Master Thesis
Computer Science
Thesis no: *MCS-2002-3*
August 2002

StepTree

A File System Visualizer

Thomas Bladh

Department of
Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Contact Information:

Author:	Thomas Bladh
Address:	Lindblomsvägen 97, 37233 Ronneby Sweden
E-mail:	tibsoft@swipnet.se

University advisor:
Professor Rune Gustavsson
Department of Software Engineering and Computer Science

Department of
Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet	: www.bth.se/ipd
Phone	: +46 457 38 50 00
Fax	: + 46 457 271 25

ABSTRACT

A 3D visualization tool for file system hierarchies is presented. The visualization technique used is based on the Tree-map / nested Venn diagram concept and is capable of visualizing metrics and attributes such as size, change and file type for thousands of nodes simultaneously. Size is visualized through node base area, change is visualized through the ghosting and hiding of unchanged nodes and file type is visualized through colors. Actions such as navigation and selection are performed exclusively in 3D. Finally a method for improving the visibility of nodes through the equalization of sibling nodes is proposed.

Keywords: tree-maps, hierarchies, visualization, file system.

CONTENTS

1 INTRODUCTION........ 1

1.1 POSSIBLE USER QUESTIONS........ 1
1.1.1 Where am I?........ 1
1.1.2 What type of file is this? 2
1.1.3 Where has my harddrive space gone? 2
1.1.4 Have important files been recently changed?........ 2
1.1.5 What are the structural and contentual relationships?........ 2
1.1.6 Summary 2
1.2 PROVIDING ANSWERS 3
1.2.1 Visualization methods for large Hierarchies 3
1.2.2 Focus and Context Techniques 3
1.2.3 Ghosting, Hiding and Grouping 3
1.2.4 Node Metrics 3
1.3 THE PROBLEM OF VANISHING NODES 4

2 METHODS........ 5

2.1 LITERATURE STUDY 5
2.2 EXPERIMENT 5

3 INFORMATION VISUALIZATION 7

3.1 NODE METRICS 7
3.2 VISUALIZING HIERARCHIES 8
3.3 TREE-MAPS 9

4 STEPTREE........ 12

4.1 EQUALIZATION 13
4.2 LAYOUT ALGORITHM 14
4.3 COLORS 15
4.4 VISUALIZING CHANGE 15
4.5 LABELS 16
4.6 NAVIGATION & SELECTION 16
4.7 TECHNICAL ISSUES........ 16

5 RESULTS........ 18

5.1 STEPTREE AND USER QUESTIONS 18
5.1.1 Where am I?........ 18
5.1.2 What type of file is this? 18
5.1.3 Where has my harddrive space gone? 18
5.1.4 Have important files been recently changed?........ 19
5.1.5 What are the structural and contentual relationships?........ 19
5.2 VANISHING NODES AND EQUALIZATION 20

6 DISCUSSION........ 22

6.1 ABOUT THE RESULTS 22
6.1.1 User Questions 22
6.1.2 The Significance of Equalization........ 23
6.2 REFLECTIONS........ 24
6.3 FUTURE WORK........ 24
6.4 SUMMARY 25

7 ACKNOWLEDGEMENTS........ 26

8 REFERENCES 27

9 LIST OF FIGURES 29

APPENDIX A - STEPTREE DOCUMENTATION
1. ABOUT STEPTREE 1
2. COPYRIGHT & DISCLAIMER 2
3. SYSTEM REQUIREMENTS 2
4. NAVIGATING THE SCENE 3
5. NAVIGATING THE TREE 3
6. VISUALIZATION SETTINGS 4
7. CONTROL SETTINGS 6
8. SOLVING PROBLEMS 6
APPENDIX B – RECURSEFILES FUNCTION

1 INTRODUCTION

This thesis is about the visualization of hierarchies. Hierarchies are evident all around us, our family structures, company management, governments and the chain reactions of nuclear fission; all are examples of more or less formal hierarchies. Hierarchies have a prominent role in computer science, evident for example in the hierarchical structuring of most programming languages, hierarchical file systems, the underlying AI of chess playing programs and decision support systems to name a few.

The focus of the thesis is the visualization of file system hierarchies. I was especially interested in the simultaneous visualization of the metrics size and change. It was for this reason that my attention was turned towards the Tree-map \ nested Venn-diagram type of visualizations. Tree-maps are known for their excellence at visualizing the size of nodes in file systems. Would it also be possible to extend the concept to cover other metrics, such as change, while still retaining that excellence? Will there be any penalties for doing this and. if so, can these be mitigated? The means by which I have tried to answer these questions is through the development of the experimental application StepTree.

A great deal of research is available which explores the possibilities of Tree-maps. Tree-maps can be used to visualize any hierarchical data structure but is most often used for the visualization of file systems. See [Shneiderman 1992], [Turo & Johnson 1992], [Bruls et al. 2000] and [van Wijk & van de Wetering 1999]. When visualizing file systems, size is often mapped to the base area of nodes in the visualization. This trait is also mirrored in two earlier 3D adaptations of the Tree-map / nested Venn diagram concept (similar to StepTree). These are:

FSV <http://fsv.sourceforge.net> (21 August 2002) by Daniel Richard G. and **VisFS** <http://www.heiko-schmidt.info/project/visfs/visfs_de.html> (21 August 2002) by Heiko Schmidt.

Upon review it became evident that all variations on the Tree-map theme (flat as well as three-dimensional) where size is mapped to base area seem to suffer from the same serious problem. The problem has to do with assuring that all nodes, even those without on-disk size are guaranteed at least some level of visibility. Assuring the visibility of all nodes is imperative if we wish to visualize other metrics besides size. This is the central problem that I propose to have solved in this thesis.

1.1 Possible User Questions

To highlight the more common problems when visualizing a file system hierarchy I will present a number of questions a user might ask about a file system while browsing through it.

1.1.1 Where am I?

This is a question that most file system browsers and even command line tools can answer today. It would seem important not to forget this when implementing a new file system visualization tool. One should try to answer not only the questions that new and innovative visualization concepts might enable us to answer but also the questions that people have been asking and will continue to ask in the future.

1.1.2 What type of file is this?

This is another basic question. The file type is essential for the user to determine which action should be taken in a given situation. Is this a zip archive or an executable? Which answers the question: Will I have to open this and extract the contents or can I just run it?

1.1.3 Where has my harddrive space gone?

It is easy to accidentally forget about large files or directories which should have been deleted. Most commonly used file system browsers do not visually represent size in any way. This means that finding the offending files would involve searching for them manually or with a search tool. Searching manually in a file system hierarchy of perhaps thousands of nodes is a very time consuming task. Search tools, although somewhat useful, are nevertheless an interruption of your browsing activity (you have to actively invoke them). Search tools are often good at finding large individual files but not at locating large directories.

1.1.4 Have important files been recently changed?

This question has to do with the insecurity caused by the limited insight into the often vast amounts of data residing on even modest systems today. How can you be sure that your system has not been tampered with in some way either by a malicious human intruder, virus, trojan or worm? It is very difficult, if not impossible, to answer this question with current types of file system browsers. There are specialized checksum-based detection tools but these will only give you lists of changed files from monitored directories without any form of overview.

1.1.5 What are the structural and contentual relationships?

Insight into these types of relationships between files and directories is essential for successful navigation of a file system hierarchy. Even today’s somewhat primitive file system browsers offer this insight to some degree. More advanced visualizations might help answer more specific questions such as: “Where are all my PDF files located?”, “Are they spread out or concentrated in one location?”.

1.1.6 Summary

Enabling the user to answer all of these questions quickly and reliably would be the ideal for any file system browser. The problem is that enabling the user to answer questions about harddrive space might make it harder to answer a question about change and the same goes for most of the combinations. This is especially true for Tree-maps. There is also a question of which of these actually needs to be answered and that depends on the type of user the visualization is geared towards. A systems administrator might be interested in harddrive space and changes to the file system whereas a normal user would perhaps be more interested in usability related questions such as “Where am I?” and “What type of file is this?”. An important part of this thesis work has been to develop a visualization application which can answer as many of them as possible, if not all, preferably at the same time.

1.2 Providing Answers

Creating a tool capable of answering the questions from section 1.1 could be aided by looking into the following areas.

1.2.1 Visualization methods for large Hierarchies

There are a number of visualization types aimed at large hierarchies. For example Cone Trees [Cockburn & McKenzie 2000], Tree-maps [Shneiderman 1992] and the circular SunBurst [Stasko et al. 2000]. Tree-maps and SunBurst are both well suited for answering questions about a single metric such as size but might need some rethinking to answer questions related to combinations of metrics.

1.2.2 Focus and Context Techniques

Focus and Context techniques are used for visualizations where you wish to be able to focus on a certain feature without loosing context. This could be thought of as zooming in on a particular city on a map and still see it in the context of the entire country. A well known technique in this category is fisheye views [Furnas 1981] where the analogy of a fisheye lens is used to describe the relation between the focus and the “big picture”. Focus and Context techniques can simply be said to help achieve overview. Focus and Context techniques would help answer questions about location as well as about structural and contentual relationships.

1.2.3 Ghosting, Hiding and Grouping

These are ways of handling the display of nodes deemed to be of lesser importance. To G*host a node* means to de-emphasize it (e.g. making it transparent), *Hiding* means to simply not display it and *Grouping* refers to combining nodes into super nodes that could represent entire sub-trees in the hierarchy [Herman et al. 2000]. Making a decision on which nodes to group, ghost or hide is in most cases, in one way or another, based on node metrics.

1.2.4 Node Metrics

A good definition of node metrics is given by [Herman et al. 2000] they define a node metric as:

“[…] a measure that is associated with a node in the graph.”

Node metrics could be used to help decide which nodes to emphasize based on features we are looking for in the hierarchy (such as change). If we were forced to show only a subset of the entire hierarchy we could use node metrics to decide which nodes of the hierarchy to select for display. Node metrics could be based on application independent factors such as distance from focus, number of children, etc. as well as application dependent factors such as the number of changed files in a directory or the time elapsed since a certain file was changed. It is important not only that you use node metrics but also how you apply them. As mentioned previously you can use node metrics to decide which nodes to *hide*, *ghost* or *group*. Other applications of node metrics are simple graphical characteristics such as color and size applied to the nodes.

We react much faster to purely visual characteristics such as color, size and shape than we do to symbols [Cobb 1997]. If we had not worked this way, activities dependent on reflexes such as driving a car or riding a bike would in all likelihood have been almost impossible for us to master. The visualization of node metrics is vital for success as the proper use of them could help us answer all of the questions from the previous section.

1.3 The Problem of Vanishing Nodes

The purpose of this thesis is, to recap, partly to create a visualization tool capable of answering as many of the questions from section 1.1 as possible. In addition I strongly felt that I had to solve a serious problem that I found with earlier Tree-map/Venn diagram type visualizations and to see if I could devise and implement a reasonable solution to it.

The problem I found sprang out of a difficulty in answering questions about size at the same time as other metrics with Tree-maps and it is to my knowledge found in all space filling visualizations based on the nested Venn diagram concept. It is described in [Turo & Johnson 1992] as the problem of nodes “dropping” out of the visualization. The phenomenon is caused by direct mapping between size on-disk and size in the visualization.

The problem in short is that a file without size will not be visible at all and a file with a very small size in relation to its parent will become virtually invisible. The nature of most file systems is such that some files will be tiny, perhaps only a few bytes, while others might be several hundred megabytes, or more, in size. When a direct mapping between on-disk size and size in the visualization is used this difference of several orders of magnitude makes it impossible to assure, or at least improve the odds, that small and empty files will be visible. The problem of answering questions about for example size and change at the same time should now be obvious for how do you visualize change to nodes in the file system if those nodes can hardly be seen by the user. A tentative solution proposed by Turo & Johnson [Turo & Johnson 1992] is that of assigning a minimum size for nodes but they acknowledge that this is not a solution without problems and they do not describe it in greater detail.

2 METHODS

Two principal research methods were used for this thesis, a literature study was conducted, and an experimental application was developed. The central part of the thesis is the creation of the experimental application. The literature study should be regarded as an introduction to visualization research as well as a foundation for many of the choices made during the design and development of the application.

2.1 Literature Study

The literature study was focused on research papers and other resources from the field of Information Visualization. Further focus has been placed on the visualization of hierarchies and in particular on the space filling approach to hierarchy visualization. Tree-maps, especially, are covered in greater detail in section 3.3. A greater area has of course been covered in preparation for and during the work but much that was deemed to be peripheral has been omitted from this study.

2.2 Experiment

After reading a great deal about information visualization and the multitude of visualization types conceptualized by innovative researchers all over the world my attention was finally caught by one type of visualization. The type in question was the two-dimensional space filling Tree-map concept [Shneiderman 1992]. This type did at first seem somewhat confusing and unsuited for achieving the kind of intuitive insight I was looking for. I soon understood however, as the creator undoubtedly already had, that beneath the superficial confusion was a visualization method of great potential. Because it is space filling it is very compact, affording effortless overview and using the available screen space well. Tree-maps used for the visualization of file systems are superior to anything else I have seen when it comes to communicating the size of files to the user. It is thus ideally suited to answering the third question posed in section 1.1:

"Where has my harddrive space gone?"

There are two major problems with flat Tree-maps however. The first problem is related to its potential at answering the fifth question from section 1.1:

"What are the structural and contentual relationships?"

Tree-maps are indeed excellent at visualizing aspects of content (i.e. size, file-type etc.) they are not however as well suited for the visualization of structural relationships. The reason for this is that, in this type of visualization, the leaf nodes of the tree tend to obscure the rest of the nodes. Even if they do not, as is the case with offset Tree-maps [Turo & Johnson 1992], the flat projection of the map will make mental separation of the structure into layers difficult.

The second problem, the problem of vanishing nodes from 1.3 has to do with the way traditional Tree-maps visualize size. The direct relationship between actual size (on-disk) and size in the visualization causes files with an actual size of zero as well as empty directories not to be shown at all. Even non-empty files can virtually disappear if their size in relation to the size of their sibling group is too small. Having small or empty files and directories disappear might not be a problem if all we are interested in visualizing is size but it needs to be solved if we want to visualize other metrics, such as change, reliably.

The experimental application was developed as a proof of concept for solutions to both problems described above. The basic premise was to create a Tree-map type of visualization that solved these problems.

A decision to create a 3D version of this basic visualization type was made and was motivated by the failure of the flat implementations to address the the problem of visualizing structural relationships. It was thought that adding a third dimension would help visualize structural relationships by allowing for the separation of the map into separate layers. One problem of extending a flat visualization into three dimensions is that you get what is essentially a new infinity of options.

Development was carried out in Borland Delphi (Object Pascal). The reasons for choosing Delphi were many. Firstly Object Pascal is, as the name suggests, an Object Oriented language. For example creating a representation of the file system tree in memory is made easy when files, directories and drives can be handled as objects. Details such as the creation of windows and handling of events are abstracted from the operating system making porting a possibility (There is Object Pascal for Linux in the form of the development environment Kylix). Finally the development of a 3D graphics application benefits greatly from the fact that headers for the popular 3D API OpenGL are included and ready to use in the Delphi distribution.

3 INFORMATION VISUALIZATION

This thesis is concerned with the subject of "Information Visualization" and thus most of the literature reviewed falls into this category. Information Visualization is an important field within computer science as it connects with almost all of the sub fields. Regardless of whether you are working with Artificial Intelligence, Databases, Decision Support Systems etc., systems will inevitably generate data. This data will sooner or later be visualized in one way or another. With ever increasing amounts of data residing in file systems, databases and on networks we have a distinct problem. How do we make sense of it all?

The definition of "Information Visualization" I find most fitting is the one given by Card et al. [Card et al. 1999]:

> *"The use of computer-supported, interactive, visual representations of abstract data to amplify cognition."*

The relevant area for this thesis is the visualization of structured information. Herman et al. [Herman et al. 2000] offer a good definition of structured data in the form of a question: *"is there an inherent relation among the data elements to be visualized?"*. If the answer is *ye*s then the data is structured, if it is *no* then it is unstructured. A file system is structured in that it is already (before visualization) ordered as a hierarchy.

Much research into the visualization of the opposite i.e. unstructured data is focused on adding structure to the unstructured data "after the fact". Structure is added by for example grouping information with similar semantic contents together. A structured hierarchical file system can be thought of as originally unstructured data, data that has been put into a structure (the file system hierarchy) by some sort of human intervention, either by the user, a developer or an application (written by the developer). It is this structure that is subsequently visualized. To directly visualize unstructured information on the other hand, often requires that we add structure to the information for it to make any sense. Structure helps us understand connections between individual elements.

Central problems relating to the interaction with large hierarchies are that you tend to loose the sense of overview and have problems finding the features you are looking for. These are nicely summarized by Herman et al. [Herman et al. 2000] as "Where am I?" and "Where is the file that I'm looking for?".

3.1 Node Metrics

The term "node metric" is defined by Herman et al. [Herman et al. 2000] as: *"[…] a measure that is associated with a node in a graph"*. I will use the term to refer to this definition throughout the thesis. Node metrics can be either content-based or structure-based. A content based metric for a file in a file system hierarchy could for example be its size, type or the time of the last access. Structure-based metrics could be the degree of a node (the number of edges connected to it) or the distance from focus. Metrics can also be combined to form new metrics. Node metrics are used to influence the display of the structure either by altering the structural layout or by changes in size and color of nodes. The ways you can use node metrics is essentially unlimited. You could create a system where the icons representing directories are enlarged or shrunk depending on how many files of a certain type they contain. Another way could be to display files in brighter colors depending on how well they are expected to compress. This type of implementation could help identify files which “are not what they seem”, for example a supposed text file which is actually an executable.

3.2 Visualizing Hierarchies

Hierarchies can be visualized in a great number of ways. The most common visualization method for file system hierarchies is the node and indentation variant of the node and link method used in for example the Microsoft Explorer and Nautilus (Linux/Gnome) browsers. In this type of implementation only those areas manually expanded are visible at any one time. As nodes are expanded downwards a full expansion would require a great deal of scrolling and be practically impossible to overview. The screenshot below is of my preliminary reference directory (PDF files and prepared reference texts) and shows Microsoft Explorer in action. There are two panes, one for the tree view (left) and one for the simpler directory view (right).

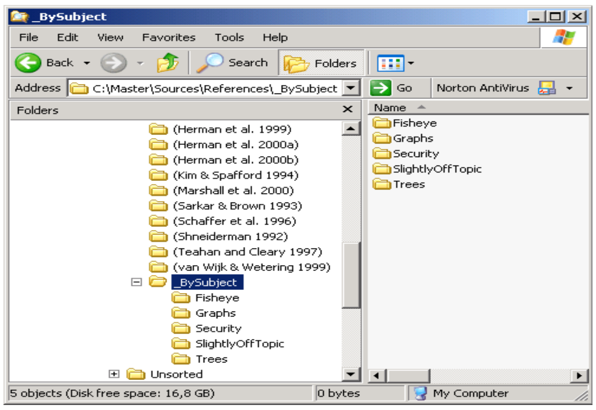


Figure 3.1: Screenshot of tree view (left pane) in Microsoft Explorer.

Another common type of visualization is the node and link type of diagram where each node represents an individual item in the hierarchy and lines the structural relation between them. This method is popular for displaying hierarchies in scientific disciplines such as mathematics, physics and computer science. It has not however proved as popular in commercial applications for the display of file system hierarchies.

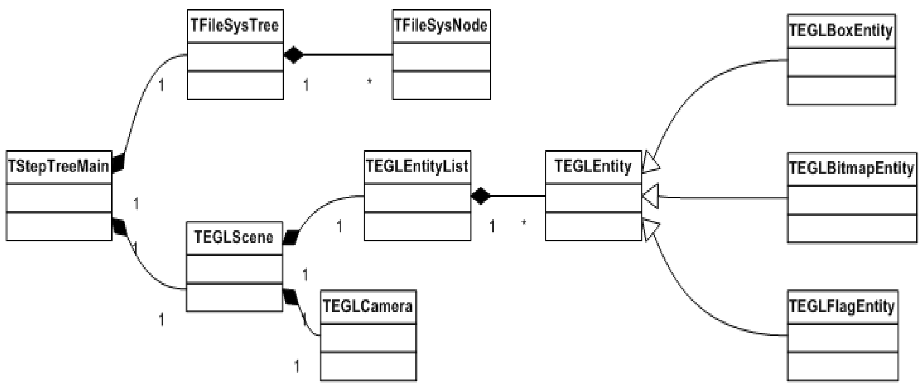


Figure 3.2: An example of a node and link diagram depicting the class hierarchy of the StepTree application.

A problem with both the node and indentation and the node and link visualization methods is that they make inefficient use of the available screen space as you can see in figures 3.1 and 3.2 there is a lot of white space in these renditions. This might not be a problem for a handful of objects but quickly becomes one when thousands of objects needs to be visualized as the whole structure will no longer fit on the screen.

A better method in this regard is the space filling approach where all of the screen space or at least a large portion of it is used, allowing for a greater number of nodes to be displayed at once. In the space filling approach a set space (for example the entire screen surface) represents the entire hierarchy. Examples of space filling visualizations include the rectangular Tree-map [Shneiderman 1992] and the circular SunBurst [Stasko et al. 2000]. An important thing to recognize is that a space filling hierarchical visualization need not necessarily be two dimensional it can conceivably be anywhere from 1 to n dimensional (whether or not this is practically feasible is a different matter).

The rest of this subject overview will be centered on the space filling approach. There are a number of reasons for this which I will detail below:

- At an early stage I decided on using OpenGL for the experimental visualization I was developing. I wanted the ability to zoom in and out of the visualization as well as the ability to move freely about the rendered scene, something which would have been quite time consuming to implement without the use of a 3D API.

- To keep complexity at a minimum, both in development and in use, the rectilinear geometry of the space filling Tree-map approach seemed ideal. With an OpenGL implementation it is also important, for performance reasons, to limit the number of polygons per node. This is the principal reason behind choosing an approach similar to that used in Tree-map visualizations instead of the circular SunBurst.

- Tree-maps are compact, affording greater overview than is possible with node and link diagrams.

3.3 Tree-maps

Tree-maps are essentially nested Venn diagrams and are often used for the visualization of file systems where size is mapped to area. Tree-maps visualize hierarchies through enclosure unlike the node and link type of visualizer which visualizes hierarchies through connections. Using the two dimensional space filling approach is a clever and simple way of displaying a hierarchy as it allows the contents of an entire structure (or a great deal of it) to be displayed at once. Tree-maps are most often used for the visualization of file systems, where the area occupied by a particular node (in relation to the whole) is proportional to the size of the file it represents. The original Tree-map as described by Shneiderman [Shneiderman 1992] had a few problems however. One problem was that the somewhat simplistic area division method originally used had a tendency to generate very thin vertical or horizontal slices for small files in otherwise sizeable directories, as is evident in Figure 3.3 below. A second problem was that even though it affords a good overview of the relative sizes of different files it is harder to grasp the structural relationships between files and directories on different levels in the hierarchy. One cause for this is that leaf nodes (i.e. files) obscure other nodes (i.e. directories). Another contributing cause is that there is little or no sense of depth in a flat Tree-map. When a hierarchy is projected as a traditional tree (as is the case in most browsers) you get an immediate sense of the level a node is on, this is not the case with the basic Tree-map.

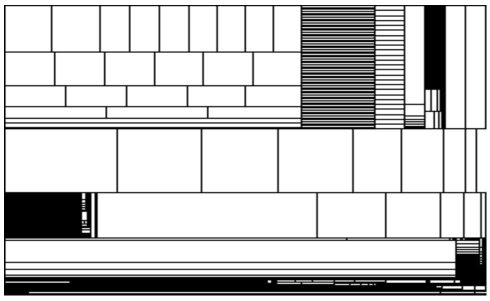


Figure 3.3: A basic Tree-map.

The good news is that a number of improvements to the original Tree-map concept have been published and that much interesting research is going on at the moment. The two most notable improvements are Cushion Tree-maps [van Wijk & van de Wetering 1999] and Squarified TreeMaps [Bruls et al. 2000]. Cushion Tree-maps are essentially a shaded version of the original Tree-map visualization. In my opinion the shading helps you see the individual cushions as objects rather than as simply an area. You get a better feel for where the edges are and can more easily see which files belong in which directory.

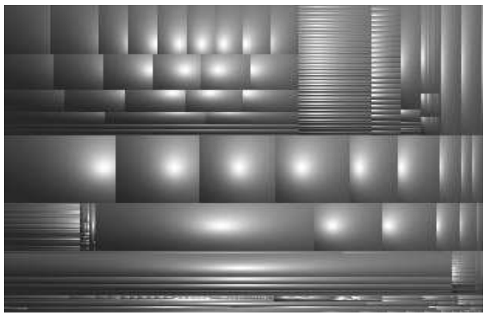


Figure 3.4: Cushion Tree-map.

Squarified Tree-maps use a different subdivision algorithm which is more complex in that it not only generates rectangles with a certain area but also makes them as square as possible (aspect ratio as near 1 as possible). The result is the virtual elimination of the extremely long and thin rectangles evident in figures 3.3 and 3.4.

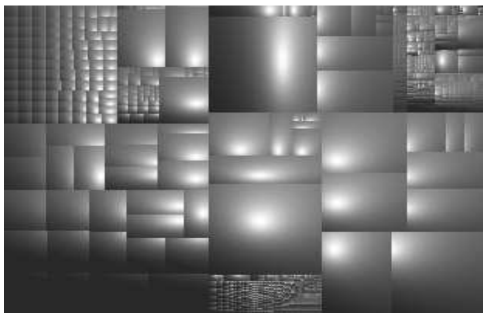


Figure 3.5: Squarified Cushion Tree-map.

Tree-maps have been used in areas other than the visualization of file system hierarchies. Alternative applications include photo browsing with PhotoMesa

[Bederson 2001], stock portfolio visualization [Jungmeister & Turo 1992] as well as for viewing information on tennis matches [Jin & Banks 1997].

4 STEPTREE

StepTree is an experimental 3D visualization tool for file system hierarchies. It is, in its current incarnation, capable of visualizing the following metrics and attributes:

- **Size**
- **Change**
- **File Type**

It was developed as an experimental and exploratory investigation into the possibilities of using a variation in 3D on the Tree-map / nested Venn diagram theme to visualize effectively not only size but also other metrics. By "effectively" is meant that the visualization of size should not, to a significant extent, adversely affect the visualization of other metrics (see section 2.2 for a more in-depth description of this problem). With this stated purpose in mind I have focused on the tree layout process while letting other, less central aspects such as navigation and selection, remain somewhat primitive.

For reasons detailed in the method and literature review sections (2 and 3) I focused on the Tree-map / nested Venn diagram visualization type. As I described in the subject overview, the rectilinear geometry of Tree-maps is easier to work with than the more complex layouts such as Cone Trees or Hyperbolic Trees. This is especially true because I am using the OpenGL API to render the visualization in three dimensions. The layout of traditional node and link diagrams in three dimensions would be quite a project in itself and would perhaps have been too time consuming for this project. The choice of the Tree-map visualization is not based solely on ease of development however. Tree-maps are very compact and are likely (I believe) to be easier for users to navigate and interact with.

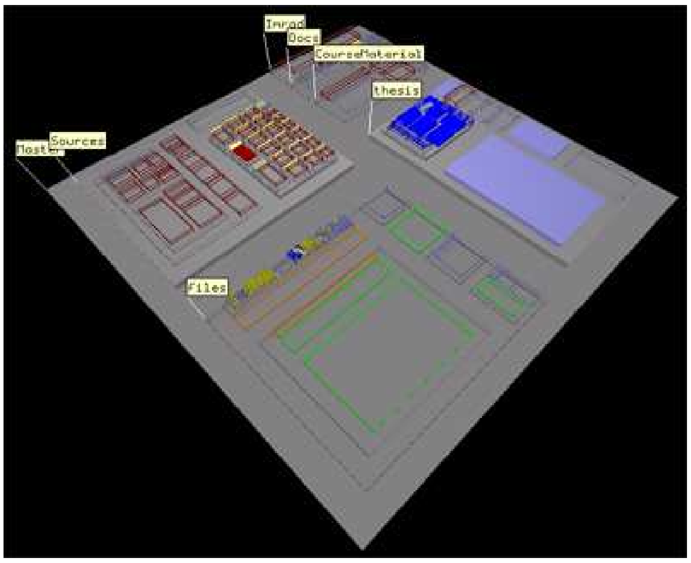


Figure 4.1: StepTree Screenshot.

Figure 4.1 above shows a sample directory with nodes changed within the last day drawn solid and all other nodes drawn as wire frames. Note that directories (grey) containing solid files are also drawn solid. File type is associated with color. In this view we have pdf files (dark red), bmp (blue), txt (yellow) and exe (green) to name a few.

4.1 Equalization

Equalization as I have implemented it in StepTree is a relatively simple method of redistributing area from large nodes to smaller ones within a group of siblings without changing the total area of the group. The function below is applied to all children in a sibling group, equalizing them to a degree depending on the equalization constant used. The same equalization constant is used for the whole tree.

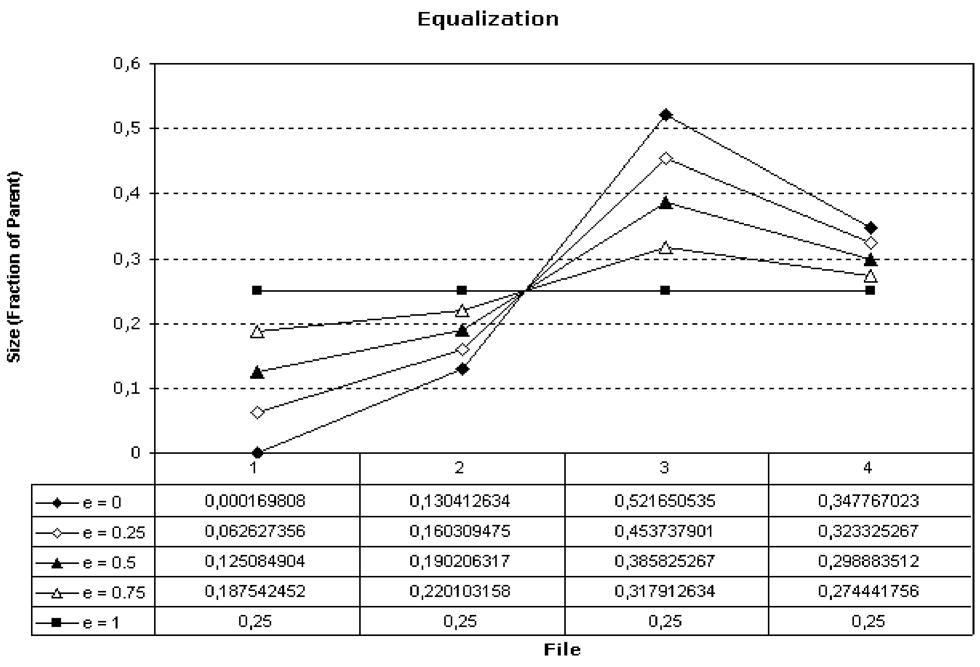
Equalization function:

$$v_{eq} = v + (a - v) * e$$

Where:

v	=	Initial area of the child expressed as a fraction of the area of the parent.
a	=	Average child area for the sibling group.
e	=	Equalization constant ($0 \le e \le 1$).
v_{eq}	=	Equalized area fraction

The initial value of v is the fraction of the parents size this particular child represents on-disk. The equalized output (v_{eq}) is the fraction of the parent's area the child will represent in the visualization. Setting the equalization constant to 1 will result in a completely equalized sibling group where all nodes are given the same fraction of the parent area (the average). Setting the constant to zero will result in v_{eq} being equal to v which of course means no equalization.



	1	2	3	4
◆ e = 0	0,000169808	0,130412634	0,521650535	0,347767023
◇ e = 0.25	0,062627356	0,160309475	0,453737901	0,323325267
▲ e = 0.5	0,125084904	0,190206317	0,385825267	0,298883512
△ e = 0.75	0,187542452	0,220103158	0,317912634	0,274441756
■ e = 1	0,25	0,25	0,25	0,25

Figure 4.2: A diagram showing the gradual equalization of four files.

Figure 4.2 above shows the gradual equalization of four files making up a sibling group. The initial values in the “e = 0” series shows the distribution of size as it is on-disk. The values in the “e = 0.25” series have been equalized 25 percent and so on. Each step brings the values closer to the average sibling size.

Even if this function is simple to the point of being trivial it serves the important purpose of making this type of hierarchical visualization, where size is mapped to area, viable for the visualization of other metrics besides size. Files without size as well as empty directory structures would not be visible at all without equalization or a similar redistribution function. Files with very small sizes relative to the size of their parent directory would also become all but invisible. This disappearance of files would be unacceptable in a case where you wish to visualize change as well as size (as is the case with StepTree). Making things visible is after all what visualization is all about and visualizing changes to small files or the creation (creation can be considered change) of new empty files or directories might be just as important as changes to larger files and directories.

Equalization is a compromise as it distorts the visualization to a degree depending on the equalization constant used (available as a slider setting in the settings window of the StepTree application). Two files of equal size on disk may for example appear to be of different size in the visualization if they have different parent directories.

Note that in my implementation the equalization step is followed by an atrophy step (described in 4.2) where the area used by children is shrunk by a set fraction in relation to the parent to expose underlying structure. The equalization and atrophy steps are listed in the settings window as distortions and can both be completely disabled.

4.2 Layout Algorithm

The graph is laid out by a recursive algorithm implemented as the function *RecurseFiles* (see Appendix B). The first call to this function specifies the root node of the visualized subset of the file system (c:, d:, c:\temp etc.) and also the coordinates and dimensions of the box that should represent the whole of this subset. RecurseFiles calls itself once for every child, placing the child nodes as dictated by a layout algorithm, which I will describe below.

The layout I used was inspired by the algorithm for squarification described by Bruls et al. [Bruls et al. 2000] although the one I settled on is somewhat simpler. The first examples of the Tree-map visualization [Shneiderman 1992] used an even simpler algorithm for the subdivision of an area into smaller sub areas. This early algorithm suffered from the problem that nodes tended to be drawn with wildly differing aspect ratios, some would be fairly square while others would be almost absurdly elongated. It is for example much harder to compare the size of objects if they vary in shape than it is if they are roughly the same shape. Uniformly square nodes could also, arguably, be considered more aesthetically pleasing. This is why using a squarification algorithm for the layout is desirable. Optimal squarification however is not computationally feasible [Bruls et al. 2000] this is why a solution that is merely “good enough” is used instead.

The first step in our Step-tree layout is the creation of a box with suitable dimensions and placement in the 3D space to serve as a foundation for our tree. This box will represent the root of the visualized subset of the file system. I decided on making it 100 units wide, 100 units deep and 2 units high. The decision was based purely on aesthetics.

When laying out the children of the root node I begin by sorting them by size in ascending order. All children are then equalized as described in the previous section. The actual placing of nodes on top of the parent is based on the notion of a line. Nodes on the same line all have the same depth and a line will always be as wide as the parent node is wide. The first child starts a new line. This initial one-node line is, of course, as wide as the parent is wide and as deep as it needs to be for it to use up its fraction of the parents area. We now add the next child (if there is one) obeying the same rules and see if the average aspect of the whole line improves (gets closer to 1) or worsens as a result of the addition. If it worsens we go back one step and keep the previous line, creating and placing the nodes it contains and then start a new line with the rejected child. If it improves we simply continue to add children to the new line until it finally worsens or we reach the last child. Some might wonder about the reason behind my use of an ascending order sort. The reason is simply to ensure that as many as possible of the small nodes achieve a good level of "squareness" by placing these first. The larger nodes can afford to run low on depth and might be somewhat elongated without also becoming too thin. Small nodes often became virtually invisible due to extreme elongation when a descending order sort was used.

Before *RecurseFiles* calls itself with the coordinates generated by the layout algorithm for a particular child these coordinates are sent through the atrophy function. The atrophy function scales the child down in size by a predefined fraction while still keeping it centered on the same point on top of its parent. This is especially important as it exposes the structure of the tree, creating the step pyramid effect.

The process described above is applied recursively for every node containing child nodes (invariably directories).

4.3 Colors

Color is applied to nodes depending on their file type. The relationship between file types and colors is established through an editor built in to the visualization tab of the settings window. There are two special case "file types" however. The first is the color assigned to directories (by default grey) and the second is the default color for files of types not yet assigned their own color. These two special color mappings can only be edited, not deleted.

Assigning different colors to different types of nodes can for example help you see with a glance which type of file is dominant in the visualized subset. Files of similar type can also be assigned similar colors. In the default color map for StepTree, for example, files with an extension indicating raw text contents (ASCII, ANSI) are all colored in different shades of a yellow-white base color.

4.4 Visualizing Change

Change is as mentioned previously tracked by ghosting or hiding unchanged nodes. A node is considered changed at the date of its last modification. If this date, as reported by the system, turns out to be earlier than the creation date then the date of creation is used instead.

The date of change determines whether a node should be drawn as a solid. If this date falls within the timeframe specified in the Visualization tab of the Settings window (default is 14 days) then the node is considered solid. Whether to use ghosting or hiding is also determined through settings as described in the StepTree documentation (Appendix A). Ghosting is accomplished by drawing wire frames instead of a solid boxes for ghosted nodes. Hiding is accomplished by simply omitting unchanged nodes from the drawing process.

Hiding unchanged nodes can be extremely effective when trying to spot changes to the file system as these are effectively singled out. Changes are also propagated down in the hierarchy. The date of the last change assigned to a directory is determined by the most recent change date among all of its descendants. What this means is that you don’t need to show the whole structure to know that a particular branch of the hierarchy has changed. If a directory is solid it will contain a file or directory that has either been modified within the specified timeframe or been created within it.

4.5 Labels

Labels in StepTree are implemented in the form of flags. Permanent labels are only assigned to directories on the first two levels (0 and 1) of the visualized subset (where 0 is the root of the subset). Files are not assigned permanent labels. Information about individual files is displayed by pressing the right mouse button while at the same time pressing CTRL with the mouse pointer over the box representing the node in question. Labeling of nodes could of course have been accomplished in a number of different ways. The main problem of labeling nodes of a hierarchy visualization where thousands may be visible at the same time is that you might end up with a veritable forest of labels where very few individual labels would be intelligible. One could imagine solutions where the appearance of labels is triggered by proximity to the observer, the cursor or any number of variations on that theme.

4.6 Navigation & Selection

Navigation through the 3D scene rendered by StepTree is accomplished through tried and true mouse and keyboard interaction. Due to time constraints the navigation was not a main priority. The requirement here was only that it worked and worked robustly. The navigation method is essentially a free floating version of the type of navigation used in most of the current 3D games. This to ensure that there will be at least one major group of users for which the use of StepTree will be somewhat intuitive. More in-depth descriptions of how navigation and selection works can be found in the included StepTree documentation (Appendix A).

4.7 Technical Issues

Designing a graphics intensive application such as StepTree does of course require that one take into consideration more than just the theoretical. Even though theory does not require a limit to the number of nodes the visualization can support the hardware most often does.

The implementation of the selection mechanism, for example, imposes definite limits on how many nodes can be drawn at once. A user can with the current method select one of 32766 unique nodes. StepTree has been implemented in such a way that information is propagated up towards the root. It is therefore not necessary to visualize the whole tree in order to see if something has changed. It seems that 32766 nodes should be quite enough for most conceivable purposes. Another drawback to visualizing a great number of nodes is that it seems to confuse the user somewhat, making navigation more difficult.

Another issue is that you do not wish to alienate people by having the application hang just because he or she happens to choose extreme settings. I have solved this by making it possible to abort lengthy operations by pressing ESC. An additional security measure was the introduction of a node limit value configurable through the Visualization tab of the Settings window. The limit is initially set to 5000 nodes which is a number most reasonably modern computers can handle.

Nodes are in this implementation drawn as simple boxes. The only box for which the bottom is rendered is the box representing the root of the visualized subset. The reason for this is that all other boxes rest on another box. This means that n visualized nodes equals $(n * 5) + 1$ polygons (quads). This is just one of many ways in which performance is taken into consideration.

Any visualization is limited by the available hardware. The original Tree-map concept is mainly limited by screen resolution and color depth. With virtually every graphics card sold today being able to handle resolutions of at least 1600x1200 pixels at 32 bpp, this is not something to worry much about anymore. The main issue when rendering a Tree-map in 3D lies in the assumption that you will want to rotate and move through the 3D scene in real time. It is this real time interaction that requires computing power, especially when you have thousands of objects to handle. To simply render a Tree-map type visualization once using a 3D API (e.g. OpenGL) would be not be much more demanding than rendering one using conventional “flat” graphics. It is the dynamic nature of most 3D implementations, including this one, that taxes system resources.

5 RESULTS

The first aim of this thesis work was to create a 3D visualization, based on the Tree-map / nested Venn diagram concepts, capable of answering the questions proposed in section 1.1. The second aim was to overcome the problem of vanishing nodes (see 1.3) which is found in current implementations of this type of visualization, both flat and three dimensional. In this chapter I will show how StepTree answers the former and solves the latter.

5.1 StepTree and User Questions

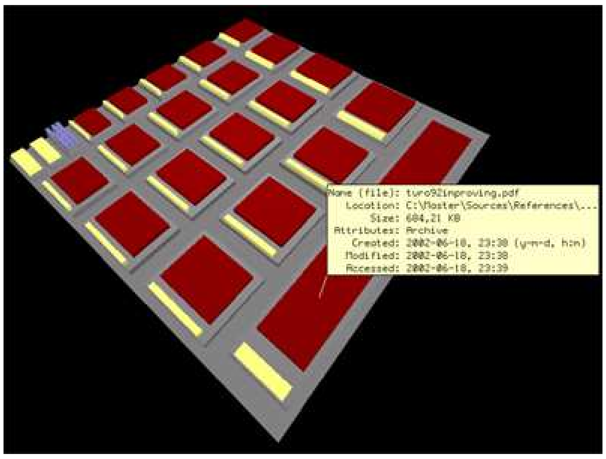
In this section I will list the aforementioned questions and then I will show how StepTree can help provide answers to them.

5.1.1 Where am I?

This question is answered in StepTree by somewhat primitive means. The path to the current location is displayed as the caption of the main window. This is the question where there is the most room for improvement.

5.1.2 What type of file is this?

There are two ways of answering this question in StepTree. First, StepTree uses different colors for different file types. Once you learn the color scheme (or create your own), learning the type of a file from simply looking at it is quite easy (see Figure 5.1 below). The second way is by displaying detailed information about a node (see 4.5 and Appendix A).



```
Name (file): turo92improving.pdf
   Location: C:\Master\Sources\References\...
       Size: 684,21 KB
 Attributes: Archive
    Created: 2002-06-18, 23:38 (y-m-d, h:m)
   Modified: 2002-06-18, 23:38
   Accessed: 2002-06-18, 23:39
```

Figure 5.1: *Screenshot of a directory containing pdf files (dark red) and text files (light yellow). One of the files also has a detailed information label.*

5.1.3 Where has my harddrive space gone?

StepTree maps the size of files on-disk to the base area of the node in the visualization. Visualizing just two levels of the tree will show you which branch is the largest and you can navigate further from there. Increasing the number of visualized levels will show you in greater detail the distribution of size further into each branch. Figure 5.2 below shows two screenshots of the same directory clearly illustrating this.

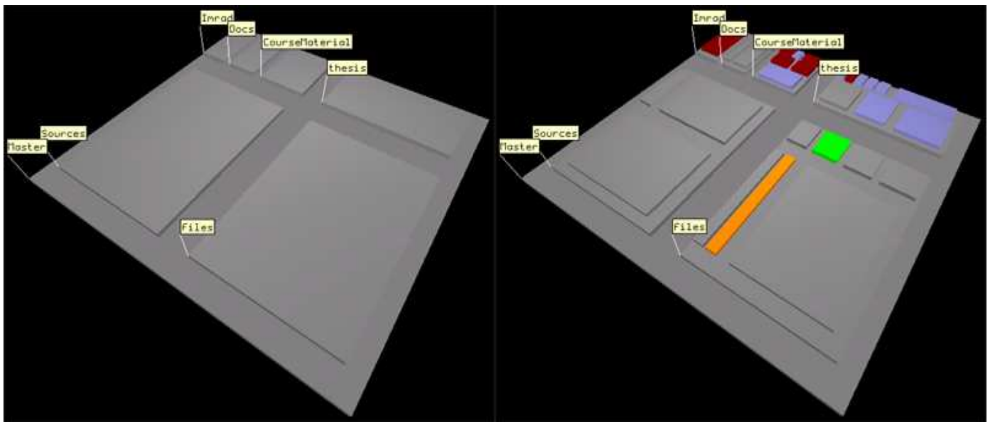


Figure 5.2: Two screenshots of the same directory taken in StepTree. The lef t screenshot shows 2 levels and the right 3.

5.1.4 Have important files been recently changed?

In the StepTree visualization tool, change is linked to solidity as you can clearly see in Figure 5.3 below. The solid node in question had been changed within the specified time period, which in this case was set to one day before the present. All non-solid (and thus less important) nodes can be either ghosted or hidden depending on the current setting. Ghosted nodes are drawn as wire frames while hidden nodes are not drawn at all (obviously).

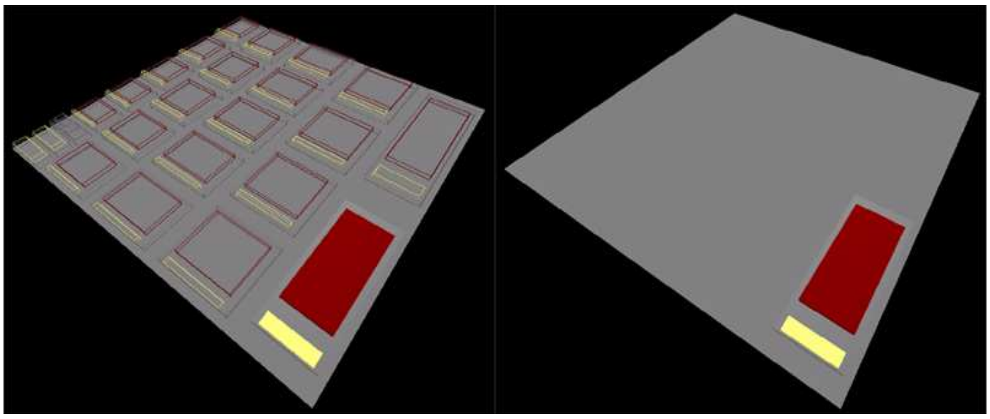


Figure 5.3: Visualizing change with solidity. In the left screenshot non-solid nodes are ghosted and in the right they are hidden.

5.1.5 What are the structural and contentual relationships?

Answering this question is dependent on having enough of the internal structure exposed. In StepTree the exposure of internal structure is controlled by the “Atrophy” value. An atrophy of 20 percent for example results in every level of the hierarchy taking up 20 percent less space than the previous. The higher the atrophy the more area of internal nodes is exposed. Setting it too low will result in a Tree-map like birds eye view where the leaf nodes cover the internal nodes completely, which makes selections and getting a good structural overview much more difficult. Setting it too high on the other hand will result in levels shrinking too rapidly making nodes further down in the hierarchy difficult to see. The range of possible values for atrophy in StepTree is 0 to 50 percent.

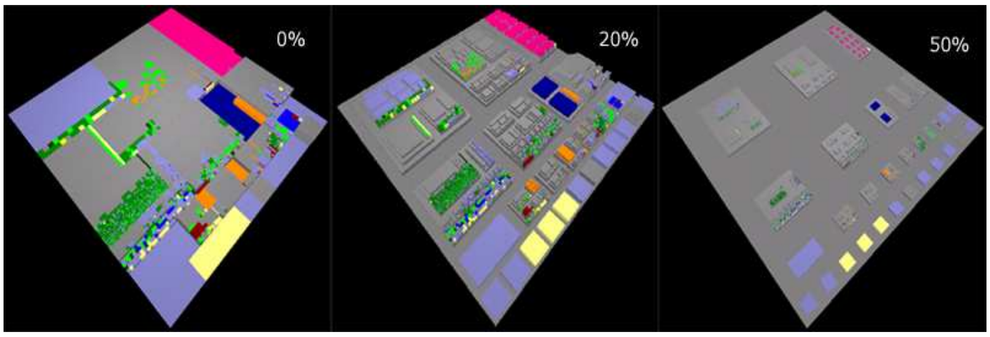


Figure 5.4: Views of the same base directory with different atrophy values.

5.2 Vanishing Nodes and Equalization

An important problem I set out to solve (see 1.3) was that of making sure that small nodes or nodes without size do not vanish. As previously mentioned, nodes (files or directories) that have no on-disk size may fall out of the visualization; they will not be visible due to the use of a direct mapping of on-disk size to area. Zero bytes equal zero area. Very small files or directories may also suffer virtually the same fate due to the often great differences in size (several orders of magnitude); they will simply be drowned out by their neighbors. The solution I found was that of equalizing sibling nodes, a process that is described in section 4.1. In Figure 5.5 below it is shown visually what the positive and negative effects of equalization are.

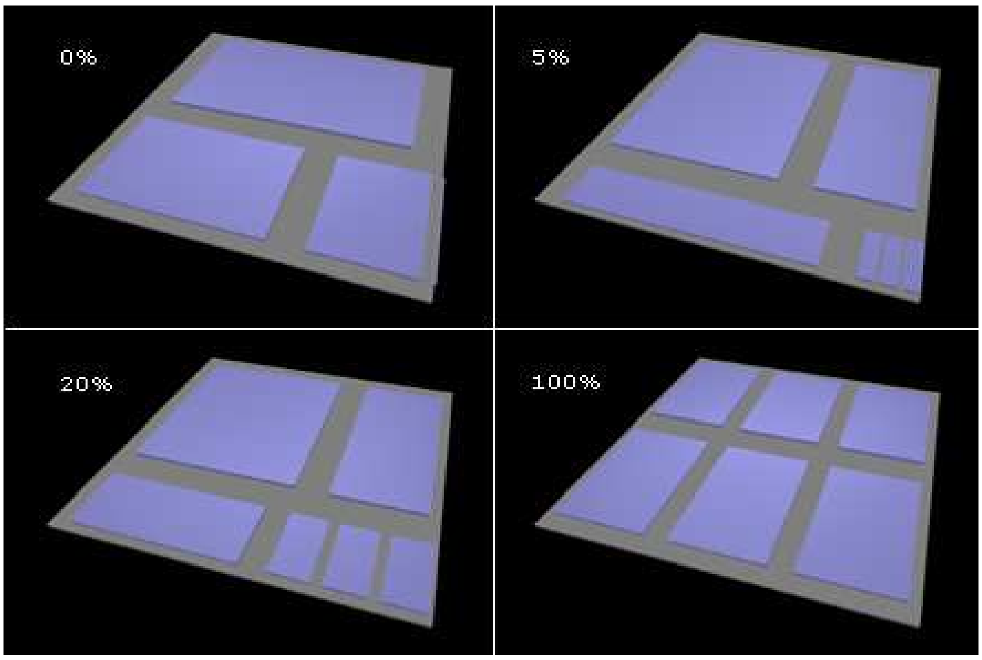


Figure 5.5: Four views with different equalization constant values of the same directory containing six files.

The directory visualized in Figure 5.5 above contains six files with the sizes 1, 2, 3, 1048576, 2097152 and 4194304 bytes respectively. The great difference in size between the first three and the last three files is used to emphasize how equalization improves the visualization of files which are orders of magnitude smaller than their siblings. It is not at all apparent in the top-left view with no equalization that there are in fact six files in the directory. The three files of 1, 2 and 3 bytes in size are all in this case rendered as very thin slices barely visible in the lower right corner of the image. With just 5 percent equalization in the top-right view one can begin to see that there are more files in the directory and at 20 percent it becomes plain that there are in fact three more. Finally in the bottom-right view at 100 percent equalization all six files seem to be of equal size. An equalization of 100 percent results in all sibling nodes being assigned the same area (hence equalization). Therefore with equalization set at 100 percent, StepTree cannot be said to visualize size at all. Equalization is useful but it should be used with moderation as it can cause excessive distortion. One scenario, which was mentioned previously, was that of two files of equal size being given different base areas in the visualization because they have different parent directories.

6 DISCUSSION

In this chapter I will summarize and explain the significance of findings from the previous chapter. Reflections on issues that surfaced during the project will be given as well as ideas for future work in the area and possible future development of the experimental application StepTree.

6.1 About the Results

In this section I will discuss the significance of the results from the previous chapter as pertains to the answering of the proposed user questions and equalization as a solution to vanishing nodes.

6.1.1 User Questions

I believe I succeeded in creating a visualization tool capable of answering all of the user questions from chapter one with varying degrees of success. The ways in which these questions can be answered in the StepTree application were visually demonstrated in the previous chapter.

The first question; *"Where am I?"* is answered by StepTree even though the method used (displaying the path at the top of the window) is virtually identical to the one used in standard file system browsers (such as Microsoft Explorer and Nautilus) and cant be seen as innovative. It is here that I see the most room for improvement. The focus of the project was somewhat shifted from this basic browsing question when the central issue of node visibility and the possible solution in the form of the equalization algorithm surfaced early in the project. One possible solution could for example be to provide a flat overview map showing the part of the tree you are currently viewing. Another could be to keep a simplified "stairway" of labeled directories leading to the one currently viewed. This would not lead to any significant performance degradation as only one box per ancestor directory needs to be rendered. Paths seldom exceed a handful of directories.

The method used to answer the second question; *"What type of file is this?"* is through assignment of colors a method first suggested in the context of Tree-maps by Shneiderman [Shneiderman 1992] and later used in the flat Tree-map visualizer SequoiaView from Technische Universiteit Eindhoven. Using colors for file types gives you a powerful visual cue as to the nature of the directory you are looking at. Does it contain a mix of different file types or just a single type? Does it contain executables or documents? These questions can be answered at a glance instead of by scrolling through several screens full of textual information.

The original Tree-map concept, where on-disk size is mapped directly to area in the visualization, is very good at representing size and is thus ideally suited to answer the third question; *"Where has my harddrive space gone?"* but this type of visualization is not as good at answering questions about structural relationships. StepTree can be placed somewhere in the middle on this scale. It visualizes size but not in the same uncompromising way as the traditional Tree-map. In StepTree a layout step called level atrophy is used to create a step pyramid like effect where the underlying structure of the hierarchy is exposed. This makes files of equal size on different levels in the hierarchy appear of different size in the visualization. The equalization of sibling nodes to prevent nodes from dropping out of the visualization further compromises the visualization of size. Reliability diminishes the further you get from the current focus. However unless you use extreme equalization settings you will be able to tell, accurately, which subdirectory is the largest and which file in the current directory is the largest.

The problem of answering the fourth question; *"Have important files been recently changed?"* at the same time as answering the third question, *"Where has my harddrive space gone?"* turned out to be more difficult than I had expected and this became the catalyst for my introduction of the equalization algorithm. Without equalization certain files would become all but invisible and visualizing change in them would become more difficult if not impossible. Change is propagated towards the root from the leaf nodes of the hierarchy in such a way that the most recent change in a branch of the hierarchy is reflected in the root of that branch. This means that it is easy to follow a trail of change up towards a changed file without displaying the whole hierarchy. Also if unchanged nodes are hidden a greater portion of the hierarchy can be shown (performance increases as fewer polygons have to be rendered) and greater patterns of change can be observed.

Answering the fifth question; *"What are the structural and contentual relationships?"* is accomplished in a number of ways. StepTree visualizes structural relationships through containment in the same way as nested Venn diagrams. All files in a particular directory are placed on top of said directory. To see the relationship between a node and its ancestors it is obviously important to see not only the node but also its ancestors. In the original Tree-map, leaf nodes obscured underlying nodes making such relationships less apparent. The atrophy of child nodes in relation to their parent creates a gap through which the parent node and in turn its parent node etc. can be seen and their relationship appreciated. Finally the visualization of content in the form of the connection of file type to color and on-disk size to area in conjunction with the above mentioned visualization of structural relationships enables you to answer more specific questions which are both structural and content related. Examples of such questions would be; "Where are all my PDF files located?" and "Are they spread out or concentrated in one location?".

6.1.2 The Significance of Equalization

I introduce equalization of sibling nodes as a solution to the problem of files dropping out of the visualization due to small on-disk size or no on-disk size. It has been shown that equalization can make otherwise virtually invisible files visible by redistribution of size between sibling nodes. The test case described in section 5.2 and in particular the graphical evidence in the form of Figure 5.5 should be ample proof of this. Equalization is significant for this type of visualization in that it makes the visualization of other metrics besides size worth while by assuring the visibility of the node.

A problematic issue with equalization that should be emphasized is the issue of increased unreliability. Equalization of siblings will regardless of how small an equalization constant is used cause the visualization of relative size to become somewhat distorted for all but those nodes originating in the same sibling group. A node that is larger (on-disk) than another from the same sibling group will always be visualized as larger (albeit perhaps only slightly) unless they are 100 percent equalized. There is however no guarantee for the same happening if they are from different sibling groups. This problem is only minor if a relatively low equalization constant is used (20 percent and under) and I believe that the pros greatly outweigh the cons in this case.

It would in all fairness be possible to solve the small size problem by some clever method of zooming but this would not help us if we wish to see patterns of change (as we are not likely to get a very good sense of overview if we were zoomed in to one particular tiny file). Another possible alternative solution would be to link size and change in such a way that changed files automatically increase in size. This would however run the risk of ruining the visualization of both metrics as it would be almost impossible to know if the size of a file in the visualization is linked to actual on-disk size or a recent change.

6.2 Reflections

Working on this thesis project has made me realize that there is no substitute for actually trying ideas out through implementation. Problems surface during development and testing which may not be apparent during the research phase or even the design phase. Some of these problems can be addressed within the timeframe of the project itself while others may give opportunity for further research in the future.

Something I realized while developing StepTree was that three dimensional user interfaces, such as StepTree, running on top of an operating system using the distinctly two dimensional desktop metaphor will inevitably cause somewhat of a clash in the moment of transition. The reasons for this might be that there are two different ways of navigating, making selections as well as viewing the two environments. Some visualization tools try to solve this by merging the two metaphors into the same application. The file system visualization tool VisFS for example has a traditional flat node and indentation style tree browser beside a more innovative 3D visualization. In VisFS this traditional browser serves as the sole method of navigating through the 3D visualization, providing the user with a familiar interface to an otherwise alien concept. It can be seen as serving the purpose of a hallway or an entrance linking the user to the 3D visualization through something he or she is already familiar with. A non experimental, purely three dimensional method of visualization would in my opinion have to run in an environment which is already three dimensional in order for it to become a real success. If it is not to be run on such an environment it would probably be a good idea to add a more familiar method of navigation to bridge the gap. Based on this I believe that a good idea for further development of StepTree would be to offer just such a bridge, to include a more traditional flat tree browser while at the same time allowing fully functional 3D navigation and selection. Another idea would be to include a user interface similar to the one exemplified by StepTree into an immersive 3D environment. These environments have existed for quite some time, mainly in the form of 3D games, both on-line and off-line. On-line games which are increasingly becoming popular are places where thousands of players congregate, not just to play the game but also to a great extent to socialize. If you are immersed in such a game it is often a great annoyance when the outside, in this case the flat operating system, has to be accessed for a purpose such as sending a file to a friend. What if the file system could be accessed from the 3D world, the pertinent file located and sent without having to make the transition?

6.3 Future Work

The StepTree application should provide ample opportunity for further research and improvements. Areas such as usability and practical applications would probably be rewarding to look into. The visualization of change exemplified in StepTree could if coupled with a truly secure file system (e.g. where meta data on files and directories cannot be changed at will) be used in for security purposes. It would also be interesting to look into how this type of visualization performs in true virtual reality. Answering the browsing question “Where am I?” should be looked into in greater detail as I sense that a better solution could be found. It would also be interesting to see in greater detail how different levels of equalization of sibling nodes affect usability and reliability.

6.4 Summary

The development of StepTree has been both experimental and exploratory. Some aspects of the design were controlled from the beginning while others had to be added as solutions to various problems that surfaced during development. All in all I feel that this project has been a success. StepTree is obviously not the final answer to the problem of hierarchy visualization but I believe that it might serve as an example, a reference point from which to make comparisons and draw further conclusions in the future. I would very much like to see further work done with this application as a basis.

7 ACKNOWLEDGEMENTS

I would like to thank my advisor Professor Rune Gustavsson for his support, his insightful comments, his patience in listening to my ideas and for his guidance during this period. I would also like to thank Frans Mårtensson for taking the time to read this thesis, giving comments on it as a whole and pointing out details and areas that needed further attention.

8 REFERENCES

Bederson, B. B. (2001). *PhotoMesa: A Zoomable Image Browser Using Quantum Treemaps and Bubblemaps*. UIST 2001, ACM Symposium on User Interface Software and Technology, CHI Letters, 3(2), pp. 71-80.

Bruls M, Huizing K, van Wijk J J (2000). *Squarified Treemaps*. In Proc. of Joint Eurographics and IEEE TCVG Symp. on Visualization (TCVG 2000). IEEE Press, pp. 33-42.

Card S. K., Mackinlay J. D., and Shneiderman B. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufman, 1999.

Cobb, T. (1997). Cognitive efficiency: Toward a revised theory of media. Educational Technology Research & Development, 45 (4), 21-35.

Cockburn A and McKenzie B. *An evaluation of cone trees*. In People and Computers XV (Proceedings of the 2000 British Computer Society Conference on HumanComputer Interaction.), University of Sunderland, September 2000, Springer-Verlag, In Press <http://citeseer.nj.nec.com/cockburn00evaluation.html>

Furnas G W. *The FISHEYE view: a new look at structured files*. Bell Laboratories Technical Memorandum #81-11221-9, October 12, 1981.

Jin L. and Banks D. C. (1997). *TennisViewer: A Browser for Competition Trees*. IEEE Computer Graphics and Applications, July/August, 63-65.

Jungmeister W.-A. and Turo D. 1992. *Adapting treemaps to stock portfolio visualization*, University of Maryland, Center for Automation Research technical report CAR-TR-648 (also CS-TR-2996, SRC-TR-92-120).

Herman I, Melançon G, Marshall M S. *Graph visualization and navigation in information visualization: A survey*. IEEE Transactions on Visualization and Computer Graphics, 6(1):24-43, 2000. <http://citeseer.nj.nec.com/herman00graph.html>

Shneiderman B, *Tree visualization with tree-maps: A 2-d space-filling approach.* ACM Transactions on Graphics 11, 1 (1992), 92-99. <http://citeseer.nj.nec.com/shneiderman91tree.html>

Stasko J, Catrambone R, Guzdial M and McDonald K. *An evaluation of space-filling information visualizations for depicting hierarchical structures.* Technical Report GIT-GVU-00-03, Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Atlanta, GA, February 2000. <http://citeseer.nj.nec.com/stasko00evaluation.html>

Turo D. and Johnson B. *Improving the visualization of hierarchies with treemaps: Design issues and experimentation*. In Proceedings of IEEE Visualization '91, pages 124--131. Kaufman & Nielson Ed., October 1992. <http://citeseer.nj.nec.com/turo92improving.html>

van Wijk J and van de Wetering H. *Cushion Treemaps: Visualization of Hierarchical Information.* In proceedings of INFOVIS'99 (San Francisco, October 25-26, 1999), pages 73-78.

9 LIST OF FIGURES

3.1 Screenshot of tree view (left pane) in Microsoft Explorer.

3.2 An example of a node and link diagram depicting the class hierarchy of the StepTree application.

3.3 A basic Tree-map.

3.4 Cushion Tree-map.

3.5 Squarified Cushion Tree-map.

4.1 StepTree Screenshot.

4.2 A diagram showing the gradual equalization of four files.

5.1 Screenshot of a directory containing pdf files (dark red) and text files (light yellow).One of the files also has a detailed information label.

5.2 Two screenshots of the same directory taken in StepTree. The left screenshot shows 2 levels and the right 3.

5.3 Visualizing change with solidity. In the left screenshot non-solid nodes are ghostedand in the right they are hidden.

5.4 Views of the same base directory with different atrophy values.

5.5 Four views with different equalization constant values of the same directory containing six files.

Appendix A

StepTree Documentation

CONTENTS

1. ABOUT STEPTREE 1
2. COPYRIGHT & DISCLAIMER 2
3. SYSTEM REQUIREMENTS........ 2
4. NAVIGATING THE SCENE 3
5. NAVIGATING THE TREE........ 3
6. VISUALIZATION SETTINGS........ 4
7. CONTROL SETTINGS 6
8. SOLVING PROBLEMS........ 6

1. ABOUT STEPTREE

StepTree is a 3D file system visualization prototype for large file system hierarchies. It is capable of representing *size*, *change* and *file type* visually in the same view. The name StepTree comes from the similarity between this type of visualization and a step pyramid. A step pyramid can be said to be a tree with the base step as its root and a branching factor of one.

StepTree was created as part of a master thesis project conducted at the Department of Software Engineering and Computer Science at Blekinge Institute of Technology in Sweden.

Features:

- Visualizes the relative sizes of nodes (directories and files).

- Size metrics for nodes with the same parent can be equalized. Equalization makes even files of zero bytes visible. Use this with caution however as it introduces errors (two files of equal size on disk may appear to be of different size in the visualization if they are on different levels in the tree).

- Visualizes change. Can de-emphasize or prune away nodes which have not been modified for a set number of days. De-emphasized nodes are drawn as wire frames.

- StepTree can handle a large number of nodes. The current maximum is set to 32766. This particular limitation is caused by the selection method currently in use. Your graphics hardware might of course further limit the number of nodes you will be able to display.

What am I trying to do?

I wanted to look at the possibility of using 3D adaptations of Venn diagrams to visualize not only size but also change at the same time. If I had not implemented equalization of size metrics, for example, this would not have been practical (as some nodes would have been too small to see). Once I saw that it would be practical to visualize change I looked into a number of ways of visualizing it. The solution I finally selected was pruning or de-emphasizing unchanged (and thus uninteresting) nodes. My choice on how to visualize de-emphasized nodes was, to a large degree, based on performance and only partly on aesthetics. Wire frames for example although not overly ugly are not as nice as transparent solids. Transparency however has the distinct drawback of draining performance, especially when thousands of nodes are rendered. My goal was also that it should be possible to see where changes had been made without necessarily seeing the whole structure. Changes made to files far down in the hierarchy needs to be visible higher up. Thus the age in days of a change to a particular file is propagated upwards towards the root, updating (if less than what is already there) the values of directories along the way. This means that there is always a path of solid nodes from the root of the visualized subset to any solid node.

2. Copyright & Disclaimer

CONTACT:

Author: Thomas Bladh
E-mail: tibsoft@swipnet.se

3. System Requirements

These are minimum or close to minimum requirements.

Hardware:

Processor:	166MHz Pentium
Memory:	64MB Ram
Harddrive:	At least one :)

Software:

OS:	Windows 98, Windows Me*, Windows 2000 or Windows XP with OpenGL installed. ** = Not tested.*
Graphics:	Desktop color depth should be at least 16bpp.

4. NAVIGATING THE SCENE

Moving and looking around is done by using the keys assigned for this purpose as well as by changing the placement of the mouse pointer within the window. The possible movements and their default keys are:

Forward	A
Backward	D
Up	E
Down	SPACE
Left Strafe	S
Right Strafe	F
Look Up	Up Arrow (*and upper edge mouse pointer placement*)
Look Down	Down Arrow (*...lower edge...*)
Look Right	Right Arrow (*…right edge…*)
Look Left	Left Arrow (*…left edge…*)

If the mouse pointer is placed near the edge of the window the camera will turn in the direction indicated by the placement of the mouse pointer. The way the on-screen axes are interpreted can be reversed in the “Controls” tab of the “Settings” window (see **Control Settings**). It should be noted that forward motion is also bound (hard coded) to the right mouse button. Future versions of StepTree should also allow mouse buttons to be re-assigned.

For more info on key mappings see the **Control Settings** topic.

5. NAVIGATING THE TREE

Entering Directories

To enter a directory you simply left click on it (In the default color scheme, all directories are grey and all files colored). To go back one level you simply click the current base node of the visualized subset.

Opening & Executing Files

To open or execute a file you hold down the CTRL key and left click on it.

Information

To view detailed information about a particular node (file or directory) you hold down the CTRL key and then press the right mouse button with the mouse pointer over the node. The information will be visible as long as you keep the mouse button pressed.

6. VISUALIZATION SETTINGS

On this tab you can change settings affecting the visual appearance of the visualization.

Distortions

There are currently two types of distortion which can be applied to the visualization. The first is node atrophy and the second is equalization. With both of these set to zero the birds eye view of the visualization is similar to that of a flat Tree-map.

The node atrophy percentage stands for how much the area of each new level should be reduced in relation to the previous. An atrophy of 20 percent means that the area of each new level will be 80 percent of the area of the previous.

An explanation of equalization is a bit more complex. The metric derived from the sizes of nodes sharing the same particular parent can be seen as points in a graph. The equalization value decides how much this “graph” should be smoothed out towards the average. An equalization of 100% means that the graph becomes a vertical line (all node metrics are the same and the base size of a node in the visualized subset bears no relation to its size on disk) whereas an equalization of 0% means that there is a direct and unaltered relationship between the base size of a node in the visualization and its actual size on disk (except of course for the distortion introduced by atrophy).

Visualize Modifications

The visualization of modifications can either be enabled or disabled. If it is disabled all nodes are drawn solid. If enabled only those nodes modified within a specified time period are drawn solid.

There are two settings controlling this feature. The first is, as mentioned previously, the number of days after being modified that a node is drawn solid. This value can be set to between 1 and 365 days. The second determines how non-solid nodes are drawn. The options are either to draw them as wire frames or not draw them at all.

Color Mapping

This section is a simple editor used to assign colors to different file types (such as exe, dll and mp3) as well as directories. You have the option of either *adding*, *editing* or *removin*g an item here and its use should be quite self explanatory. There are two fixed entries in this list that you should know about. The first is the “<dir>” entry which specifies the color to use for directories. The second is the “<file>” entry which specifies a default color to use for files of types not yet assigned their own color.

Detail

Here you specify the number of levels of the tree you wish to show. Keep in mind that setting this value to high might degrade performance.

As a safety measure a setting is included to specify a maximum number of nodes that the application is allowed to display at once. This protects you from accidentally making changes which could generate too many nodes and potentially flood the memory causing the computer to hang. Be careful not to set this value above what your computer can easily handle.

Misc

Startup Root:

Here you can specify the location you wish the application to start in. This can be the absolute root, a fixed disk or a directory on a fixed disk. The absolute root is selected simply by opening the root selection browser navigating to where you see the fixed disks (`[-C-], [-D-]` etc.) and clicking on ok without selecting anything. Note that the absolute root is represented by an empty path (as it is above the drives in the tree). The startup root textbox should therefore be empty if the absolute root was selected.

Enable Label Flags:

This enables the rendering of flags with names of first level (in the visualized subset) directory nodes (drives are classified as directories).

7. Control Settings

On this tab you can map keyboard keys to various functions as well as change how the mouse axes are interpreted.

Mapable Functions

Forward	Forward motion.
Backward	Backward motion.
Up	Move up.
Down	Move down.
Left Strafe	Sideways motion, left.
Right Strafe	Sideways motion, right.
Look Up	*(pretty obvious)*
Look Down	*(pretty obvious)*
Look Right	*(pretty obvious)*
Look Left	*(pretty obvious)*
Add Level	Add a level to the visualized subset.
Rem Level	Remove a level from the visualized subset.
Toggle Flags	Enable and disable label flags.

Mouse

The two mouse settings are changed using check boxes labeled “invert x axis” and “invert y axis”. These simply reverse how the mouse placement is handled on the respective axis.

8. Solving Problems

Lockups:

If you should accidentally try to render more nodes than your computer can handle you can abort the operation by pressing ESC. You will be presented with the choice of aborting or continuing, if you abort your settings will be lost (to make sure erroneous settings are purged) and the application terminates. When you restart after an abort it should be as if the application was newly installed.

There are of course better solutions to this problem but this is, after all, only a prototype :)

Navigation & Selection Problems

If you experience problems selecting nodes (ending up in the wrong directory etc.) you might want to try increasing the color depth to 24bpp.

Appendix B

RecurseFiles Function

```
//---------------------------------------------------------------------------
// RecurseFiles
//---------------------------------------------------------------------------
// INPUT:   Node    - Pointer to a node in the FileSysTree structure from which
//                    to start creating entities.
//          x, y, z - Coordinates in the scene where Node will be placed.
//                    (actually an entity representing a Node) All Node's
//                    descendants will be placed on top of this entity.
//          w, h, d - Node's dimensions. All descendants will be smaller or
//                    in some rare cases equal in size to Node.
//          Level     Level to assign to Node. Node's children will have Level
//                    set to Level + 1.
//
// EXTERNAL:
//          Settings.Atrophy      - Double (0 - 0.5)
//                                  How much each successive level should
//                                  shrink.
//          Settings.Equalization - Double (0 - 1)
//                                  Amount of equalization.
//          Settings.Levels       - Word
//                                  Maximum number of levels to display.
//          Settings.NodeLimit    - LongWord (1-1000000)
//                                  Maximum number of nodes to allow.
//          Settings.NonSolids    - (nsWireframe, nsInvisible, nsIgnore)
//                                  Enumerated variable describing how non-solid
//                                  nodes should be drawn.
//          Settings.SolidDays    - Word (1-360)
//                                  Number of days after being modified that
//                                  a node is represented as "solid".
//
// OUTPUT:  A boolean value indicating whether the number of created entities
//          has exceded Settings.NodeLimit. If this happens the recursion is
//          halted and no more entities are created. It is then up to the caller
//          to clear the Static entity list, lower the value of Settings.Levels
//          and try again.
//
// NOTE:    Uses a number of external as well as local methods.
//---------------------------------------------------------------------------
function TStepTreeMain.RecurseFiles(Node: TFileSysNode;
x, y, z, w, h, d: Double; Level: Integer): Boolean;
var
      BoxEntity: TEGLBoxEntity;
      FlagEntity: TEGLFlagEntity;
      Child: TFileSysNode;
      NodeLine: TNodeList;
      i, Children: Integer;
      NodeFraction: Double;
      nx, nz, nw, nh, nd: Double;
      rx, rz, rw, rh, rd: Double;
      WasAdded: Boolean;
      ChildList: TNodeList;
      FlagCoords: TGLArrayf3;
      Solid: Boolean;

      //---------------------------------------------------------------------
      //RenderLine
      //---------------------------------------------------------------------
      // INPUT: N/A
      //
      // OUTPUT: Returns True if successful and False if not.
      //
      // NOTE:   Renders a line of boxes.
      //---------------------------------------------------------------------
      function RenderLine: Boolean;
      var
            i: Integer;
            LineArea: Double;
      begin
            Result := True;
            LineArea := GetLineArea(NodeLine, w);
            nd := LineArea / w;
            // create the boxes
            for i := 0 to Length(NodeLine) - 1 do
            begin
                  // calculate node width
                  nw := (NodeLine[i].Area / LineArea) * w;
                  // and node height
                  nh := h;
                  rx := nx; rz := nz; rw := nw; rh := nh; rd := nd;
                  NodeAtrophy(rx, rz, rw, rh, rd, Settings.Atrophy);
                  // Place the box
                  Result := RecurseFiles(NodeLine[i].Node,rx,y+h,rz,rw,rh,rd,Level+1);
                  if (Result = False) then Exit;
                  // add node width to x coord
                  nx := nx + nw;
            end;
            // add to depth (next row)
            nz := nz + nd;
            // reset the x coord
            nx := x;
      end;
begin
      Result := True;

      // Failsafe.
      Application.ProcessMessages;
      if AbortSignaled then
      begin
            Result := False;
            Exit;
      end;

      // If the node limit was exceeded.
      if (Scene.StaticEntities.Count > Settings.NodeLimit) then
      begin
            Result := False;
            Exit;
      end;

      // Determine if this node is solid.
      Solid := (Node.Modified <= Settings.SolidDays);
      // If this node is a non-solid and the Config specifies that non-solids
      // should not be drawn then we bail out early.
      if not Solid and (Settings.NonSolids = nsInvisible) then Exit;

      nx := x; nz := z;

      // Create a box entity and assign it to our node.
      BoxEntity := TEGLBoxEntity.Create;

      with BoxEntity do
      begin
            DomainObject := Node;
            // Set dimensions and coordinates for the node.
            Coords[_X] := x;
            Coords[_Y] := y;
            Coords[_Z] := z;
            Dimensions[_W] := w;
            Dimensions[_H] := h;
            Dimensions[_D] := d;
            // Should the bottom be drawn (only the base node needs this).
            DrawBottom := Level = 1;
            // Non-solid nodes should be drawn as wireframes unless the settings
            // indicate that this should be ignored.
            WireFrame := not Solid and not (Settings.NonSolids = nsIgnore);

            if (Node.NodeType = ntDirectory) then
            begin
                  // Color and add the directory to the scene as a box.
                  Color := TRGBA(Settings.GetColorForType('<dir>'));
                  // Should we add a flag.
                  if (Level in [1..2]) and (Node.Name <> '') then
                  begin
                        // Create, Position and Print the flag.
                        FlagEntity := TEGLFlagEntity.Create;
                        FlagCoords[_X] := x;
                        FlagCoords[_Y] := y + h;
                        FlagCoords[_Z] := z + d;
                        PrintBitmap(Node.Name, FlagEntity.Bitmap, 0);
                        FlagEntity.Coords := FlagCoords;
                        FlagEntity.PoleHeight := 10;
                        // Add it to the scene.
                        Scene.AddEntity(FlagEntity, ecDynamic, 0);
                  end;
            end
            else begin
                  // Color and add the file to the scene as a box.
                  Color := TRGBA(Settings.GetColorForType(GetFileExt(Node.Name)));
            end;
            // Wireframed nodes are rendered slightly darker than other nodes to
            // further distinguish them from solid ones.
            if WireFrame then
            begin
                  Color[_R] := Round(Color[_R] * 0.8);
                  Color[_G] := Round(Color[_G] * 0.8);
                  Color[_B] := Round(Color[_B] * 0.8);
            end;
      end;
      Scene.AddEntity(BoxEntity, ecStatic, 0);

      // If we have reached the maximum number of levels we exit here.
      if (Level + 1 > Settings.Levels) then Exit;

      // Get the number of children and exit if there are none.
      Children := Node.ChildCount;
      if Children = 0 then Exit;

      // Add all children to the child list.
      SetLength(ChildList, Children);
      for i := 0 to Children - 1 do
      begin
            Child := Node.GetChild(i);
            // Calculate an equalized fraction for each child.
            NodeFraction := ScaleAndEqualize(   Child.Size,
                                                Node.Size,
                                                Children,
                                                Settings.Equalization);
            ChildList[i].Area := NodeFraction * (w * d);
            ChildList[i].Node := Child;
      end;

      // Order children by size (for an improved layout).
      OrderNodeListBySize(ChildList, False);

      for i := 0 to Children - 1 do
      begin
            // Try to add a child to the line (this will fail if the aspect
            // of the new line is worse (farther from 1) than the old line.
            WasAdded := AddToLine(ChildList[i], NodeLine, w);
            // If the child was not added.
            if not WasAdded then
            begin
                  // We render this line and start a new one.
                  Result := RenderLine;
                  ClearLine(NodeLine);
                  AddToLine(ChildList[i], NodeLine, w);
                  // If this was the last child we render the new line also.
                  if (i = (Children - 1)) then
                  begin
                        Result := RenderLine;
                        ClearLine(NodeLine);
                  end;
            end
            else begin
                  // If this was the last child we render and clear the line we have.
                  if (i = (Children - 1)) then
                  begin
                        // Try to render the line.
                        Result := RenderLine;
                        ClearLine(NodeLine);
                  end;
            end;
            if (Result = False) then Exit;
      end;
end;
```