks.

It would also be interesting to apply expand-ahead to other tree representations, such as various node-link style layouts, and to adapt the expand-ahead algorithm for more general kinds of graphs or data structures where elements can be collapsed or expanded.

7.3.3 Genealogical Graphs

Dual-trees can show many generations vertically, but have a horizontal extent limited to ancestors or descendants of 2 root nodes. Although these roots can be changed interactively to browse a data set, only a fraction of a large data set may be visible at any given time. To increase the horizontal extent of nodes shown, without introducing long or crossed edges, the dual-tree scheme could be generalized to a sequence of N trees, laid out left-to-right and alternating between ancestor and descendant trees, all shown at once. Another possibility is to embed combinations of ancestor and descendant trees in 3D, e.g. by arranging intersecting trees on perpendicular planes. Use of 3D could eliminate the need to rotate subtrees, possibly giving the user a more consistent view of the data.

It would also be useful to have graphical representations that are oriented toward higher-level groupings of individuals, such as family units. For example, a viewer for a dual-tree $A(x) \cup D(y)$ could be augmented to also show siblings of nodes in $A(x)$, and spouses of nodes in $D(y)$. This would increase the crowding of nodes somewhat, but would make complete nuclear families visible.

The subtree-drag-out widget could be adapted for other kinds of graphs, allowing neighbourhoods of nodes to be selected (perhaps with something like lasso selection, or by dragging out a 2D region that is constrained along the direction of edges) and then collapsed, displaced, or manipulated.

Rather than always displaying a subset of the genealogical graph, another valid direc-

tion to pursue is to develop a layout for entire genealogical graphs that allows interactive adjustment of the tradeoffs between generational alignment, edge placement, and other aspects. To perform such adjustment, perhaps the user would drag node(s) or nuclear families from one location to another to roughly suggest how to rearrange things, and in response the layout algorithm would recompute the positions of nodes and edges so they again adopt neatly chosen positions.

Finally, visualizations might be designed to emphasize other aspects of genealogical data, such as complicated family relationships, distance between pairs of individuals, "outliers" or unusual relationships (e.g. intermarriage), and geographic migration of families over time [Kapler and Wright, 2004].

7.3.4 Beyond the Case Studies

We invite researchers in scientific and information visualization to study examples of popup widgets, menuing techniques, gestural input, and other input techniques in HCI, and to think about adapting them for interfaces for visualizing data. It seems likely that many opportunities for innovation remain here, and there are probably new dimensions that could be added to the taxonomy of parameter manipulation (Chapter 6) that have not yet been considered.

Continued advances in computer hardware and software will inevitably be accompanied by an accumulation of more examples of highly interactive visualizations. As interaction in visualizations becomes more rapid and state transitions more frequent, the design of the depiction and control of transitions will become more important. With this comes the opportunity to develop a unified understanding of the various taxonomies for visualization that have been proposed, in this work and in work by others, and to develop more complete design guidelines or design methodologies that build on those proposed so far.

Bibliography

[Accot and Zhai, 1997] Johnny Accot and Shumin Zhai. Beyond fitts' law: Models for trajectory-based hci tasks. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 295–302, 1997.

[Accot and Zhai, 2002] Johnny Accot and Shumin Zhai. More than dotting the i's — foundations for crossing-based interfaces. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 73–80, 2002.

[Achugbue, 1981] James O. Achugbue. On the line breaking problem in text formatting. In *Proceedings of ACM SIGPLAN SIGOA Symposium on Text Manipulation*, pages 117–122, 1981.

[Agarawala and Balakrishnan, 2006] Anand Agarawala and Ravin Balakrishnan. Keepin' it real: Pushing the desktop metaphor with physics, piles and the pen. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 1283–1292, 2006.

[Agrawala *et al.*, 2000] Maneesh Agrawala, Denis Zorin, and Tamara Munzner. Artistic multiprojection rendering. In *Proceedings of Eurographics Workshop on Rendering Techniques*, pages 125–136, 2000.

[Agrawala *et al.*, 2003] Maneesh Agrawala, Doantam Phan, Julie Heiser, John Haymaker, Jeff Klingner, Pat Hanrahan, and Barbara Tversky. Designing effective step-

by-step assembly instructions. In *Proceedings of ACM SIGGRAPH*, pages 828–837, 2003.

[Agur and Lee, 1999] Anne M. R. Agur and Ming J. Lee, editors. *Grant's Atlas of Anatomy*. Lippincott, Williams and Wilkins, 10th edition, 1999.

[Amar and Stasko, 2004] Robert Amar and John Stasko. A knowledge task-based framework for design and evaluation of information visualizations. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)*, pages 143–149, 2004.

[Amar *et al.*, 2005] Robert Amar, James Eagan, and John Stasko. Low-level components of analytic activity in information visualization. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)*, pages 111–117, 2005.

[Anton, 1988] Howard Anton. *Calculus, with Analytic Geometry*. John Wiley and Sons, 3rd edition, 1988.

[Baecker and Small, 1990] Ronald M. Baecker and Ian Small. Animation at the interface, 1990. A chapter (pp. 251–267) in Brenda Laurel, editor, The Art of Human-Computer Interface Design, Addison-Wesley.

[Baecker, 1967] Ronald M. Baecker. Planar representations of complex graphs. Technical Report 1967-1, Lincoln Laboratory MIT, February 1967.

[Barlow and Neville, 2001] Todd Barlow and Padraic Neville. A comparison of 2-d visualizations of hierarchies. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)*, pages 131–138, 2001.

[Bartram *et al.*, 1995] Lyn Bartram, Albert Ho, John Dill, and Frank Henigman. The continuous zoom: A constrained fisheye technique for viewing and navigating large information spaces. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST)*, pages 207–215, 1995.

[Bartram, 1997] Lyn Bartram. Can motion increase user interface bandwidth? In *Proceedings of IEEE Conference on Systems, Man, and Cybernetics*, pages 1686–1692, 1997.

[Bartrolí *et al.*, 2001] Anna Vilanova Bartrolí, Rainer Wegenkittl, Andreas König, and Eduard Gröller. Nonlinear virtual colon unfolding. In *Proceedings of IEEE Visualization (VIS)*, 2001.

[Baudisch and Gutwin, 2004] Patrick Baudisch and Carl Gutwin. Multiblending: displaying overlapping windows simultaneously without the drawbacks of alpha blending. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 367–374, 2004.

[Beaudouin-Lafon and Mackay, 2000] Michel Beaudouin-Lafon and Wendy E. Mackay. Reification, polymorphism and reuse: Three principles for designing visual interfaces. In *Proceedings of ACM Advanced Visual Interfaces (AVI)*, 2000.

[Beaudouin-Lafon, 2001] Michel Beaudouin-Lafon. Novel interaction techniques for overlapping windows. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST)*, pages 153–154, 2001.

[Beddow, 1990] Jeff Beddow. Shape coding of multidimensional data on a microcomputer display. In *Proceedings of IEEE Visualization (VIS)*, pages 238–246, 1990.

[Bederson and Hollan, 1994] Benjamin B. Bederson and James D. Hollan. Pad++: A zooming graphical interface for exploring alternate interface physics. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST)*, pages 17–26, 1994.

[Bell and Feiner, 2000] Blaine A. Bell and Steven K. Feiner. Dynamic space management for user interfaces. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST)*, pages 239–248, 2000.

[Bell *et al.*, 2001] Blaine Bell, Steven Feiner, and Tobias Höllerer. View management for virtual and augmented reality. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST)*, pages 101–110, 2001.

[Bertin, 1967] Jacques Bertin. *Sémiologie graphique: Les diagrammes, Les réseaux, Les cartes*. Éditions Gauthier-Villars, Paris, 1967. (2nd edition 1973, English translation 1983).

[Bertin, 1977] Jacques Bertin. *La graphique et le traitement graphique de l'information*. Flammarion, Paris, 1977.

[Bier *et al.*, 1993] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. Toolglass and magic lenses: The see-through interface. In *Proceedings of ACM SIGGRAPH*, pages 73–80, 1993.

[Bier *et al.*, 1994] Eric A. Bier, Maureen C. Stone, Ken Fishkin, William Buxton, and Thomas Baudel. A taxonomy of see-through tools. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 358–364, 1994.

[Blizzard Entertainment, 2000] Blizzard Entertainment. Diablo II, 2000. http://www.blizzard.com/diablo2/. Accessed June 2005.

[Brill *et al.*, 1994] Manfred Brill, Hans Hagen, Hans-Christian Rodrian, Wladimir Djatschin, and Stanislav V. Klimenko. Streamball techniques for flow visualization. In *Proceedings of IEEE Visualization (VIS)*, pages 225–231, 1994.

[Bruyns and Montgomery, 2002] C. D. Bruyns and K. Montgomery. Generalized interactions using virtual tools within the Spring framework: Cutting. In *MMVR Medicine Meets Virtual Reality*, 2002.

[Buchheim *et al.*, 2002] Christoph Buchheim, Michael Jünger, and Sebastian Leipert. Improving Walker's algorithm to run in linear time. In *Proceedings of Symposium on Graph Drawing (GD)*, pages 344–353, 2002.

[Burtnyk *et al.*, 2002] Nicholas Burtnyk, Azam Khan, George W. Fitzmaurice, Ravin Balakrishnan, and Gordon Kurtenbach. StyleCam: Interactive stylized 3D navigation using integrated spatial & temporal controls. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST)*, pages 101–110, 2002.

[Burtnyk *et al.*, 2006] Nicolas Burtnyk, Azam Khan, George Fitzmaurice, and Gordon Kurtenbach. ShowMotion: Camera motion based 3D design review. In *Proceedings of ACM Symposium on Interactive 3D Graphics (I3D)*, 2006.

[Buxton, 1986] William Buxton. Chunking and phrasing and the design of human-computer dialogues. In *Proceedings of IFIP World Computer Congress*, pages 475–480, 1986.

[Cabral and Leedom, 1993] Brian Cabral and Leith (Casey) Leedom. Imaging vector fields using line integral convolution. In *Proceedings of ACM SIGGRAPH*, pages 263–270, 1993.

[Callahan *et al.*, 1988] Jack Callahan, Don Hopkins, Mark Weiser, and Ben Shneiderman. An empirical comparison of pie vs. linear menus. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 95–100, 1988.

[Card and Mackinlay, 1997] Stuart K. Card and Jock D. Mackinlay. The structure of the information visualization design space. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)*, pages 92–99,125, 1997.

[Card *et al.*, 1996] Stuart K. Card, George G. Robertson, and William York. The WebBook and the Web Forager: An information workspace for the world-wide web. In

Proceedings of ACM Conference on Human Factors in Computing Systems (CHI), pages 111–117, 1996.

[Card *et al.*, 1999] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman, editors. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers, 1999.

[Card, 2003] Stuart K. Card. Information visualization, 2003. Chapter 28 (pp. 544–582) of Julie A. Jacko and Andrew Sears, editors, The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications, Lawrence Erlbaum Associates.

[Carpendale and Montagnese, 2001] M. Sheelagh T. Carpendale and Catherine Montagnese. A framework for unifying presentation space. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST)*, pages 61–70, 2001.

[Carpendale *et al.*, 1997a] M. Sheelagh T. Carpendale, David J. Cowperthwaite, and F. David Fracchia. Extending distortion viewing from 2D to 3D. *IEEE Computer Graphics and Applications (CG&A): Special Issue on Information Visualization*, 17(4):42–51, 1997.

[Carpendale *et al.*, 1997b] M. Sheelagh T. Carpendale, David J. Cowperthwaite, and F. David Fracchia. Making distortions comprehensible. In *Proceedings of IEEE Symposium on Visual Languages (VL)*, pages 36–45, 1997.

[Carpendale *et al.*, 1999] M. Sheelagh T. Carpendale, D. J. Cowperthwaite, M. Tigges, A. Fall, and F. D. Fracchia. The Tardis: A visual exploration environment for landscape dynamics. In *Proceedings of SPIE Conference on Visual Data Exploration and Analysis VI*, pages 110–119, 1999.

[Carpendale, 1999] M. Sheelagh T. Carpendale. *A Framework for Elastic Presentation Space*. PhD thesis, School of Computing Science, Simon Fraser University, Burnaby, Canada, March 1999.

[Chambers *et al.*, 1983] John M. Chambers, William S. Cleveland, Beat Kleiner, and Paul A. Tukey. *Graphical Methods for Data Analysis*. Wadsworth, 1983.

[Chen *et al.*, 2003] Min Chen, Deborah Silver, A. S. Winter, Vikas Singh, and N. Cornea. Spatial transfer functions — a unified approach to specifying deformation in volume modeling and animation. In *Proceedings of Eurographics/IEEE TVCG Workshop on Volume Graphics (VG)*, pages 35–44,163, 2003.

[Chernoff, 1973] Herman Chernoff. The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association*, 68(342):361–368, June 1973.

[Chi and Riedl, 1998] Ed H. Chi and John T. Riedl. An operator interaction framework for visualization systems. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)*, 1998.

[Chi, 2000] Ed H. Chi. A taxonomy of visualization techniques using the data state reference model. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)*, pages 69–75, 2000.

[Chorin and Marsden, 1993] Alexandre J. Chorin and Jerrold E. Marsden. *A Mathematical Introduction to Fluid Mechanics*. Springer-Verlag, 3rd edition, 1993.

[Chuah and Roth, 1996] Mei C. Chuah and Steven F. Roth. On the semantics of interactive visualizations. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)*, pages 29–36, 1996.

[Cleveland and McGill, 1988] William S. Cleveland and Marylyn E. McGill, editors. *Dynamic Graphics for Statistics*. Wadsworth, 1988.

[Cleveland, 1985] William S. Cleveland. *The Elements of Graphing Data*. Hobart Press, 1985.

[Cleveland, 1993] William S. Cleveland. *Visualizing Data*. Hobart Press, 1993.

[Cohen and Brodlie, 2004] Marcelo Cohen and Ken Brodlie. Focus and context for volume visualization. In *Proceedings of the Theory and Practice of Computer Graphics 2004 (TPCG'04)*, pages 32–39, 2004.

[Colby and Scholl, 1991] Grace E. Colby and Laura R. Scholl. Transparency and blur as selective cues for complex visual information. In *Proceedings of SPIE Vol. 1460 Image Handling and Reproduction Systems Integration*, pages 114–125, 1991.

[Coleman, 2004] Patrick Sean Coleman. Interactive control of nonlinear projection for complex animated scenes. Master's thesis, Department of Computer Science, University of Toronto, Toronto, Canada, November 2004.

[Conner *et al.*, 1992] D. Brookshire Conner, Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, Robert C. Zeleznik, and Andries van Dam. Three-dimensional widgets. In *Proceedings of ACM Symposium on Interactive 3D Graphics (I3D)*, pages 183–188, 1992.

[Cormen *et al.*, 1990] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[Cox *et al.*, 1998] Donald A. Cox, Jasdeep S. Chugh, Carl Gutwin, and Saul Greenberg. The usability of transparent overview layers. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI) conference summary*, pages 301–302, 1998.

[Davis, 1995] Marc Davis. Media streams: An iconic visual language for video representation, 1995. In Ronald M. Baecker, Jonathan Grudin, William A. S. Buxton, and Saul Greenberg, editors, Readings in Human-Computer Interaction: Toward the Year 2000, Morgan Kaufmann Publishers.

[de Leeuw and van Wijk, 1993] Willem C. de Leeuw and Jarke J. van Wijk. A probe for local flow field visualization. In *Proceedings of IEEE Visualization (VIS)*, pages 39–45,CP–5, 1993.

[Di Battista *et al.*, 1999] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

[Dragicevic, 2004] Pierre Dragicevic. Combining crossing-based and paper-based interaction paradigms for dragging and dropping between overlapping windows. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST)*, pages 193–196, 2004.

[Drebin *et al.*, 1988] Robert A. Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. In *Proceedings of ACM SIGGRAPH*, pages 65–74, 1988.

[Driskill and Cohen, 1995] Elana Driskill and Elaine Cohen. Interactive design, analysis, and illustration of assemblies. In *Proceedings of ACM Symposium on Interactive 3D Graphics (I3D)*, pages 27–33, 1995.

[Eades *et al.*, 1993] Peter Eades, Tao Lin, and Xuemin Lin. Two tree drawing conventions. *International Journal of Computational Geometry and Applications*, 3(2):133–153, 1993.

[Eades, 1984] Peter Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[Engel *et al.*, 2001] Klaus Engel, Martin Kraus, and Thomas Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *SIGGRAPH/EUROGRAPHICS Workshop On Graphics Hardware*, 2001.

[Fitts, 1954] Paul M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6):381–391, June 1954. (Reprinted in Journal of Experimental Psychology: General, 121(3):262–269, 1992).

[Frick *et al.*, 1994] A. Frick, A. Ludwig, and H. Mehldau. A fast adaptive layout algorithm for undirected graphs. In *Proceedings of Symposium on Graph Drawing (GD)*, 1994.

[Furnas and Bederson, 1995] George W. Furnas and Benjamin B. Bederson. Space-scale diagrams: Understanding multiscale interfaces. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 234–241, 1995.

[Furnas and Zacks, 1994] George W. Furnas and Jeff Zacks. Multitrees: Enriching and reusing hierarchical structure. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 330–336, 1994.

[Furnas, 1981] George W. Furnas. The FISHEYE view: a new look at structured files. Technical Report #81-11221-9, AT&T Bell Laboratories, Murray Hill, New Jersey, 1981. 19 pages.

[Furnas, 1986] George W. Furnas. Generalized fisheye views. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 16–23, 1986.

[GenoPro Inc., 2005] GenoPro Inc. GenoPro, 2005. http://www.genopro.com/. Accessed April 2005.

[Gibson *et al.*, 1998] S. Gibson, C. Fyock, E. Grimson, T. Kanade, R. Kikinis, H. Lauer, N. McKenzie, A. Mor, S. Nakajima, H. Ohkami, R. Osborne, J. Samosky, and K. Sawada. Volumetric object modeling for surgical simulation. *Medical Image Analysis*, 2(2):121–132, 1998.

[Grossman *et al.*, 2001] Tovi Grossman, Ravin Balakrishnan, Gordon Kurtenbach, George Fitzmaurice, Azam Khan, and William Buxton. Interaction techniques for 3D modeling on large displays. In *Proceedings of ACM Symposium on Interactive 3D Graphics (I3D)*, pages 17–23, 2001.

[Guimbretière and Winograd, 2000] François Guimbretière and Terry Winograd. FlowMenu: Combining command, text, and data entry. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST)*, pages 213–216, 2000.

[Guralnik, 1984] David B. Guralnik, editor. *Webster's New World Dictionary*. Warner Books, Inc., 2nd edition, 1984.

[Gutwin *et al.*, 2003] Carl Gutwin, Jeff Dyck, and Chris Fedak. The effects of dynamic transparency on targeting performance. In *Proceedings of Graphics Interface (GI)*, 2003.

[Harel, 1988] David Harel. On visual formalisms. *Communications of the ACM (CACM)*, 31(5):514–530, May 1988.

[Harrison and Vicente, 1996] Beverly L. Harrison and Kim J. Vicente. An experimental evaluation of transparent menu usage. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 391–398, 1996.

[Harrison *et al.*, 1995] Beverly L. Harrison, Gordon Kurtenbach, and Kim J. Vicente. An experimental evaluation of transparent user interface tools and information content. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST)*, pages 81–90, 1995.

[Henze, 1998] Chris Henze. Feature detection in linked derived spaces. In *Proceedings of IEEE Visualization (VIS)*, pages 87–94,522, 1998.

[Herman and Liu, 1979] G. T. Herman and H. K. Liu. Three-dimensional display of human organs from computed tomograms. *Computer Graphics and Image Processing*, 9:1–21, 1979.

[Herman *et al.*, 2000] Ivan Herman, Guy Melançon, and M. Scott Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 6(1):24–43, January 2000.

[Herndon and Meyer, 1994] Kenneth P. Herndon and Tom Meyer. 3D widgets for exploratory scientific visualization. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST)*, pages 69–70, 1994.

[Hinckley *et al.*, 1998] Ken Hinckley, Randy Pausch, Dennis Proffitt, and Neal F. Kassell. Two-handed virtual manipulation. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 5(3):260–302, September 1998.

[Hochheiser and Shneiderman, 2002] Harry Hochheiser and Ben Shneiderman. A dynamic query interface for finding patterns in time series data. In *Extended abstracts of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 522–523, 2002.

[Hoffman, 1999] Gary B. Hoffman. Genealogy in the new times, 1999. http://www.genealogy.com/genealogy/61_gary.html.

[Höhne *et al.*, 1992] Karl Heinz Höhne, Michael Bomans, Martin Riemer, Rainer Schubert, Ulf Tiede, and Werner Lierse. A volume-based anatomical atlas. *IEEE Computer Graphics and Applications (CG&A)*, 12(4):72–78, July 1992.

[Hultquist, 1992] J. P. M. Hultquist. Constructing stream surfaces in steady 3D vector fields. In *Proceedings of IEEE Visualization (VIS)*, pages 171–177, 1992.

[Igarashi and Hinckley, 2000] Takeo Igarashi and Ken Hinckley. Speed-dependent automatic zooming for browsing large documents. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST)*, pages 139–148, 2000.

[Interrante and Grosch, 1997] Victoria Interrante and Chester Grosch. Strategies for effectively visualizing 3D flow with volume LIC. In *Proceedings of IEEE Visualization (VIS)*, pages 421–424, 1997.

[Interrante *et al.*, 1996] Victoria Interrante, H. Fuchs, and S. Pilzer. Illustrating transparent surfaces with curvature-directed strokes. In *Proceedings of IEEE Visualization (VIS)*, pages 211–218, 1996.

[Ishak and Feiner, 2004] Edward W. Ishak and Steven K. Feiner. Interacting with hidden content using content-aware free-space transparency. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST)*, pages 189–192, 2004.

[Islam *et al.*, 2004] Shoukat Islam, Swapnil Dipankar, Deborah Silver, and Min Chen. Spatial and temporal splitting of scalar fields in volume graphics. In *Proceedings of IEEE Symposium on Volume Visualization and Graphics (VolVis)*, pages 87–94, 2004.

[Johnson and Shneiderman, 1991] Brian Johnson and Ben Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *Proceedings of IEEE Visualization (VIS)*, pages 284–291, 1991.

[Kadmon and Shlomi, 1978] Naftali Kadmon and Eli Shlomi. A polyfocal projection for statistical surfaces. *The Cartographic Journal*, 15(1):36–41, June 1978.

[Kajiya and Von Herzen, 1984] James T. Kajiya and Brian P. Von Herzen. Ray tracing volume densities. In *Proceedings of ACM SIGGRAPH*, pages 165–174, 1984.

[Kandogan and Shneiderman, 1996] Eser Kandogan and Ben Shneiderman. Elastic windows: Improved spatial layout and rapid multiple window operations. In *Proceedings of ACM Advanced Visual Interfaces (AVI)*, pages 29–38, 1996.

[Kandogan and Shneiderman, 1997] Eser Kandogan and Ben Shneiderman. Elastic windows: Evaluation of multi-window operations. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, 1997.

[Kanitsar *et al.*, 2003] Armin Kanitsar, Rainer Wegenkittl, Dominik Fleischmann, and Meister Eduard Gröller. Advanced curved planar reformation: Flattening of vascular structures. In *Proceedings of IEEE Visualization (VIS)*, 2003.

[Kapler and Wright, 2004] Thomas Kapler and William Wright. GeoTime information visualization. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)*, pages 25–32, 2004.

[Khan *et al.*, 2005] Azam Khan, Ben Komalo, Jos Stam, George Fitzmaurice, and Gordon Kurtenbach. HoverCam: Interactive 3D navigation for proximal object inspection. In *Proceedings of ACM Symposium on Interactive 3D Graphics (I3D)*, 2005.

[Kindlmann and Durkin, 1998] Gordon Kindlmann and James W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Proceedings of IEEE Symposium on Volume Visualization and Graphics (VolVis)*, pages 79–86, 1998.

[Kirsh and Maglio, 1994] David Kirsh and Paul P. Maglio. On distinguishing epistemic from pragmatic action. *Cognitive Science*, 18(4):513–549, Oct-Dec 1994.

[Klassen and Harrington, 1991] R. Victor Klassen and Steven J. Harrington. Shadowed hedgehogs: A technique for visualizing 2D slices of 3D vector fields. In *Proceedings of IEEE Visualization (VIS)*, pages 148–153, 1991.

[Kleiner and Hartigan, 1981] Beat Kleiner and John A. Hartigan. Representing points in many dimensions by trees and castles. *Journal of the American Statistical Association*, 76(374):260–269, June 1981.

[Kniss *et al.*, 2001] Joe Kniss, Gordon Kindlmann, and Charles Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of IEEE Visualization (VIS)*, pages 255–262, 2001.

[Knuth, 1968] Donald Ervin Knuth. *The Art of Computer Programming, Volume I: Fundamental Algorithms*, pages 309–310. Addison-Wesley, 1968.

[Kosara *et al.*, 2001] Robert Kosara, Silvia Miksch, and Helwig Hauser. Semantic depth of field. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)*, pages 97–104, 2001.

[Krasner and Pope, 1988] G. E. Krasner and S. T. Pope. A description of the model-view-controller user interface paradigm in the Smalltalk-80 system. *Journal of Object Oriented Programming*, 1(3):26–49, 1988.

[Kurtenbach and Buxton, 1993] Gordon Kurtenbach and William Buxton. The limits of expert performance using hierarchic marking menus. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 482–487, 1993.

[Kurtenbach *et al.*, 1999] G. Kurtenbach, G. Fitzmaurice, R. Owen, and T. Baudel. The hotbox: Efficient access to a large number of menu-items. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, 1999.

[Kurzion and Yagel, 1997] Yair Kurzion and Roni Yagel. Interactive space deformation with hardware-assisted rendering. *IEEE Computer Graphics and Applications (CG&A)*, 17(5), 1997.

[Laidlaw, 1995] David H. Laidlaw. *Geometric Model Extraction from Magnetic Resonance Volume Data*. PhD thesis, California Institute of Technology, 1995.

[LaMar *et al.*, 2001] Eric LaMar, Bernd Hamann, and Kenneth I. Joy. A magnification lens for interactive volume visualization. In *IEEE Pacific Conference on Computer Graphics and Applications*, pages 223–232, 2001.

[Lamping and Rao, 1994] John Lamping and Ramana Rao. Laying out and visualizing large trees using a hyperbolic space. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST)*, pages 13–14, 1994.

[Lamping *et al.*, 1995] John Lamping, Ramana Rao, and Peter Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 401–408, 1995.

[Landauer and Nachbar, 1985] T. K. Landauer and D. W. Nachbar. Selection from alphabetic and numeric menu trees using a touch screen: Breadth, depth, and width. In *Proceedings of ACM CHI 1985 Conference on Human Factors in Computing Systems*, pages 73–78, 1985.

[Laramee *et al.*, 2004] Robert S. Laramee, Helwig Hauser, Helmut Doleisch, Benjamin Vrolijk, Frits H. Post, and Daniel Weiskopf. The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum*, 23(2):203–221, 2004.

[Lee and Bederson, 2003a] Bongshin Lee and Benjamin B. Bederson. Favorite folders: A configurable, scalable file browser. Technical Report HCIL-2003-12, CS-TR-4468, UMIACS-TR-2003-38, University of Maryland, Computer Science Department, College Park, Maryland, 2003. 10 pages.

[Lee and Bederson, 2003b] Bongshin Lee and Benjamin B. Bederson. Favorite folders: A configurable, scalable file browser (demo paper). In *ACM Symposium on User Interface Software and Technology (UIST) Conference Supplement*, pages 45–46, 2003.

[Leung and Apperley, 1994] Ying K. Leung and Mark D. Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 1(2):126–160, June 1994.

[Levoy, 1988] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications (CG&A)*, 8(3):29–37, May 1988.

[Li *et al.*, 2004] Wilmot Li, Maneesh Agrawala, and David Salesin. Interactive image-based exploded view diagrams. In *Proceedings of Graphics Interface (GI)*, pages 203–212, 2004.

[Lieberman, 1994] Henry Lieberman. Powers of ten thousand: navigating in large information spaces. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST)*, pages 15–16, 1994.

[Lieberman, 1997] Henry Lieberman. A multi-scale, multi-layer, translucent virtual space. In *Proceedings of IEEE Conference on Information Visualization (IV)*, pages 124–131, 1997.

[Löffelmann *et al.*, 1997a] Helwig Löffelmann, Lukas Mroz, and Eduard Gröller. Hierarchical streamarrows for the visualization of dynamical systems. In *Proceedings of EUROGRAPHICS Workshop on Visualization in Scientific Computing*, pages 203–209, 1997.

[Löffelmann *et al.*, 1997b] Helwig Löffelmann, Lukas Mroz, Eduard Gröller, and Werner Purgathofer. Stream arrows: Enhancing the use of stream surfaces for the visualization of dynamical systems. *The Visual Computer*, 13(8):359–369, 1997.

[Lorensen and Cline, 1987] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of ACM SIGGRAPH*, pages 163–169, 1987.

[MacKenzie, 1992] I. Scott MacKenzie. Fitts' law as a research and design tool in human-computer interaction. *Human-Computer Interaction*, 7:91–139, 1992.

[Mackinlay *et al.*, 1991] Jock D. Mackinlay, George G. Robertson, and Stuart K. Card. The perspective wall: Detail and context smoothly integrated. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 173–176, 1991.

[Mackinlay, 1986] Jock D. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics (TOG)*, 5(2):110–141, April 1986.

[Mander *et al.*, 1992] Richard Mander, Gitta Salomon, and Yin Yin Wong. A 'pile' metaphor for supporting casual organization of information. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 627–634, 1992.

[Mann and Rockwood, 2002] Stephen Mann and Alyn Rockwood. Computing singularities of 3D vector fields with geometric algebra. In *Proceedings of IEEE Visualization (VIS)*, pages 283–290, 2002.

[McCormick *et al.*, 1987] Bruce H. McCormick, Thomas A. DeFanti, and Maxine D. Brown. Visualization in scientific computing. *Computer Graphics*, 21(6), November 1987.

[McGuffin and Balakrishnan, 2005a] Michael J. McGuffin and Ravin Balakrishnan. Fitts' law and expanding targets: Experimental studies and designs for user interfaces. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 12(4):388–422, December 2005.

[McGuffin and Balakrishnan, 2005b] Michael J. McGuffin and Ravin Balakrishnan. Interactive visualization of genealogical graphs. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)*, pages 17–24, 2005.

[McGuffin and schraefel, 2004] Michael J. McGuffin and m. c. schraefel. A comparison of hyperstructures: Zzstructures, mSpaces, and polyarchies. In *Proceedings of 15th ACM Conference on Hypertext and Hypermedia (HT)*, pages 153–162, August 2004.

[McGuffin *et al.*, 2002] Michael McGuffin, Nicolas Burtnyk, and Gordon Kurtenbach. FaST Sliders: Integrating Marking Menus and the Adjustment of Continuous Values. In *Proceedings of Graphics Interface (GI)*, pages 35–41, May 2002.

[McGuffin *et al.*, 2003] Michael J. McGuffin, Liviu Tancau, and Ravin Balakrishnan. Using deformations for browsing volumetric data. In *Proceedings of IEEE Visualization (VIS)*, pages 401–408, October 2003.

[McGuffin *et al.*, 2004] Michael J. McGuffin, Gord Davison, and Ravin Balakrishnan. Expand-Ahead: A space-filling strategy for browsing trees. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)*, pages 119–126, 2004.

[McGuffin, 2002] Michael McGuffin. A content-centric model of interaction between users and software, and some thoughts regarding meta-interfaces, 2002. Unpublished webpage. http://www.dgp.toronto.edu/~mcguffin/research/contentCentrism/.

[Misue *et al.*, 1995] Kazuo Misue, Peter Eades, Wei Lai, and Kozo Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages and Computing*, 6(2):183–210, 1995.

[Munzner, 1997] Tamara Munzner. H3: Laying out large directed graphs in 3D hyperbolic space. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)*, pages 2–10, 1997.

[Munzner, 1998] Tamara Munzner. Drawing large graphs with H3Viewer and Site Manager. In *Proceedings of Symposium on Graph Drawing (GD)*, 1998.

[Neth and Payne, 2002] Hansjörg Neth and Stephen J. Payne. Thinking by doing? Epistemic actions in the Tower of Hanoi. In *Proceedings of the Twenty-Fourth Annual Conference of the Cognitive Science Society*, pages 691–696, 2002.

[Nguyen and Huang, 2002] Quang Vinh Nguyen and Mao Lin Huang. A space-optimized tree visualization. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)*, pages 85–92, 2002.

[Nielson and Hamann, 1990] Gregory M. Nielson and Bernd Hamann. Techniques for the interactive visualization of volumetric data. In *Proceedings of IEEE Visualization (VIS)*, pages 45–50, 1990.

[North *et al.*, 1996] Chris North, Ben Shneiderman, and Catherine Plaisant. User controlled overviews of an image library: A case study of the visible human. In *Proceedings of ACM Digital Libraries '96*, pages 74–82, 1996.

[North, 2000] Christopher Loy North. *A User Interface for Coordinating Visualizations Based on Relational Schemata: Snap-Together Visualization.* PhD thesis, Department of Computer Science, University of Maryland, College Park, Maryland, 2000.

[Owada *et al.*, 2004] Shigeru Owada, Ayumi Akaboya, Frank Nielsen, Fusako Kusunoki, and Takeo Igarashi. Kiru ("Cut", in Japanese). In *12th Workshop on Interactive Systems and Software (WISS 2004)*, pages 1–4, 2004.

[Owada, 2005] Shigeru Owada. *Practical Volume Graphics.* PhD thesis, School of Information Science and Technology, University of Tokyo, Tokyo, Japan, 2005.

[Pflesser *et al.*, 1995] B. Pflesser, U. Tiede, and K. H. Höhne. Towards realistic visualization for surgery rehearsal. In *Computer Vision, Virtual Reality and Robotics in*

Medicine, Proc. CVRMed '95 (N. Ayache, ed.), vol. 905 of Lecture Notes in Computer Science, pages 487–491, 1995.

[Pickett and Grinstein, 1988] Ronald M. Pickett and Georges G. Grinstein. Iconographic displays for visualizing multidimensional data. In *Proceedings of IEEE Conference on Systems, Man, and Cybernetics*, pages 514–519, 1988.

[Plaisant *et al.*, 2002] Catherine Plaisant, Jesse Grosjean, and Benjamin B. Bederson. SpaceTree: Supporting exploration in large node link tree, design evolution and empirical evaluation. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)*, pages 57–64, 2002.

[Playfair, 1786] William Playfair. *The Commercial and Political Atlas.* London, 1st edition, 1786. (3rd edition reproduced in The Commercial and Political Atlas and Statistical Breviary, by William Playfair, edited and introduced by Howard Wainer and Ian Spence, 2005).

[Playfair, 1801] William Playfair. *The Statistical Breviary.* London, 1801. (Reproduced in The Commercial and Political Atlas and Statistical Breviary, by William Playfair, edited and introduced by Howard Wainer and Ian Spence, 2005).

[Pook *et al.*, 2000] Stuart Pook, Eric Lecolinet, Guy Vaysseix, and Emmanuel Barillot. Control menus: Excecution and control in a single interactor. In *Extended abstracts of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 263–264, 2000.

[Read, 2000] Dwight W. Read. Formal analysis of kinship terminologies and its relationship to what constitutes kinship. *Mathematical Anthropology and Cultural Theory: An International Journal*, 1(1), November 2000. 46 pages.

[Reeves, 1983] William T. Reeves. Particle systems—technique for modeling a class of fuzzy objects. In *Proceedings of ACM SIGGRAPH*, pages 359–376, 1983.

[Reingold and Tilford, 1981] Edward M. Reingold and John S. Tilford. Tidier drawings of trees. *IEEE Transactions on Software Engineering*, SE-7(2):223–228, March 1981.

[Rekimoto and Green, 1993] Jun Rekimoto and Mark Green. The information cube: Using transparency in 3D information visualization. In *Proceedings of the Third Annual Workshop on Information Technologies & Systems (WITS'93)*, pages 125–132, 1993.

[Rezk-Salama *et al.*, 2001] C. Rezk-Salama, M. Scheuering, G. Soza, and G. Greiner. Fast volumetric deformation on general purpose hardware. In *Proceedings of ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 17–24, 2001.

[Robertson and Mackinlay, 1993] George G. Robertson and Jock D. Mackinlay. The document lens. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST)*, pages 101–108, 1993.

[Robertson *et al.*, 1989] George G. Robertson, Stuart K. Card, and Jock D. Mackinlay. The cognitive coprocessor architecture for interactive user interfaces. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST)*, pages 10–18, 1989.

[Robertson *et al.*, 1991] George G. Robertson, Jock D. Mackinlay, and Stuart K. Card. Cone trees: animated 3D visualizations of hierarchical information. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 189–194, 1991.

[Robertson *et al.*, 1993] George G. Robertson, Stuart K. Card, and Jock D. Mackinlay. Information visualization using 3D interactive animation. *Communications of the ACM (CACM)*, 36(4):56–71, April 1993.

[Robertson *et al.*, 2002] George Robertson, Kim Cameron, Mary Czerwinski, and Daniel Robbins. Polyarchy visualization: visualizing multiple intersecting hierarchies. In

Proceedings of ACM Conference on Human Factors in Computing Systems (CHI), pages 423–430, 2002.

[Rohde *et al.*, 2004] Douglas L. T. Rohde, Steve Olson, and Joseph T. Chang. Modelling the recent common ancestry of all living humans. *Nature*, 431(7008):562–566, September 2004.

[Rosenblum, 1994] Lawrence J. Rosenblum. Research issues in scientific visualization. *IEEE Computer Graphics and Applications (CG&A)*, 14(2):61–63, March 1994.

[Sarkar and Brown, 1992] Manojit Sarkar and Marc H. Brown. Graphical fisheye views of graphs. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 83–91, 1992.

[Schlienger *et al.*, 2006] Céline Schlienger, Pierre Dragicevic, Claire Ollagnon, and Stéphane Chatty. Les transitions visuelles différenciées: principes et applications. In *Compte rendu de la Conférence Francophone sur l'Interaction Homme-Machine (IHM)*, pages 59–66, 2006.

[Schroeder *et al.*, 1991] W. J. Schroeder, C. R. Volpe, and W. E. Lorensen. The stream polygon: A technique for 3D vector field visualization. In *Proceedings of IEEE Visualization (VIS)*, pages 126–132,417, 1991.

[Sears and Shneiderman, 1994] Andrew Sears and Ben Shneiderman. Split menus: Effectively using selection frequency to organize menus. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 1(1):27–51, March 1994.

[Sellen *et al.*, 1992] Abigail J. Sellen, Gordon P. Kurtenbach, and William A. S. Buxton. The prevention of mode errors through sensory feedback. *Human Computer Interaction*, 7(2):141–164, 1992.

[Shneiderman, 1982] Ben Shneiderman. The future of interactive systems and the emergence of direct manipulation. *Behaviour and Information Technology*, 1:237–256, 1982.

[Shneiderman, 1983] Ben Shneiderman. Direct manipulation: a step beyond programming languages. *IEEE Computer*, 16(8):57–69, August 1983.

[Shneiderman, 1992a] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, 2nd edition, 1992.

[Shneiderman, 1992b] Ben Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Transactions on Graphics (TOG)*, 11(1):92–99, January 1992.

[Shneiderman, 1994] Ben Shneiderman. Dynamic queries for visual information seeking. *IEEE Software*, 11(6):70–77, November 1994.

[Shneiderman, 1996] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of IEEE Symposium on Visual Languages (VL)*, pages 336–343, 1996.

[Shoumatoff, 1985] Alex Shoumatoff. *The Mountain of Names: A History of the Human Family*. Simon & Schuster, Inc., 1985.

[Sindre *et al.*, 1993] Guttorm Sindre, Bjørn Gulla, and Håkon G. Jokstad. Onion graphs: Aesthetics and layout. In *Proceedings of IEEE Symposium on Visual Languages (VL)*, pages 287–291, 1993.

[Singh and Balakrishnan, 2004] Karan Singh and Ravin Balakrishnan. Visualizing 3D scenes using non-linear projections and data mining of previous camera movements. In *Proceedings of ACM African Graphics Conference (AFRIGRAPH)*, pages 41–48, 2004.

[Singh, 2002] Karan Singh. A fresh perspective. In *Proceedings of Graphics Interface (GI)*, pages 17–24. Canadian Information Processing Society, 2002.

[Sonnet *et al.*, 2004] Henry Sonnet, Sheelagh Carpendale, and Thomas Strothotte. Integrating expanding annotations with a 3D explosion probe. In *Proceedings of ACM Advanced Visual Interfaces (AVI)*, 2004.

[Spence and Apperley, 1982] Robert Spence and Mark D. Apperley. Data base navigation: an office environment for the professional. *Behaviour and Information Technology*, 1(1):43–54, 1982.

[Spence and Wainer, 2005] Ian Spence and Howard Wainer. Playfair, William, 2005. An entry in Encyclopedia of Social Measurement, volume 3, pp. 71–79.

[Spence, 2001] Robert Spence. *Information Visualization*. ACM Press, 2001.

[Spence, 2002] Robert Spence. Rapid, serial and visual: a presentation technique with potential. *Information Visualization*, 1(1):13–19, March 2002.

[Spence, 2005] Ian Spence. No humble pie: The origins and usage of a stastitical chart. *Journal of Educational and Behavioral Statistics*, 30(4):353–368, Winter 2005.

[Stevens, 1946] Stanley Smith Stevens. On the theory of scales of measurement. *Science*, 103:677–680, 1946.

[Stoev and Straßer, 2002] Stanislav L. Stoev and Wolfgang Straßer. A case study on automatic camera placement and motion for visualizing historical data. In *Proceedings of IEEE Visualization (VIS)*, pages 545–548, 2002.

[Stone *et al.*, 1994] Maureen C. Stone, Ken Fishkin, and Eric A. Bier. The movable filter as a user interface tool. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 306–312, 1994.

[Sugiyama *et al.*, 1981] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(2):109–125, February 1981.

[Sutherland, 1963] Ivan E. Sutherland. Sketchpad: A man-machine graphical communication system. In *Proceedings of AFIPS Spring Joint Computer Conference*, pages 328–346, 1963.

[Tan *et al.*, 2001] Desney S. Tan, George G. Robertson, and Mary Czerwinski. Exploring 3D navigation: Combining speed-coupled flying with orbiting. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 418–425, 2001.

[Teoh and Ma, 2002] Soon Tee Teoh and Kwan-Liu Ma. RINGS: A technique for visualizing large hierarchies. In *Proceedings of Symposium on Graph Drawing (GD)*, pages 268–275, 2002.

[Thomas and Cook, 2005] James J. Thomas and Kristin A. Cook, editors. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. August 2005. http://nvac.pnl.gov/agenda.stm.

[Tory and Möller, 2004] Melanie Tory and Torsten Möller. Rethinking visualization: A high-level taxonomy. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)*, pages 151–158, 2004.

[Tory *et al.*, 2005] Melanie K. Tory, Simeon Potts, and Torsten Möller. A parallel coordinates style interface for exploratory volume visualization. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 11(1):71–80, January 2005.

[Tufte, 1983] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 1983.

[Tufte, 1990] Edward R. Tufte. *Envisioning Information*. Graphics Press, 1990.

[Tufte, 1997] Edward R. Tufte. *Visual Explanations: Images and Quantities, Evidence and Narrative*. Graphics Press, 1997.

[Tukey, 1977] John Wilder Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977.

[Tversky *et al.*, 2002] Barbara Tversky, Julie Bauer Morrison, and Mireille Betrancourt. Animation: can it facilitate? *International Journal of Human-Computer Studies*, 57:247–262, 2002.

[Tweedie, 1997] Lisa A. Tweedie. Characterizing interactive externalizations. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 375–382, 1997.

[van Wijk and Nuij, 2003] Jarke J. van Wijk and Wim A. A. Nuij. Smooth and efficient zooming and panning. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)*, 2003.

[van Wijk, 1991] Jarke J. van Wijk. Spot noise: Texture synthesis for data visualization. In *Proceedings of ACM SIGGRAPH*, pages 309–318, 1991.

[Vázquez *et al.*, 2002] Pere-Pau Vázquez, Miquel Feixas, Mateu Sbert, and Antoni Llobet. Viewpoint entropy: A new tool for obtaining good views of molecules. In *Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization (2002)*, pages 183–188, 2002.

[Venolia and Neustaedter, 2003] Gina Danielle Venolia and Carman Neustaedter. Understanding sequence and reply relationships within email conversations: A mixed-model visualization. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 361–368, 2003.

[Viega *et al.*, 1996] John Viega, Matthew J. Conway, George Williams, and Randy Pausch. 3D magic lenses. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST)*, pages 51–58, 1996.

[Viola *et al.*, 2004] Ivan Viola, Armin Kanitsar, and Meister Eduard Gröller. Importance-driven volume rendering. In *Proceedings of IEEE Visualization (VIS)*, pages 139–145, 2004.

[Walker II, 1990] John Q. Walker II. A node-positioning algorithm for general trees. *Software—Practice and Experience*, 20(7):685–705, July 1990.

[Ward, 2002] Matthew O. Ward. A taxonomy of glyph placement strategies for multidimensional data visualization. *Information Visualization*, 1:194–210, 2002.

[Ware and Lewis, 1995] Colin Ware and Marlon Lewis. The DragMag image magnifier. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)*, pages 407–408, 1995.

[Ware, 2000] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers, 2000.

[Wesson *et al.*, 2004] Janet Wesson, MC du Plessis, and Craig Oosthuizen. A ZoomTree interface for searching genealogical information. In *Proceedings of ACM AFRIGRAPH '04*, pages 131–136, 2004.

[Westover, 1990] Lee Westover. Footprint evaluation for volume rendering. In *Proceedings of ACM SIGGRAPH*, pages 367–376, 1990.

[Wetherell and Shannon, 1979] Charles Wetherell and Alfred Shannon. Tidy drawings of trees. *IEEE Transactions on Software Engineering*, SE-5(5):514–520, September 1979.

[White and Jorion, 1992] Douglas R. White and Paul Jorion. Representing and computing kinship: A new approach. *Current Anthropology*, 33(4):454–463, Aug.-Oct. 1992.

[Wilkinson, 1999] Leland Wilkinson. *The Grammar of Graphics*. Springer, New York, 1999.

[Wittenbrink *et al.*, 1996] Craig M. Wittenbrink, Alex T. Pang, and Suresh K. Lodha. Glyphs for visualizing uncertainty in vector fields. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 2(3):266–279, September 1996.

[Wong and Bergeron, 1997] Pak Chung Wong and R. Daniel Bergeron. 30 years of multidimensional multivariate visualization, 1997. Chapter 1 (pp. 3–33) of Gregory M. Nielson, Hans Hagen, and Heinrich Müller, editors, Scientific Visualization: Overviews, Methodologies, and Techniques, IEEE Computer Society.

[Wong *et al.*, 2003] Nelson Wong, Sheelagh Carpendale, and Saul Greenberg. EdgeLens: An interactive method for managing edge congestion in graphs. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)*, pages 51–58, 2003.

[Woods, 1984] David D. Woods. Visual momentum: a concept to improve the cognitive coupling of person and computer. *International Journal of Man-Machine Studies*, 21:229–244, 1984.

Appendix A

Survey of Techniques for Managing Occlusion

Part of the work done for the first case study, on volume visualization, involved considering the general question of how to deal with occlusion, and surveying existing work on occlusion and how to manage it. What follows is a more extensive literature survey than that given in Chapter 3. Volumetric data is of significant interest here, but we also look to occlusion in visualization of other kinds of data, and to analogous or related techniques in 2D and so-called $2\frac{1}{2}$D spaces.

We first give a breakdown of strategies for reducing, alleviating, or managing occlusion, establishing a simple taxonomy. We then consider each of these strategies in turn in its own subsection, and survey the corresponding literature.

A.1 Occlusion in 2D and 3D: Strategies

Inter-object occlusion in a 3D scene is partly a symptom of having too much data, i.e. too many objects or geometric elements. However, even with small data sets, occlusion is sometimes a basic consequence of visualizing data that has an inherent or preferred embedding in space, for example the 3D configuration of parts that compose a mechan-

ical device, or volumetric data sets from medical scans. Even with data that does not have an inherent spatial embedding, techniques for mitigating occlusion are still useful, for example after having chosen an embedding for the data which happens to produce occlusion.

The following general strategies will be considered for reducing, alleviating, or managing occlusion.

- **Show less data:** Reduce the amount of data displayed, or reduce the number of geometric elements shown. This may involve displaying only a subset of data (e.g. through filtering, cutting away portions of data, or displaying only certain extracted features) or aggregating data into a coarser form (e.g. through downsampling, or extraction of higher level features or summaries).

- **Modify the elements displayed:** Change one or more properties of the individual elements. One possibility here is to change the opacity of the elements: making some or all of the elements semi-transparent may greatly reduce occlusion. Another possibility is to scale each of the elements down in size, without translating them, to introduce more space between objects. (Note, however, that this is equivalent to translating all elements away from each other, and then applying a global scale factor; we consider schemes like this in the next category.) Various other possibilities may exist here, however we limit our consideration to the use of transparency in § A.3.

- **Rearrange elements:** Change the arrangement of elements in space, i.e. the location, orientation, or shape of elements or portions of elements. We include here a broad set of spatial transformations, ranging from piecewise-rigid transformations like exploded views (where elements are translated away from each other) to piecewise continuous non-uniform deformations.

- **Use camera control:** Changing the point of view and orientation of the camera

is part of the status quo for interactive viewing of 3D scenes, however, there are various novel ways of facilitating camera control.

- **Change the projection method:** Non-linear projections of 3D surfaces onto 2D screens can sometimes eliminate much or all occlusion.

Carpendale [1999, chapter 1] distinguishes between changes in the *representation* (e.g. the embedding) of data, and changes in the *presentation* (e.g. rendering parameters, or relatively cosmetic properties) of data. With respect to this, the strategies listed above can all be thought of as ways of changing the presentation of the data, except possibly for the 3rd strategy (rearranging elements). The status of the 3rd strategy is debatable, depending on which space we consider as containing the data's representation: the one before spatial transformation (in which case the transformation is a change in presentation), or the one after (in which case the transformation modifies the representation).

Although we will survey work within all of the above strategies, spatial rearrangement of data will be of particular interest, in particular of volumetric data, as a means of alleviating occlusion. We will also see that spatial rearrangement can achieve some of the same goals as focus+context schemes, which have traditionally been applied to 2D spaces.

The following sections survey each of the strategies listed above, in order.

A.2 Showing Less Data

A.2.1 Geometric Slicing and Cutting

Two basic subsets of volumetric data sets are (1) thin, planar slices or cross-sections, which may be axis-aligned or not, and (2) data sets produced by applying a boolean mask, e.g. intersecting the volume with a half-space. The latter corresponds to tools that "carve away" or cull parts of the volume, such as cutting planes that truncate the

volume, cutting boxes, and cutting spheres (Figure A.1).

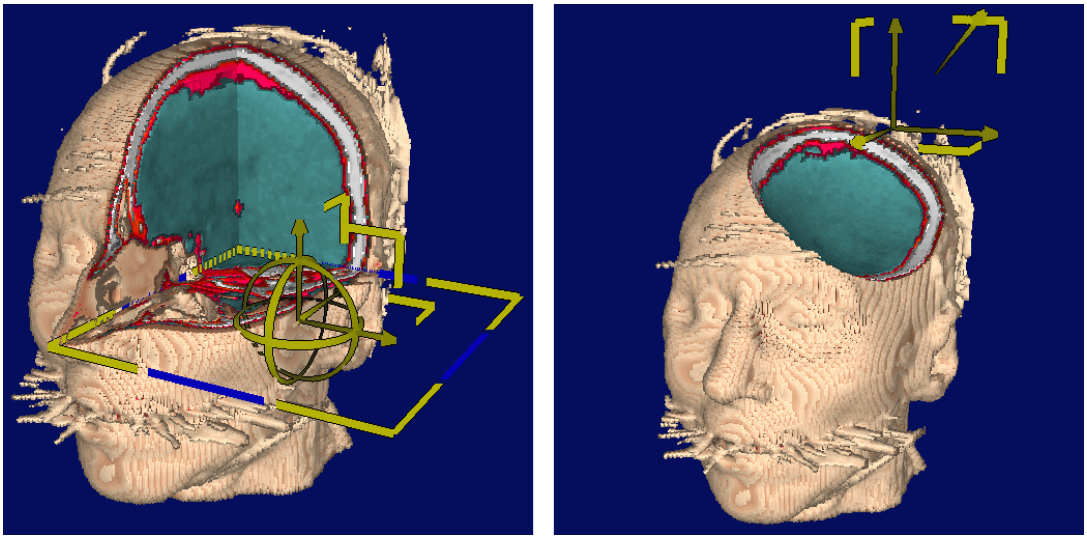

Figure A.1: The data set from Figure 3.1 being browsed with different cutting tools. *Left:* A cutting box. *Right:* A cutting sphere.

See [Nielson and Hamann, 1990; Höhne *et al.*, 1992; North *et al.*, 1996] for examples of user interfaces supporting browsing of such subsets. Browsing and modifying such subsets, e.g. by swimming cuts or slices through the volume, allows every voxel to be made visible, but may not present the data to the user in the most natural way. For example, it is difficult to mentally reconstruct 3D curved surfaces or objects by looking at a series of cross-sections.

A.2.2 Extracted Subsets and Features

Other subsets can be produced by automatic or semi-automatic extraction from the original data set.

Segmentation of the volume, which is often done at least partially automatically, allows the system to display or hide each object independently, allowing the user to expose as much of the surface of any object as desired, at the cost of temporarily hiding other objects.

Isosurfaces [Lorensen and Cline, 1987] can be extracted in a fully automatic fashion

from a continuous scalar field, and may be displayed alone, or with other surfaces or segmented objects.

Vector fields, such as velocity fields of fluids, are often continuous, and usually their essential topology is more important to the user than the vector values at each voxel. Much work (e.g. [Mann and Rockwood, 2002]) has been directed at detecting and extracting the critical points, lines, and surfaces within a vector field, since these features are often sparsely distributed, and displaying them can drastically reduce occlusion and aid comprehension.

A.2.3 Subsampling

Here we consider various ways of sampling the data, typically at many regularly spaced locations, and displaying the results of such samples. Each sample can be though of as a probe into the data. By having fewer samples than there are voxels or data points, occlusion between elements can be reduced while still giving the user an overview of the data. Although this subsampling coarsens the data, most or all of the essential features of the data may still be present.

One form of subsampling involves sampling at or around *points* in the volume. The value associated with each sample may be evaluated only at this point, or may be an average over the vicinity around the point. Each sample can then be displayed as an icon or glyph at the sample's location. The glyph may be, for example, a coloured cube, or arrow, or ellipsoid, depending in part on whether the data is scalar, vector, tensor, etc.

Arrow plots (also called hedgehogs) [Klassen and Harrington, 1991; Wittenbrink *et al.*, 1996] use arrow glyphs to display vector fields in 2D and 3D. In arrow plots, the length of arrows often represents vector magnitude, which means that long arrows may overlap and occlude each other, and/or short arrows may be difficult to see. There is a tradeoff between density of arrows, occlusion, and how clearly magnitude is displayed. An alternative is to use colour to express magnitude, but this may make relative comparisons

of magnitude more difficult. Yet another possibility is offered by dense texture-based techniques such as spot noise [van Wijk, 1991] and Line Integral Convolution (LIC) [Cabral and Leedom, 1993] [Laramee *et al.*, 2004]. In 2D, these can make the direction of a vector field apparent at almost every pixel, but leave no room to use length as the magnitude indicator; instead, colour may be used. Applying LIC in 3D however leads again to occlusion problems [Interrante and Grosch, 1997].

Moving particles [Reeves, 1983] are also used to visualize vector fields, and these can be thought of as moving point probes.

The value probed at a single location may have many degrees of freedom associated with it (e.g. a tensor with many components), and may require a complicated visual glyph to express the value completely. In this case, it may be better to have only a single probe [de Leeuw and van Wijk, 1993; Herndon and Meyer, 1994] or a small array of probes [Herndon and Meyer, 1994] that the user manually positions, rather than have a large array of automatically sampled points. The probe of de Leeuw and van Wijk [1993] also allows the user to leave virtual copies of it at selected points.

For a survey of glyphs used in 2D, see [Ward, 2002].

In contrast to the point sampling or point probes just considered, another strategy is to sample *lines* or *curves* within the data set, and display these. Such curves may be generated by first choosing a set of seed points and then computing a curve through each point, but the samples are nevertheless essentially lines rather than points.

Examples of such sampled curves are streamlines or fluxlines, which are everywhere tangent to a vector field (which may itself be the gradient $\nabla\phi$ of some scalar field ϕ). When displayed, these curves may have the colour along them varied to indicate the magnitude of the vector field along the curve, and may also have tick marks, arrows, or other glyphs (such as stream polygons [Schroeder *et al.*, 1991]) along them to show additional properties.

Other curves and variations on curves which have been used in fluid visualization are

- pathlines: also called particle trajectories, these are the path taken by a massless point particle over time [Chorin and Marsden, 1993, p. 15]

- streaklines[1]: these correspond to a set of particles at a given time, where the particles were introduced at a common location over an interval of time, modelling dye injected into the fluid

- timelines: these correspond to a set of particles or points at a given time, where the particles started at different initial locations

- streamribbons: a ribbon-like strip of polygons, that may have its 2 edges lie along streamlines, or that may be centred on a single streamline and have its orientation depend on some parameter such as curl [Schroeder *et al.*, 1991]

- streamtubes: a streamline drawn with variable thickness [Schroeder *et al.*, 1991]

- rakes of streamlines (e.g. [Herndon and Meyer, 1994]), or other bundles of streamlines

Displaying curves sampled at discrete locations rather than point glyphs can somewhat increase occlusion, but can remarkably improve the clarity of the topology of the data set (Figure A.2). This is partly because curves are continuous along one spatial direction, which can make important variations in the data visible, while reducing the need to sample along the two perpendicular directions.

Extending the notion of curve samples yields *surface* samples, such as isosurfaces [Lorensen and Cline, 1987] of a 3D scalar field, which are analogous to iso-curves of a 2D scalar field; stream surfaces [Hultquist, 1992], generated by tracing the streamlines originating at points along a curve (note that, in static flow, streamlines and timelines

[1]Note that, in stationary flow, where the vector field does not change over time, streamlines and pathlines and streaklines all coincide.

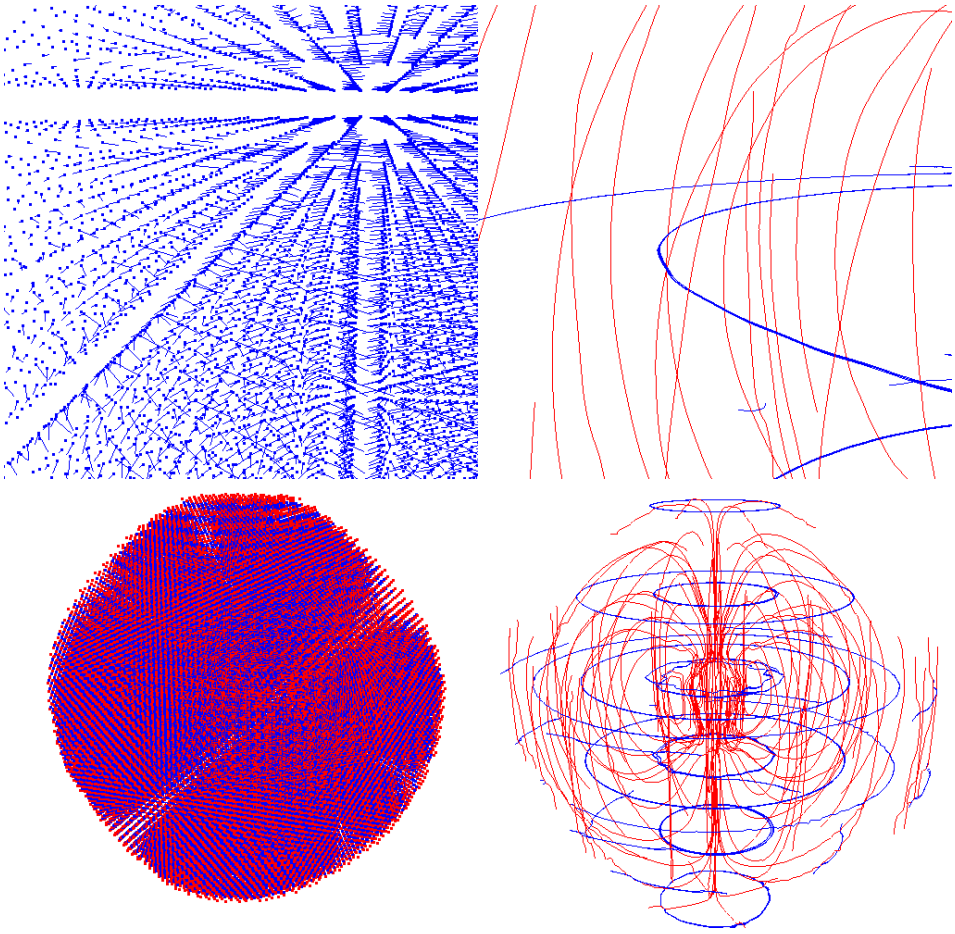

Figure A.2: Four different views of a 3D electromagnetic field, showing all combinations of a close-up view (top) or zoomed-out overview (bottom), with either arrow plots (left) or fluxlines (right). The electric and magnetic fields are in blue and red, respectively. Where arrow plots do not suffer from excessive density or occlusion, overall directional trends in the data are still difficult to perceive, even with interactive rotation. Fluxlines reveal the structure of the field fairly clearly, despite being relatively sparse.

on a stream surface are everywhere perpendicular); and time surfaces, an extension of timelines.

Just as glyphs can be drawn along a sampled curve, they can also be drawn at regular locations on a sampled surface. Streamballs [Brill *et al.*, 1994] are related to this idea. Another example is streamarrows [Löffelmann *et al.*, 1997b; 1997a], which are arrow-shaped planar patches drawn on a stream surface, or displayed as arrow-shaped holes in a stream surface. Streamarrows drawn as holes make flow direction more apparent along

the surface while also reducing occlusion of objects behind the surface.

Another way to reduce occlusion created by a surface is to draw strokes that suggest the curvature of the surface [Interrante *et al.*, 1996], allowing the user to see through the spaces between strokes.

A.3 Using Transparency

Here we consider reducing inter-object occlusion by changing the opacity of rendered elements.

A.3.1 Volume Rendering and Transfer Functions

Perhaps the earliest renderings of volumetric data were reported in [Herman and Liu, 1979], where no transparency is used, and the volume is divided into two disjoint sets using thresholding. Each voxel of one set is then drawn as a small cube, producing a "cuberille" rendering. Other simple techniques for rendering voxel data, including a "tiny cubes method", are reported in [Nielson and Hamann, 1990].

In contrast, *direct volume rendering* (DVR, or sometimes simply *volume rendering*) refers to rendering where voxels have fractional opacity values that are blended or accumulated, especially in a fashion that approximates physical media. DVR can be performed with both forward rendering techniques, such as compositing [Drebin *et al.*, 1988], splatting [Westover, 1990], and 3D texture mapping (increasingly well supported on modern graphics hardware); as well as with inverse rendering techniques, such as raycasting or ray tracing [Kajiya and Von Herzen, 1984; Levoy, 1988]. Associated with DVR is the notion of a *transfer function*, which assigns to voxels their optical properties such as colour, opacity, and emittance. The transfer function is usually a function of the voxel's data value and/or the classification [Drebin *et al.*, 1988; Levoy, 1988] of voxels, and may also be a function of other variables, such as the gradient

of the data [Kniss *et al.*, 2001], or the location of the voxel.

Changing the transfer function can be used to emphasize, deemphasize, or de-occlude regions within the volume. Because transfer functions have so many degrees of freedom, and can influence a rendered image in non-intuitive ways, users may spend much time adjusting a transfer function and still fail to discover important features in a data set. An active area for research is improvements to user interfaces for specifying transfer functions [Kniss *et al.*, 2001] as well as automatic generation or adjustment of transfer functions [Kindlmann and Durkin, 1998].

DVR is often contrasted with maximum intensity projection (MIP), a rendering technique where a maximal (opacity or colour) value is found along each ray from the eye and projected onto the corresponding pixel. Although opacity values are not accumulated or blended, MIP can nevertheless be thought of as another form of transparency. The resulting image is loosely analogous to an x-ray negative, in that the volume can be thought of as blocking the transmission of back lighting (albeit in a physically inaccurate way), rather than reflecting front lighting. Rendering can be faster with MIP than with DVR, because the final image is independent of the order in which elements are processed, obviating the need for a z-ordering. See Figure A.3.

Depth perception in DVR and MIP renderings is best if aided by motion (parallax) cues and/or stereopsis. The ability to rotate a volume and see it re-render in real time is highly desirable. Even in this case, however, certain internal regions of the data may remain completely occluded under all external viewpoints, because the voxel values there may contribute negligibly or not at all to the final renderings. These internal regions can be made visible by adjusting the transfer function, but this in turn makes other regions invisible.

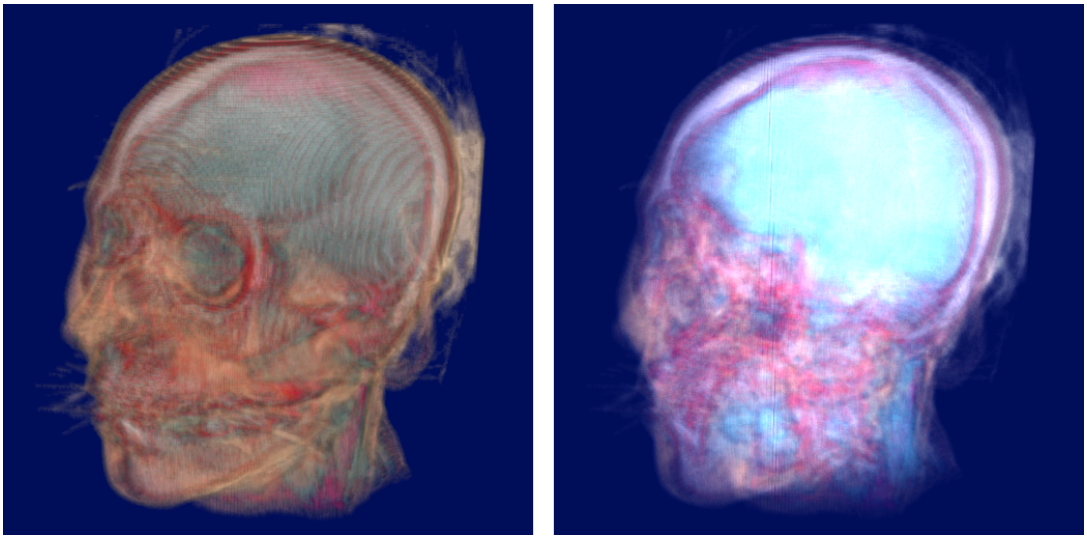

Figure A.3: The data set from Figure 3.1 rendered with transparency, using splatting. *Left:* Direct volume rendering (DVR). *Right:* A rendering somewhat similar to maximum intensity projection (MIP), where the blending function additively (and hence commutatively) accumulates colours, without requiring that voxels be z-ordered.

A.3.2 Other Uses of Transparency in 2D and 3D

In 2D, there are examples of user interfaces that blend 2 or more semi-transparent layers, to make more information available to the user.

Colby and Scholl [1991] combine overlapping layers of geographical information, using increased transparency and blurring to deemphasize certain layers in favour of other ones (blurring for deemphasis is also used in Semantic Depth of Field [Kosara *et al.*, 2001]). The Macroscope [Lieberman, 1994; 1997] also blends overlapping layers, each one showing a map at a different scale, allowing for focus and context to be combined through transparency. Although less sophisticated, interfaces displaying just two overlapping maps at different scales [Cox *et al.*, 1998] have been incorporated into commercial software games [Blizzard Entertainment, 2000].

Transparency has also been used in more generic user interface elements such as windows and widgets. Harrison et al. [1995; 1996] investigated semi-transparent menus designed to be usable while allowing the user to benefit from the background information visible through them. Alpha-blended windows are now supported in various operating

systems, window managers, and other software systems, some of which allow the opacity of auxiliary panels of widgets to be dynamically adjusted based on the location of the mouse cursor [Gutwin *et al.*, 2003]. Alternatives to traditional alpha-blending have also been proposed, such as multiblending [Baudisch and Gutwin, 2004], which can enhance the visibility of edges or important visual features that might otherwise be washed out or obscured.

Aside from using cursor position, transparency can also be automatically adjusted based on the current arrangement of objects with respect to the camera view. Free-space transparency [Ishak and Feiner, 2004] reduces the opacity of white space in front windows, to make content behind the white space more visible. Importance-driven rendering [Viola *et al.*, 2004], a technique for 3D volume rendering, can make the front layer of a data set less opaque wherever important data is behind it, and updates the opacity as the volume is rotated. These two techniques are complementary: one removes unimportant elements from front layers, while the other ensures that important elements in back layers are visible through the front. Each of these implies a different prioritization of layers, assuming that the front or back layers are more important, respectively.

The transparency of regions of layers or objects can also be adjusted in a manual but very light-weight fashion, using magic lenses in 2D [Bier *et al.*, 1993; Stone *et al.*, 1994; Bier *et al.*, 1994] or 3D [Viega *et al.*, 1996]. These allow the user to quickly switch between different rendering modes or transparency settings simply by selecting a region of space, which is much more direct than, say, adjusting an abstract curve representing a transfer function. The cutting tools of Figure A.1 can be seen as special cases of magic lenses, where the opacity of voxels within the lens is set to zero. One limitation of magic lenses is that the user can typically only switch between a small number of discrete rendering modes, which may work well with only a small number of layers, but has limited flexibility in cases involving many layers or objects.

The recent work in automatic adjustment of transparency [Ishak and Feiner, 2004;

Viola *et al.*, 2004] of layers or regions of layers is promising, as it may alleviate the need for the user to manually select layers of interest, or to adjust a transfer function. More generally, existing 2D user interfaces have shown that success can be achieved with 2 or perhaps a few more layers. However, transparency is only a partial solution to the general occlusion problem, as it does not scale up to large numbers of layers or overlapping objects. Ironically, this can be interpreted as an advantage. The Information Cube [Rekimoto and Green, 1993] is a visualization of a tree as recursively nested semi-transparent boxes. Because of alpha blending, the lower levels of the tree are increasingly obscured, a feature which the authors present as desirable, because it hides details that would be distracting otherwise.

A.4 Rearranging Elements (e.g. Deformation)

We now consider spatial transformations, such as exploded views and deformations, for dealing with the occlusion problem.

The strategies in § A.2 and § A.3 have a common weakness, which was pointed out by Carpendale et al. [1997a]. Although transparency and removal of outer data both make inner data more visible, they also result in loss of context. This can make it difficult for users to form an integrated mental picture of the entire volume.

The strategy considered here involves changing the positions of data points, without removing any data. This has the potential to reduce occlusion and expose more inner surfaces or regions of the data, while retaining the displaced data in the periphery, helping the user maintain a mental context. Appropriately chosen spatial transformations, or mappings of the form $f: \mathbb{R}^3 \mapsto \mathbb{R}^3$, could, for example, split open a volume, showing displaced structures side-by-side, making it easy for the user to see how they connect, and allowing the user to mentally stitch them together into a whole. Deformations with familiar real-world analogues (e.g. cutting and peeling the skin off a fruit, or the layers

off an onion) are also likely to be readily understood by users.

The notion of showing inner portions of a data set without completely removing the surrounding data is akin to focus+context schemes that allow a user to "zoom in" on data of interest, while using remaining screen space to show the surrounding context. Traditional research in focus+context schemes has centred on the presentation of 2D spaces, using mappings of the form $f: \mathbb{R}^2 \mapsto \mathbb{R}^2$. We thus begin by reviewing this work, and then consider other work directly concerned with elimination of occlusion through other spatial transformations or displacement of graphical elements.

A.4.1 Focus+Context Schemes in 2D

Focal (zoomed in) and contextual (zoomed out) views of a 2D space can be presented to a user in separate viewports or windows, which may overlap (e.g. as with insets, or Manhattan lenses, or DragMag [Ware and Lewis, 1995], or small "radar" overviews) or be tiled, and be of the same or different sizes in screen space, depending on their relative importance. Having separate windows creates a divided attention problem, and requires more effort for the user to mentally integrate the two images. Overlapping windows can somewhat reduce the division of attention, by showing a magnified image close to its location in an overview, but this in turn creates occlusion of some of the overview.

It is arguably preferable to display a single image which visually integrates both the focal and contextual regions, i.e. a detail-*in*-context visualization, at the cost of distorting some of the 2D space. These are also called fisheye visualizations, because the visual effect can be reminiscent of a wide-angle camera lens. An early but non-interactive instance of this appeared in the cartography community [Kadmon and Shlomi, 1978]. Seminal work in the computer science community, which formalized the fisheye concept, was [Furnas, 1981; 1986]. This was followed by many subsequent proposed schemes. Some schemes map the original 2D space onto a surface in 3D that is then projected back down to 2D (these can be described as $\mathbb{R}^2 \mapsto \mathbb{R}^3 \mapsto \mathbb{R}^2$ mappings) [Spence and Apperley, 1982;

Mackinlay *et al.*, 1991; Robertson and Mackinlay, 1993; Carpendale and Montagnese, 2001] and others distort the 2D space directly ($\mathbb{R}^2 \mapsto \mathbb{R}^2$ mappings) [Kadmon and Shlomi, 1978; Sarkar and Brown, 1992]. Still other schemes use transformations inspired by the properties of hyperbolic space [Lamping and Rao, 1994; Lamping *et al.*, 1995; Munzner, 1997]. Leung and Apperley [1994] performed a comparative analysis of many techniques, clarifying the differences between them by comparing their transformation functions and magnification functions. Carpendale gives an extensive survey of techniques [Carpendale, 1999, chapter 2], and later Carpendale et al. [2001] implemented a system capable of generating, and interpolating between, many previously described fisheye schemes.

Although all the above techniques can be thought of as $\mathbb{R}^2 \mapsto \mathbb{R}^2$ mappings, and therefore candidates to be generalized to 3D, it must be kept in mind that 2D focus+context techniques are primarily concerned with combining views that are at different scales, rather than eliminating occlusion.

A.4.2 Occlusion in 2D and $2\frac{1}{2}$D

Occlusion in $2\frac{1}{2}$D spaces (i.e. 3D spaces with a fixed camera plane and objects that face the camera) is perhaps most commonly encountered in virtual desktops and windowing systems that support overlapping windows. Occlusion becomes more of a problem as users open more windows, and need to switch between windows more frequently, or need to view the output of many windows simultaneously.

Elastic Windows [Kandogan and Shneiderman, 1996; 1997] eliminate inter-window occlusion completely, while allowing users to flexibly reorganize windows with recursively nested containers that can be resized. However, the amount of data that users can see at any given time is still limited by the physical screen size and resolution. So, although windows are never occluded, they may be squeezed to unusably small sizes.

Mander et al. [1992] describe different ways of browsing virtual piles of documents, in some cases emulating the effect of riffling or thumbing through a physical pile of paper.

The WebBook [Card *et al.*, 1996] allows flat pages to be flipped in 3D, giving the user brief glimpses of their content as the pages are rotated in quick succession (this idea of rapid exposure to a sequence of visual stimuli is revisited by Spence [2002]). Beaudouin-Lafon [2001] designed novel interaction techniques for overlapping windows that are also inspired by physical piles of paper. One of these allows a user to peel back the corner of one or more windows, to take a peek at occluded windows. Dragicevic [2004] extended Beaudouin-Lafon's work to enable a user to drag an icon or other item into a destination window that might be occluded by several other windows. This is achieved by peeling window corners in response to the user *crossing* [Accot and Zhai, 1997; 2002] window edges with the pointing device while dragging. Even more recent techniques for rapidly flipping or riffling through virtual piles are given by Agarawala and Balakrishnan [2006].

Bell and Feiner [2000] define a data structure and algorithms for storing and querying the available free space in a 2D space populated by full-space rectangles. This enables automatic allocation strategies to be implemented that avoid or reduce occlusion. For example, the position of a new window can be automatically chosen to fall within existing empty space, or windows can be automatically repositioned if the user covers their initial locations with a new window.

Beyond the domain of window management, occlusion in 2D also occurs in drawings of graphs, e.g. in the form of edge crossings. Drawings of graphs with a large number of edges may suffer from edge congestion: edges may cover each other, and cover nodes, and data beneath the graph drawing. Wong et al. [2003] developed an EdgeLens tool that temporarily curves some or all edges under a lens without changing the positions of nodes. This can open up "holes" allowing the user to see some information more easily. In some cases, edge connections are easiest to see when the EdgeLens is actively moved by the user, creating motion cues from the varying curvature of edges.

A.4.3 Occlusion in 3D: Rigid Transformations

We now consider research in reducing occlusion through rigid transformations (i.e. translations and rotations).

Exploded diagrams show the components of an object (e.g. a mechanical assembly) rigidly displaced away from each other, to show essential features of the components (e.g. connecting elements, such as screw holes or plugs) and how they fit together into a whole. Translating components away from each other can eliminate all inter-component occlusion, and displaying dashed lines can help the viewer see how and where they fit together.

Various systems (e.g. [Driskill and Cohen, 1995; Agrawala *et al.*, 2003]) have been created to help automate the generation of exploded diagrams, however the output of these systems is a set of static images, intended possibly for technical illustrations. Li et al. [2004] developed a system for creating interactive exploded diagrams, however these are constructed from 2D images, and do not allow the user to rotate objects in 3D.

Sonnet et al. [2004] developed an interactive 3D system where the user can explode different regions of a 3D model with a point probe, and also see annotations on components dynamically appear.

Perhaps one remaining research direction for interactive 3D exploded diagrams would be to dynamically update the displacement of components based on the camera view, rather than (or in addition to) the location of a point probe. Displacements could be computed to avoid inter-component overlap in screen space.

Related to this, the work of Bell et al. [2001] involves automatically computing the positions of billboard annotations of a static 3D scene. As the camera view changes, the positions of annotations in screen space can be updated to avoid occlusion. Unfortunately, this can lead to frequent rearrangement of annotations, whose motion can be distracting. Note also that the connections between annotations and the objects they annotate can be drawn very simply, as simple line segments. In other situations where the "connections"

are not so simple, e.g. if we wish to avoid occlusion between deformable subsets of a data set, then the connections may themselves create significant occlusion.

A.4.4 Occlusion in 3D: Deformations

Here we consider spatial transformations of volumetric data (or other 3D data) that are not limited to rigid translations and rotations. As already alluded to, these can be used to browse volumetric data by, e.g., splitting open or peeling away parts of a data set, de-occluding the interior, while retaining the displaced parts on the screen to provide context.

The simulation of physically realistic deformations has practical applications in computer animation as well as in medicine. There are also techniques for harnessing recent flexible graphics hardware to accelerate such simulation (e.g. [Rezk-Salama *et al.*, 2001]). High-fidelity simulations of surgical procedures (e.g. [Pflesser *et al.*, 1995; Gibson *et al.*, 1998]) can be used for education, training, rehearsal and planning, and can involve the use of virtual reality, haptic feedback, and the simulation of the physical properties, such as elasticity and hardness, of the tissues being operated on.

Although such physically realistic simulation is not necessary for *browsing* data, surgical tools provide a useful metaphor for designing browsing tools. For example, Bruyns and Montgomery [2002] describe virtual tools that look and behave like scalpels, scissors, and forceps, allowing a 2- or 3-dimensional mesh to be cut and peeled open. Such tools could be useful for exploring data hidden beneath layers of a mesh. Furthermore, there are deliberately non-physical operations which might be useful in browsing, such as creating a cut and "swimming" the location of the cut through the data.

Chen et al. [2003] give a formalization of spatial transformations for volumetric data, and give examples of transformations such as splitting (considered in more detail in [Islam *et al.*, 2004]), squeezing, sweeping, tearing open, and turning inside out. These deformations are considered at a general level, as useful for modeling, animation, and

visualization of volumes.

We now consider some categories of deformations that have been explored in previous work.

Splitting Open. Kurzion and Yagel [1997] describe a "discontinuous ray deflector" that gives the appearance of cutting into a volume and spreading open the voxels. This is similar to the "book" metaphor used by Carpendale et al. [1999] where data are spread open like pages of a book. The same book metaphor has been used in traditional anatomical diagrams, where organs are shown cut in half and spread apart on consecutive pages of a book [Agur and Lee, 1999, for example pp. 622–623, 718, 719].

Peeling. Laidlaw [1995] segmented scans of a banana and a human hand, and created images of their skin peeling off. His implementation did not, however, support real-time interaction. Owada et al. [2004] [Owada, 2005, chapter 7] developed an interactive system inspired by culinary sculpting of food. A "knife" widget can be invoked and controlled with strokes and mouse dragging, and used to cut into the volume, and peel off thin slices.

Radial distortions. Radial fisheye distortions have also been applied to volume data. LaMar et al. [2001] describe a focus+context technique that magnifies a region inside a volume. Cohen and Brodlie [2004] combine 3D magnification with 2D magnification, to obtain a fisheye effect in object space and/or in image space. These kinds of distortions, however, do little to reduce occlusion, and mainly just increase the visibility of small details. The magnified region is only visible to the user if the surrounding data is semi-transparent, or if a cutting plane is used to reveal the inside. In contrast, Carpendale et al. [1997a] describe a *visual access distortion* technique that clears a path of visibility to a point of interest by radially pushing occluding data away from the line of sight. This "cleared path" remains on the line of sight as the scene is rotated, giving the appearance of a constantly shifting deformation. Thus, the rotation and deformation of the data are coupled.

A.5 Using Camera Control

Given a 3D scene, there may be points where the camera can be placed to minimize occlusion, and/or make the information in the scene easiest to perceive and interpret. Although the user can discover such points by manually controlling the camera, it may be more efficient to determine the location of such points for the user, and then constrain or at least guide the camera to them. The StyleCam [Burtnyk *et al.*, 2002] implements different interaction techniques for guiding the user's view to pre-selected points of view or regions of favourable vantages. Techniques for automatically selecting or extracting key points of view are discussed in [Vázquez *et al.*, 2002; Stoev and Straßer, 2002; Singh and Balakrishnan, 2004].

Although less related to occlusion reduction, there has also been work in making cameras respond more automatically to input, so that useful camera motions are easier to execute. Zooming level, or the height of a camera, has been coupled to the user's speed in world space in 2D [Igarashi and Hinckley, 2000] and 3D [Tan *et al.*, 2001] so that the speed of "visual flow" in screen space is approximately constant. The HoverCam [Khan *et al.*, 2005] can make viewing of a 3D surface approximately as easy as scrolling a 2D plane, e.g. by automatically re-orienting the camera when the user reaches the edge of an object's face.

Changes in the camera view can only reduce occlusion if there are points in a 3D scene where objects are not occluded by other objects. In the case of volumetric data, the inner region of a volume may not be visible from any point of view.

A.6 Changing the Projection Method

The previous section assumed a traditional perspective or orthographic camera projection, where changes of the camera view correspond simply to changes in the position or orientation of the camera. Here we consider changes in the projection method for the

camera, i.e. changes in the way the user's 2D view is generated from the 3D scene.

In computer graphics, projections are usually thought of as forward mappings from points in 3D space to points in the 2D camera plane. However, it is also worthwhile to consider more general projection methods which can be defined as an inverse mapping from points in the camera plane to (straight or curved) rays in 3D space. With such a method, one point in 3D space may correspond to multiple locations on the camera plane, as can occur in real world situations involving reflection or refraction.

Such general projection methods, which may be non-linear or non-affine, and/or involve combining or stitching together many simpler projections, have the ability to show different sides of an object in a single image, or show a single object from multiple perspectives. The depictions of objects that result can be inspired by styles of past artistic movements, and have applications in modern non-photorealistic rendering, but may also be useful in visualization. Examples of such work include [Agrawala *et al.*, 2000], who combine images from multiple cameras into a single image, and [Singh, 2002], who defines a single virtual camera that smoothly interpolates views from other perspective cameras. Singh's [2002] work was later applied in a context where the views of the perspective cameras were chosen automatically [Singh and Balakrishnan, 2004]. A survey of work on non-linear and non-affine projection is found in Coleman [2004].

In a similar spirit, unfolding or flattening of 3D structures can been used to create a single 2D image which preserves much of the important original information. Bartrolí et al. [2001] applied this idea to unfolding scans of colons, and Kanitsar et al. [2003] applied it to flattening vascular structures. Because the output is 2D, navigation of the information becomes far easier, and occlusion may be completely eliminated, though some of the information in the original 3D relationships is necessarily lost.

Spheres are one of the simplest 3D surfaces, however projections of the earth's surface onto 2D maps have long been an important issue in cartography. Many projections have been proposed, each with its own advantages and disadvantages with respect to various

criteria such as preservation of area, of lengths, of directions, etc. One of the best known is the Robinson projection, developed in 1963 by Arthur H. Robinson, which offers a compromise on many of the criteria, and which was adopted by the National Geographic Society of the United States between 1988 and 1998. The Robinson projection continues to be used in many textbooks.

Many projections from 3D scenes to 2D images are equivalent to some deformation of the 3D scene. These projections can be seen as $\mathbb{R}^3 \mapsto \mathbb{R}^2$ mappings, whereas deformations from 3D to 3D that are followed by a traditional (e.g. perspective) projection are $\mathbb{R}^3 \mapsto \mathbb{R}^3 \mapsto \mathbb{R}^2$ mappings. Thus, each perspective view of each deformation can be implemented as a non-linear projection of the original, undeformed data.

There are, nevertheless, important differences between the two approaches. An $\mathbb{R}^3 \mapsto \mathbb{R}^3$ deformation results in a 3D object. While this object may have an unfamiliar shape and be highly distorted, its 3D shape can be ascertained through interactive rotation, i.e. by viewing it from different angles. In contrast, the 2D result of a non-linear projection may also appear highly distorted, and be difficult to understand even with interactive adjustment of the projection's parameters.

On the other hand, projections also offer one advantage over deformations. Not every projection can be implemented as a distortion. As explained above, with the more general kind of projection methods, a single point in 3D may correspond to multiple points in 2D, e.g. when showing a single element of a scene from multiple perspectives. Deformations, on the other hand, cannot "duplicate" points (at least, not under the definition of deformation we have been considering) and displace the copies to different locations.

Projections may be effective at eliminating occlusion in simple 3D scenes, where most of the information is on or near a surface of the object(s). If the output is a single 2D image that contains all of the essential information, this can vastly simplify user interaction. However, if the surface is highly non-planar (e.g. has a high topological

genus), or much information is located throughout a volume of space, then projection may not be effective, or be very difficult to understand and adjust interactively.

A.7 Observations

As already pointed out, removal of data (§ A.2) and transparency (§ A.3) have the disadvantage that the de-occlusion of some data requires that other data be removed or be made less visible. This might create a loss of context for the user.

If the user is focused on one region of a data set, making that focal region visible is essential. This corresponds to what Chris Henze [1998] has called a *location query*, where the user wants to know "what exists at a particular field location?". Seeing how the local region is integrated with the surrounding context might also be useful, but is perhaps not essential. On the other hand, in the case of a *condition query*, "where in the field are particular conditions satisfied?" [Henze, 1998], the user needs *all* the pertinent data to be visible. Having some of it removed could be very detrimental to the task.

The other strategies surveyed (§ A.4–A.6) do not remove any data. Camera control (§ A.5) does not change the embedding of the data, only the viewpoint, and thus has fairly limited power to address occlusion, especially in the case of volumetric data. Projection (§ A.6) and deformation (§ A.4) are related in their use of distortion of the data's embedding, although they each have pros and cons with respect to each other. Projection can be mathematically more general (at least, in the sense explained in § A.6), and its output is 2D, which in some cases is sufficient. However, for general volumetric data, it seems likely that projections are too surface-oriented to be useful in all cases. Projection and deformation also have a shorter history in the visualization literature than removal of data and use of transparency, and may yet yield many fruitful results.

One overall positive observation to make is that *all* of the general strategies surveyed in § A.2–A.6 can be combined when visualizing a data set, possibly complementing each

other. Thus, the choice of what tradeoffs to make can be left to a large extend to the user, based on the nature of the data being examined. Although a given strategy may have disadvantages with respect to others, there is still value in pursuing it as a research direction, for the sake of its own advantages. Ultimately, the best strategies can be combined after they have each been explored to a sufficient extent.