Visualizing Hierarchical Data

Ioannis Tsiompikas

May 11, 2009

Contents

1 Hierarchical Data

Hierarchies are an extremely poppular way of organizing information, processes, systems, or even people. Whenever something can be subdivided into parts, that's automatically a hierarchy. Whenever a big task can be broken up into sub-tasks, that's a hierarchy. As a result hierarchically-structured data are extremely common.

Here are some examples of hierarchical data:

- Filesystems: most operating systems use hierarchies of directories, to organize files in data storage devices, such as disks and solid-state memory.
- Classes used in object oriented programming form inheritance hierarchies.
- Usenet newsgroups are organized hierarchically.
- The internet domain name system organizes hosts in a hierarchy of domains and subdomains.
- Languages are hierarchical, with letters forming words, words combine into sentences, which make up paragraphs, etc.

2 Techniques for Hierarchical Data Visualization

Hierarchical data structures are essencially trees, i.e. directed acyclic graphs, so any graph visualization techniques apply. However, it generally tends to be a much easier problem than generic graph visualization, due to the fact that most trees, those where each node can only have one parent, are planar graphs.

2.1 Linked Nodes

The most obvious way to visualize a tree structure, is to arrange its nodes on a plane and link them with lines. There are three common ways to do that: layered trees, radial trees, H-V (horizontal-vertical) trees, and indented trees.

2.1.1 Layered Trees

Layered tree layouts are commonly encountered in data structure literature to show tree stuctures such as binary search trees. In this layout, the root of the node is placed at the very top (or less commonly, bottom) of the image, and its children are placed below it, such that siblings share the same vertical position. Also, nodes are connected together visually with lines or arrows running from each parent to its child nodes (see figure 1).

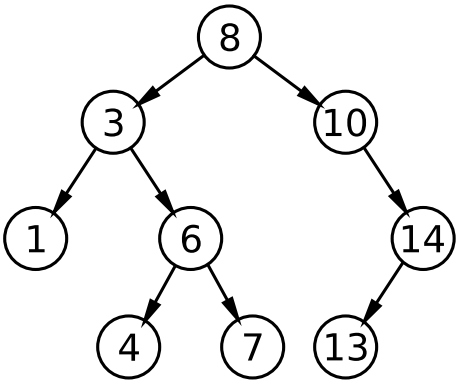

Figure 1: A typical example of a layered tree layout for a binary search tree.

2.1.2 Radial Trees

Radial tree layouts, start with the root in the middle of the image, and place its child nodes in a circle arround it. Succeding layers of the typical layered tree layout, are thus placed at increasingly greater radii from the center. Child nodes are again connected to their parents with lines (figure 2).

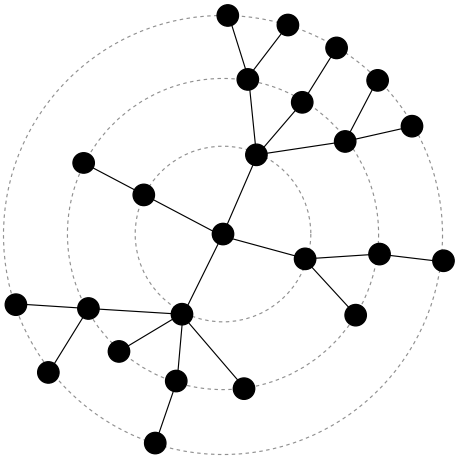

Figure 2: Radial tree layout.

An interesting variation of the radial tree layout, is the *hyperbolic tree*[1]. The hyperbolic tree allocates more image space to the nodes in the center of the view, and compress the more distant nodes to the edge of the image. This is done by laying out the tree nodes in hyperbolic space, and the whole graph into a unit disk in euclidean space for rendering (figure 3).

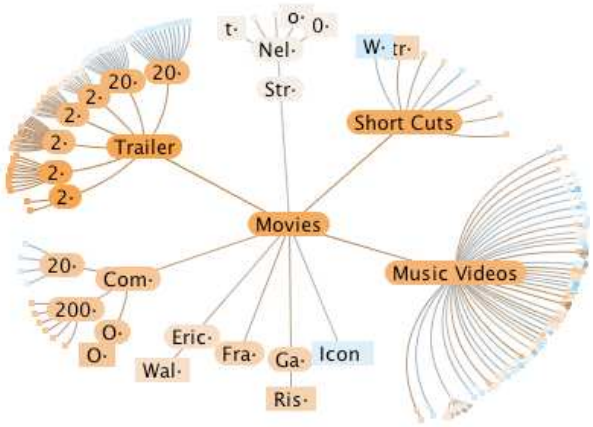

Figure 3: Hyperbolic tree

2.1.3 H-V Trees

Horizontal-Vertical tree placement is typically encountered in lisp data structure visualizations (cons cells). H-V trees place children nodes either horizontally or vertically aligned with its parent (see figure 4).

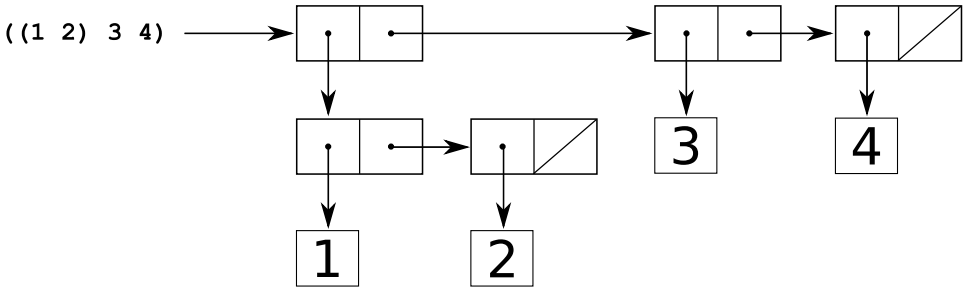

Figure 4: H-V tree placement of lisp cons cells.

2.1.4 Indented Tree Layout

Finally, possibly the simplest way to visualize a tree structure, is to traverse the tree pre-order, and simply list all nodes one under the other, with children indented horizontally in relation to their parent. This method, probably due to its simplicity, is used by almost *all* file managment applications, to show the hierarchy of directories in a filesystem (figure 5).

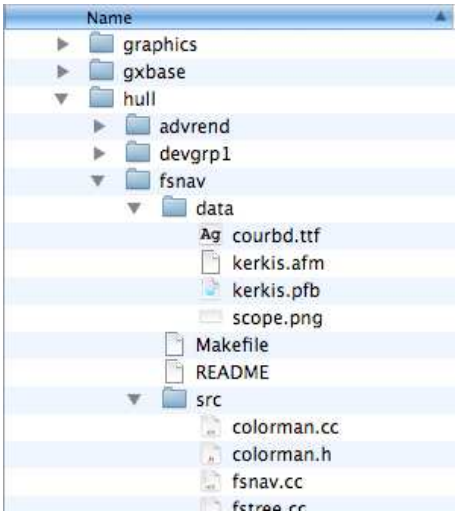

Figure 5: Indented filesystem tree in the MacOS X Finder.

2.2 Treemaps

A very space-efficient tree visualization technique is called *treemap*[2]. In treemaps, a shape representing the whole tree (usually a rectangle) is recursively subdivided horizontally and vertically, so that child nodes are placed within the 2D boundrary of their parent.

A common application of this technique is used to visualize filesystem hierarchies, where each rectangle (directory) is subdivided into parts whose area are proportional to their file size, if they represent files, or the sum of their children's sizes, if they represent subdirectories (figure 6).

Many variations of the basic treemap algorithm exist, such as *cushion treemaps*[3], which use shading to convey nesting level (figure 8), voronoi treemaps[4], circular treemaps, and 3D treemaps (figure 7).

2.3 Icicle and Sunburst Trees

The icicle technique, uses multiple layers of rectangles to visualize the tree structure, where each child sits ontop the parent's rectangle. The sunburst diagram is exactly the same, but drawn radially as co-centric circles instead of rectangles (figure 9).

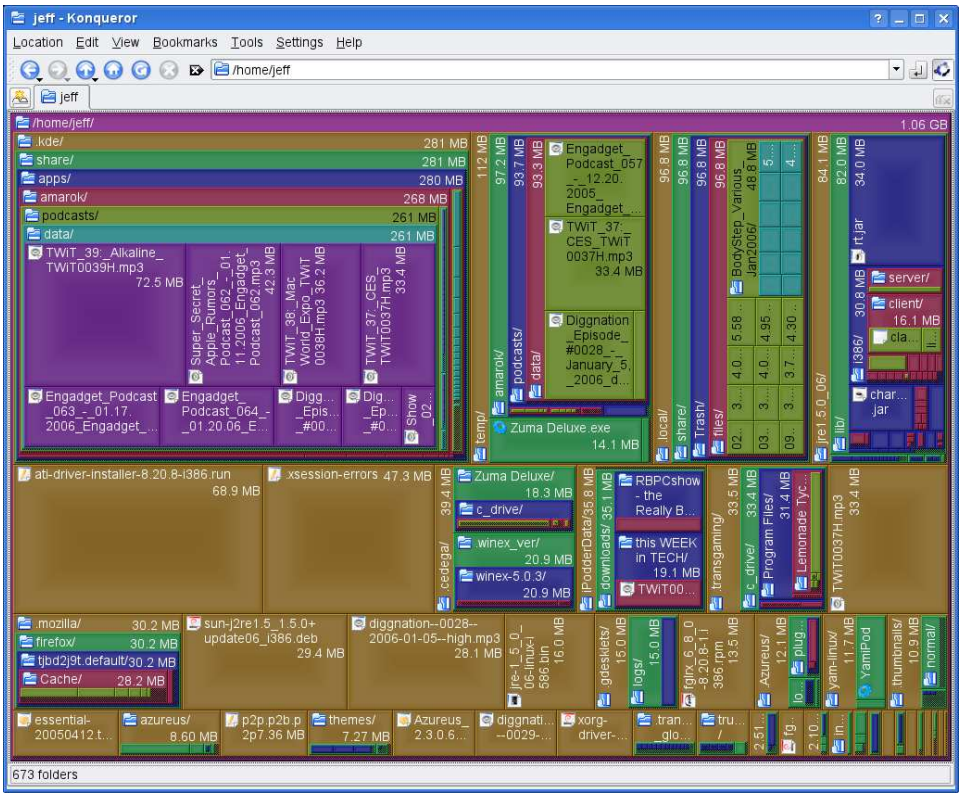

Figure 6: Konqueror's treemap view of a filesystem hierarchy

Figure 7: FSV (FileSystem Visualizer) 3D treemap

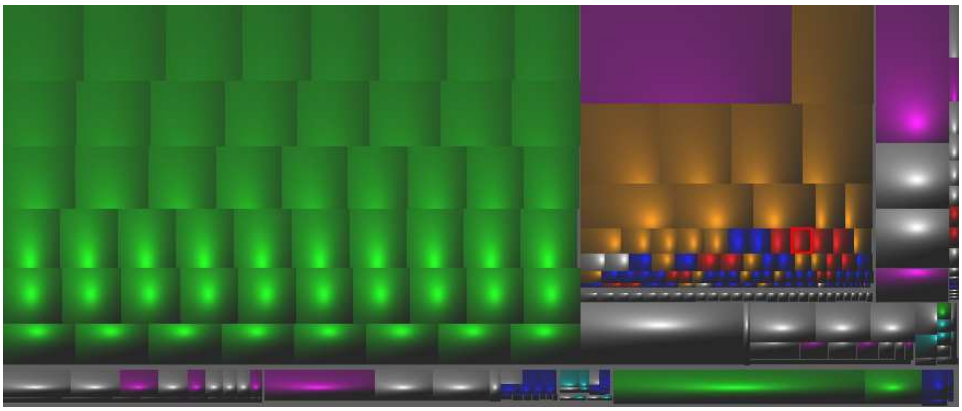

Figure 8: KDirStat's cushion treemap

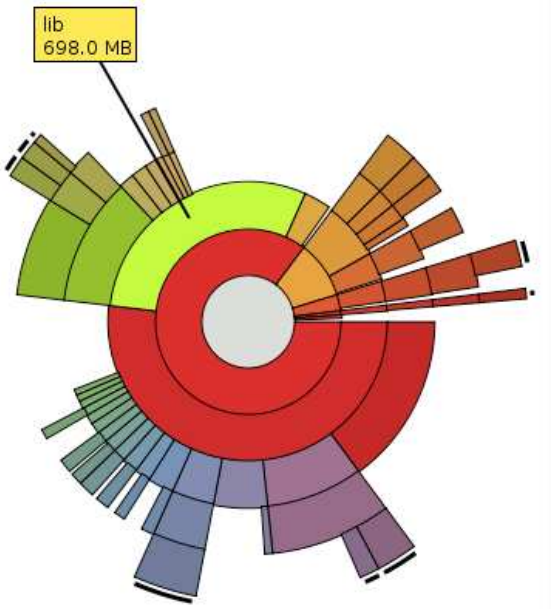

Figure 9: Sunburst tree visualization

3 Fsnav, a 3D Filesystem Navigator

3.1 Conception

The method I've chosen to implement for my filesystem visualization program is a 3D version of the 2D linked-node layered tree technique. The idea was to write something as similar as possible to the old SGI FSN program.

FSN was an experimental 3D filesystem navigator for Silicon Graphics IRIX systems, written in IrisGL during the early 90s (figure 10).

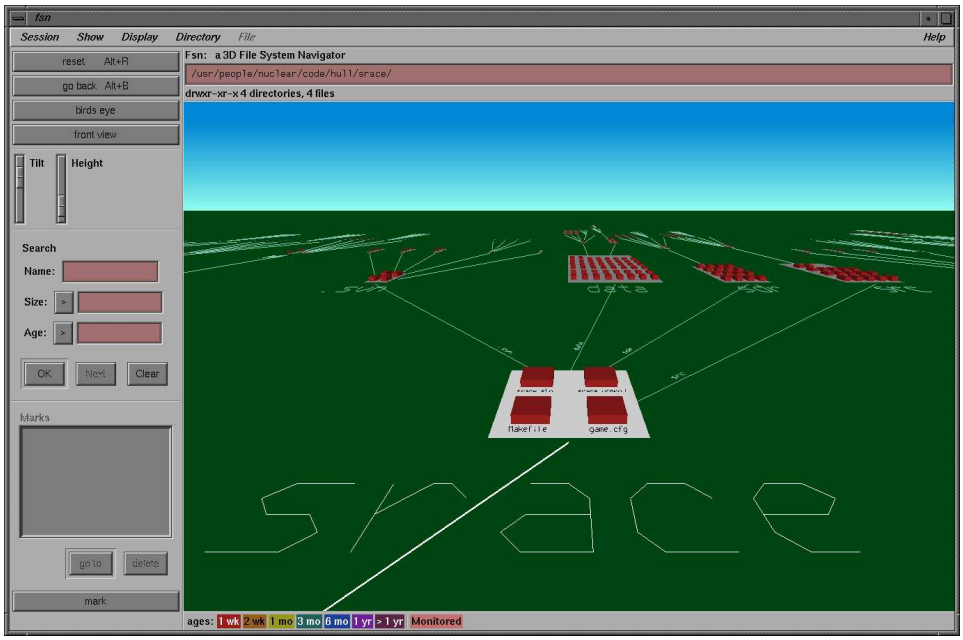

Figure 10: SGI 3D filesystem navigator (FSN).

FSN visualizes directories with 3D boxes arranged on a ground plane, connected together with line segments to show parent-child relationships. On top of each directory box, there are smaller boxes represeting the files contained in that directory.

The user can navigate the filesystem by clicking on the links between directories, at which point the program takes control of the camera an flies towards the target directory. When the user clicks on a file box, a spotlight cone appears on that file, and more information about the file are shown in a simple text widget at the left toolbar (figure 11).

Although its usability and usefullness is questionable, FSN became famous due to its appearance in the first Jurassic Park film, in a scene where a little girl uses a computer to lock a door, before two velociraptors manage to enter the room. In that scene, after announcing that she knows how to

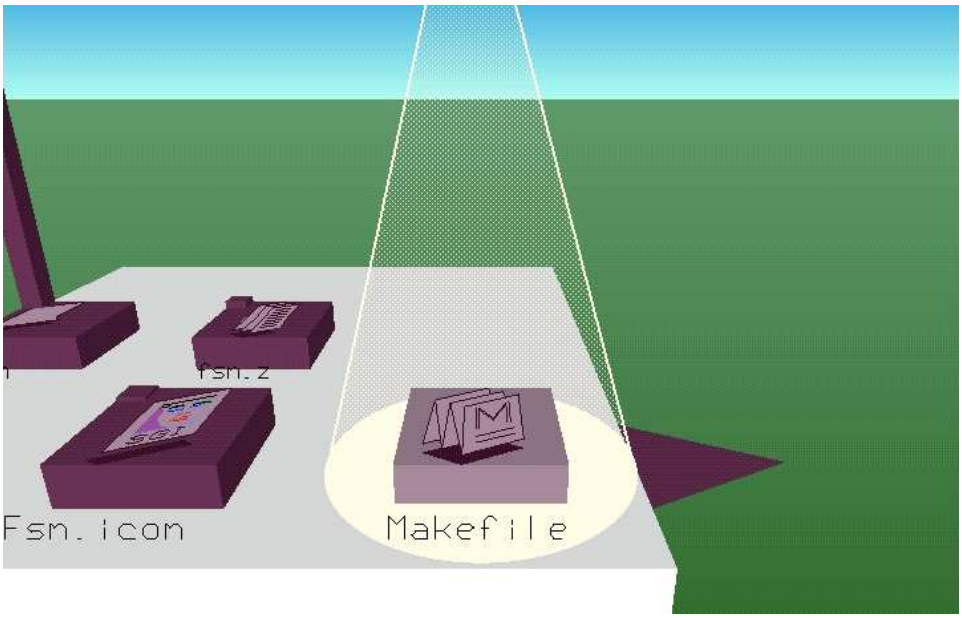

Figure 11: SGI FSN file detail view

use the computer since it's a UNIX system, the little girl starts navigating around a fancy (for the time) 3D environment looking for the appropriate program that will lock the doors.

3.2 Fsnav Design & Implementation

The main goal of fsnav is to accurately capture the look & feel of the original FSN application, without necessarily being an exact replica (figure 12). The most important aspects that had to be duplicated where deemed to be the following:

- Similar 3D visualization of the filesystem hierarchy with boxes and links between them.
- Ability to focus on any file and receive more detailed information beyond its filename which should always be visible.
- Automatic point-and click navigation through the hierarchy, with camera transitions.

To achieve a usable result, the last item is very important. If the camera just snapped to the target directory without a transition, the user would easily become disoriented. That transition is initiated by a double click on any file or directory, but not on the links as with the original FSN application.

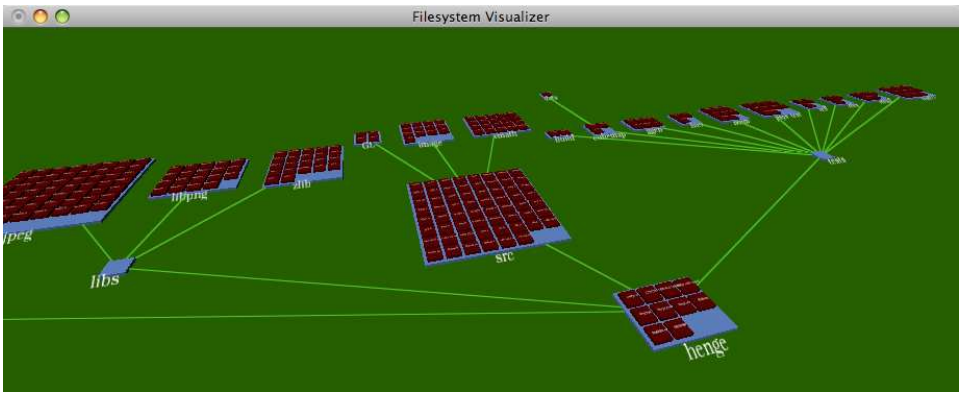

Figure 12: Fsnav 3D filesystem navigator

Selection and double-clicking on files and directories is implemented by creating two points on the near and far planes, corresponding to the mouse location in homogenous clip space, and reverse-transforming those points to world coordinates thus obtaining a mouse ray. That mouse ray is subsequently used to calculate intersections with all boxes in the environment, the nearest of which is used as the selected target node.

Departing from the original program, it was decided to allow the user to freely rotate the view around the target file/directory, by dragging the mouse with the left mouse button pressed. Also the user may zoom in and out by holding the right mouse button while dragging.

When the user selects a file by clicking on it, detailed information about that file is overlaid on the 3D viewport next to the file box as shown in figure 13.

The information provided in the file details are:

- File size.
- Permissions (rwx for user, group, and others).
- User name and user id.
- Group name and group id.
- Access, modification, and creation times.

Furthermore, in order to make it easy to quickly browse the details of many files without having to click on each one of them, if the user holds the spacebar pressed, then file details are shown, in the same manner, for every file under the mouse cursor.

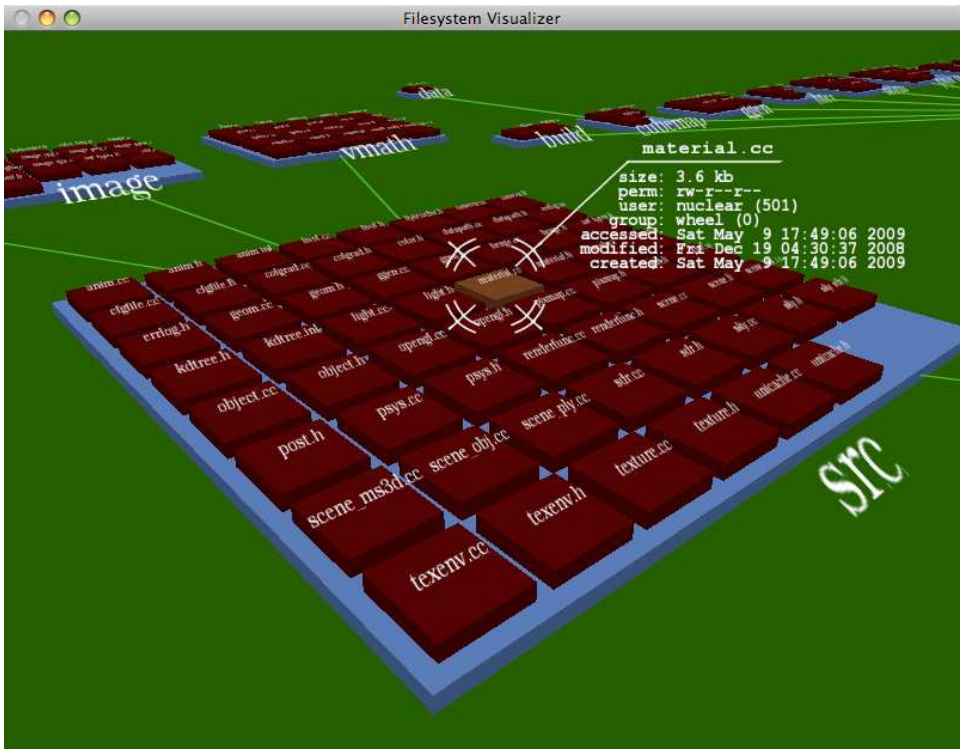

Figure 13: Fsnav detailed file information

3.3 Implementation Details

Fsnav was written in C++, and uses the following libraries:

- OpenGL and GLUT.
- Freetype for truetype/opentype/type1 font rendering.
- vmath, my reusable C/C++ 3D math library.
- libimage, a library I've written for loading and saving multiple image file formats.
- libpng and libjpeg are used indirectly trhough libimage.
- zlib is used by libpng.

Fsnav is a UNIX program, and should be able to run on any UNIX system with an OpenGL implementation. It's been tested and known to work under GNU/Linux, MacOS X, FreeBSD, and IRIX. A windows executable, which is sadly a requirement for the coursework, was compiled using the Cygwin POSIX emulation library. File permissions and other details that

are not applicable under windows, are mapped to sane defaults by cygwin, so everything works fine (tested on a virtual machine running windows xp).

4 Fsnav Manual

Welcome to fsnav, a 3D filesystem navigator inspired by the old experimental FSN program by Silicon Graphics. Unlike FSN, fsnav will compile and run on any UNIX system with an OpenGL implementation. For any questions or bug reports, please contact me at `nuclear@member.fsf.org`.

4.1 License

Fsnav is free software. You may freely modify and redistribute it under the terms of the GNU General Public License version 3, or at your option any later version published by the Free Software Foundation.

4.2 Installation

First, make sure you have all the following dependencies installed at your system: opengl, glut (or freeglut), libpng, libjpeg, zlib, and freetype. Then just run `make` to compile fsnav, and `make install` (as root) to install it.

4.3 Usage

To launch the the filesystem navigator, just type `fsnav` followed by the directory you wish to use as the root for the visualization. Fsnav traverses all subdirectories of the specified directory, collects file statistics and builds the filesystem tree, which might take some time for very large hierarchies (for instance if you try to visualize the whole filesystem). If no argument is specified to fsnav, it will start from the current directory.

4.3.1 Controls

Initially when fsnav starts, the camera is centered arround the root of the tree. You may rotate the view freely horizontally and vertically by holding down the left mouse button and dragging the mouse around. You may also zoom in or out by holding down the right mouse button, and moving the mouse vertically.

In order to navigate to other subdirectories, just rotate the view so that the target directory is visible, and double click on it.

Selecting any file by clicking on it, will show details about that file such as filesize, permissions, etc. Alternatively, you may hold down the spacebar and hover the mouse over any file to see its details.

At any point you may exit by pressing escape.

References

[1] J. Lamping, R. Rao, and P. Pirolli, "A focus+context technique based on hyperbolic geometry for visualizing large hierarchies," in *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, (New York, NY, USA), pp. 401–408, ACM Press/Addison-Wesley Publishing Co., 1995.

[2] B. Shneiderman, "Tree visualization with tree-maps: 2-d space-filling approach," *ACM Trans. Graph.*, vol. 11, no. 1, pp. 92–99, 1992.

[3] J. J. Van Wijk and H. van de Wetering, "Cushion treemaps: Visualization of hierarchical information," in *INFOVIS '99: Proceedings of the 1999 IEEE Symposium on Information Visualization*, (Washington, DC, USA), p. 73, IEEE Computer Society, 1999.

[4] M. Balzer, O. Deussen, and C. Lewerentz, "Voronoi treemaps for the visualization of software metrics," in *SoftVis '05: Proceedings of the 2005 ACM symposium on Software visualization*, (New York, NY, USA), pp. 165–172, ACM, 2005.