Sparks of Innovation in Human-Computer Interaction,
B. Shneiderman, Ed., Ablex Publ., Norwood, NJ (1993)

6.3 Treemaps: a space-filling approach to the visualization of hierarchical information structures

Brian Johnson
Ben Shneiderman

Abstract

This paper describes a novel method for the visualization of hierarchically structured information. The treemap visualization technique makes 100% use of the available display space, mapping the full hierarchy onto a rectangular region in a space-filling manner. This efficient use of space allows very large hierarchies to be displayed in their entirety and facilitates the presentation of semantic information.

Proc. of the 2nd International IEEE Visualization Conference (San Diego, Oct. 1991) 284-291.

Introduction

A large quantity of the world's information is hierarchically structured:manuals, outlines, corporate organizations, family trees, directory structures, internet addressing, library cataloging, computer programs... and the list goes on. Most people come to understand the content and organization of these structures easily if they are small, but have great difficulty if the structures are large.

We propose an interactive visualization method for presenting hierarchical information called treemaps. We hope that the treemap approach is a step forward in the visualization of hierarchical information, and that it will produce benefits similar to those achieved by visualization techniques in other areas.

As humans we have the ability to recognize the spatial configuration of elements in a picture and notice the relationships between elements quickly. This highly developed visual ability allows people to grasp the content of a picture much faster than they can scan and understand text (Kamada, 1988).

The treemap visualization method maps hierarchical information to a rectangular 2-D display in a space-filling manner; 100% of the designated display space is utilized. Interactive control allows users to specify the presentation of both structural (depth bounds, etc.) and content (display properties such as color mappings) information. This is in contrast to traditional static methods of displaying hierarchically structured information, which generally make either poor use of display space or hide vast quantities of information from users. With the treemap method, sections of the hierarchy containing more important information can be allocated more display space while portions of the hierarchy which are less important to the specific task at hand can be allocated less space (Furnas 1986; Henry & Hudson, 1990).

Treemaps partition the display space into a collection of rectangular bounding boxes representing the tree structure (Shneiderman, 1990). The drawing of nodes within their bounding boxes is entirely dependent on the content of the nodes, and can be interactively controlled. Since the display size is user controlled, the drawing size of each node varies inversely with the size of the tree (i.e., number of nodes). Trees with many nodes (1000 or more) can be displayed and manipulated in a fixed display space.

The main objectives of our design are:

Efficient space utilization

Efficient use of space is essential for the presentation of large information structures.

Interactivity

Interactive control over the presentation of information and real time feedback are essential.

Comprehension

The presentation method and its interactive feedback must facilitate the rapid extraction of information with low perceptual and cognitive loads.

Esthetics

Drawing and feedback must be esthetically pleasing.

Hierarchical information structures contain two kinds of information: structural (organization) information associated with the hierarchy, and content information associated with each node. Treemaps are able to depict both the structure and content of the hierarchy. However, our approach is best suited to hierarchies in which the content of the leaf nodes and the structure of the hierarchy are of primary importance, and the content information associated with internal nodes is largely derived from their children.

Motivation: current methods and problems

This work was initially motivated by the lack of adequate tools for the visualization of the large directory structures on hard disk drives.

Traditional methods for the presentation of hierarchically structured information can be roughly classified into three categories: listings, outlines, and tree diagrams. It is difficult for people to extract information from large hierarchical information structures using these methods, as the navigation of the structure is a great burden and content information is often hidden within individual nodes (Vincente, Hayes, & Williges, 1987).

Listings are capable of providing detailed content information, but are generally very poor at presenting structural information. Listings of the entire structure with explicit paths can provide structural information, but require users to parse path information to arrive at a mental model of the structure. Alternatively, users may list each internal node of the hierarchy independently, but this requires users to manually traverse the hierarchy to determine its structure. Outline methods can explicitly provide both structural and content information, but since the structural indentation can only be viewed a few lines at a time, it is often inadequate (Chimera, Wolman, Mark, & Shneiderman, 1991).

The number of display lines required to present a hierarchy with both the listing and outline methods is linearly proportional to the number of nodes in the hierarchy. These methods are inadequate for structures containing more than a few hundred nodes. A great deal of effort is required to achieve a mental model of the structure in large hierarchies using these methods.

Tree drawing algorithms have traditionally sought efficient and esthetically pleasing methods for the layout of node and link diagrams. These layouts are based on static presentations and are common in texts dealing with graph theory and data structures. They are excellent visualization tools for small trees (Bruggemann-Klein & Wood, 1989; Henry & Hudson, 1990; Kamada, 1988; Knuth, 1973; Robertson, Mackinlay & Card, 1991). However, these traditional node and link tree diagrams make poor use of the available display space. In a typical tree drawing more than 50% of the pixels are used as background. For small tree diagrams this poor use of space is acceptable, and traditional layout methods produce excellent results. But for large trees, traditional node and link diagrams can not be drawn adequately in a limited display space. Attempts to provide zooming and panning have only been only partially successful (Henry & Hudson, 1990).

Another problem with tree diagrams is the lack of content information; typically each node has only a simple text label. This problem exists because presenting additional information with each node quickly overwhelms the display space for trees with more than just a few nodes.

The presentation of content information in all of these traditional methods has usually been text-based, although tree diagrams are a graphically based method capable of making use of many of the visualization techniques presented in this paper. Unfortunately, global views of large tree diagrams require the nodes to be so small that there is virtually no space in which to provide visual cues as to node content.

Treemaps efficiently utilize the designated display area and are capable of providing structural information implicitly, thereby eliminating the need to explicitly draw internal nodes. Thus, much more space is available for the rendering of individual leaf nodes, and for providing visual cues related to content information.

Treemaps provide an overall view of the entire hierarchy, making the navigation of large hierarchies much easier. Displaying the entire information structure at once allows users to move rapidly to any location in the space. As Beard states in his paper on navigating large two-dimensional spaces (Beard & Walker, 1990), "If the two-dimensional information space fits completely onto a display screen, there is no navigation problem .... Users are never lost because they can see the complete information space."

A directory tree example

Obtaining information about directory trees was the initial motivation for this research and provides a familiar example domain. For illustrative reasons, the hierarchy in this example is small and nodes have only an associated name and size. While reading through this example, think about how the techniques described would scale up to a directory tree containing 1000 files. An Apple Macintosh screen snapshot showing a treemap of 1000 files from one of our laboratory's hard disk drives follows this example.

Presenting directory structures is a very practical problem. The following are the methods widely available today:

- Command Line Listing (e.g., UNIX "ls", DOS "dir");
- Outlines (e.g., UNIX "du", Microsoft Windows);
- Windowing (e.g., Macintosh Finder);
- Tree Drawings (e.g., OpenWindows File Manager).

We are not aware of approaches that provide a visual representation of the relative sizes of files or directories.

Even moderately sized directory trees are difficult to visualize using standard operating system interfaces. With command line interfaces such as UNIX "ls" or DOS "dir", only the immediate children of any directory are listed. An overall view of the directory tree must be pieced together by traversing the various paths and listing the immediate children of the currently active directory.

Desktop metaphors and their windowing strategies are another alternative. One of the problems with windows is that they often obscure each other, and users may spend much of their time arranging windows. Also, the tree structure is not apparent unless windows have been carefully placed. Desktop icons generally show only the type of the file. Much richer visual mappings are possible but are currently not available, for instance, the depth of an icon's shadow could be used to indicate file size.

We will use a small directory tree hierarchy as an example. Tree A depicted in Figures 1 through 7 contains 23 nodes; of these, 6 are directories (internal nodes) and 17 are files (leaf nodes). This tree is structured such that among siblings, file nodes always precede directory nodes.

In Figure 1 we see an outline view similar to the presentations provided by PCShell under DOS, the UNIX command "du", or Microsoft Windows 3.0. This presentation requires 23 lines; a structure with 1000 files would require a minimum of 1000 lines in order to present both directories and files.

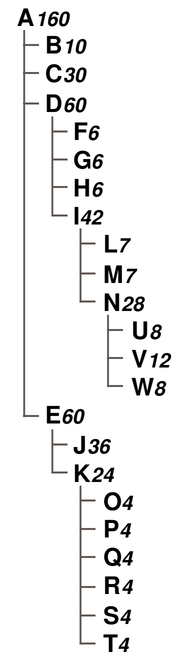

Figure 1. Outline

Figure 2 presents a typical tree diagram; such drawings can be found in graph theory textbooks. This tree drawing approach is similar to the presentation method used by the OpenWindows File Manager. Directory trees with 1000 files cannot be drawn all at once on a typical screen (if all files are at the same level, each file node will have less than one pixel in which to draw itself). The problem becomes even more severe when real file names are used as node labels.

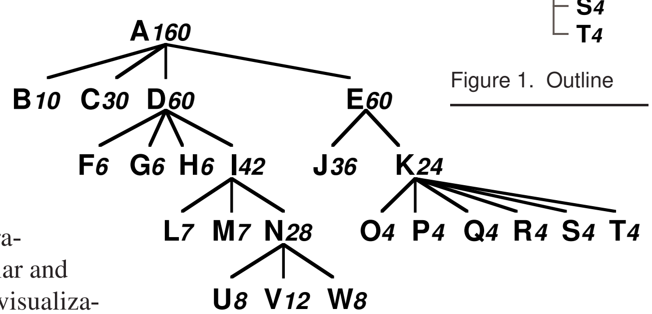

Figure 2. Tree diagram

Figure 3 presents the same information in yet another manner, as a Venn diagram. We use this figure for illustrative purposes as a familiar and often used set theoretic visualization technique. It is an intermediate step which facilitates the transition from traditional presentations to treemaps. This is an odd use of Venn diagrams, as one does not usually think of files and directories as sets. However, simple directory structures can be thought of as set theoretic collections of files, using only the containment (subset) property. Note that each node has been drawn proportionate to its size.

The space required between regions would certainly preclude this Venn diagram representation from serious consideration for larger structures. Note that this "waste" of space is also present in traditional tree diagrams. Using boxes

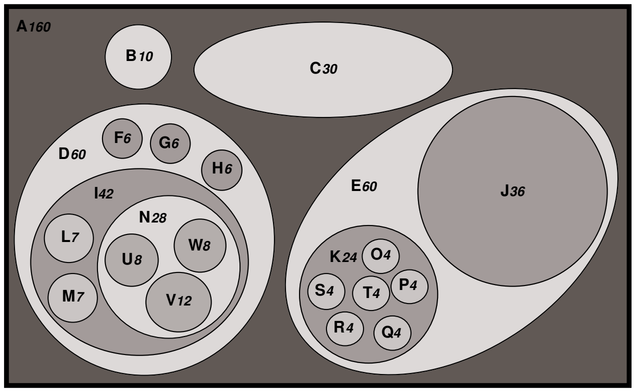

Figure 3. Venn diagram

instead of ovals and a bin-packing algorithm could partially solve this space problem. But bin-packing is an NP-complete problem and does not preserve order.

Figure 4 is a box-based Venn diagram which illustrates a more efficient use of space and is an excellent tool for the visualization of small hierarchies. But even the small degree of nesting present in this technique renders it unsuitable for the presentation of large hierarchies. Fortunately space efficient results can be achieved without bin-packing, using our "slice and dice" treemap approach, a simple linear method in which the algorithm works top-down. An analogy should quickly illustrate this concept. If the hard disk drive were a large, flat, rectangular cheese, one could certainly slice it into chunks representing the size of each top level directory. Applying this slice and dice algorithm recursively to each piece of the cheese, and rotating the slicing direction 90 degrees at each recursive step, would result in the treemap of Figure 5.

Figure 5 simply eliminates the nesting offset used to seperate objects at each level. If we wanted to distribute our cheese to 17 people based on their weights,

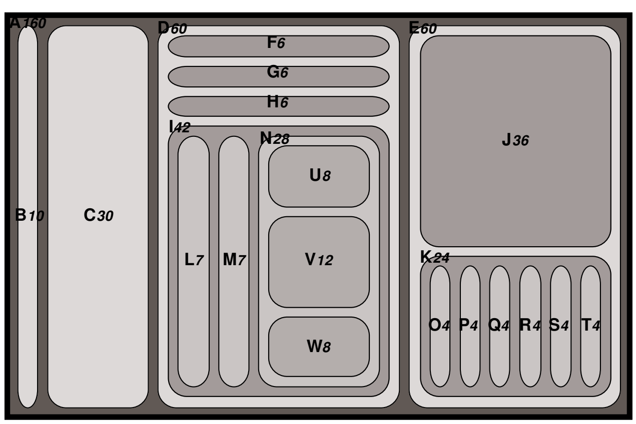

Figure 4. Nested treemap

Figure 5 would give us a slicing diagram. This weight-proportionate distribution is one of the important features of treemaps. The treemap snapshots of Figures 6 and 7 (see color plates) are the full color, machine generated screen snapshots of Figures 4 and 5. All screen snapshots in this paper have been made while using our TreeViz application on an Apple Macintosh II.

Figure 8 (see color plates) is a screen snapshot showing a treemap of 1000 files. A simple color mapping has been used to code some of the various Macintosh file types: treemap applications are red; all other applications are purple; system files are green; picture files are magenta; text files are yellow; archive files are cyan; and all other file types not currently of primary interest are gray. This treemap shows 21 root level files on the left, followed by 19 root level directories moving across to the right. Detailed file information is displayed in a pop-up dialog window as the mouse is dragged over files in the display.

In this directory structure it can be observed that purple application files are generally the largest files on this disk, and take up relatively the same percentage of overall disk space as system related (green) files. A duplicate set of files exists just

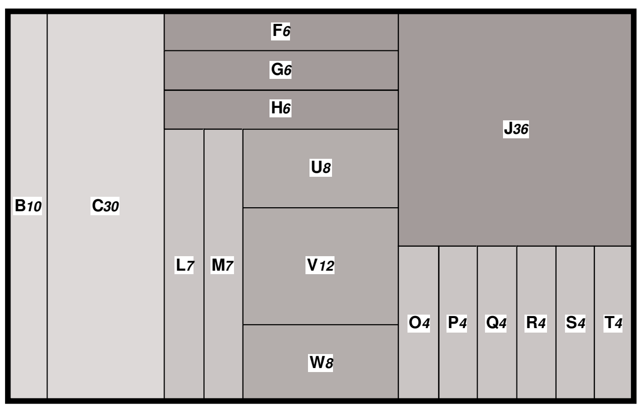

Figure 5. Treemap

to the right of the vertical green bar. The files in this root level folder can be seen duplicated one level down in subfolders, as repeating geometric patterns offset 900 from their parent.

Since this treemap portrays the overall allocation of disk space, the largest files can be located quite easily. Sorting a large directory listing by size would also make finding the largest files easy, but these files would not be presented in their original context. In addition, sorting a list on two or more properties (i.e., size and type) makes presentation of the results difficult. Treemaps make finding the largest system, application, and picture files on the disk as easy as finding the largest green, purple, and magenta rectangles in Figure 8. This is one simple example of the visual display properties possible; further discussion is contained in section 4.2.

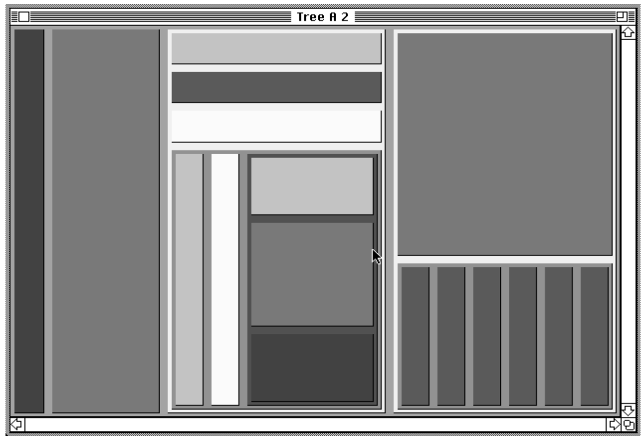

Figure 6. Nested treemap

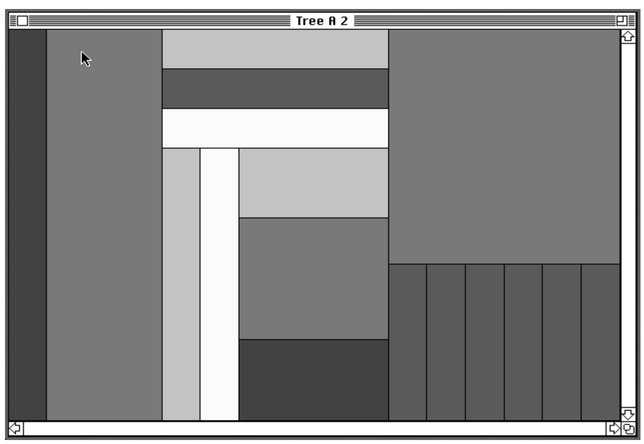

Figure 7. Non-nested treemap

The treemap method

Displaying a directory tree while fully utilizing space and conveying structural information in a visually appealing and low cognitive load manner is a difficult task, as these are often opposing goals. Our interactive approach to drawing directory trees allows users to determine how the tree is displayed. This control is essential, as it allows users to set display properties (colors, borders, etc.) maximizing the utility of the drawing based on their particular task.

Structural information: partitioning the display space

Treemap displays look similar to the partition diagrams of quad-trees and k-D trees. The key difference is the direction of the transformation. Quad-trees create hierarchical structures to store 2-D images efficiently (Samet, 1989), while

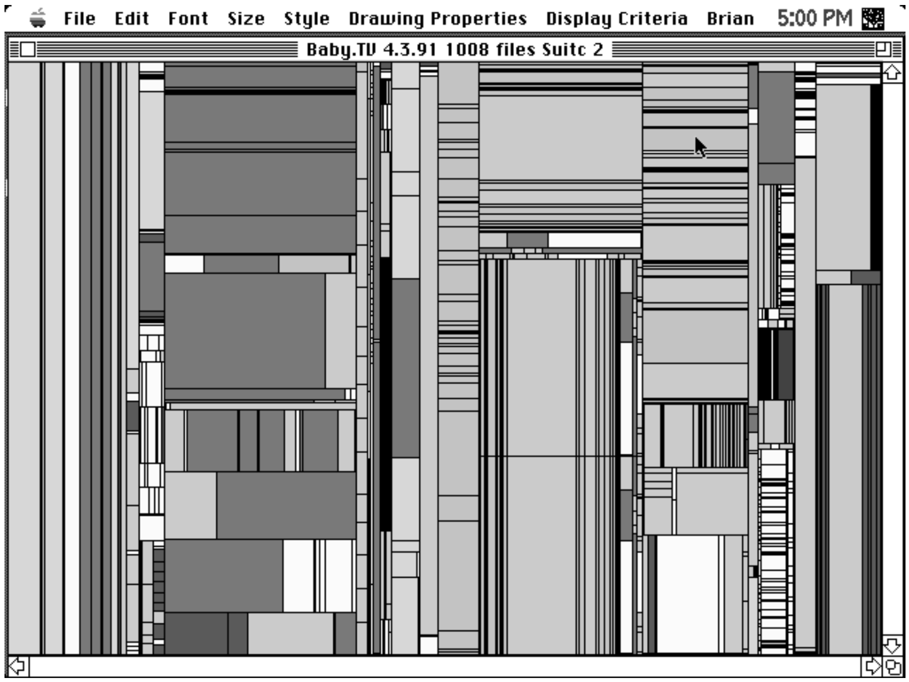

Figure 8. Treemap with 1000 Files

treemaps present hierarchical information structures efficiently on 2-D display surfaces.

Treemaps require that a weight be assigned to each node; this weight is used to determine the size of a nodes bounding box. The weight may represent a single domain property (such as disk usage or file age for a directory tree), or a combination of domain properties (subject to Property 4 below). A node's weight (bounding box) determines its display size and can be thought of as a measure of importance or degree of interest (Furnas, 1986).

The following relationships between the structure of the hierarchy and the structure of its treemap drawing always hold:

Properties

1) If Node 1 is an ancestor of Node 2, then the bounding box of Node 1 completely encloses, or is equal to, the bounding box of Node 2.
2) The bounding boxes of two nodes intersect if one node is an ancestor of the other.
3) Nodes occupy a display area strictly proportional to their weight.
4) The weight of a node is greater than or equal to the sum of the weights of its children.

Structural information in treemaps is implicitly presented, although it may also be explicitly indicated by nesting child nodes within their parent. Nesting provides for the direct selection of all nodes, both internal and leaf. The space required for nesting reduces the number of nodes which can be drawn in a given display space and hence reduces the size of the trees that can be adequately displayed compared to non-nested drawings (Travers, 1989).

A non-nested display explicitly provides direct selection only for leaf nodes, but a pop-up display can provide path information as well as further selection facilities. Non-nested presentations cannot depict internal nodes in degenerate linear sub-paths, as the bounding boxes of the internal nodes in the sub-path may be exactly equal. Such paths seldom occur and tasks dependent on long chains of single child nodes will require special treatments.

Content information: mapping content to the display

Once the bounding box of a node is set, a variety of display properties determine how the node is drawn within it. Visual display properties such as color (hue, saturation, brightness), texture, shape, border, blinking, etc., are of primary interest, but the interface will not limit users to purely visual properties (Ding & Mateti, 1990). Color is the most important of these visual display properties, and it can be an important aid to fast and accurate decision making (Hoadley, 1990; MacDonald, 1990; Rice, 1991). Auditory properties may also be useful in certain circumstances. Nodes may have many domain dependent properties, in which case a rich set of mappings exists between content information and display properties.

The drawing of individual nodes within their bounding boxes determines the content information statically presented in a treemap. The number and variety of domain properties that can be statically coded in the drawing of the tree is limited. As Kuhn states, "Since human perception imposes an upper bound on the complexity of graphic representations, only a small number of relations can be shown." (Ellson, 1990; Kuhn, 1990) Interactive control of the drawing is therefore critical because the mapping of content information to the display will vary depending on the information users require. Dynamic feedback is provided by a pop-up window which displays information about the node currently under the cursor.

For example, files could have weights (display size) proportional to their creation date, color saturation dependent on their last modification date, and pitch (tone heard while crossing border) based on size. Using this scheme it is easy to locate old files which have changed recently, and as the cursor crosses into their bounding box a deep tone tells users that the file is large even before they read the information about that file.

Algorithms

Algorithms are given to draw a treemap and to track cursor movement in the tree. The algorithms may be applied to any tree, regardless of its branching degree. Both algorithms appear on the following page as Figures 9 and 10.

```
DrawTree() The node gets a message to draw itself         The Root node is set up prior to the
                                                             original recursive call
{   doneSize = 0;                                         The percent of this node's subtree
                                                             drawn thus far
    PaintDisplayRectangle();                              The node sends itself a Paint
                                                             Message
    switch (myOrientation) {                              Decide whether to slice this node
                                                             horzontally or vertically
        case HORIZONTAL:
            startSide = myBounds.left;                    Set start for horizontal slices
        case VERTICAL:
            startSide = myBounds.top;                     Set start for vertical slices
    }
    if (myNodeType == Internal) {                         Set up each child and have it draw
                                                             itself
        ForEach (childNode) Do {
            childNode->SetBounds(startSide, doneSize, myOrientation); Set childs bounds
                                                             based on the parent partition
                                                             taken by previous children of
                                                             parent
            childNode->SetVisual();                       Set visual display properties (color,
                                                             etc.)
            childNode->DrawTree();                        Send child a draw command
}}}

SetBounds(startSide, doneSize, parentOrientation)
{   doneSize = doneSize + mySize;                         How much of the parent will have
                                                             been allocated after this node
    switch (parentOrientation) {                          Decide which direction parent is
                                                             being sliced
        case HORIZONTAL:
            myOrientation = VERTICAL;                     Set direction to slice this node for its
                                                             children
            endSide = parentWidth * doneSize / parentSize; How much of the parent will have
                                                             been sliced after this node
            SetMyRect(startSide + offSet,                 Left side, Offset controls the nesting
                                                             indentation
                parentBounds.top + offSet,                Top
                parentBounds.left + endSide - offSet,     Right
                parentBounds.bottom - offSet);            Bottom
            startSide = parentBounds.left + endSide;      Set start side for next child
        case VERTICAL:
            myOrientation = HORIZONTAL;                   Set direction to slice this node for its
                                                             children
            endSide = parentHeight * doneSize / parentSize;
            SetThisRect(parentBounds.left + offSet,       Left side
                startSide + offSet,                       Top
                parentBounds.right - offSet,              Right
                parentBounds.top + endSide - offSet);     bottom
            startSide = parentBounds.top + endSide;       Set start side for next child
 }}
```

Figure 9. Drawing AlgorithmThe basic drawing algorithm produces a series of nested boxes representing the structure of the tree.

```
FindPath(point the Point)
{   if node encloses thePoint then
    for each child of thisNode do {
    path = findPath(thePoint);
       if (path != NULL) then
          return(InserlnList(this Node, path));          Add child to path
}
return (NULL);                                          Start path, the Point is in this node,
                                                           but not in any of its children
}
```

Figure 10. Tracking algorithm

The cursor tracking algorithm facilitates interactive feedback about the tree. Every point in the drawing corresponds to a node in the tree. While the current tracking point (from a mouse or touchscreen input device) is in a node, the node is selected and information about it is displayed.

Drawing algorithm

The treemap can be drawn during one pre-order pass through the tree in O(n) time, assuming that node properties (weight, name, etc.) have previously been computed or assigned. The current algorithm has been implemented in object-oriented Think C on a Macintosh II. The drawing algorithm proceeds as follows:

1) The node draws itself within its rectangular bounds according to its display properties (weight, color, borders, etc.).

2) The node sets new bounds and drawing properties for each of its children, and recursively sends each child a drawing command. The bounds of a node's children form either a vertical or horizontal partitioning of the display space allocated to the node.

Tracking algorithm

The path from the root of the tree to the node associated with a given point in the display can be found in time proportional to the depth of the node.

In our implementation, when a node draws itself it stores its bounding box in an instance variable. Every point in the treemap corresponds to a node in the hierarchy; in addition, every node is contained in the bounding box of the root node. Recall that each node's bounding box completely encloses the bounding boxes of its children, and that the bounding boxes of sibling nodes never overlap. Finding the path to a node containing a given point thus involves only a simple descent through one path in the tree, until the smallest enclosing bounding box is found.

Coping with size

A typical 13-inch display has a resolution of 640 x 480, or roughly 300,000 pixels. Drawing an 80mb directory tree (weight = disk usage) on such a display requires that each pixel represent 260 bytes, i.e., there are roughly 4 pixels per

Kilobyte. Assuming that such a directory structure may contain roughly 3,000 files (as on one of our lab's hard disks) implies that there are approximately 100 pixels per file on average. A box with 10 pixels per side (roughly $4mm^2$) is easily selectable using a standard mouse or touchscreen device (Sears & Shneiderman, 1990). This average case analysis is only part of the story since file sizes may vary widely.

The range of file sizes on our hard disk varied from a few hundred bytes to well over one million bytes. In the treemap of Figure 8, groups of very small files often become completely black regions as there is only enough space to draw their borders. Magnification over these regions or zooming can provide access to these files. But since the assignment of node weights can be user controlled, presumably the nodes with the greatest weights are of greatest interest and the nodes with the smallest weights are of least interest.

Future research directions

Further research includes the exploration of alternate structural partitioning schemes, appropriate visual display of both numeric and non-numeric content information, dynamic views such as animated time slices, and operations on elements of the hierarchy. Standard operations such as zooming, marking, selecting and searching also invite designers to explore variations on the treemap strategy.

Dr. Ram Naresh-Singh, a visiting research scientist in our lab, is working on an alternate directory only approach to partitioning the display which we have termed "top-down". His implementation on a Sun Sparcstation preserves the traditional notion of having the root node at the top and the leaves at the bottom.

Animation, or time-sliced displays, could provide insight into evolving structures. For example, the hierarchical organization of a university could be mapped from the university level (root), to the college level, to the department level, to the research lab level. If weights were assigned based on personnel resources, it would be easy to see the structure of the university based on the distribution of employees, and hence understand its strengths and weaknesses. Furthermore, if the saturation of red was proportionate to the funds spent at each node, and the saturation of cyan (the inverse of red) was proportionate to the funds allocated, nodes (labs, departments, colleges) which were on budget would be shades of gray (equal amounts of red and cyan), nodes over budget would become increasingly red, and nodes under budget would become increasingly cyan. The magnitude of the nodes funding would range from black (small budgets and expenditures) to white (large budgets and expenditures). If a series of these displays are generated based on data over the last ten years, it would be possible to see how funding and personnel resources have evolved and been distributed within the university.

The range and variety of potential applications of this technology is vast. For instance, stock market portfolios are often hierarchically structured. Animations over time of financial portfolios could be a valuable application of this technology.

Conclusion

We believe that space-filling approaches to the visualization of hierarchical information structures have great potential. The drawing algorithm we have given is quite general, and the numerous possibilities for mapping information about individual nodes to the display are appealing. The treemap approach to visualizing hierarchical structures enables meaningful drawings of large hierarchies in a limited space.

Acknowledgments

We would like to acknowledge the support of the members of the Human-Computer Interaction Lab, whose suggestions and criticisms have been greatly appreciated. They have forced us to prove the value of treemaps and allowed us to hone our presentations of the idea.

References

Beard, D. V., Walker II, J. Q. (1990) Navigational techniques to improve the display of large two-dimensional spaces. *Behaviour & Information Technology*, Vol. 9, No. 6, 451-466.

Brüggemann-Klein, A., Wood.D. (July 1989) Drawing trees nicely with tex, *Electronic Publishing*, Vol. 2, No. 2, 101-115.

Card, S. K., Robertson, G. G., Mackinlay, J. D. The information visualizer, an information workspace, *Proc of ACM CHI'91,* Conference on Human Factors in Computing Systems, Information Visualization, 181-188.

Chimera, R., Wolman, K., Mark S., Shneiderman, B. (Feb. 1991) Evaluation of three interfaces for browsing hierarchical tables of contents, Technical Report CAR-TR-539, CS-TR-2620, University of Maryland, College ParK.

\Cox, D. J., The art of scientific visualization,(March 1990) *Academic Computing*, 20.

Ding, C., Mateti, P. (May 1990) A framework for the automated drawing of data structure diagrams, *IEEE Transactions on Software Engineering*, Vol. 16, No. 5, 543-557.

Ellson, R., Visualization at work (March 1990) *Academic Computing*, 26.

Feiner, S. (March 1988) Seeing the forest for the trees: hierarchical display of hypertext structures, *ACM Proc. COIS'88* , Conf. on Office Information Systems (Palo Alto, CA), 205-212.

Furnas, G. W. (1986) Generalized fisheye views, *Proc. of ACM CHI'86* , Conference on Human Factors in Computing Systems, Visualizing Complex Information Spaces, 16-23.

Henry, T. R., Hudson, S. E. (May 1990) Viewing large graphs, Technical Report 90-13, University of Arizona.

Hoadley, E. D. (February 1990) Investigating the effects of color. *Communications of the ACM*, Vol. 33, No. 2, 120-139.

Kamada, T. (Dec. 1988) On visualization of abstract objects and relations, Ph.D. thesis, University of Tokyo, Department of Information Science, 7-3-1 Hongo, Bunkyo-ku, Tokyo, 113 JAPAN.

Knuth, D. E. (1973) *Fundamental Algorithms, Art of Computer Programming,* vol. 1, Addison-Wesley, Reading, MA, 2nd edition.

Kuhn, W. (1990) Editing spatial relations, *Proc. of the 4th International Symposium on Spatial Data Handling*, (Zurich, Switzerland) 423-432.

Rice, J. R. (March 1991) Ten rules for color coding, *Information Display*, Vol. 7. No. 3, 12-14.

Robertson, G.G., Mackinlay, J. D., Card, S. K. (1991) Cone trees: animated 3d visualizations of hierarchical information, *Proc. of ACM CHI'91,* Conference on Human Factors in Computing Systems, Information Visualization, 189-194.

Samet, H. (1989) *Design and Analysis of Spatial Data Structures*, Addison-Wesley Publishing Co., Reading, MA.

Sears, A., Shneiderman, B. (April 1991) High precision touchscreens: design strategies and comparisons with a mouse. *International Journal of Man-Machine Studies*, Vol. 34, NO. 4, 593-613.

Shneiderman, B. (Sept. 1990.) Tree visualization with tree-maps: a 2-d space-filling appoach, *ACM Transactions on Graphics,* Vol. 11, No. 1, 92-99.

Travers, M. (1989) A visual representation for knowledge structures, *ACM Hypertext'89 Proc.*, Implementations and Interfaces, 147-158.

Tufte, E.R. (1983) *The Visual Display of Quantitative Information*, Graphics Press, Cheshire, CT.

Vincente, K. J., Hayes, B. C., Williges, R. C. (1987) Assaying and isolating individual differences in searching a hierarchical file system, *Human Factors*, Vol. 29, No. 3,:349-359.