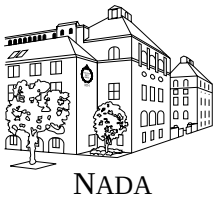
STOCKHOLMS
UNIVERSITET

3D Graphics
in the User Interface of a File System Browser

Niklas Höglund

TRITA-NA-E04038



NADA

Numerisk analys och datalogi
KTH
100 44 Stockholm

Department of Numerical Analysis
and Computer Science
Royal Institute of Technology
SE-100 44 Stockholm, Sweden

3D Graphics in the User Interface of a File System Browser

Niklas Höglund

TRITA-NA-E04038

Master's Thesis in Computer Science (20 credits)
within the First Degree Programme in Mathematics and Computer Science,
Stockholm University 2004
Supervisor at Nada was Kai-Mikael Jää-Aro
Examiner was Lars Kjelldahl

Abstract

This Master's project examines whether 3D graphics can be used to advantage in the user interface of a file system browser. Different ways of visualizing file systems were examined. A prototype of a file system browser using 3D graphics was then implemented, with the goal of making interaction in 3D as fast or faster than it is in a 2D browser. When the prototype was tested on users, it was found that it was usually slower than the browser in Windows XP, but that the prototype could be improved by fixing many known problems.

3D-grafik i användargränssnitt till filsystemsbläddrare

Sammanfattning

Målet med det här examensarbetet var att undersöka om användandet av 3D-grafik kan vara till fördel i användargränssnittet till filsystemsbläddrare. Olika sätt att visualisera filsystem har undersökts, och en prototyp av en tredimensionell filsystemsbläddrare har sedan implementerats, med målet att interaktionen i 3D ska vara minst lika snabb som i en tvådimensionell bläddrare. Prototypen har sedan testats på användare, varvid det visat sig att interaktionen med den oftast är långsammare än med den vanliga tvådimensionella bläddraren i Windows XP, men även att det finns potential för förbättringar om kända problem med prototypen löses.

Contents

1 Introduction **1**
- 1.1 Background and purpose . . . 1
- 1.2 Method . . . 2
- 1.3 Report organization . . . 3

2 First user study **4**
- 2.1 How do people use file managers? . . . 4
- 2.2 Graphical file managers compared to the command shell . . . 8
- 2.3 Conclusions . . . 9

3 Hierarchy visualization techniques **10**
- 3.1 Space-filling tree-maps . . . 10
- 3.2 Squarified space-filling tree-maps . . . 11
- 3.3 Cone trees . . . 12
- 3.4 The Information Cube . . . 13

4 Study of existing programs for visualizing file systems **15**
- 4.1 fsv . . . 15
- 4.2 StepTree . . . 19
- 4.3 3DOSX . . . 20
- 4.4 Xcruiser . . . 21
- 4.5 tdfsb . . . 25

5 Discussion of the techniques **28**
- 5.1 Why is navigation important? . . . 28
- 5.2 Tree-maps . . . 28
- 5.3 Cone trees . . . 29
- 5.4 Xcruiser, tdfsb . . . 30
- 5.5 Rich icon representations . . . 31
- 5.6 Local or global visualization . . . 31

6 Prototype **33**

6.1 Wanted features 34
6.2 Object placement 35
6.3 Filtering 37
6.4 Implementation 38
6.5 Collision avoidance 39
6.6 Navigation 42
6.7 Correlation problem 44
6.8 Grid layout 46

7 Second user study **48**
7.1 User comments 49
7.2 Conclusions 52

8 Discussion of results **54**
8.1 The implemented visualization 54
8.2 How important is fast visual navigation? 54
8.3 Would a 3D interface be better than a 2D counterpart? 55
8.4 Human-like interaction 56
8.5 Different views for different tasks 57
8.6 Future work 57

References **58**

Chapter 1

Introduction

1.1 Background and purpose

When I started to work on this Master's project, the question I posed was: Can 3D graphics be used to improve the user interfaces of regular 2D applications? This is clearly too wide and vague a question for a Master's project. After discussions with my supervisor Kai-Mikael Jää-Aro, Lars Kjelldahl and Sören Lenman, I decided to narrow the question down to look only at a specific application: the file manager.

The main purpose of this project is to examine if a 3D user interface for managing files can provide advantages over a conventional 2D interface such as the Explorer in Microsoft Windows or Nautilus in GNOME.

While thinking about how this can be accomplished I asked myself some other things that may be important in answering the main question.

I realized there must be some kind of visual objects that represent files and directories, but how should these be placed in the 3D space? Are there existing layout algorithms for 2D, 2.5D or 3D that are appropriate? The traditional layout used in Windows and usually in Linux is to place all icons in a regular 2D grid. This would not work well in 3D as it would be very hard to see the objects that end up in the middle of a 3D grid. The Amiga and Mac OS allow users to place objects manually, and can also order objects automatically if requested. Manual placement enables users to place objects as they want them, but may not be good on directories that come from other systems, as objects in them have not been manually placed.

Maybe some ways of visualizing the file system are good when performing some tasks, and other ways are good for other tasks? Adding more options where you can today select between "icon view" and "list view" today would probably be a good thing to do.

Another important question is how navigation should function to make it as easy and as fast to work in a 3D environment as well as in a 2D environment? If navigation is more tedious or difficult than in current systems, then many people would probably not want to use it unless it also contains substantial improvements in other areas.

I also thought that maybe good graphics together with zoom could be the basis of a system that would be more intuitive for people that are not used to computers. I have seen non-technical people having problems understanding the hierarchy, and having problems moving things from one folder to another. If things like this can be made easier, more like the real world or like in a computer game, then that may help people who do not use computers much. Zoom may be an intuitive way to move around in a hierarchy, and it might also be an intuitive way of viewing objects. If the icon for a picture file is a miniature of the picture, then zooming in on that icon could show the entire picture. This is done in a nice way in the PhotoMesa picture viewer [1] developed as part of research on zooming user interfaces in the Human-Computer Interaction Lab at the University of Maryland. In the same way, zooming in on a text file could show the text in that file, and then typing on the keyboard could edit the text. This could also make it easy to find places in text files, as you could zoom out to get an overview and then zoom in on the correct place. This would provide similar advantages as automatically zooming out documents when scrolling quickly through them [12] to avoid the effect that it is impossible to read or recognize anything when text scrolls by too quickly.

I will not be able to implement a word processor in 3D in this project, nor most other things in the last paragraph, but I think the ideas have some merit. Also, these things can be implemented in 2D graphics, so they are not really pertinent to the main question of this project.

To summarize, the goal of this project is to answer the question of whether 3D graphics can make a file system browser more useful than 2D graphics can. To answer this question it is of course necessary to understand how a 3D browser should work to be as good as possible, which is what most of this report discusses.

1.2 Method

I will create a prototype of a 3D file system browser and then compare this to a traditional 2D file system browser in a user study. If this comparison shows my prototype to have advantages, that will at least partially answer the main question of this project. If it does not show advantages, that will not be proof that 3D cannot be advantageous, as the prototype may simply not be good enough, but evaluating the prototype and its problems should

probably give a good hint to what can be done. Insight into the area will also be gained by studying other similar programs, studying literature, designing and implementing the prototype and writing this report.

1.3 Report organization

I initiate this report with a user study described in chapter 2 to examine how people use file managers. This is mostly to get some insight into what tasks are performed and what might be improved upon.

To see how hierarchies can be visualized I look at existing techniques in chapter 3 and examine various programs using these techniques and other ones in chapter 4. I discuss the techniques in chapter 5.

After that, I describe the prototype 3D browser I have implemented, and the difficulties I had to overcome during this implementation, in chapter 6.

Chapter 7 continues with another user study to evaluate the prototype, and lastly, chapter 8 has a summary of the results, and relates this to the goals of this project and ties this into what can be done in future research.

Chapter 2

First user study

The purpose of this user study that I performed in the beginning of this project was to see how people use file managers. The goal was to gain some understanding of what problems existing file managers have, and what can be made better.

2.1 How do people use file managers?

I tried to find as many actions as possible that people do with files and directories. To do this, I interviewed family and friends, and some people I study computer science with. All in all eight persons, three family, of which one uses computers regularly and the other two occasionally, one friend, who uses computers fairly regularly, and four computer science students. All but two of these are male, and ages vary between 16 and 55 (but most are around 25–30).

I also walked around in the computer labs at the Department of Numerical Analysis and Computer Science, of the Royal Institute of Technology to see what people there did, and I added some things I remembered people I know (or myself) often doing.

I have not tried to use a representative choice of people in any way. The goal was to find as many actions as possible, not to order them after frequency of use, or any other way. The list below is not ordered in any specific way. The bullets under a heading are alternative ways of performing the same action. Different people do it different ways. Sometimes several of the actions I have listed are done together, such as "create directories", "move files into directories" and "rename files".

The list below includes file management actions regardless of what program is used to perform them. Usually, either a graphical file manager such as Windows Explorer or GNOME Nautilus is used, or a command shell (bash or tcsh). Sometimes a specific GUI is used for a certain task.

I have not tried to be exact in the steps of performing an action. When I say "click" below, it may be that a double click is required. Also, different programs sometimes have slight variations that I have not listed.

Find and open documents

- Click through folders and then on the document icon.
- `cd skolan/exjobb; emacs tidsplan.txt`.

Start programs

- Click on a link on the desktop background.
- Click on a link in a start menu.
- Write the name of the program in on a command line.
- Look for an executable file on the hard drive in typical folders such as "`C:\Program Files`" and then click on it.

Organize: create directories

- Right click and select "New folder"; write the name.
- `mkdir "directory name"`.

Organize: move files into directories

- Drag files and drop them on directories.
- Select files; "cut" them; go to the destination and "paste" them.
- `mv a b/`.

Organize: rename files

- Right click and select "rename".
- `mv name new_name`
- Select the file, click on the file name and type the new one.

Copy files between computers

- `rsync -rv exjobb shell.nada.kth.se:`
- Drag and drop between "My Computer" and "Network Neighborhood".
- Use a program like for example WinSCP or an FTP program.
- Sending mail to oneself, and receiving it on another computer, usually through web mail.
- Using a network file system.

Unpack downloaded archive files

- Click on the archive; choose extract; choose destination.
- Click on the archive; drag and drop the contents to the destination
- `tar xjf xxx.tar.bz2`

Add music to play list of music player

- Drag and drop from file manager to play list window.
- Press a button in the music player and select files in the file dialog that appears.
- Use a function in the music player to scan the hard drive for music files.

Copy pictures from digital camera to picture directory

- Using a wizard that appears automatically when the camera is connected.
- Open "My Computer" and the unit of the digital camera; open the destination; create a folder for the pictures; drag them to the destination.
- Using a specific application, often supplied by the camera vendor.

Make backup copies of directories[1]

- Right click a directory and select duplicate; rename the result to xxx-20030123.
- `cp -a xxx xxx-20030123`
- `cd xxx; cvs commit -m "..."`

Organize digital camera pictures

- Like organizing files, but it is important to be able to see miniatures of the pictures, as pictures coming from digital cameras have meaningless filenames.

Copy music from CD

- Start CD ripping program; select all tracks; press the start button.
- `cd music; mkdir Reverence; cd Reverence; cdparanoia -B; oggenc -q -1 *; rm *.wav`

Free hard-drive space

- Look for large unnecessary files or directories and delete them. Right click on icons and go to properties to see how much space a directory uses (including its contents). Do this for many directories and the delete some of them.
- `du -s *`; delete things that seem unnecessary and use much space.

Clean up

- Look around in different directories and delete uninteresting or no longer important files or directories.

[1] Proper backups should of course not be made like this, but I have not seen anyone make a proper backup. Making a safety copy of a directory containing source code before doing extensive changes is common, though.

2.2 Graphical file managers compared to the command shell

Everyone I have interviewed seems to think it is easier to learn to use graphical file managers. It can be frustrating to use written commands before learning enough of them, but in a graphical environment it is usually possible to get a list of available commands.

In Windows, almost everyone I have spoken with uses the built-in file manager, but on Unix systems most use a command shell.[2] Many think that the shell is more powerful than graphical file managers; it is not just a file manipulation tool, but can do many other things too.

Opinions vary about which is fastest to work with once you have learned them both. Some say the shell is fastest and some say file managers are. It would be interesting to measure which is fastest, but that is outside the scope of this project. The answer might be that whatever a user is most used to is fastest. Bruce Tognazzini from Apple, in his "Tog on Interface" [17], discusses whether it is best to use the mouse or keyboard shortcuts (like ctrl-s to save). According to him, they have found out, among other things, that "mousing" is faster than "keyboarding". According to their tests, subjects consistently report that using the keyboard is faster than using the mouse, while the stopwatch consistently says it is the other way around. Apparently, deciding what key combination to use is done by the higher levels of the brain, while reaching for and moving the mouse is done by lower levels. Because of this, the higher levels of the brain has to wait while the body moves the mouse. This makes the mouse seem slower. This argument might not hold for often-used keyboard shortcuts, as there is hardly any conscious though involved when using them. Tognazzini is not very specific on what was tested, and he does not provide a reference to any published material on how the test was performed. His book has been cited much on the net, for example on Slashdot[3], where some people agree and many others disagree with his results.

My conclusion is that if the task is complex enough, like programming or shell scripting, a text interface is almost necessary, while simple tasks can probably be done faster in a GUI interface. But how complex must tasks be before text-based interfaces are faster?

Two persons I have spoken with have said that though they use shell commands most of the time, they sometimes use graphical file managers when

[2]Most people I have interviewed who use Unix have learned to use it here in the Department of Numerical Analysis and Computer Science, so it is possible that the education here influences this result.

[3]http://slashdot.org/askslashdot/01/11/01/1858233.shtml

they do not know what program can open a file. Then they double click on the file in the file manager and the right program is started.

2.3 Conclusions

The individual actions described in section 2.1 may be interesting for developers of the systems used there, but I do not see them as relevant to the overall design of something so different as a 3D file manager. When it comes to design of specific features to support the actions mentioned here, it is probably a good idea to look at how they are implemented in existing programs, but first an overall design must be decided upon, and if that differs much this user study is probably not relevant for further design.

I cannot see any obvious major problems with current systems from this user study. Many systems could probably benefit from a simplification and homogenization of the user interface or commands available, but many users would not like the systems to be changed at all unless the changes were significant enough to merit relearning the interface.

Some people complain about not knowing how to find a command to perform a specific action in command shells. A good, and obviously available, help system could solve this problem, though reading help texts is still probably slower than looking through menus. Documentation and manual pages are often written in a technical language that can be hard to understand for new users.

Many of the actions in section 2.1 involve finding specific files or directories, sometimes many in succession such as when navigating deep down into the hierarchy to find a specific file. If this could be made faster that would be a good advantage. It would be interesting to compare existing systems regarding user performance when finding files and directories. Is a system where you type the name of what you want to find best, or one where icons are alphabetically sorted, or one where icons are placed by the user when the file or directory is created and the position remembered?

Chapter 3

Hierarchy visualization techniques

Before designing a file system browser, it is of course a good idea to examine how others have made hierarchy visualizations. This chapter describes some relevant methods.

I have limited myself to visualizations that are used when visualizing hierarchies, though there are of course many other kinds of visualization that could be good when showing various kinds of information, and probably many file systems too. To create a general tool that can show any file system, even those where the hierarchical nature is important, it is necessary that the visualization shows the hierarchy in some way.

I am not sure that having a hierarchical file system is the best way. Most people seem to think about files as objects of some kind and of directories as groups to put related files in. That these should be mapped into a hierarchy is not obvious, and there are other solutions. Instead of browsing to a location where a document was put, doing a search for "large document written me long ago" may be a better way. GNOME Storage (http://www.gnome.org/~seth/storage/) and Microsoft Longhorn WinFS both seem to work in this direction.

3.1 Space-filling tree-maps

The idea of space-filling tree-maps [16] is to visualize a tree by dividing a rectangle into smaller rectangular objects, one rectangle for each node in the hierarchy. These rectangles have size proportional to some node property (usually file size, if the tree is a file system).

The algorithm for placing the rectangles is pretty simple:

1. Decide on a position and size for the root rectangle that contains all other rectangles. This can be set to fill the entire screen area, for example.

2. Divide the root rectangle into as many sub-rectangles as there are children of the root node. Each of these rectangles is as high as the root rectangle, so they go from its bottom to its top. Each sub-rectangle has width proportional to its size divided by the total size of all children. If the rectangle represents a file, its size is the size of that file in. For a directory, the sum of all sizes of its contents is used.

3. Repeat step 2 and 3 for every sub-node of the root node, with the determined rectangle for that node as root node. Alternate the direction of the subdivision this time, so that sub-nodes are placed vertically if they were placed horizontally this time, and horizontally otherwise. Alternating the direction of subdivision makes it possible to see to which level of the hierarchy rectangles belong.

In a small hierarchy with the nodes /a, /b, /c/d, /c/e, /f/g, /f/h/i and /f/h/j, the tree-map would look as in figure 3.1.

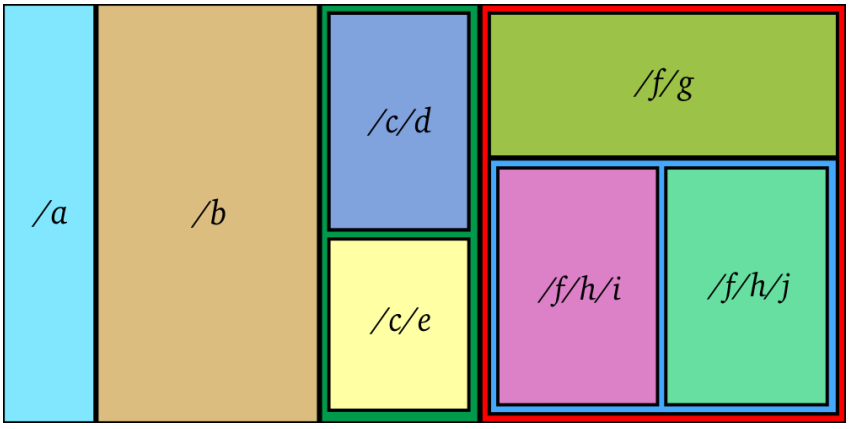


Figure 3.1. Simple tree-map example.

3.2 Squarified space-filling tree-maps

An extension of the tree-map concept was proposed [4] in which the layout is changed so that all rectangles are as square as possible.

The result looks like in figure 4.6, except that in the StepTree figure, borders have been added between rectangles. The borders make the figure easier

to read, but rectangle size is no longer exactly proportional to file/directory size, only approximately. (If the scale is one byte per pixel, a directory with ten thousand one-byte files would have to have 30 401 pixels of border to have borders around all files. A directory with just one file of ten kilobytes would then have to have 10 000 bytes of file and 30 401 bytes of border, or more than three times more border than content. This example is of course exaggerated, but some real directories have lots of tiny files and others have just a few very large files.)

Square rectangles have several advantages. In the original tree-maps rectangles often become very thin, and almost impossible to see. Also, square rectangles use less space for borders, and their sizes can be changed in smaller (and thus more accurate) steps. (Imagine having a rectangle of $1 \times 1\ 024$ pixels covering the entire height of the screen. It can only be increased to $2 \times 1\ 024$ pixels, while a rectangle of $32 \times 32 = 1\ 024$ pixels can be increased many steps before reaching 32×64 pixels.)

There are disadvantages too, though. The alternating direction of layout (horizontal, vertical, horizontal, and so on) in the original tree-maps made the structure of the tree easier to see. In both tree-map variants rendering techniques such as borders around all non-leaf nodes and cushion tree-maps [18] can be used to show structure. In cushion tree-maps each rectangle is rendered using different shades of grey to look three-dimensional (somewhat like a cushion lying on the screen). This makes it easier to see the extent of the rectangles, and appear to work well without the alternating directions.

Another drawback of squarified tree-maps, due to the positioning method used, is that any ordering between nodes at the same level of the tree is not retained in the visualization of squarified tree-maps, while it was in the original tree-maps.

Anyway, squarified tree-maps seem to be the most popular variant.

3.3 Cone trees

Cone trees [15] are basically an extension of the normal two-dimensional trees we often draw (see figure 3.2 for an example). The difference is that instead of placing all child nodes along a horizontal line, they are placed on a horizontal circle below the root node. This can look as in figure 3.3.

When there are many children to a node, they often occlude each other. A common navigation technique is then to rotate this subtree, so that the interesting child nodes can be seen in front of the subtree.

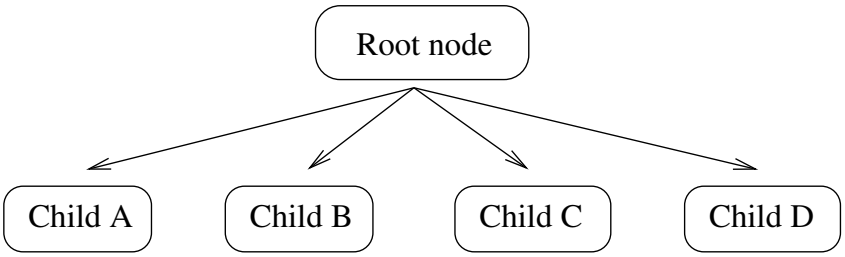


Figure 3.2. 2D tree.

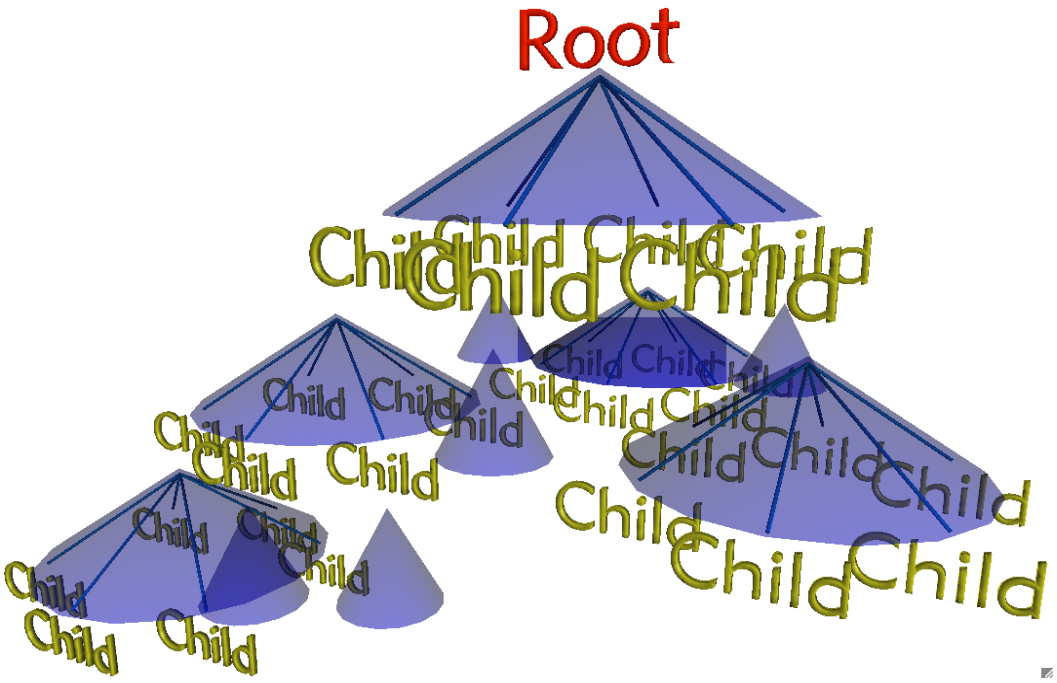


Figure 3.3. Cone tree example.

3.4 The Information Cube

In "The Information Cube" [14], every node in a hierarchy is visualized as a semi-transparent cube. The contents of a node are shown as cubes within its cube. What you see if you look at this visualization is lots of cubes within cubes within cubes, and so on.

A screen shot of a program using this technique is in figure 3.4.

Finding a layout that uses the space in a cube optimally is an NP complete problem, but that is not really a problem as using the space optimally would not be very good anyway. It would cause the cubes to occlude each other too much. It is better to use a rather sparse layout than a tightly packed layout.

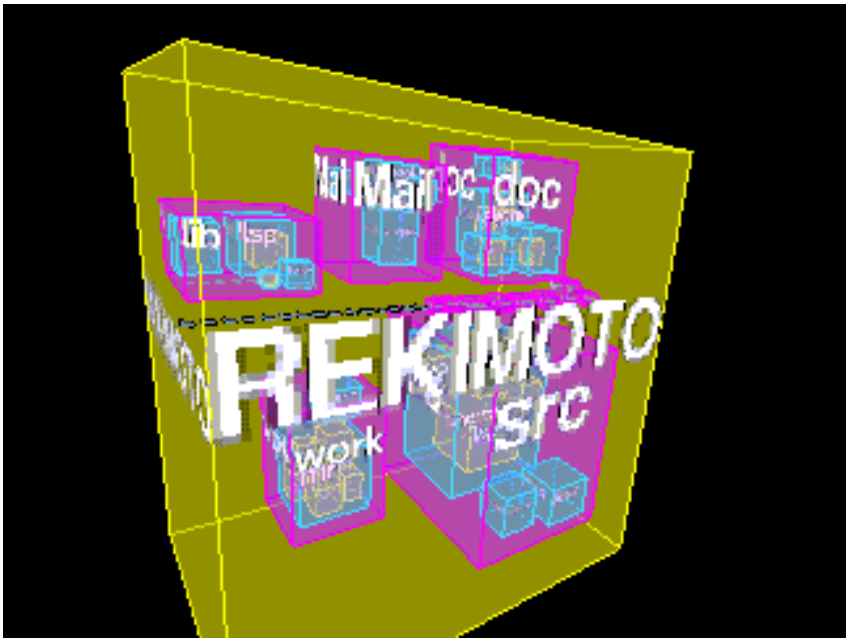


Figure 3.4. The Information Cube.

According to its creators, the information cube has the advantages that understanding it is easy and that it produces a good detail level, neither too low nor too high, as deeper levels of the hierarchy become opaque enough to disappear.

Occlusion, they say, is a problem, making it hard to see objects that end up behind other objects. In their implementation (described in [14]), this is countered by having the image follow head movements, by using stereoscopic view and by using interactive rotation.

It is of course also possible to zoom in and focus on an inner cube. In the described virtual reality interface this is done by pointing on a cube and making a gesture.

Chapter 4

Study of existing programs for visualizing file systems

In this chapter I describe some programs that visualize file systems. All of these programs, except 3DOSX, only let you look at the file system, so they are not really file managers. They can of course be used practically to find out things about a file system, but I have the impression the programs are meant more as demos of how the visualizations work than as useful day-to-day tools.

3DOSX also has standard file manipulation operations such as copying and deleting files. According to its web pages (http://www.acm.uiuc.edu/macwarriors/eoh2k2/3dosx/), it aspires to be a replacement for the Finder in Macintosh OS X. A common opinion in discussion fora is that 3DOSX is cool, but that the finder is more practical.

To get a feel for how popular these programs are, I checked the SourceForge download statistics for the programs that are available there. Xcruiser has been downloaded 1 019 times since February 2003 and fsv 7 436 since august 2001. As a comparison, GAIM, a common IM application, is downloaded about 6 000 times every day. These statistics are not exact, as some of the programs are downloaded from other sources too. I know Xcruiser and GAIM, but not fsv, are available in the Debian GNU/Linux distribution.

4.1 fsv

fsv (http://fsv.sourceforge.net) has two kinds of visualization, called MapV and TreeV.

MapV

MapV is quite similar to squarified tree-maps, but it uses another algorithm to place objects that does not always produce square rectangles.

If viewed from the positive y direction MapV looks just like the two-dimensional tree-maps described in section 3.2. Every file and directory is drawn as an axis-aligned block (which of course looks like a rectangle when viewed from the positive y direction), where the size of the block is proportional to the file size if the block represents a file, or the sum of the sizes of all contained files for a directory. Along the y direction, the recursion level is displayed. Every file and directory is drawn as a block on top of the directory it is located in. As all directory blocks have equal height, it is easy to see how deep down in the hierarchy any visible object is by looking at how high up it is placed. Files and directories are placed on top of their parent directory ordered by their size, so that the smallest objects are closest to the camera, if the camera points in its default direction.

Directories can be opened or closed. If a directory is closed, only its block is drawn with its name on it. If it is opened all its contents are drawn on top of it.

Different types of files (image files, text files, ...) are drawn in different colors.

Figure 4.1 shows how the Linux kernel source code tree can look visualized with all directories closed. In figure 4.2 one directory is opened and in figure 4.3 all are opened.

Note that it can be hard to find small files, as they occupy little screen area, and as files are ordered by their sizes, not their names. I believe this visualization can be good when looking at the sizes of objects. It would be a good visualization to use when looking at what can be deleted to free drive space, for example, but not when searching for a small text file.

I find it is quite easy to get where I want in fsv. To view a file system entity[1] close up, all you have to do is to click on it. If you want to view a directory with expanded items in it you have to click on the border around the directory, as clicking on a sub-entity would look at that instead.

It is also possible to zoom in and out by holding the middle button and moving the mouse up or down. You may have to move the mouse far to get to where you want, though. This could probably be made smoother by using mouse acceleration or higher mouse pointer speed.

[1] In this text I use the word entity to refer to file system entities (such as files, directories and symbolic links).

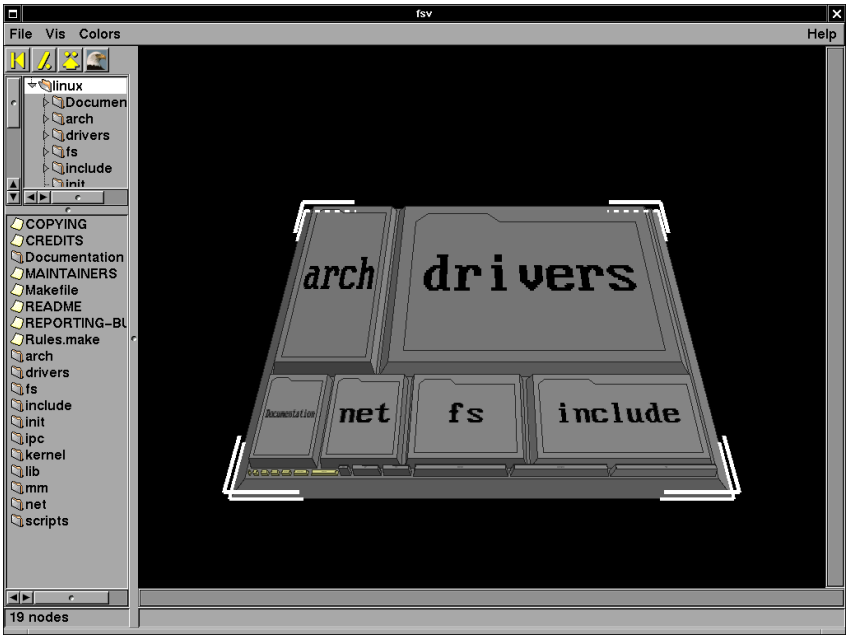


Figure 4.1. MapV in fsv with all directories closed.

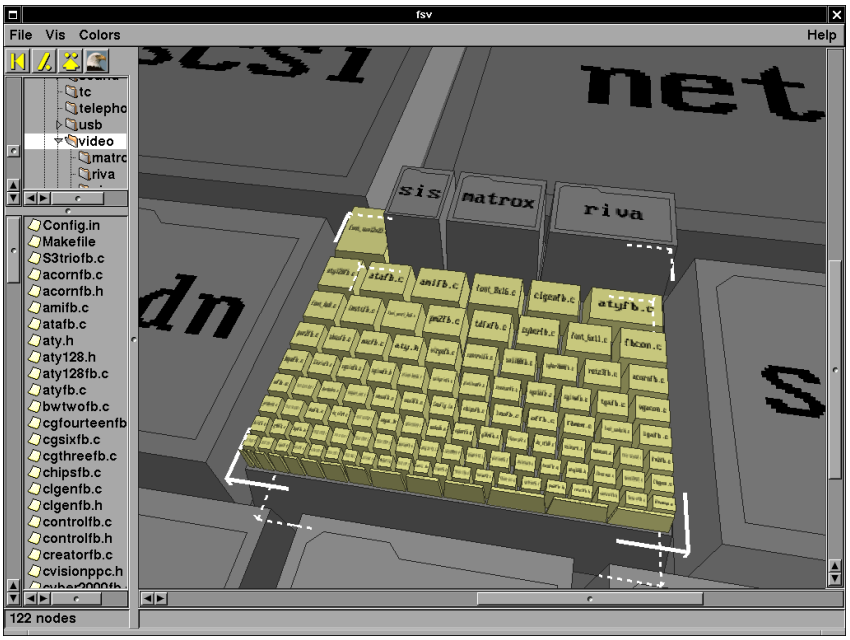


Figure 4.2. MapV in fsv with one directory opened.

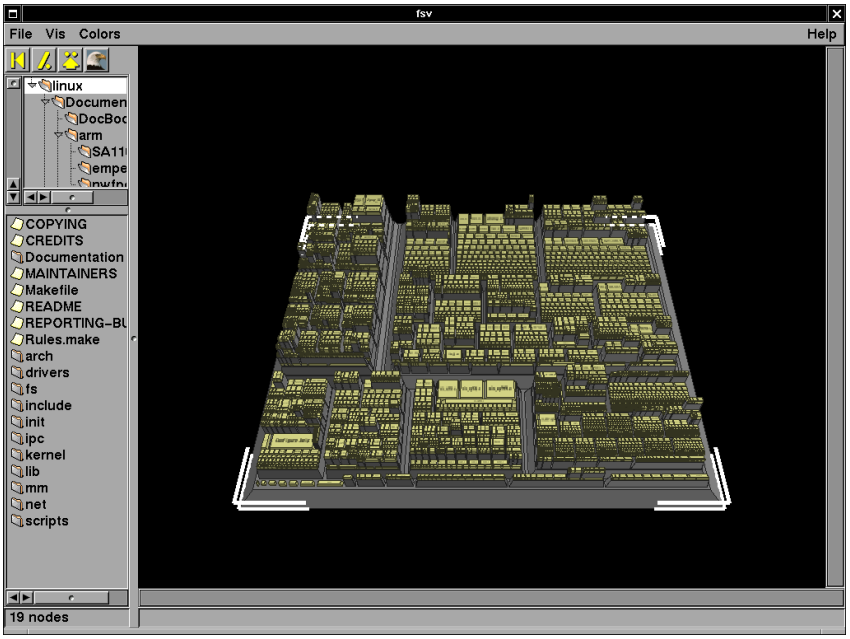


Figure 4.3. MapV in fsv with all directories opened.

It is not possible to scroll sideways. If you look at a couple of files in a folder, and want to see other files you may have to zoom out first, and then zoom in on another place.

TreeV

In a way, TreeV is the opposite of MapV. All files and directories have the same bottom area, but the height is proportional to their size. (In MapV the height was constant, but the bottom area proportional to the size.) To avoid occlusion, the blocks are still ordered from smallest to largest so it can still be hard to find a file you know the name but not the size of.

In TreeV, the contents of a directory are not placed on top of the block of the directory when it is opened. Instead, the directory is replaced by a symbol signifying that it has been opened, and it is shown with its contents next to its parent directory. Look in figure 4.4 for an example, where "ftape" contains three directories: "compressor", "zftape" and "lowlevel". On top of ftape these subdirectories have been replaced by crosses, and they have been "opened up" next to ftape, and connected by red lines.

Seen from above, the root directory is placed in the center, its subdirectories in a circle section around it, their subdirectories in circle sections outside their respective parents, and so on (see figure 4.5). In TreeV, objects do not have fixed places like in MapV. Instead, when new directories are opened other

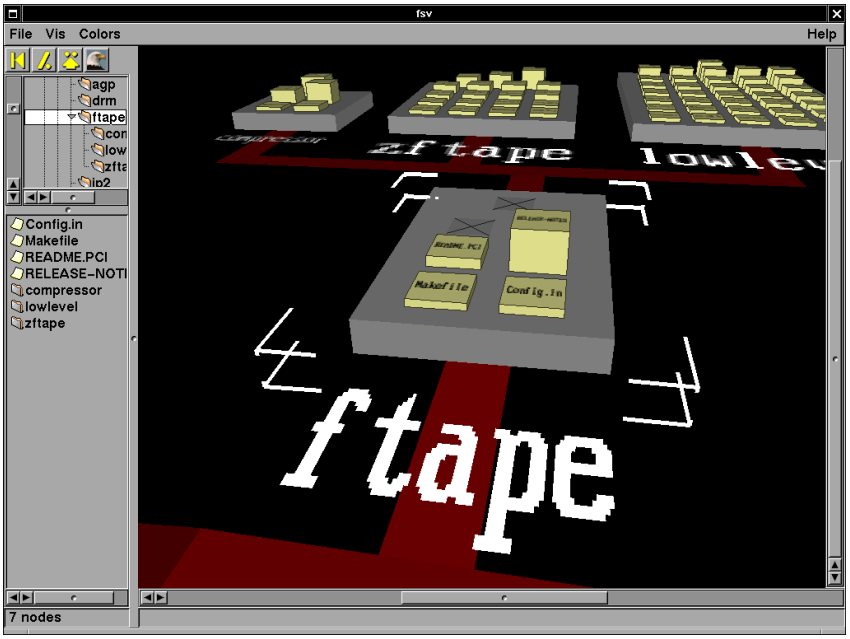


Figure 4.4. TreeV in fsv.

directories glide apart to make space. If the diameter of all the circle sections together on a level of the hierarchy is too large to fit around the entire circle, the radius is increased to make it fit.

A problem with TreeV is that if more than 10–20 subdirectories are opened, the whole layout gets so wide it can be hard to find a specific directory.

4.2 StepTree

Another program visualizing file systems is StepTree [2]. It uses a variant of squarified space-filling tree maps (see section 3.2), and size is the primary property shown here just as in fsv. The visualization is overall very similar to that used in MapV in fsv, with just a couple of differences. In order for relatively small entities not to disappear entirely it uses an equalization step to take some size from larger files/directories to the smaller ones.

StepTree also draws recently modified entities, and directories containing recently modified entities, as solid objects, while others are drawn as wire frames. This works very well to show what has recently been changed (making it possible to find things you have worked on recently quicker, for example, or to find what files have been modified in a shared project). Look in the screen shot in figure 4.6 for an example where some folders have files modified in the

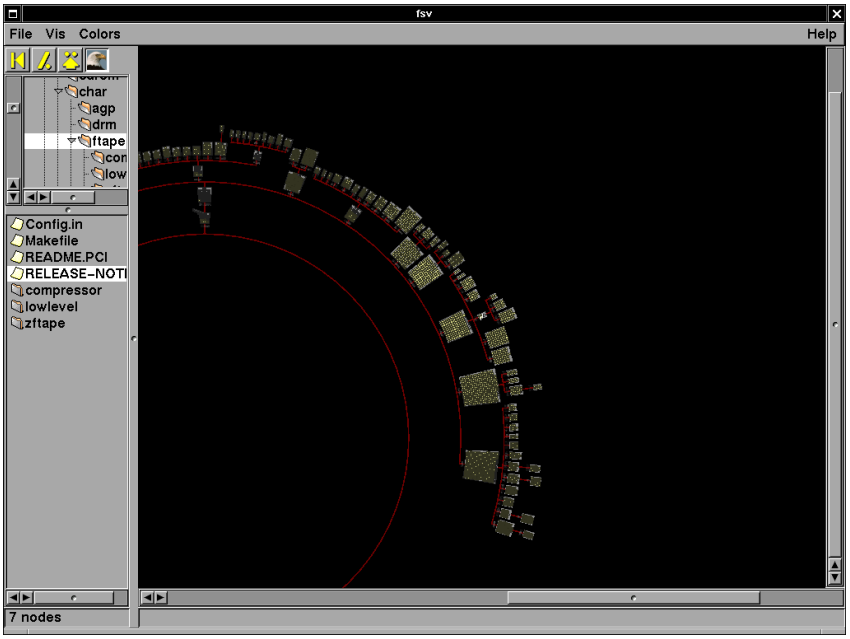


Figure 4.5. TreeV in fsv from above.

last 14 days and other ones do not. The limit of 14 days is of course arbitrary, but unfortunately there can only be one limit active at the same time.

Just as in fsv, different colors are used for different types of files.

StepTree does not show file names at all, and only few directory names, so it is not a good tool for finding files by name. It may be possible to improve this somewhat, but as displayed objects have widely varying sizes I do not believe that it is possible to make it easy to search for files by browsing names and still maintain approximate proportionality between directory size on screen and on disk. The name of a file could be written on the box representing the file, but for small files that box is very small so you would have to zoom in much to be able to look through all file names.

If the name is known, it would be possible to allow typing in a string in a text box and draw all files with names containing that string in some special way making them easy to spot. Not drawing files that do not match at all may be another good option.

4.3 3DOSX

In 3DOSX (http://www.acm.uiuc.edu/macwarriors/eoh2k2/3dosx/), every directory is represented as a disc, and the entities in that directory are visualized as icons above the circumference of the disc (see figure 4.7). Icons representing

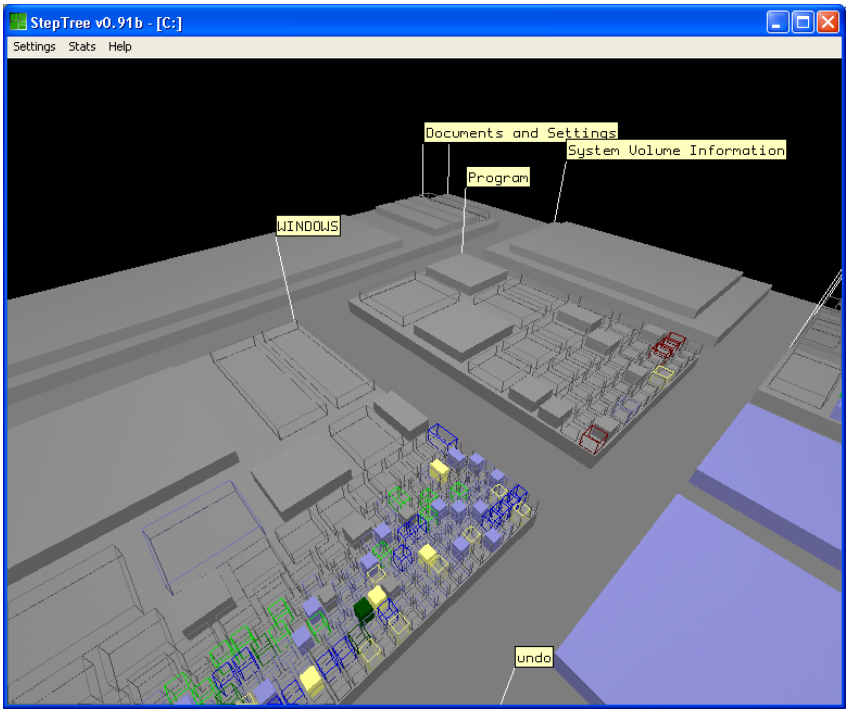


Figure 4.6. The StepTree visualization.

directories are connected to discs containing their contents by a cylinder. This is basically a variant of cone trees (see section 3.3), but upside down. Only the base of the cone is drawn, with a transparent cylinder where the actual cone would be in the original cone trees.

Navigation is basically done by rotating the current disc and selecting the icon of a directory to be moved to its disc.

4.4 Xcruiser

Xcruiser (http://xcruiser.sourceforge.net/) visualizes the file system hierarchy as a universe with galaxies in it. Every directory is shown as a galaxy, most often containing other galaxies (other directories), and stars (files). See figure 4.8 for an example. The stars have mass ($= r^3$) proportional to the file size, and the file name determines position and color (figure 4.9). Entities with short names are shown close to the center of the galaxy, and entities with long names close to its edge. Those with similar names are shown close to each other, so related files often show up as star clusters. Exactly how the similarity of names is decided is not specified in the documentation, but files with common prefixes in their names appear in groups. Soft links are shown

Figure 4.7. 3DOSX.

as worm holes, which are drawn as green curves through space. This can be very interesting. Notice for example how many symbolic links there are in `/dev` on my computer (figure 4.10). There are also quite many in `/usr/bin` (figure 4.11).

Figure 4.8. Files are shown as stars in Xcruiser.

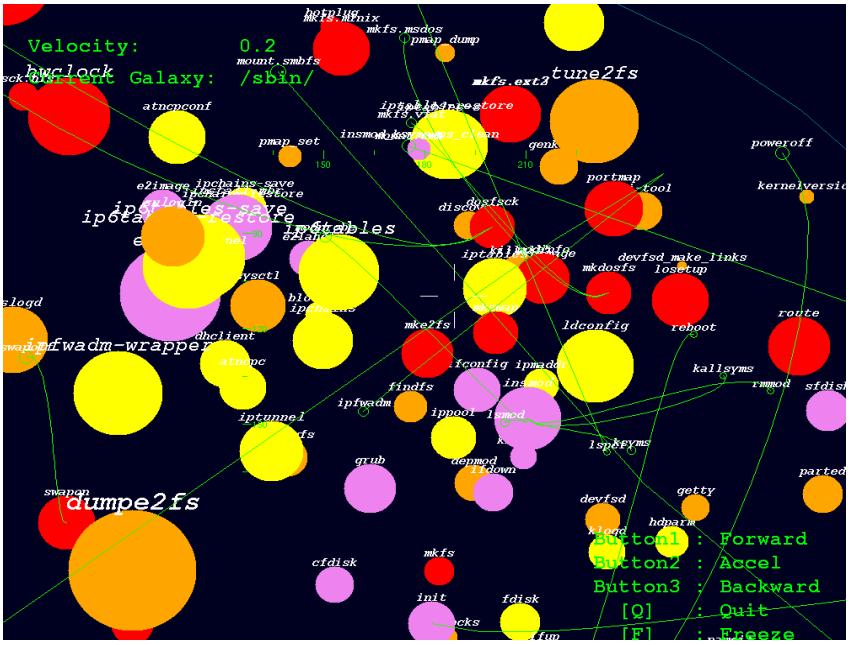


Figure 4.9. File size is proportional to the mass of the stars.

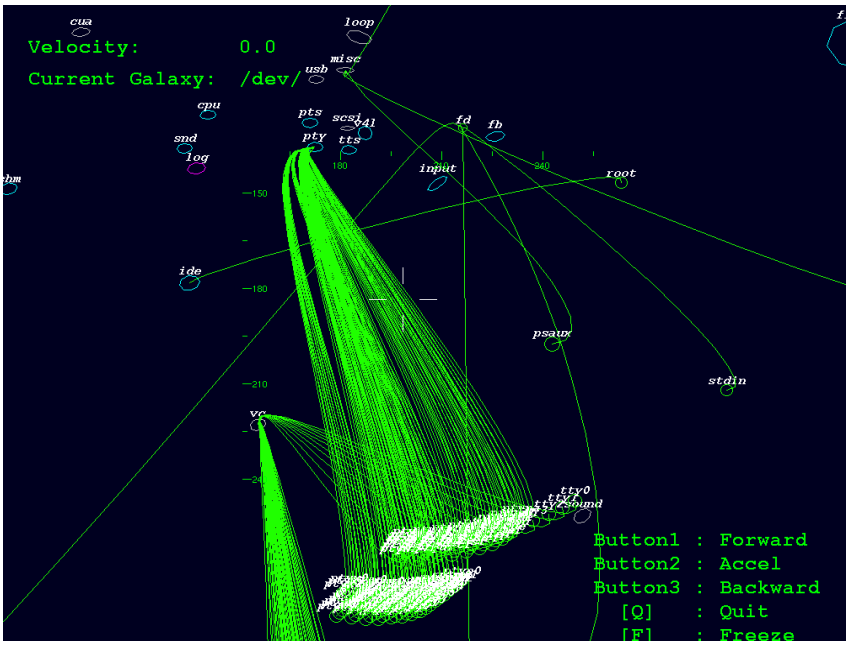


Figure 4.10. I never knew how many links there are on my computer until I ran Xcruiser.

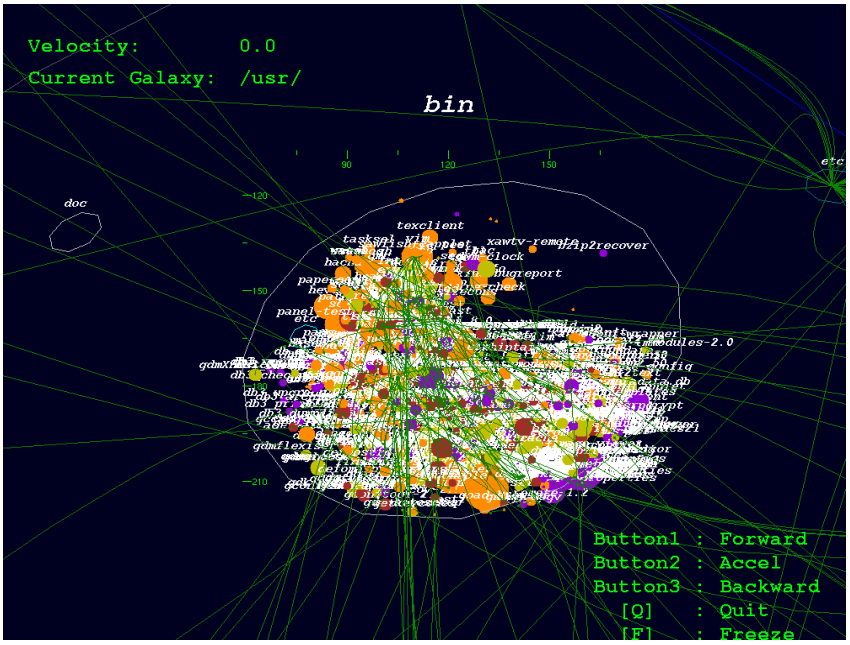


Figure 4.11. `/usr/bin` also contains many links. It actually seems to be quite a mess.

Xcruiser shows symbolic links in a very nice way. Normally I have not much noticed them at all, but Xcruiser displays them in a way that shows both the source and target of a link visually. It can also display structure by placing related files together, if they have similar names. As an example, /dev/tty∗ are all placed in a dense row in figure 4.10.

As the objects are placed freely in space it can be quite hard to find what you want to find. If you are in the middle of a galaxy there are objects all around you, and most of "around you" is located outside what is visible on the screen. There seem to be no order to how names are placed, at least no easily understood order.

I find it quite hard to navigate in Xcruiser. To make it faster to go to distant galaxies there is a constant acceleration while moving forward, so the speed gets high after a while. There is also a button to move quicker, but two distinct speeds are not enough. When entering a "sub galaxy", there is an automatic speed change that reduces the speed to what is appropriate in the reduced scale of the "sub galaxy". The speed selection does not work perfectly. Sometimes the speed is far too low, wasting time, and other times it is far to high, making it hard to aim correctly without stopping often and adjusting the direction. Aiming is quite hard overall, and could maybe be made easier. It might be better to use other controls, enabling the user to select the speed more directly. One way would be to change the speed by rolling the mouse

wheel, and/or fine tuning the current speed selection code so that it always works well.

Many of the problems with the navigation in Xcruiser were described in 1990 by Mackinlay et al. [13]. They propose an algorithm where a target is selected and motion then starts quickly and then slows down exponentially as the target closes in. This algorithm only works if one has a target to move to, and could be made to allow moving to either galaxies to get an overview of their contents or to individual files. The current navigation metaphor of flying in a spaceship allows for better general movement, and some of the charm of Xcruiser is flying around and exploring, which would not work as well with targeted movement. Even with the current movement scheme it may be good to start adjust the speed to be exponentially slower the closer the camera is to the closest objects in the middle of the screen, and reversely to speed it up exponentially as it moves away.

The current controls are very easy to learn, though, and that is important considering that the average user probably uses Xcruiser for a short while. Xcruiser is more of a demo than a useful program.

4.5 tdfsb

In tdfsb (http://www.hgb-leipzig.de/~leander/TDFSB/) (version 0.0.7), all entities are placed on a rectangular floor. If seen from above, they appear to be placed on a grid in alphabetic order from right to left, and from front to back. There seems to be some randomness to the placement, but this is probably caused by object size being proportional to file size.

Some kinds of files are displayed in interesting ways:

- The 3D icons for entities one has no read access to are put inside a semi-transparent red block.
- Symbolic links are shown as semi-transparent grey blocks containing the object the link points to.
- Images and movie files[2] are shown on the surfaces of rectangles. Only the first frame of a movie is shown unless one selects to play it, in which case the movie is played on the surface of the rectangle.
- Sound files[3] are shown as a set of speakers. If one chooses to play a sound file, its speakers start vibrating; this way it is easy to see which file is being played.

[2]Only MPEG 1 movies are supported.

[3]Only MP3 files are supported.

Due to the way icons are placed, many icons occlude those further back. Large files get very large graphical representations that occlude even more. This can be helped by flying up above the ground level, and getting a birds eye view.

Navigation in tdfsb is straightforward. Six keys on the keyboard move the camera[4] forward, back, left, right, up and down. The viewing direction is changed by moving the mouse, and a right click moves the camera close to a chosen file. It sometimes takes some time to go where you want to go, though, and I think the system could be made faster to work with. An idea would be to allow clicking on the ground to go to that place quickly, instead of holding down the key that moves forward until you get there. The difference between clicking on an object to view it and clicking on the ground to go there could be that current height above ground could be maintained. To view an object closely the camera must be placed in front of that object, but if you want to look around in a different part of the world it is only annoying that most of the view gets covered by a single object.

There are screen shots of tdfsb in figures 4.12 and 4.13.

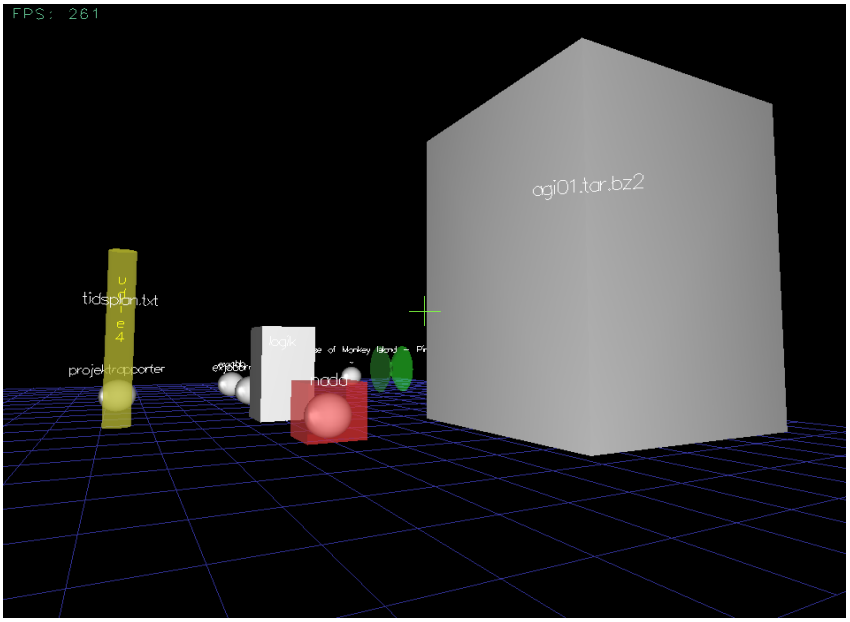


Figure 4.12. A text file, a directory for which I lack read permission and a large file as shown by tdfsb.

[4]With camera I mean the standard computer graphics term to refer to the computer screen (or "view point") as an object placed in the virtual 3D world.

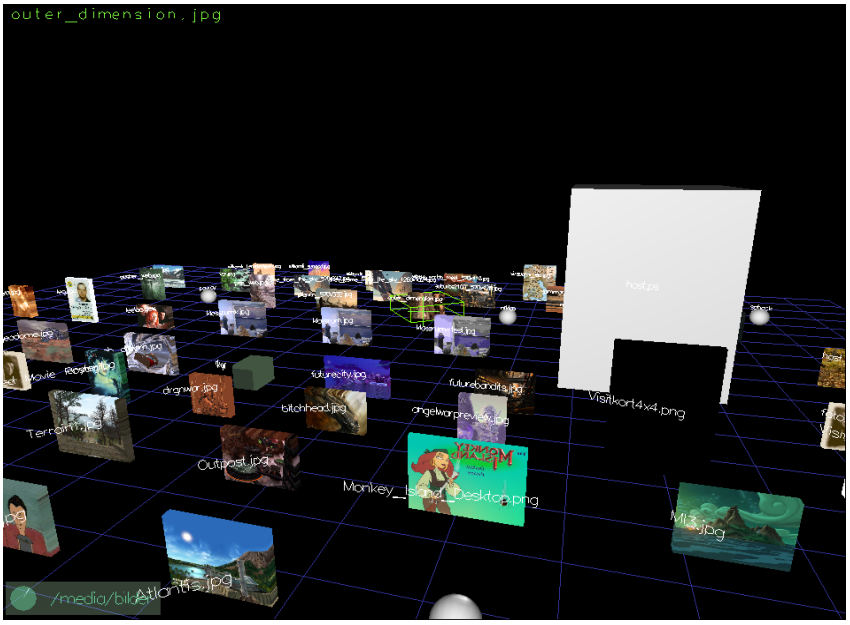


Figure 4.13. Some image files as viewed in tdfsb.

Chapter 5

Discussion of the techniques

5.1 Why is navigation important?

My goal in this project is to see if it is advantageous to use advanced graphical methods in a file system browser. A file system browser is a tool that is meant to be used, and nobody wants to use a tool that is tedious to use. That is why an emphasis must be placed on navigation. One must quickly and without fuss be able to move anywhere.

Many of the programs tested in chapter 4 were quite tedious to move around in. I believe the authors did not consider navigation a very important issue, as the programs were meant as demos of their respective visualization methods, but not to be practically used.

I have a different goal, and consider navigation important. I described in chapter 4 how navigation is handled in various programs. The navigation methods used were not always optimal for rapid navigation. I will discuss here how they could have been implemented.

5.2 Tree-maps

Those of the programs tested that are based on space-filling approaches are all more or less flat. In these programs navigation can be quite simple. The camera can be considered flying above the visualized hierarchy. It can be brought closer to an area of interest to view more details, or further away to gain an overview. It can be moved horizontally to view what is outside the borders of the screen, and it can be rotated to view objects from different angles.

All these actions could easily be handled with a mouse, for example like this: rolling the mouse wheel moves the camera up or down, clicking on an

object moves it to the center of the screen (horizontal movement) and holding the right button down and moving the mouse around rotates the camera.

These techniques use the screen-space very efficiently. They also have very little occlusion, and they are good at showing which entities use disk space. They show the hierarchy pretty well, but file and directory names are hard to see. In the original variant objects can be ordered alphabetically within the same hierarchy level, but not in squarified tree-maps. The biggest problem with names, though, is that objects typically are very small on the screen so the names do not fit. For most tasks users perform on file systems, file names are important, so the lack of clearly visible filenames is a big drawback. It would of course be possible to make the name pop up when the mouse is dragged over a file or directory, but if you want to search for a file by reading through file names this is not a good solution. (If you know what the file name is, though, it could be quicker to let the computer do the searching, unless manual browsing is faster than telling the computer to search and writing the desired name.) It would also be possible to write the filename with small letters on the small files, and use zooming to read through the names.

These techniques could not be used as the only means for browsing file systems, but they would be a good alternative when freeing disk space, for example.

It can also be a drawback that there is so much information shown at the same time. If all that is needed for a certain task is a small fraction of that information, this can make it unnecessarily hard to find what one looks for. In other cases, though, this is an advantage, as you can see so much at the same time.

5.3 Cone trees

A problem with the cone tree layout (see figure 3.3) is that icons are only placed in one dimension (along a line around the base of the cone). Because of this, it is only possible to see the names of a handful of entities within the same directory without rotating the disc. This makes it tedious to get to a node, even if you know the name of it, if there are many nodes in the same cone. It is my experience with 3DOSX that you may have to scroll very far, and as you do not see more than at a maximum of ten or twenty names at once you have to read the names of all icons as they go by, and thus cannot scroll very quickly.

Cockburn and McKenzie have evaluated cone trees [7] and found that their performance deteriorated rapidly as the branching factor of the data increased. They used branching factors between 6 and 20 in their test. Directories with many more entities than that are quite common. In their test, users performed

the required tasks only a little quicker using the normal trees than with the cone trees when the branching factor was 6. For the high branching factor (20) the difference was a lot larger.

An advantage with cone trees, though, is that several levels of the hierarchy can be seen at the same time (for an example, see figure 3.3). Because of this you can quickly get an idea of the hierarchical structure of the tree, and maybe also learn to navigate to nodes in this structure directly without going by intermediate levels of the hierarchy.

To alleviate the problem that occurs with a high branching factor, it might be possible to place children within cones with many children in some other way to show more children at the same time. Maybe several circles of child nodes could be placed on top of each other. Having several child nodes on top of each other could make it harder to see what is a new level of hierarchy, I believe, but this might be solved by drawing the cone trees in a way that clearly shows the separation between hierarchy levels.

Another way might be to place nodes in the interior of the circles (the bases of the cones). Even more nodes would occlude each other, but making it possible to hide everything above the currently focused node might still make the tree useful. There would not be so much occlusion if the tree was viewed from above and the nodes above the currently focused node were hidden or transparent.

5.4 Xcruiser, tdfsb

Both Xcruiser and tdfsb visualize file systems as a 3D space containing objects representing file system entities. This is very close to the traditional icon metaphor, but the icons have been replaced by three-dimensional objects and the "desktop" has been replaced by a 3D empty space. This gives both new possibilities and problems.

A 3D space has another dimension of freedom to place objects in. This makes it possible to place objects in different interesting ways. However, it also leads to problems when navigating. tdfsb has a quite regular placement of objects, and only shows one directory at a time. All objects are placed on a plane, so the program does not fully use all three dimensions. It is possible to get an overview of all objects by moving away from them, as they are quite tightly clustered together.

Xcruiser uses all three dimensions fully, and it shows many levels of the hierarchy simultaneously. This makes it hard to find what one looks for, in my opinion. Space is not only vast in the direction you look, but it is also possible to rotate to look in lots of other directions. It is possible to zoom out

a level in the hierarchy, to get an overview, and then zoom back in, but doing this can be a time-consuming process.

5.5 Rich icon representations

Stephanie Houde and Gitta Salomon examined how people recognize and find real world objects and discussed how icons could be made to support this process [11]. Many characteristics, such as size, color, shape, specific words, typography or images, etc. are used when recognizing objects, and sometimes particularly distinctive objects are used as reference points near which we remember other objects to be located. Also, we are good at ignoring objects of other types or looks than the kind we look for. When looking for a large dictionary, we quickly glance past video cassettes and paperbacks, and when looking for something red, we ignore objects of other colors.

I believe this should be used in file managers. Recently, it has become common to use thumbnails as icons for image files, and sometimes a frame from the beginning of an animation is used to represent that file. (tdfsb and Nautilus in GNOME do this.) This could be taken even further by using differently sized icons for different objects, and by showing miniatures of many other kinds of documents too. Also, it could be a good idea to look at what icons are used and make sure it is easy for the human visual system to quickly tell them apart.

5.6 Local or global visualization

Advantages of only showing one directory at a time are that it demands much less computer power and that the screen gets less small detail that can be hard to perceive. On the other hand, showing all files and directories at once has the advantage that the user can learn spatial locations of a certain file or directory in the "information landscape". Then it is possible to bypass several levels of the hierarchy and go directly to the wanted object. Another advantage of a "global" visualization is that it is easy to see properties of parts of the hierarchy. In the MapV visualization in fsv it is easy to compare how much space different directories take, for example, or to see that one directory contains many small files while another has fewer large ones.

MapV is a typical example of a "global" visualization, though directories can be closed to reduce detail.

A computer usually has a huge amount of files, though, so it can be very demanding to show them all in full detail. The computer I am sitting at right now writing this has 307 588 files and directories. It might be possible to

use level of detail algorithms to show only part of this without having this omission be visible.

A hybrid variant is also possible, where a few levels of the hierarchy are shown at once. A room full of book shelves full of books, is an example of this. Every book shelf could be one directory, every book a document.

Cone trees are also suitable to show a few levels of hierarchy at once.

Another hybrid is this: The user is hanging inside a sphere. Other objects lie on the inside of the lower half of the sphere. Some of these objects are directories, shown as transparent spheres in their turn containing more objects. This way, two or three levels of the hierarchy can be seen and manipulated at once. As only the lower half of the sphere is used, it is possible to see everything in a directory by just being high up in it and looking down, or to see more detail by moving closer to interesting parts.

Chapter 6

Prototype

The existing programs described in chapter 4 are good for certain purposes, like determining the size of file system objects. I want to determine, however, if a 3D file system browser can replace a 2D file system browser, and to do that the 3D browser must be general enough to enable people to perform all or almost all tasks that are usually performed with a file system browser.

I only had a limited time to create a prototype file system browser that shows this is possible, so the prototype cannot be very featureful, but only allows navigating through a file system visualized in some way, while still showing that the visualization and navigation methods used are practical.

Looking at existing programs, I eliminated the techniques that seemed worst from a general browser view-point. Cone trees are not very good at handling trees with large branching factors, and tree maps are not good at file names. This more or less rules these out for looking for files with known names in medium and large-sized directories, which is done often. Some variant of "objects placed in a 3D space" seemed best, like Xcruiser and the Information Cube. The Information Cube provides the best overview, as you can rotate around everything, and then zoom in to view details, while in Xcruiser you are in the middle of everything looking out.

The Information Cube has a problem with occlusion, though, but that might be solved by placing objects so they do not occlude much. One way would be placing all objects on a single plane, but then we would have a 2D file system browser. Some other placement may be able to use the extra dimension without having much occlusion problems.

6.1 Wanted features

Before implementing the prototype, I wanted to summarize what can be desired of a file system browser. As I am mostly focusing on how files are located, most of these wishes relate to that.

There are different ways of finding a file or directory. If the name of a file is known, it is often possible to type the first letters of this name to select a file, and it is also possible, though generally slower, to scan through a sorted list of icons. Some systems rely primarily on preserved icon positions, so that the user can remember where on the screen the icon is located, though these systems generally allow alphabetic sorting too when things turn into a mess. One person I have spoken with often sorts some unorganized files in date order, and then looks through all files of similar age to the file he looks for, and another often looks through screens full of miniatures of photos to locate interesting ones.

The following are all features that would be good in a visualization, though it would probably be impossible to have all of them at once:

1. That all files are organized in alphabetic order so it is easy to search for names.

2. That all files are organized in date order, so it is easy to find files you know when they were created, modified or last read.

3. That all files have persistent positions, so it is easy to remember where commonly used files are placed. A side benefit of this is that it makes it possible to remember the look of a directory.

4. That the user can place files himself, in a way logical to him.

5. That the view is compact, fitting as much as possible at once on the screen.

6. That it is easy to locate files based on content.

7. That all related files are shown together (in a cluster).

8. That it is easy to recognize files of a specific type, for example by having different kinds of icons. This can speed up finding files, when the type it known.

9. That it is easy to locate picture files, even if their file names are unknown. (Say for example if all files are named “DSC000??.JPG”.

10. That several, preferably all, levels of the hierarchy are shown at once.

11. That it is easy to see how much space different files and directories use.

It would also be good if the program would learn usage patterns, and unobtrusively suggest the file or directory it believes is currently wanted.

6.2 Object placement

A simple way to solve the occlusion problem in the Information Cube would be to let the user place all objects. This would have the advantage that the user will know where all objects are, so he can easily find them, and it will probably result in a good placement. (At least for this user.) Its biggest problem is with files that are automatically generated and files or directories that come from different systems not using 3D positions. These files have to be placed automatically by the browser. If these are only a few files they can be placed at some default location, and the user can then place them where he wants them later. There can also be a "clean up" command that places all objects in a directory in alphabetic (or other) order. This is how the browser on the Amiga works.

Another advantage of manual placement is that objects keep their position until manually moved. This makes it easy to find commonly used objects.

Most common today in file system browsers seems to be to use automatically placed icons, and this is probably best for my prototype as all existing files will come from external systems. Manual placement may make it easier to experiment with different ways of organizing files, but it may be quite subjective and harder to evaluate. It is also less intuitive to position objects in three dimensions than in two dimensions, using a mouse.

It is possible to keep the advantage of files having a fixed position by making the position a function of file system properties and not dependant on other files, or in other words:

$$(x, y, z) = \mathbf{F}(\text{name, size, mod-time, } \dots)$$

Properties that are available for almost all files are name, size, change time, modification time, access time, file system type (directory, regular file, FIFO, socket, device node), type of regular file (image, document, music, text), owner, group, access rights, number of hard links, i-node number, IO block size and device number. There are additional properties available to some file types. Music files, for example, also often have properties such as author, album, title, length, bit rate and encoding quality.

If files are placed one after another, as in the Explorer in Windows using alphabetic order, they cannot retain their positions if new files are inserted or if files are removed. If they are not densely packed, placing them only based

on properties should be workable. It is still possible to have files in alphabetic order by designing the function **F** appropriately, but they cannot be densely packed. If files are too densely packed, the probability of two files ending up in the same position is significant. On the other hand, if files are placed too far apart, it could be very hard to get an overview.

I realized before starting to implement a prototype file system browser that object placement will probably require some experimentation to get right. I intended to first try displaying the different properties as listed in table 6.1. As file names are so important, it may be better to use more than one dimension for them, but I wanted to start with something simple.

Table 6.1. Possible mapping between file system properties and display parameters.

file system property	display parameter
file name	y position
modification time	x position
change time	z position
access time	brightness
log(size)	size
type	shape and color

In early tests, I noticed a problem with using different sizes for different objects. It makes it hard to see which objects are close and which far away, and does thus partially remove the benefit of having depth. It is still possible to rotate and to see that objects are in front of other objects, so all feeling for depth is not lost, but it is harder to perceive. In the prototype I implemented, I ended up using a simpler mapping, shown in table 6.2. This could probably be improved upon, but there have been so many things that can be improved upon that I have not gotten around to this one.

Table 6.2. Mapping between file system properties and display parameters used in the prototype file system browser.

file system property	display parameter
file name	y position
log(size)	x position
log(time since last modification)	z position
type	shape and color

The use of logarithms is intended to show the magnitude of size and age. There are many small files and a few that are huge. If a linear scale had been used, all small files would end up at the same place and huge ones far

away. With a logarithmic scale, the files are more evenly spread, and thus more information is conveyed.

In order to map file names to the y axis, I created a function that converts a name to a number in the interval $[0, 1]$. First I created a function that converts a character, c, to a number, $p(c)$. $p(c)$ is the position of c in the string $s =$" 0123456789abcdefghijklmnopqrstuvwxyzåäö.", or $1 + \text{len}(s)$ if c is not in s. For example, $p(\text{"d"}) = 15$ and $p(\text{"\#"}) = 42$ (the last because "#" is not part of s).

To convert an entire name, where $c_1, \ldots, c_n$ are the characters, the function is

$$\text{nameToNum}(c_1, \ldots, c_n) = \sum_{i=1}^{n} \frac{p(c_i)}{(\text{len}(s) + 1)^i}.$$

This makes sure that if name a is before name b in alphabetic order,

$$\text{nameToNum}(a) < \text{nameToNum}(b).$$

The function could of course be improved to handle more uncommon characters, but that was not necessary for me.

6.3 Filtering

One goal that is difficult to obtain is to be able to find files based on content. For picture files this is easy to do by displaying the actual picture instead of an icon. This is not really computer-aided searching, just a way of browsing picture content quickly. If the number of pictures is huge, they must still all be looked through.

There are also systems that allow searching in picture files through advanced feature recognition systems, for example. I have tested one where searching was done by selecting the pictures that are most similar to the wanted picture from a screenful of miniatures. This produced another screenful of miniatures where the search could be refined, and this continued until the wanted picture is found.

Browsing through miniatures does not work well for text files, as a text file contains much more text that what fits in an icon. What is possible is to add a filter so that only files that match a selected pattern are shown, and directories that contain such files.

An advanced searching system could map the score of a file to its brightness, so that a file with a score of zero would be black (or not drawn at all), and a file with a high score would shine like a beacon. The brightness of a directory could be the brightness of its brightest file or subdirectory. This way it would be easy to browse through all relevant files.

6.4 Implementation

My prototype consists of two programs: a file system scanner and a visualization program. The scanner scans a directory and its contents and generates a list with one item per file system object in that directory. Each list item contains the file name, its path, its position in space, its type and its radius.

The visualizer uses OpenGL to show these objects and allows navigation through them. For each subdirectory the program calls the file system scanner again to get the contents of the subdirectory, and renders that within the "icon" of the subdirectory. Directories are drawn as semi-transparent cubes so that their contents can be seen through their walls.

There is a screen shot of how this looks in figure 6.1. Text files are rendered

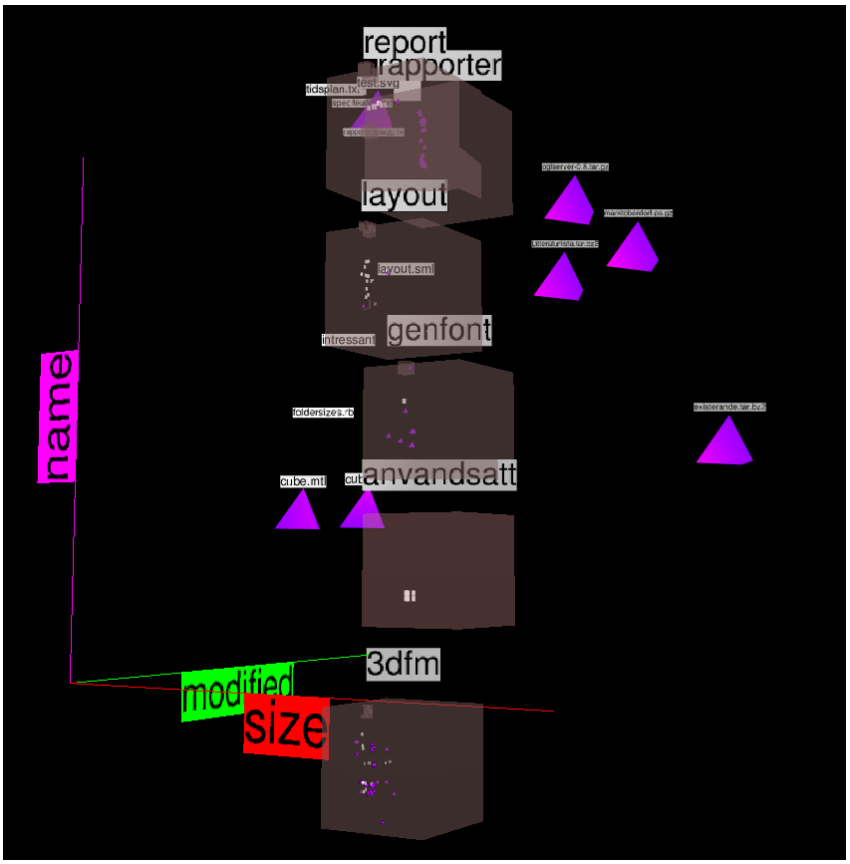


Figure 6.1. Screen shot of my prototype with intersecting objects ("report" and "rapporter").

as a set of white pages (the number of pages for a file is its number of lines divided by 66), and unknown files are drawn as tetrahedra.

I experimented with putting the entire contents of text files in their icons, but this turned out to be too slow. It would probably work if contents were only drawn if an icon filled a substantial part of the screen, or as an approximation of how a text file looks from a distance when it is further away. My level of detail implementation was not good enough at determining which files should be drawn with such detail, so for me this made the program too slow. I did not judge this to be important for this project so I decided not to put any more work in it.

6.5 Collision avoidance

A problem that is obvious from figure 6.1 is that objects can end up occupying the same space. Some way of moving "colliding" objects apart is required.

Graph layout algorithms have the same problem, and a simple technique used when doing graph layout is to minimize an energy function. The energy function is made to have higher energy levels when objects are too close, and also to have higher energy when objects are far from their original positions. I implemented this using the Quasi-Newton method for minimization of multivariable functions [3]. It seemed to work well on small data sets, but when I ran it on larger directories it became apparent this method is far too slow. It could take minutes to run on medium-sized directories.

I looked at other graph layout algorithms to find one that is faster, and can be adjusted to my problem. I found one designed by Matthew Chalmers [5] that runs in linear time in the number of nodes. After I had adopted the algorithm to my problem, every iteration did the following for each node:

1. $R \leftarrow$ select k random nodes.
2. $N \leftarrow$ the k nodes from $R \cup N$ that are closest to the position of the current node. This is a random approximation of the neighbors of the current node. Initially $N = \emptyset$.
3. $$\boldsymbol{F} \leftarrow (\boldsymbol{x}_0 - \boldsymbol{x}_i)\,|\boldsymbol{x}_0 - \boldsymbol{x}_i|^2 + \sum_{\boldsymbol{n} \in N} \frac{\boldsymbol{x}_i - \boldsymbol{n}}{\left(|\boldsymbol{x}_i - \boldsymbol{n}|^2 + 0.1\right)^2}$$
   where $\boldsymbol{x}_i$ is the position of the current node for iteration i and $\boldsymbol{x}_0$ its original position. I put the 0.1 term there to avoid dividing with zero.
4. $\boldsymbol{a} \leftarrow \frac{\boldsymbol{F}}{m}$, the acceleration of this node. The mass, m, of all nodes are a constant.
5. $\boldsymbol{v}_{i+1} \leftarrow \boldsymbol{v}_i + \boldsymbol{a}\,\Delta t$. The initial $\boldsymbol{v}_0 = \boldsymbol{0}$. Δt is the chosen time step, also a constant.

6. $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + \boldsymbol{v}_i\,\Delta t + \boldsymbol{a}\,(\Delta t)^2/2$.

This algorithm is performed a fixed number of steps, and thus its runtime is linear in the number of nodes.

Chalmers has designed other algorithms that may be even faster than the one I modified above, but they are also more complex and maybe more difficult to modify to suit this problem, if this is possible at all.

Testing of the above algorithm shows that it does not work very well. Though quite slow, it is fast enough for my prototype if few iterations are performed for small values of k, but then objects tend to end up in fairly random positions. Figure 6.2 shows how this looks with 700 iterations. With 700 iterations and a k of 5 (used here), it takes between zero and thirty or so seconds to move objects in a directory. The end result, as can be seen in the figure, is that objects are moved quite far from their original positions (shown in figure 6.1). It may work better using a larger k, an even larger number of

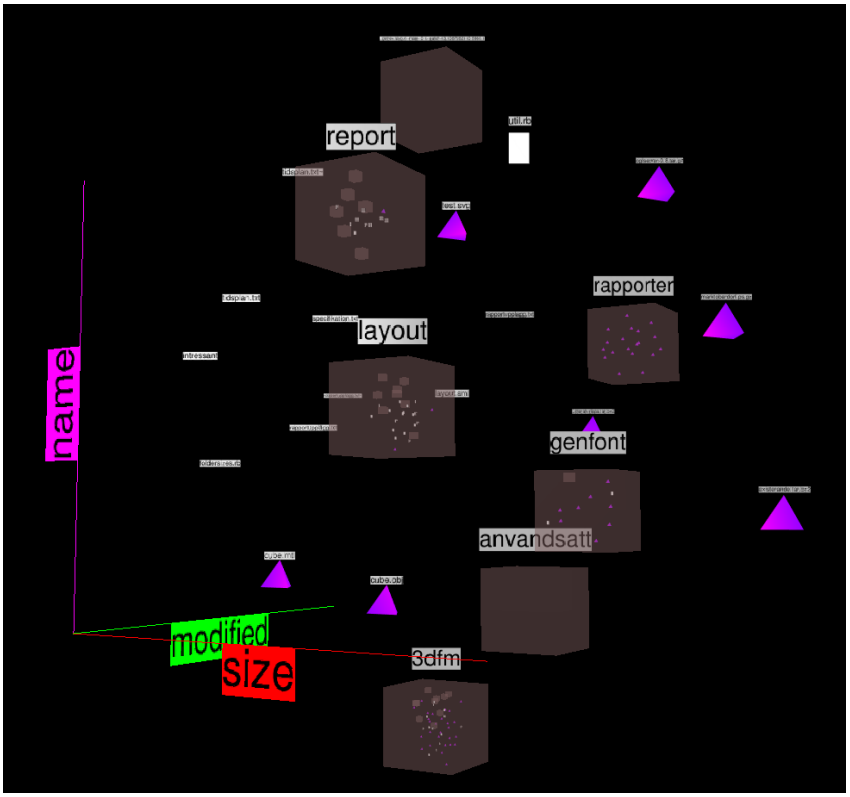


Figure 6.2. Screen shot of my prototype with spring repulsion algorithm.

iterations, a small Δt, or maybe my force function must be improved.

Instead of trying to get this to work, I decided to use a different approach. As the file names are placed on the y axis, I sort all files on their position on

this axis. Then I take one file at a time and either leave it if it does not collide with any previous file (with smaller y coordinate), or push it downwards until it does not collide with any previous file. No file is placed further up than the previous file, so it is known that files end up in ascending name order on the y axis. This also speeds the algorithm up, as it is only necessary to compare files as far back as $|y_a - y_b| < 2r$, where r is the largest radius of any file.

Though this algorithm can theoretically have a quadratic runtime it seems to always complete in less than a second. This is probably because it only does one fairly simple iteration, whereas the other algorithms do many.

As can be seen in figure 6.3, objects end up much like in figure 6.1 with this algorithm, but with no overlap. Unsurprisingly, the result tends to be more stretched out vertically than the other algorithms.

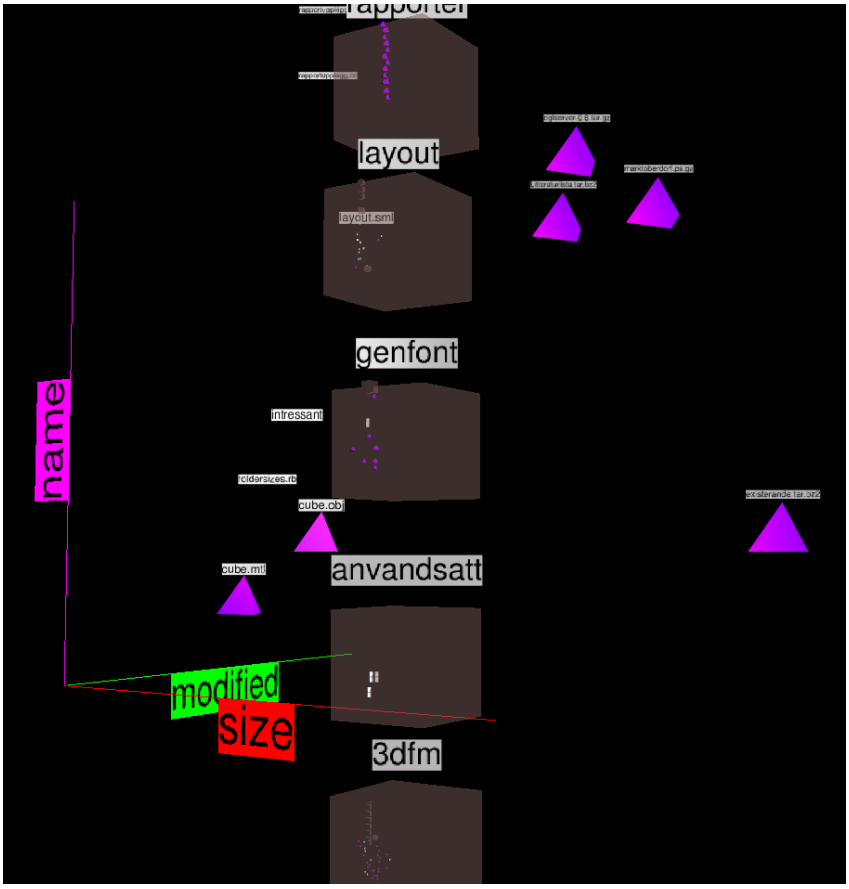


Figure 6.3. Screen shot of my prototype with colliding objects pushed apart vertically while maintaining alphabetic order.

6.6 Navigation

Focus and zoom

The way I have implemented navigation is as follows: There is always a point of focus. Clicking a file or directory with the left mouse button sets the point of focus to that object. Middle-clicking also zooms in so that the object fills a significant part of the screen. This way, it is possible to middle-click on a directory to see its contents. Then it is possible to middle-click again on a subdirectory to see its contents in turn.

Rotating the mouse wheel zooms the display in and out.

When an object is focused the camera is moved so that the center of this object is in the center of the screen. This movement is implemented the way Mackinlay et al. propose [13] so that the movement is smooth enough to keep the user from losing track of where he is and fast enough so the user does not have to wait. This is very simple to implement. Every time step, the focus point, $\boldsymbol{f}_i$, is moved to

$$\boldsymbol{f}_{i+1} = \Delta t\, k\, (\boldsymbol{v} - \boldsymbol{f}_i), \tag{6.1}$$

where Δt is the amount of time that has passed since the last update, k a constant that determines how fast the motion happens overall, $\boldsymbol{v}$ the target the focus point moves towards. This will result in an exponentially slowing velocity.

Zooming, when an object is middle-clicked, is done using almost the same algorithm. I started by using exactly the same algorithm, but this proved to have a problem. It appears that zooming in linearly at the same time as moving the focus point linearly to the target object will make the target object move outside the screen early in the zoom and then glide in from the edge of the screen when the zoom is nearly complete. I illustrate this problem in figure 6.4. The wanted behavior is for the movement and zoom to look like it follows a straight line to the target object. The zoom should look like we are really moving closer to a large object. In figure 6.5 I illustrate the wanted behavior.

Plotting Δx versus the zoom factor in a diagram from the numbers in figure 6.5 results in the diagram in figure 6.6. It appears to be a logarithmic function. I tried using the logarithm of the zoom factor instead of the zoom factor in the algorithm described above (6.1), and this seems to work quite well. The movement is not always perfect, but it is quite good. I suspect a slight adjustment could make it perfect. Such an adjustment might be using a logarithm of a different base, or inserting a suitable constant somewhere.

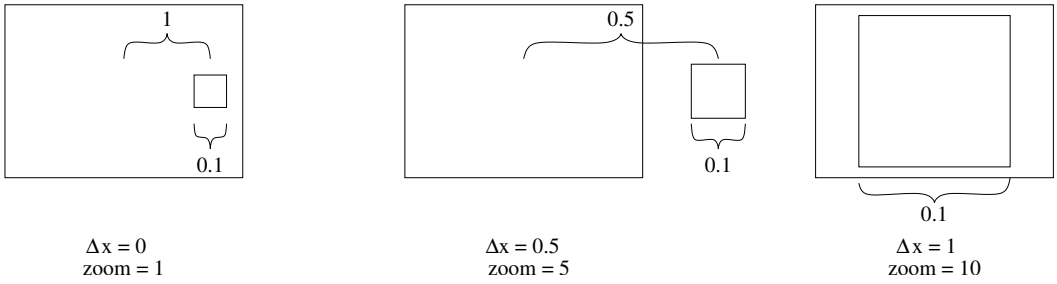


Figure 6.4. Animating the zoom factor and moving the focus point simultaneously results in what does not seem like a straight movement.

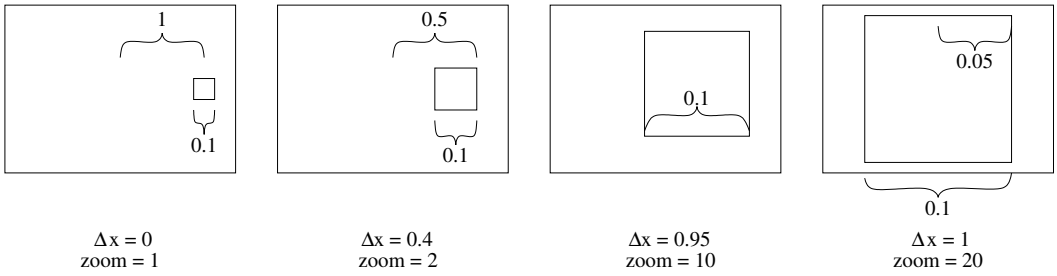


Figure 6.5. Wanted behavior for simultaneous zoom and focus movement.

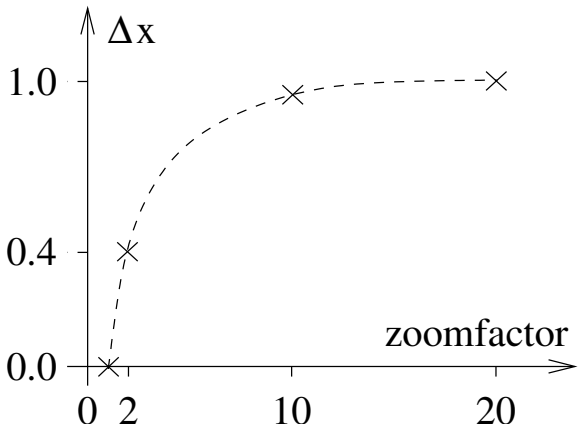


Figure 6.6. Δx versus the zoom factor for figure 6.5.

Highlighting

To make it easier to see which object would be selected on a click if many objects are clustered closely, I did as done in GTK+ (http://www.gtk.org/): highlight the object that is under the mouse. Whenever the mouse is moved, a ray is cast straight into the scene from the mouse position. The first object this ray intersects is highlighted.

Rotation

Holding the right mouse button pressed down rotates the camera around the focus point. I had intended to use "virtual sphere rotation" [6], where rotation happens as if the user grabs hold of an invisible sphere with the mouse when the button is pressed and drags the point of this sphere the mouse pointer is attached to while moving the mouse with the button pressed. Before actually finishing this implementation I realized it is probably simpler for the user to use a simpler rotation scheme. With a virtual sphere, it is possible to rotate in more ways than strictly necessary, to end up upside down or tilted sideways, for example.

I decided to just make horizontal mouse movement while the right button is pressed rotate "sideways" (around the y axis), and vertical movement rotate up and down (around the x axis). I also clamp the vertical movement so it is not possible to end up upside down (by limiting the vertical angle to the interval $[-\pi/2, \pi/2]$).

6.7 Correlation problem

When the initial problems were solved, and it was time to do more serious testing, another problem became apparent. In many directories, all files were last created, modified and accessed at almost the same time. This means they all end up in the same position on the z axis. Also, quite often all files are of almost the same size. An example of this is a copy of the Linux source code (shown in figure 6.7). Here all directories have the same size, 4 096 bytes, because directories in Linux are always an integral number of disk blocks, and all are modified when I extracted the archive the code came in. The problem with the size of directories could be solved in this case by using the cumulative size of the contents of a directory instead of using the disk size taken by the actual directory listing, but to make sure files have different modification times it would have to be made sure that all tools retain the time a human last modified a file. Many tools seem not to do this today, except if specific flags are specified.

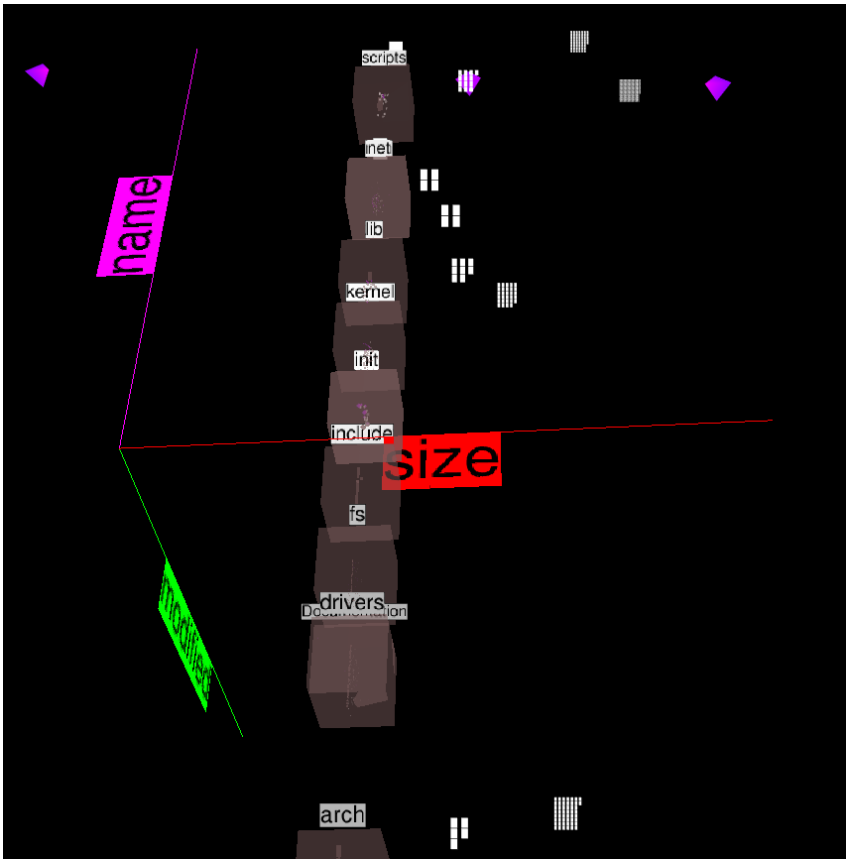


Figure 6.7. Linux source code. All directories have the same size and the same modification time.

There are not many other properties that could be used instead of those used in my prototype. Among those properties that are available for almost all files (see section 6.2 page 35), only few vary between files in the Linux source code example. Almost all files are C source code files, C headers, directories or ASCII text files containing documentation. These different types of files could be mapped to a coordinate axis with four discrete values. Filename and size (or size of contents in case of directories) also varies, and the i-node number too. The i-node number is meaningless to most people, though maybe it could be used as a random number to create a random placement that is constant over time. A pseudo-random placement is not entirely useless, as positions can be remembered, and at least one axis can be the file name. Such a placement does not provide as much information and does not help searching for information, though.

Another example of where files and directories have the same size is my copy of my CD collection. I keep every CD in a single directory. All files here

are in Vorbis format (http://www.xiph.org/ogg/vorbis/), most are of about the same length (and thus the same file size), and all songs in a specific album are ripped at the same time. All CD:s are of about the same size, so even the size of the contents of directories are about the same for all directories. The only property among those in table 6.1 that may differ between files in a directory is the file name, the i-node number and the access time, unless I last listened to the entire album at the same time. Of course, different albums are ripped at different times, so different directories have different modification times. Even among the extra properties that are often available for music files (see section 6.2), most are more or less constant in a directory. The song title differs, of course, and the length and bit rate usually varies slightly. On some CD:s different songs have different authors, but most often they do not.

The conclusion that can be drawn is that placing files on positions in space the way I have done it does not work well for all directories. Other ways, such as basing the position on all three axes on the file name, or using pseudo-random positions may work better in these cases, but they do not provide as much information to the user. In the case the visualization in my prototype works it has the advantage that it allows searching on file name, modification time, size and file type, or a combination of those.

The best way may be to use different layout methods depending on what directory is being visualized. Often used directories can have manually placed objects, others could use name, size and modification time and those where this does not work well could use only the name and maybe the i-node number to generate a position, or maybe have objects placed in tightly packed rows and columns as most 2D browsers. It might get confusing though if the layout changes when you enter a directory.

6.8 Grid layout

As an experiment, I have added the way most 2D browsers place objects to my prototype. This was very easy compared to the other layout methods I have implemented. I placed objects in columns and rows like usual in the x-y plane, and let the z position reflect the size of the file like in previous visualizations. There is a screen shot in figure 6.8.

When it was time to do a user study, I chose not to use this layout method, even though it is probably better than the other ones for some tasks, as it is very similar to traditional browsers. Because of this, a comparison would have compared specific implementations, and their problems, and not different layouts. It was more interesting to compare the other layout method to a traditional one.

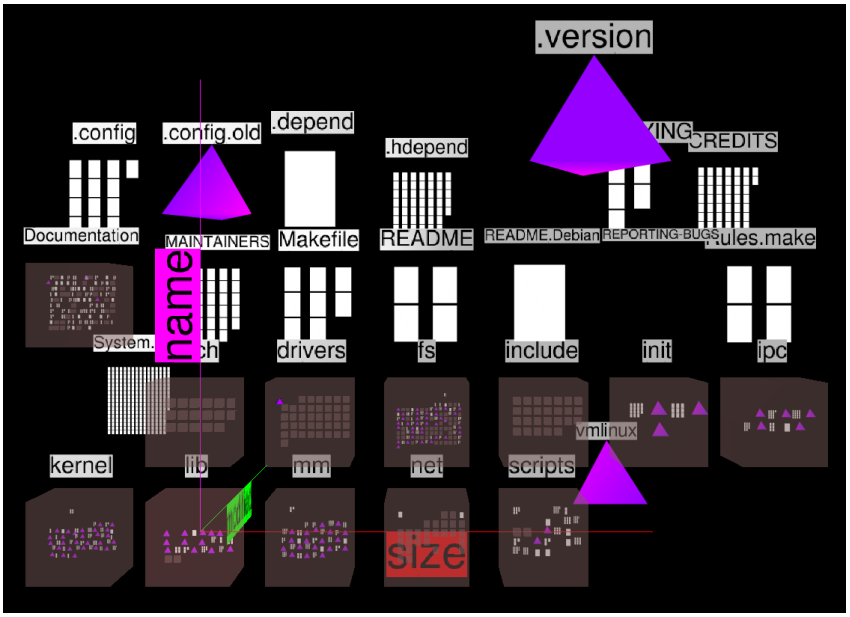


Figure 6.8. Grid layout. (Note that the labels on the coordinate axes are not correct.)

Chapter 7

Second user study

When I had implemented the prototype described above, it was time to test it by performing a user study. Mostly I wanted to see how quickly file system items could be found, and how well the implemented mouse navigation works.

I wanted to use something that is freely available on the Internet as test data, both as this makes my test repeatable and as this ensures the test uses "real-world" data. I had a CVS checkout of the game client "uclient" from the WorldForge project (http://www.worldforge.org) lying around, and as CVS seems to preserve modification times this directory worked well for me. To avoid the problem of most directories having the same size of 4 096 bytes, I changed the code to use the size of the contents of a directory as its size instead of using the size of the directory listing on disc as I had done previously.

Then I had a number of persons do a couple of actions in my prototype and in the file browser in Microsoft Windows XP, and timed all actions. I also watched the subjects to see if they had problems with any specific issue, and asked them what they thought when they were done. Every other person did the actions in my prototype browser first, and the others did them in Windows first.

The actions I had them do were:

1. Enter the directory "player".
2. Select "Makefile.am".
3. Select "player.cc".
4. Exit from the "player" directory.
5. Enter the directory "dialogs".
6. Check how many directories there are in the directory "scratch".

7. Exit from the “dialogs” directory.
8. Enter the directory “text”.
9. Enter the directory “public”.
10. Select the directory “screenshots”.

I had them repeat the above list five times, to enable them to learn how to use my program. Then I asked the following questions (here translated to English) about the files in the root directory in the hierarchy:

A. Which files have been changed lately?
B. Which two files were changed longest ago?
C. Which file is smallest?
D. Which file is largest?
E. Enter the “CVS” directory and tell me how many files and directories are in it.
F. Exit from the “CVS” directory and select the “macros” directory.

After this I had them do the first actions two times more.

In the tables below, the five repetitions of the first list are labeled 1–5, the items in the second list are labeled A–F, and the last two repetitions of the first list are labeled 6 and 7. The times for the actions when using my 3D browser are in table 7.1 and the corresponding times for the standard browser in Microsoft Windows are in table 7.2.

To not have to reboot between Linux and Windows between every test, I used two different computers. The laptop that ran Windows XP has a touch pad, and not a mouse. This surely affects the times adversely. Unfortunately I did not realize this problem until the third user as the two first ones were family and were used to that laptop and the touch pad. The third user (and some later ones) had some difficulty with the touch pad. I did not change the procedure for the last users as that would have made it harder to compare results.

7.1 User comments

After each person had completed the actions mentioned above, I asked if they had any comments. This was very informative. So was looking at them using the program and seeing what mistakes they made.

Table 7.1. Times for different actions using the prototype 3D browser. (At the question mark, I had forgotten to start the clock.)

	Test subject					
Action	**1**	**2**	**3**	**4**	**5**	**6**
1	1:46	1:05	1:41	1:37	1:19	1:51
2	49	1:56	1:19	59	40	?
3	48	43	59	1:14	41	1:00
4	40	36	1:20	1:03	39	31
5	35	41	1:27	1:04	39	35
A	50	36	29	52	19	52
B	15	23	26	24	12	57
C	5	9	10	13	9	14
D	8	7	13	9	8	10
E	11	12	28	14	9	12
F	6	7	4	7	5	6
6	32	1:05	1:04	1:08	45	53
7	30	40	1:40	32	35	1:28

Table 7.2. Times for different actions using the file browser in Microsoft Windows XP.

	Test subject					
Action	**1**	**2**	**3**	**4**	**5**	**6**
1	31	1:27	52	55	42	1:39
2	30	32	33	39	31	1:12
3	27	24	32	33	39	1:01
4	24	15	31	28	28	45
5	17	17	42	28	24	44
A	1:01	56	56	1:12	40	58
B	11	12	11	16	7	9
C	13	23	18	12	14	16
D	7	4	13	5	3	7
E	12	20	26	11	6	17
F	9	8	7	8	7	8
6	22	46	28	32	28	44
7	18	23	33	33	32	41

In Windows, after performing actions A–F, all subjects had the files ordered in a detailed list instead of as icons, as they needed to sort the files in size and date order. Because of this, it took the users longer to find folders doing actions 6–7 as the icons were not in the remembered place, and sometimes it took them longer to click on them as a list item is narrower than an icon. A similar effect happened in my 3D browser for some of the subjects, as they had the view differently rotated after A–F. This made it harder to find objects.

In Windows, objects keep their positions as long as nothing is inserted or removed, and nothing was in this test. In my browser, objects appear at different places of the screen depending on the rotation angle. My browser tries to keep object positions constant when objects are inserted or removed, but many users had problems finding objects on the screen when they had rotated the view. As it was sometimes necessary to rotate the view to find objects this was a problem.

It would have been very good if everything outside the directory you are in had been hidden, in my browser. Sometimes it was hard to tell if an object was part of the directory contents or not. This would also have reduced visual clutter.

When zooming into a directory, my browser zooms towards the focus point (which is at the center of the directory when it has been selected). Objects that are not close to this point may end up outside the screen when zooming far. If this happens, rotation or zooming out is required to be able to see the sought object.

It was suggested that it would be good if the objects were more sparsely packed, so there would be less occlusion. (This would increase the previous problem, though.)

One test subject said when all content did not fit in one screenful he found it faster to hold down the right mouse button and drag the mouse (rotation in my browser) than to drag the scroll bar at the right of the window (in Windows).

In my browser, it can be hard to find objects when they are occluded by other objects.

Many subjects tried to click on an object in a directory when it was still quite far away. I had set my browser to interpret a click as a click on the containing directory until it covered almost the entire screen. It would probably be better to start interpreting clicks as meant for something in the directory much sooner.

In my browser, several subjects often tried to click on the label above the icon of an object, instead of clicking on the actual object. I had not anticipated this as I never do this myself, and thus the program treated the label as empty

space when clicked on and did not focus on the object. (If there was another object behind the label, that would be focused instead.) One subject also suggested that it would have been good to be able to click on the label when the icons were almost on top of each other.

I had set the program to zoom in close to an object on a middle click. It would have been much better if the program zoomed further in, to the point where the contents of the directory just fit on the screen. Now, users had to do some manual zooming with the mouse wheel after the middle-click.

In my browser, it took quite long to zoom out far. This had to be done with the mouse wheel, and I had tuned it for zooming in. Using a faster zoom speed when zooming out would be a good idea. There should also be a single key press or button available to zoom out one hierarchy level quickly.

It was suggested that the zoom was sometimes too fast and sometimes too slow. Due to the discreet steps of the mouse wheel it was sometimes hard to zoom just a short way.

When told to find the smallest file, two users were confused that there were many files that the browser in Windows said were 1 kb large. It rounds sizes to entire kilobytes, but when asked to sort on size, it still sorts on exact sizes, so it was still easy to find the smallest object that way.

One user said the usual browser was much easier to use, and another said, about my browser, that if things go wrong you get entirely lost.

7.2 Conclusions

In figure 7.1 there is a graph of the average time taken for each test, and the standard deviation of these times. It is clear from this graph that the time taken in Windows XP is on average close to or lower than the time taken using my 3D browser. This is hardly surprising considering all the problems described in section 7.1. If these problems were fixed the times would probably be lowered.

That most users are used to the browser in Windows, and have used it many times, while this was the first time they used my system is probably the largest factor in the time differences. The only way to avoid this would probably have been to develop a different reference application that was made so that people would have to learn it too, to use test subjects that had never touched a computer, or to do a very long test, so that they got equally used to my browser.

After having watched all the users during this user study I tried the tests again, timing myself. Doing the sequence of actions I had the users do many times took me between 14 and 23 seconds for all seven tries. This is significantly better than the best time any of the test subjects had (30 seconds). I

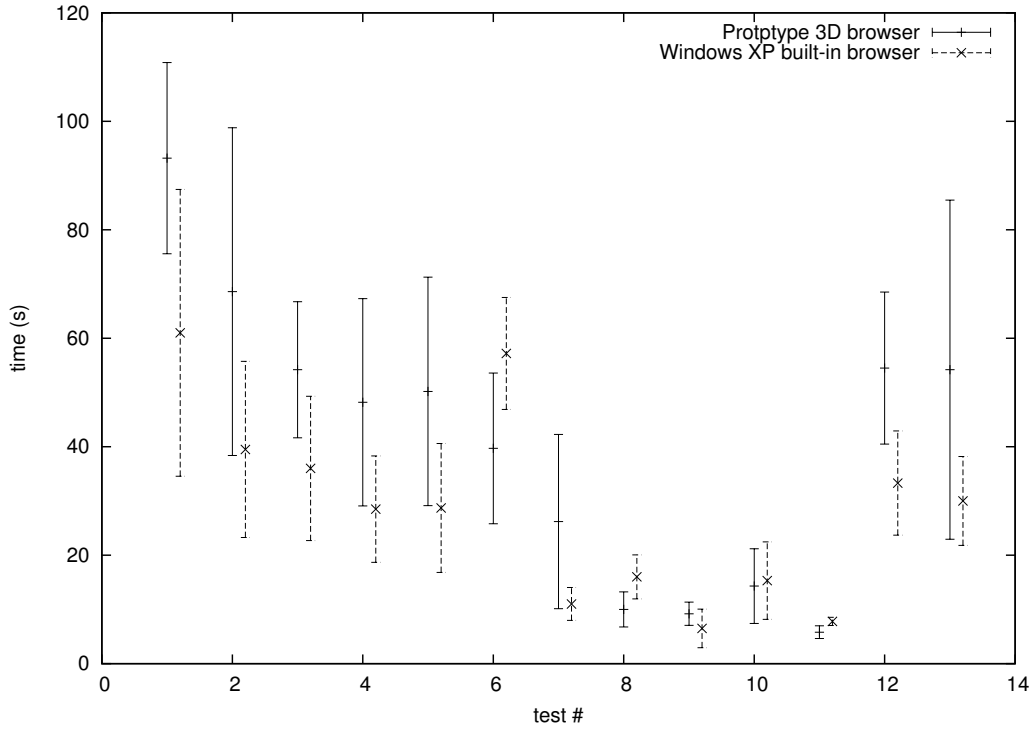


Figure 7.1. The average times taken for all tests, and the standard deviations as error bars. (Note that due to my graphing program the tests that are referred to as tests A–F in the rest of this text are numbered 6–11 in this graph, and tests 6–7 are numbered 12–13.)

have run this program many times to test it while developing it, and I have watched all the errors the test subjects had and avoided making them myself. As a comparison, my times doing the same test in Windows are 16 to 26 seconds. That times in Windows are slower may be because of the touch pad on the laptop, and maybe because all the icons in Windows look the same and are arranged in a grid. Sometimes I had to scan the names to find one, while in my program they can be recognized on their position in relation to others, as long as you make sure to view the icons from the same direction, which I made sure I always did. I also made sure to choose a direction where there was not much occlusion in this data set.

Chapter 8

Discussion of results

8.1 The implemented visualization

The visualization I have proposed in this report has some flaws. It does not work well on all kinds of directories, the display can feel a bit more "messy" (disorganized) than a regular grid, and it uses more screen space. It can also be harder to use for people that are used to different systems, but this is only a matter of learning it. If the flaws of my implementation were removed I believe this system would be better suited for some tasks than the usual 2D icon system, but worse for other tasks. It is unclear whether the visualization helps locating objects by keeping positions constant, and non-regular, as users sometimes lost themselves when rotating the view. If more visual cues were added, users might not lose themselves, and if users were more used to the system they might learn how to use it more efficiently.

The navigation system used worked quite well. I believe after using it for a while it would be about as fast as other systems, maybe faster if mouse and keyboard commands were carefully tuned. Advantages are that zooming into directories and out of them is more interactive, and provides better feedback of what is happening, and maybe that the mouse is used directly to move the view, instead of using small scroll bars that can take time to locate using the mouse. (According to the well-known "Fitts law", the smaller the target and the further away, the longer it takes to find with the mouse.)

8.2 How important is fast visual navigation?

Much emphasis in this project has been on being able to find file system objects quickly, as this was recognized as a common task. How important it is doing this quickly clearly depends on how much time is spent on file management

tasks, and I believe most people do not spend a large amount of time on this, so it could be argued that other issues are more important, like how many different tasks can be accomplished in the program, and like an having an interface that is easy to understand. Of course, there need not be a conflict between a featureful and easy to understand interface and one that is efficient when learned.

Being able to quickly navigate a hierarchy and finding objects in it is surely good, but an ability to place commonly used objects "close by" (on the desktop or panel), using shortcuts and well-ordered hierarchies can probably gain more. However, it is often seen that people do not organize their files much at all, and having navigation be fast reduces the need to spend time organizing.

It can also be argued that keyboard navigation usually is faster than visual navigation in these programs, and that keyboard navigation can be done with the same key strokes in traditional 2D icon based file system navigators and in their 3D counterpart. Keyboard navigation works well when the name of a file is known, as you only have to type the start of this name, and the computer does all the searching and then highlights the file to give feedback and selects it so it can be used. In a way, this is more of a search method than a browse method, but works well in a browser to enable local searching within a directory. When the name is not known, visual browsing of one kind or another is necessary.

8.3 Would a 3D interface be better than a 2D counterpart?

A highly graphical file system browser, whether 3D or not, could provide more graphical cues than is usually done today, like halos around recently modified files to make them easy to find, red-tinted icons on files with restricted permissions, or cleverly designed icons that are adjusted to what the human visual system easily differentiates between, just to throw out a couple of ideas.

I believe 3D graphics can be used to good advantage to make a system more interactive, regardless of whether objects are placed in 3D or not, and I believe zooming is a good thing. Zooming allows overviews, and detailed views in a natural way and it gives direct visual feedback when entering or exiting directories. (This all requires a fast-enough computer, of course, so that all motion is smooth.)

I am not as sure of 3D layouts. The one I ended up implementing works well in some cases and not so well in others. (See section 6.7 for some cases where it did not work well.) There are many possibilities in how to create

layouts, and some are almost bound to be good, as long as they can be easily navigated through and easily be gotten an overview of.

To take an example of a non-virtual 3D object space I am quite used to: my room. It has bookshelves, drawers and a desk with the objects I currently use (or have not bothered cleaning up). Most objects are not occluded. Books on bookshelves, for example, are almost always placed against the front of the shelf.

Visualizing a file system like a room might or might not work well. Leaning too heavliy on metaphors can definitely "cripple the interface with irrelevant limitations and blind designers to new paradigms more appropriate for a computer-based application" [8]. When used in an appropriate way, though, metaphors can be good, and a visualization of a file system only needs to show what files are available and allow users to select them. A room metaphor might work well, if it is not an just a cute graphical shell, but a carefully designed and implemented system, that enables users to find files easily and to open or manipulate them.

A problem with any 3D system, though, is that the screen and mouse are 2D devices. They can display and manipulate 3D objects, but doing this is more complex than doing the same things with 2D objects. Moving an object in 3D space with a mouse, for example, is necessarily more complex than moving an icon on a 2D screen. There may very well be advantages to 3D that outweigh this disadvantage, or other interaction devices that are more suitable to 3D interaction.

Another problem with a system like this is screen resolution. If you ask me for a book I know on one of the bookshelves in this room, I can probably locate it within seconds. To do the same on a computer screen, I would have to rotate the view to the right shelf just like I would have to rotate my head in the real world. Then I would have to be able to recognize the book on its coloring and text, and a book among a hundred others on a shelf a couple of meters away will be hard to see on a computer screen because of the limited resolution and small screen. One probably has to move closer to be able to read the text.

2D icon or list views have a big advantage when resolution is limited. They always display the text in an appropriate size for reading, and they display everything in a compact way.

8.4 Human-like interaction

Something that could make a large difference in usability is to be able to communicate with the computer or agents in it as we communicate with humans. Really intelligent agents like Colin in "Mona Lisa Overdrive" [9] by William

Gibson would revolutionarize the user interface, but is of course impossible to implement today. It would require speech recognition and generation, language understanding, and artificial intelligence. Unfortunately, no-one has been able to produce good enough speech recognition, as far as I know. Systems like the "August" spoken dialogue system [10] show that this may soon be possible. Artificial intelligence is even harder, but full AI would not be required for simpler agents.

8.5 Different views for different tasks

One conclusion I have drawn in this project is that different ways of visualizing a file system can be good for different purposes. Both an "icon view" and a "list view" are usually provided, and it would surely be good to also have a "tree-map view", a "cone tree view", the view I have implemented and other views also available.

The problem with having different views is that it takes time and a conscious thought to switch. Thus, it is important that there is a view that is good in most cases. For other cases, the view can be switched to a more suitable one.

8.6 Future work

What can be done to continue research from what I have done?

The most interesting thing that can be done, in my opinion, is to search for better ways to place file system objects in space, so that they are easy to find and work with.

There is also related research on other ways of storing, arranging or interacting with objects than hierarchically in directories. GNOME Storage and Microsoft WinFS are two projects that build more on searching than browsing. More advanced meta-data use would be a good thing both for searching and visualizing objects. The World Wide Web Consortium has projects related to doing this for the web, which might be usable for locally stored files too.

Lastly, it may be interesting to find a better algorithm for pushing objects apart when they end up in the same place. The algorithm I ended up using works, but its result is often stretched a bit vertically, and could theoretically be stretched too much.

References

[1] Benjamin B. Bederson. Photomesa: Quantum treemaps and bubblemaps for a zoomable image browser. In *ACM Symposium on User Interface Software and Technology (UIST 2001)*, pages 71–81, 2001.

[2] Thomas Bladh. *StepTree: A File System Visualizer*. Master's thesis, Blekinge Institute of Technology, August 2002.

[3] Garold J. Borse. *Numerical Methods with MATLAB: a resource for scientists and engineers*, chapter 10, pages 242–249. PWS Publishing Company, 1997.

[4] Mark Bruls, Kees Huizing, and Jarke J. van Wijk. Squarified treemaps. *Proceedings of Joint Eurographics and IEEE TCVG Symposium on Visualization (TCVG 2000) IEEE Press*, pages 33–42, 2000.

[5] Matthew Chalmers. A linear iteration time layout algorithm for visualising high-dimensional data. In *Proceedings of the IEEE Visualization Conference*, October 1996.

[6] Michael Chen, S. Joy Mountford, and Abigail Sellen. A study in interactive 3-d rotation using 2-d control devices. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 121–129. ACM Press, 1988.

[7] Andy Cockburn and Bruce McKenzie. An evaluation of cone trees. In *People and Computers XV: Proceedings of the British Computer Society Conference on Human Computer Interaction*, 2000.

[8] Don Gentner and Jakob Nielson. The anti-Mac interface. *Communications of the ACM*, 39(8):70–82, August 1996.

[9] William Gibson. *Mona Lisa Overdrive*. Bantam Books, 1989.

[10] Joakim Gustafson, Nikolaj Lindberg, and Magnus Lundberg. The August spoken dialogue system. In *Proceedings of Eurospeech 99*, 1999.

[11] Stephanie Houde and Gitta Salomon. Working towards rich & flexible file representations. *Adjunct proceedings InterCHI'93*, pages 9–10, 1993.

[12] Takeo Igarashi and Ken Hinckley. Speed-dependent automatic zooming for browsing large documents. In *13th Annual Symposium on User Interface Software and Technology, ACM UIST'00*, pages 139–148, November 2000.

[13] Jock D. Mackinlay, Stuart K. Card, and George G. Robertson. Rapid controlled movement through a virtual 3D workspace. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, volume 24, pages 171–176, 1990.

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

[15] George Robertson, Jock D. Mackinlay, and Stuart K. Card. Cone trees: Animated 3d visualizations of hierarchical information. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'91)*, pages 189–194, April 1991.

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

[17] Bruce "TOG" Tognazzini. *Tog on Interface*. Addison-Wesley Publishing Company, Inc., 1993.

[18] Jarke J. van Wijk and Huub van de Wetering. Cushion treemaps: Visualization of hierarchical information. In *IEEE Symposium on Information Visualization (INFOVIS'99)*, pages 73–78, 1999.