Rendering Hierarchical Data

Jarke J. van Wijk, Frank van Ham, and
Huub van de Wetering

Why is my hard disk full?
A question no doubt familiar to many readers, and one that has inspired our research for several years. Our goal is to provide more insight into large, hierarchical data sets, commonly known as trees. Hierarchical data sets can be found everywhere. A large number of items, such as files, products, employees, and stocks, can be handled and managed much more efficiently when they are grouped into larger entities. Recursive application of this approach results in a tree structure. A hierarchical file system is a prime example: The user can organize his disk, and only has to deal with a limited set of files while fulfilling a task.

But the PC disk of an average user often contains dozens of gigabytes of data, distributed over hundreds of thousands of files. In this case it becomes difficult to maintain an overview, and to determine what is cluttering the disk. Often no single, simple answer exists. Perhaps another user of the PC has installed some large programs, or has failed to cleanup after finishing a task in his or her relief at meeting a deadline. Perhaps multiple copies of the same multimedia file are stored on the disk. How can we find large files and directories and identify patterns and structures easily in such large hierarchical data structures? Automatic methods, such as searching for the largest files, fall short, and standard file browsers, using indented lists, have not been developed with this problem in mind.

We believe the best way to answer our question is to exploit the unique capabilities of the human visual system, tuned and optimized in the course of millions of years of evolution to extract information from images [9]. In other words, let us try to make synthetic images, using a wide variety of visual cues to transfer information as efficiently and effectively as possible.

Tree Visualization

We are obviously not the first to try to visualize trees. The graphical presentation of tree structures has long been studied in the graph drawing community [7]. A stan-

Jarke J. van Wijk (vanwijk@win.tue.nl), full professor, Technische Universiteit Eindhoven, the Netherlands.
Frank van Ham (fvham@win.tue.nl), PhD student, Technische Universiteit Eindhoven, the Netherlands.
Huub van de Wetering (wstahw@win.tue.nl), assistant professor, Technische Universiteit Eindhoven, the Netherlands.

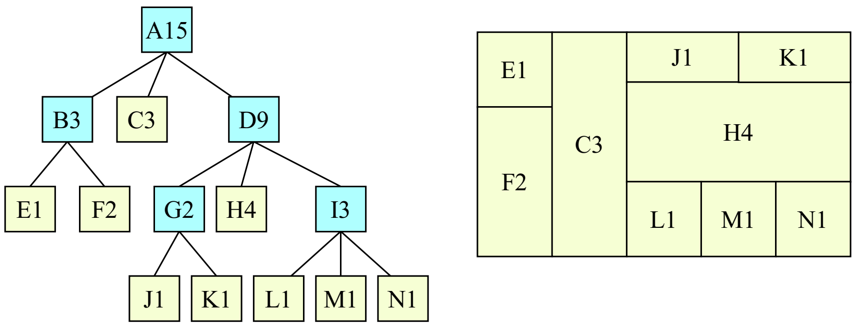

Figure 1. A node-link diagram (left) and the corresponding treemap (right).

dard and widespread representation is the node-link diagram (see Figure 1). Here, each node in the tree is mapped to an icon, typically a circle or square; each edge is mapped to a line or an arrow. A standard arrangement is to put the root of the tree at the top, and to draw the nodes in successive rows, such that each row corresponds to a layer in the tree. A variation is the use of a radial mapping, where the root node is placed in the center and nodes are placed on concentric circles. These methods work well for smaller trees, say up to a hundred nodes, but result in too much clutter in larger trees.

Tree visualization has also been intensively studied within the information visualization community. Whereas graph drawing typically aims at producing static, black-and-white diagrams, in information visualization more means are applied to enable a user to gain insight into abstract data. The use of color, multiple views, three-dimensional representations, and especially interaction enable a user to navigate through data and to understand it more efficiently. A well-known tree visualization is the cone tree [5]: a three-dimensional node-link diagram, with many options for interaction. But cone trees seem to inherit a problem of standard node-link diagrams: they fall short when it comes to large trees. Treemaps [6] were invented by Shneiderman to address this particular problem. A treemap is a space-filling display of hierarchical data, where the size of elements is mapped to the area of rectangles. Treemaps can be very effective, but can also lead to abstract images in which it is difficult to ascertain the structure of the tree.

How can we develop more effective representations of huge hierarchical data structures? An interesting aspect of information visualization is its interdisciplinary nature. Our question can be answered from viewpoints including human perception, cognitive psychology, algorithms, and graphic design. We use computer graphics as our main inspiration. That computer graphics plays a role in the complexities of visualization may seem trivial, but if we examine many of the current results in information visualization, we often find the graphical palette used is poor, often limited to uniformly shaded 2D geometric shapes. As a result, the images are often almost as abstract as the corresponding data. Our perceptual system is capable of processing many more visual cues, such as texture, smooth shading, and depth, so we can expect a higher information throughput if these graphic means are used to encode the data. The computer graphics community has devoted much effort to the rendering of highly realistic images by accurately modeling geometry and illumination. Can we use this knowledge to produce more effective visualizations? Note that we do not consider realism as an aim in itself, but just as a means to get more effective visualizations by triggering intrinsic human capabilities.

We present three methods to visualize trees that illustrate our approach. Cushion treemaps are a variation of standard treemaps, where cushions are added to visualize the structure. Beam trees are another variation on treemaps, where a stack of cylindrical beams depicts the structure. Finally, we consider the use of botanical trees as a metaphor, showing that, when done carefully, this technique can lead to useful and intriguing results.

Cushion treemaps. Each rectangle in a treemap [6], a compact 2D representation of a tree structure, represents a node of a tree, where the area of the rectangle is proportional to an attribute of the node, such as file size. As an example, Figure 1 illustrates a node-link diagram and the corresponding treemap. Each node has a label X*n*, where X is the name and *n* the size. The size of a non-leaf node is equal to the sum of the sizes of its children. The treemap is constructed as follows. We start with a given outer rectangle, which represents the root A of the tree. Next, we subdivide the rectangle from left to right into three smaller ones, where each subrectangle represents one of the children B, C, and D. The area allocated to them is proportional to their size. In the next rounds, we repeat this step recursively. Each rectangle that does not correspond to a leaf is subdivided, where the direction of subdivision alternates between horizontal and vertical. As a result, each inner rectangle represents a leaf, and the structure of the tree can be traced back from the diagram.

Treemaps have been used for many different types of hierarchical data, varying from directory structures via product catalogs to Usenet groups. Many different implementations and applications have been realized; for an up-to-date overview see [7]. Martin Wattenberg has developed a most impressive application of treemaps. His Map of the Market [10] visualizes the market-share and change of a large number of stocks, hierarchically organized into market segments. Nevertheless, for large, deep trees the structure is hard to discern. Another bad case is a balanced tree with leaves of equal size: Here the treemap degenerates into a regular grid.

Several means can be used to emphasize the structure, such as color, line width, and the nesting of rectangles. However, these do not provide a natural cue, consume space, and lead to maze-like diagrams. We considered the use of shaded geometry as an alternative, which has led to *cushion treemaps* [11]. A hierarchy of stacked cushions emphasizes the hierarchical structure of the data. The algorithm is in the spirit of the original treemap: straightforward. During the subdivision of each rectangle at every level a bump is added to each new rectangle, with the main direction of the bump being perpendicular to the direction of subdivision. As a result, each rectangle is provided with a cushion. Deep valleys between cushions indicate major subdivisions, whereas shallow valleys separate siblings.

We have used cushion treemaps extensively for the visualization of directory structures. Examples of both standard and cushion treemaps are shown in Figure 2. Color was used to indicate file type. Our running example, shown in Figures 2 to 4 is of a PC disk running MS Windows 98, which contained about 32,000 files in 1,800 folders. The main columns from left to right denote the folders *Windows* and *Program Files*, followed by a series of smaller folders. The upper half of the *Windows* folder is the *Windows\Desktop* folder. This folder contains a large number of images

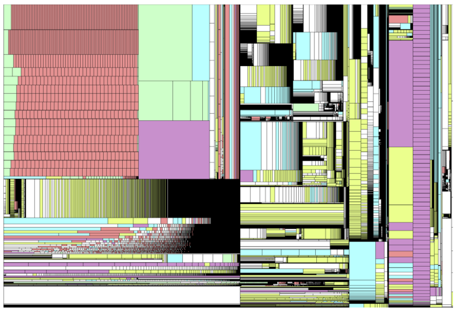
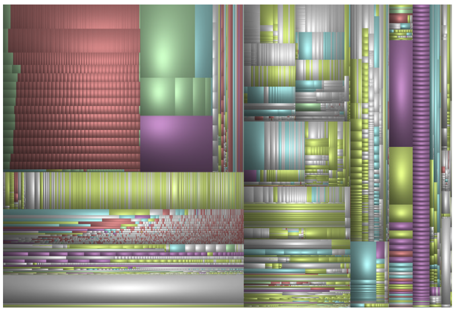
Figure 2. From left to right, standard and cushion treemap of directory structure.

(red) in separate folders (actually they are separate frames of several animations), as well as some large video-files (green) and a large archive-file (purple). The *Windows\System* folder contains a large number of libraries (yellow), the temporary Internet cache contains a large number of small files, including many images. In the lower left corner we see the file *Win386.swp* (grey). Documents and help files are shown in blue. Comparison of the standard and the cushion treemap version reveals that in the latter the hierarchical structure is much easier to detect.

The method has been embedded in a freeware utility, which we dubbed SequoiaView. Many options are provided for selection, coloring, and viewing. Most important is the added interaction: The user can easily find out which file corresponds to which rectangle, navigate through the directory structure, and obtain additional information. This utility has now been downloaded more than 200,000 times from www.win.tue.nl/sequoiaview, and many users have reported that it has been a great help in managing their disks.

Beamtrees. In this section and the next we present two more experimental approaches to visualize hierarchical data. One problem of standard treemaps is that the tree structure is not shown explicitly. For instance, non-leaf nodes cannot be pointed at. How can we visualize the non-leaf nodes? One answer is to use nested rectangles [6], but we found an alternative, which we called *beamtrees* [1].

The basic strategy is the same as that of a standard treemap: the initial rectangle is recursively sliced and diced, but instead of using all the space, we shrink each rectangle perpendicular to the direction of subdivision. Leaf nodes are gathered on one end of the rectangle, and are shown by subdividing this end. The result of this algorithm is an abstract image. The top left image of Figure 3 shows the result for the same abstract tree as in Figure 1, and we see that for instance the root A (with leaf C directly in it) visually falls apart into separate shapes.

Again, we can improve on this by using shaded geometry; here we model the tree as a stack of tubes, or circular beams. As a result, large rectangles overlapped by smaller ones are easier to discern, while the spatial ordering reflects the hierarchical order. The directory shown is the same as in Figure 2, and the same colors were used.

As a result, both leaves and most of the non-leaf nodes are displayed and can be pointed at. Also the size attribute is visualized properly: In the front view all areas are proportional to size. The user can select by which factor the beams are shrunk, and thereby make a trade-off as to how much area is used for leaves with respect to non-leaf nodes. The price to be paid is a less efficient use of screen-space in comparison to

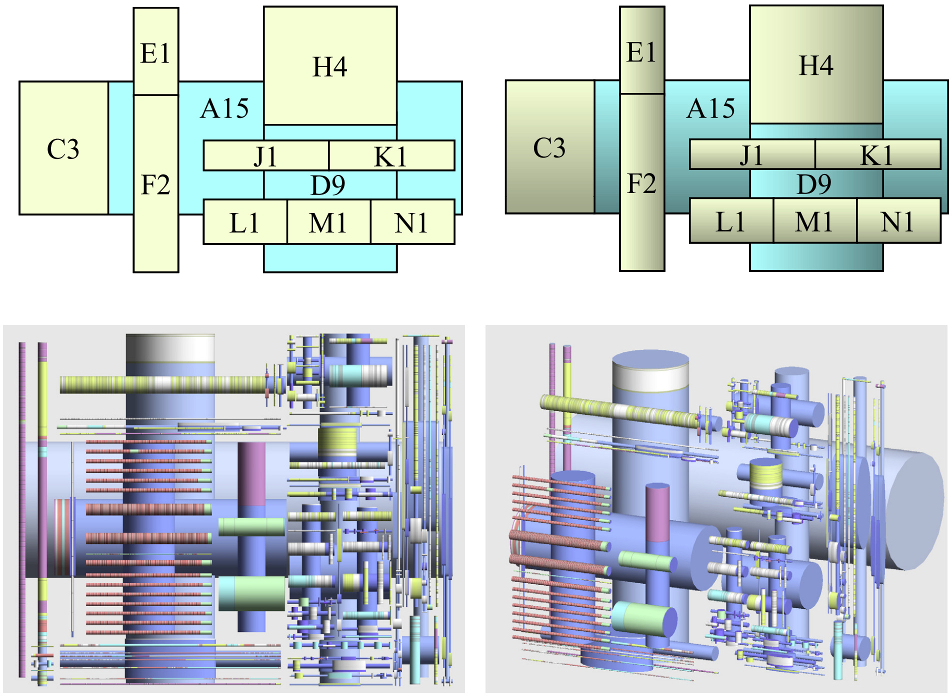

Figure 3. Beamtrees: construction, tubular version, front and oblique view of directory structure.

standard treemaps. User tests revealed that size estimation was slightly slower and harder for beamtrees than for cushion treemaps, but that tasks related to the levels of leaves (such as establishing the level) were much easier to carry out. Furthermore, the users had a strong subjective preference for 3D beamtrees.

Botanical Trees. The preceding two methods produce abstract images. Can we generate natural, 3D models of hierarchical data? The term tree is standard for hierarchical data. When we observe botanical trees, we find that the leaves, branches, and their arrangement can often easily be extracted, in spite of their very large numbers. What would happen if we try to visualize hierarchical data as botanical trees?

Together with our student Ernst Kleiberg, we have answered this question by taking advantage of research in the graphics community in this field. Many methods have been developed to generate realistic-looking trees. Of these, we found the strand model to be the most suitable for our needs. The origin of the strand model goes back as far as Leonardo da Vinci; we based our method [3] on the strand model of Holton [2]. The model is based on the vascular structure of organic objects. Each leaf is connected to a strand, which can be traced back to the root. As a consequence, the cross sectional area of each branch corresponds to the number of leaves that are attached (directly or indirectly) to it. In Holton's model, artificial trees are generated by recursively splitting branches, where the number of strands is more or less randomly distributed over the subbranches. The angles between the subbranches and their proportions are derived from the number of strands in each.

In our case, the structure of the hierarchical data is given, as well as the size attribute of the elements. As illustrated in Figure 4, we map non-leaf nodes to branches, to

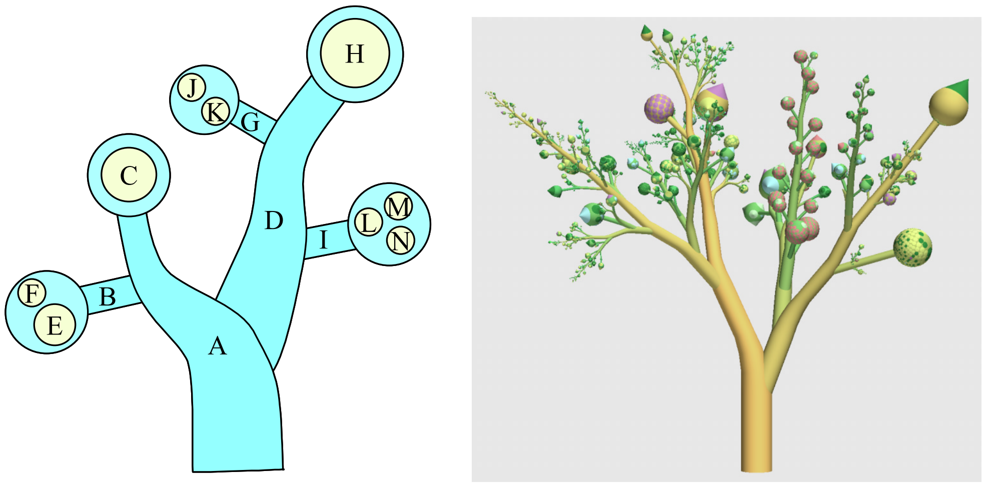

Figure 4. Botanically inspired visualization: construction and application.

which the children are attached as subbranches. The area of the cross section of the main branch is decreased for each subbranch spawned off. We found that the mapping of leaf nodes to botanical leaves with varying sizes gives cluttered and confusing images. A much better solution is to map sets of sibling nodes to fruit. We depict such a set as a sphere, where the area of the sphere is proportional to the sum of the sizes of the leaves. Each individual leaf is mapped to a cone, with size and color denoting file size and file type. A remaining problem is how to distribute the cones over the sphere in a balanced way. Again, an answer could be found in the graphics literature on botanical modeling. Lintermann and Deussen [4] have developed the concept of phi-balls to generate, among others, artificial sunflowers, pineapples, and cacti. We found that their compact and surprisingly effective algorithm could be used also to generate icons for the display of sets of elements with varying size.

This method produces intriguing models. Figure 4 shows again the same directory as the previous figures. A picture does not show the full strength: such complex three-dimensional objects have to be viewed in a setting where the user can inspect the object interactively. The reactions of users on this method are mixed so far. All agree that the images are nice, that large objects (spheres and branches) can easily be detected, and that the phi-balls provide a readable cue on the contents of a directory. However, for practical application most preferred treemaps. For us, the most important result was the discovery of yet another rich source of methods and techniques to translate abstract data into geometry. The phi-ball is a great example in this respect: a simple and elegant method to visualize a list of elements via a compact 3D icon.

Conclusion

We have presented three alternative ways to visualize hierarchical data. In all cases we used geometry and shading, aiming at visual representations that convey more insight. Have we succeeded? Only for the beamtrees we have undertaken a controlled experiment, in order to compare its usability with treemaps. However, the release of SequoiaView can also be considered as an experiment. We got many positive reactions, and from these we conclude that our cushion treemap method is effective in the real world to manage large hard disks. A strong point is the scalibility of the method: users reported that they scanned and analyzed 2.5 TB file systems effectively. Our other methods have not yet escaped from the lab yet, but we intend to develop downloadable versions, since we found the feedback from the real-world user community to be highly interesting and stimulating.

Encouraged by the results, we believe that the use of 3D computer graphics methods for information visualization can lead to much more insight, and we will pursue this path further. For many problems in information visualization, only limited solutions are available, which often do not scale up well for large quantities of data. Two challenging and important problems are the visualization of multivariate data and the visualization of networks. Our future research will aim at the development of new methods for these, and we hope that the use of methods from computer graphics will again lead to new, intriguing, and insightful visualizations.

References

1. Ham, F.J.J. van, Wijk, J.J. van. Beamtrees: Compact Visualization of Large Hierarchies. In Wong, P.C. and Andrews, K. (eds.) In *Proceedings 2002 IEEE Symposium on Information Visualization* (2002), 93–100.

2. Holton, M. Strands, gravity and botanical tree imagery. *Computer Graphics Forum 13*, 1 (March 1994), 57–67.

3. Kleiberg, E., Wetering, H. van de, and Wijk, J.J. van. Botanical visualization of huge hierarchies. In Andrews, K., Roth, S.F., and Wong, P.C. (eds.) In Proceedings 2001 IEEE Symposium on Information Visualization (2001), 87–94.

4. Lintermann, B. and Deussen, O. Interactive modeling of plants. *IEEE Computer Graphics and Applications 19*, 1 (Jan./Feb. 1999), 56–65.

5. Robertson, G.G., Mackinlay, J.D., and Card, S.K. Cone trees: animated 3D visualizations of hierarchical information. In *Proceedings of the Conference on Human Factors in Computing Systems* (1991), 189–194.

6. Shneiderman, B. Tree visualization with tree-maps: A 2-D space-filling approach. *ACM Transactions on Graphics 11*, 2 (Jan. 1992), 92–99.

7. Shneiderman, B. (2002). Treemaps for space-constrained visualization of hierarchies; www.cs.umd.edu/hcil/treemaps-history.

8. Tollis, I.G., Di Battista, G., Eades, P., and Tamassia, R. *Graph Drawing: Algorithms For the Visualization of Graphs*. Prentice Hall, 1999.

9. Ware, C. *Information Visualization: Perception for Design*. Morgan-Kaufmann, 2000.

10. Wattenberg, M. Map of the Market (1998). http://www.smartmoney.com/marketmap.

11. Wijk, J.J. van, and Wetering, H. van de. Cushion Treemaps: visualization of hierarchical information. Wills, G. and Keim. D. (eds.). In *Proceedings 1999 IEEE Symposium on Information Visualization* (1999), 73–78.