Rapid Content Browsing in Hierarchical Space-Filling Representations using Distortion Techniques

by

Kang Shi

A dissertation submitted in partial fulfillment of
the requirement for the degree of

Master of Science

Department of Computer Science
Faculty of Graduate Studies
University of Manitoba

Winnipeg, Manitoba, Canada, 2005

Abstract

Large hierarchies, such as trees or variants thereof, require complex interaction models as they typically span beyond the available display space. Space-filling visualizations, such as the TreeMap, are well-suited for displaying large hierarchies in limited viewing space. They are also designed to display the properties of nodes in hierarchies in space-filling visualizations. To browse the contents of the hierarchy, the primary mode of interaction is by drilling-down through many successive layers. In this thesis I introduce a distortion algorithm based on fisheye and continuous zooming techniques for browsing and searching data in space-filling representations, such as the TreeMap. The motivation behind the distortion approach is for assisting users to rapidly browse information displayed in the TreeMap without opening successive layers of the hierarchy. For searching tasks the distortion technique assists users in identifying the search results even when the hierarchy is dense and is capable of conveying importance level of search results.

Three experiments were conducted to evaluate the new approach. In the first experiment (N=20) the distortion approach is compared to the drill-down method. Results show that subjects are quicker and more accurate in locating targets of interest using the distortion method. The second experiment (N=12) evaluates the effectiveness of the distortion technique in a task requiring context, we define as the context browsing task. The results show that subjects are quicker and more accurate in locating targets with the distortion technique in the context browsing task. The results of both these experiments provide strong evidence that distortion based techniques applied to space-filling visualization facilitates rapid browsing. The last experiment (N=12) evaluates the effectiveness of the distortion technique for presenting search results. The results do not show any improvement in the distortion method over currently available techniques for presenting search results.

Keywords: browsing, searching, distortion, hierarchy navigation, focus+context, drill-down, space-filling visualization, TreeMap, semantic zooming.

Acknowledgement

I am greatly grateful to my supervisor Dr. Pourang P. Irani for his guidance and advice during my study. Dr. Irani is an erudite and polite professor in both research work and life. He introduced me an attractive research area, "human-computer interaction" and taught me related knowledge. I am greatly grateful for the time Dr. Irani spent and his patience in revising my thesis. I am also deeply thankful to Dr. Irani for leading me to a publication and teaching me methods of research.

I am thankful to Richard M. Zobarich who spent time on the experiment data analyses in my thesis. I am thankful to Dr. Ben Li for assisting in the development of the algorithms.

Especially, I am thankful for the support from Dr. Yangjun Chen and Dr. Michel Toulouse when I encountered difficulties.

I am greatly indebted to my parents Wujie Shi and Minhua Kang who guide me and encourage me in all aspects of my life. I am also thankful to my wife Kai

Huang for her help and encouragement.

I am thankful to the HCI Lab and all the participants in my experiments.

Last but not least, I would like to thank all my friends for their help and for the days we spent together.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Information visualization consists of using computer-supported, interactive, visual representations of abstract data to amplify cognition [6]. Visualization amplifies cognition by organizing information and thereby reducing the amount of search performed by the user for locating required data. Visual representations can also amplify cognition by increasing the memory and processing resources available to the user by off-loading internal cognitive structures onto a display. Finally, visualizations allow users to detect patterns and facilitate perceptual inferences.

However, as the visualization gets more complex (either by scaling it to larger sizes of data or due to its limited representational capacity) users will typically require more time to locate items of interest to make inferences from the display.

This problem is compounded by the fact that the choice of visual encodings and representations are chosen by the designer of the visualization and therefore users do not have a significant amount of flexibility in selecting and modifying base representations. Therefore, if items of interest are not 'immediately' perceived or if patterns are not easily detectable in the visualization users will not invest time to identify these.

In many cases, the data source of the information being visualized is structured. For instance, temperature fluctuations on a given day can be structured linearly, flight routes for an airline can be organized as a network and a library's directory can be organized hierarchically. Hierarchical structures are widely and abundantly used. A large quantity of data is organized hierarchically as it can simplify the categorization of information. Hierarchical data is typically represented in the form of a tree structure (Figure 1.1). In a tree (also referred to as a classical or conventional node-link tree), the elements or data points are represented as nodes and the hierarchical parent-to-child relationships are represented as links.

A common research theme in information visualization is to find new techniques for displaying large trees in a limited display space. As shown in figure 1.1, to locate items of interest in a tree can be problematic. A lot of white space is introduced as trees become wider and deeper. In order to view items of interest in the tree,

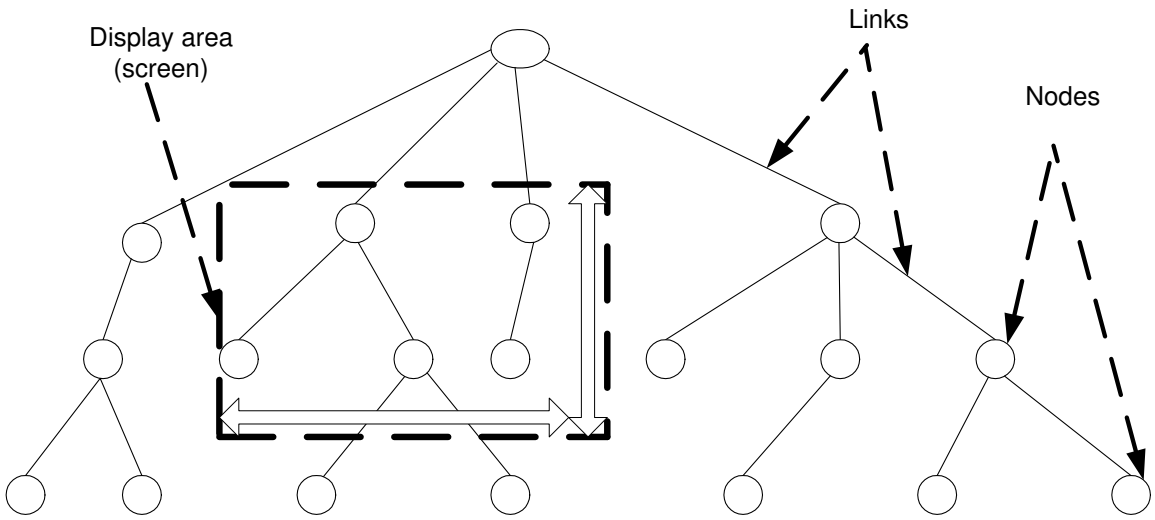


Figure 1.1: Typical hierarchical structure represented as a tree. Only part of the tree is visible in the limited rectangular display region.

users have to use the scroll bars to move the focus from one area to another. For large trees (which are deep and wide at the same time) this method of interaction is inhibitive and users may not perceive patterns in the entire collection of the data.

Several techniques have been developed to represent and display large trees in a limited display space. Space-filling techniques such as the TreeMap, is one effective approach. However, several problems arise with the standard navigation scheme in the TreeMap. In the thesis, I introduce a distortion method for navigating and browsing data in space-filling representations. Distortion techniques have been widely used to enhance information visualization tools. The basic concept

behind distortion techniques is to increase the amount of space for items with interest and decrease the space for items which are not of immediate interest to the user [8]. The distortion algorithms were developed in this thesis for browsing nodes in hierarchies, for allowing users to maintain relationships between nodes, and for visualizing search results in space-filling representations, such as the TreeMap.

1.1 Goal of the Thesis

The goal of this thesis is to create distortion techniques for browsing node information and for facilitating the visualization of search results in space-filling visualizations. Subgoals that follow from this are:

- to implement a distortion algorithm which can be applied to a well-known space-filling visualization, the TreeMap,
- to evaluate the effectiveness of the distortion algorithms.

1.2 Organization of the Thesis

The thesis is organized as follows.

Chapter 2 introduces work related to my research. It reviews techniques in the

field of information visualization for the visualization of hierarchical and space-filling hierarchical visualizations. Chapter 2 also discusses focus+context techniques, browsing techniques applied to the TreeMap, and continuous semantic zooming technique. All of these have inspired the idea of distortion in this research. Finally, chapter 2 summarizes the relevant research on search results visualization.

Chapter 3 describes the distortion algorithm. The algorithm can be divided into two parts, single node distortion (uni-distortion) and multiple node distortion (multi-distortion). Both of these algorithms are described using pseudo-code in detail.

Chapter 4 describes the user evaluation of the new techniques. Three experiments that were designed to compare the distortion techniques to conventional techniques are described and their results are discussed.

Chapter 5 summarizes the contribution of this thesis, and also outlines future directions for this work.

Chapter 2

Related Work

This thesis builds upon principles developed in two main areas of information visualization, *presentation* and *interaction*. In this chapter I present the related literature in both of these areas.

2.1 Presentation

In information visualization, presentation is concerned with how to adequately display information such that users can immediately see patterns, structures or content. One area related to this thesis is the presentation of hierarchical structures. This area focuses on different solutions for presenting large quantities of data into a limited display space while showing necessary detailed information. In this section,

I describe three areas of research which are directly related to the thesis. The first group of research presents alternative presentation methods than the traditional node-link tree for visualizing large hierarchical structures in a limited space. The second set of work describes the various techniques referred to as space-filling visualizations. These techniques are also designed with the intent of showing large hierarchies. The third category of research describes the concept of focus+context techniques and their applications.

2.1.1 Visualizing Hierarchical Structures

Hierarchical data structures are interacted with regularly. They describe the relationships among entities in organizations, in file systems, and in family genealogies. The most common form of hierarchical representation is a node-link tree. However, trees are difficult to browse and are not space efficient. A significant amount of space remains unused in the background as a result of creating an adequate layout for the nodes. Many techniques have been developed to represent and display large trees in a limited display region [22]. Two techniques which have shown promising results for depicting large trees in limited space are the Cone Trees [19] and the Hyperbolic Browser [13].

Cone Tree

A cone tree [19] (Figure 2.1) can represent large hierarchical information in a 3D

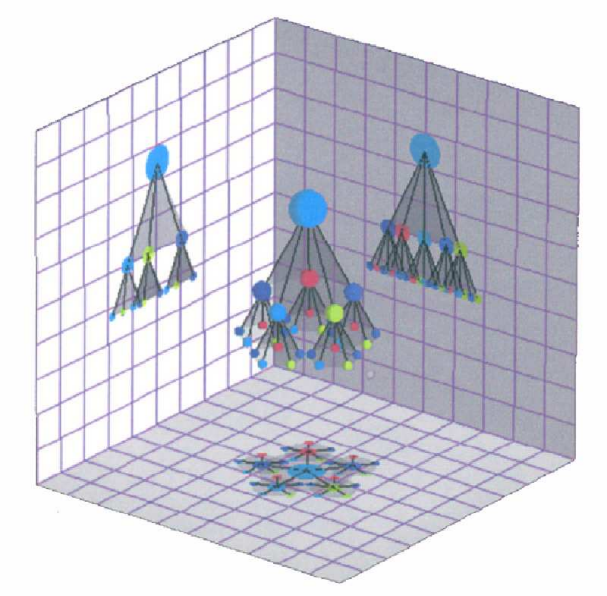


Figure 2.1: Cone tree [9]

space. The root of the tree is the apex of a cone and is placed near the top of the 3D space. The children are evenly placed around the circumference of the base. The children layers are always lower than the parent layer, and the cones in each layer have the same height. The bottom layer of the cone tree matches the width of the 3D space to insure that all nodes of the tree can be represented in the space. To browse the nodes in a cone tree, the users click on the node of interest and this action brings the target node and its path to the front. The

cones are transparently represented and the rotation is animated so that the user can easily understand and maintain the relationships between nodes in the tree. Cone trees are capable of displaying hierarchies with over a million nodes in a fixed display region. The drawback to the cone tree approach is that as a result of its 3D representation a significant amount of nodes are occluded. This has shown to degrade users' performance [19].

Hyperbolic Browser

The hyperbolic browser [13] (Figure 2.2) is another example of a hierarchical repre-

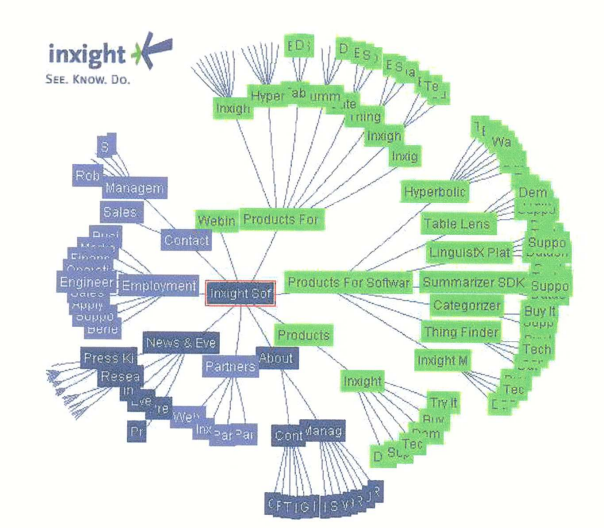


Figure 2.2: Hyperbolic Browser displaying the hierarchy of the web site inxight.com [9].

sentation capable of displaying large hierarchies. The information is laid out onto

a hyperbolic plane and is then mapped onto a circular display region. In the initial state, the root of the tree is placed at the center of the hyperbolic plane, and the children are placed around the root, level by level. The leaf nodes are placed near the edge of the plane. To browse the hierarchy, users can drag any visible node to another position to translate the view. The hyperbolic browser relies heavily on the interaction mechanisms such as selecting and dragging nodes. Without these the user cannot obtain information from the representation.

Although cone trees and the hyperbolic browser are a significant improvement over the classical node-link tree, they do not make efficient use of the entire display region. In addition, as the hierarchy gets more complex, a high degree of clutter is introduced. As a result, new techniques referred to as space-filling visualizations have been developed.

2.1.2 Space-Filling Visualizations

One approach to resolving the problem of inefficient use of space for viewing elements in a hierarchy is the use of space-filling visualizations. Space-filling visualizations [10] divide the display space into nested blocks. As shown in figure 2.3, the root of the tree has three children, so the rectangle is divided into three blocks vertically. Each child has two children so that each block is divided into 2 blocks

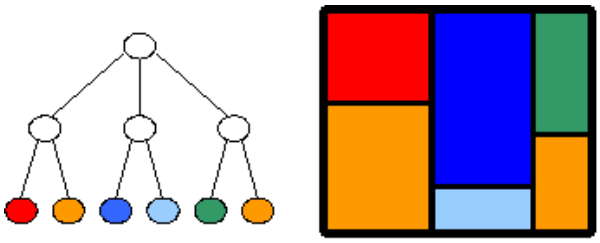


Figure 2.3: Tree and a possilbe space-filling representation. The space reserved for each node is equivalent to the weight of the node. The color of each node represents the type of each node.

horizontally. Unlike classical node-link trees, in Space-Filling visualizations, blocks represent nodes in the hierarchy and the nesting relationships among blocks represent links between parents and children. The size of each block in the space-filling visualization is the weight attributed to that node in the tree. For instance, in figure 2.3, the weight of the blue node is larger than the weight of the green node. If this representation depicted a file structure, larger files would have a larger weight and would therefore be represented using larger blocks. Furthermore, the color of the blocks is used to map some properties of the node. For example, in figure 2.3 the color of the blocks denotes the type of the nodes. The red color block could be .EXE files, the brown color blocks could be .TXT files in a file directory.

The TreeMap [12], the CushionMap [25], and the Sunburst [23] are examples of space-filling visualizations.

TreeMap

Johnson and Shneiderman [12] developed a space-filling visualization method called the TreeMap which can represent large hierarchical structures in a 2D rectangular area (The algorithm of TreeMaps is described in appendix I). TreeMaps make efficient use of the display area and provide structural information. In the TreeMap, the display area is divided into nested rectangular regions to map an entire hierarchy of nodes and their children. Each node uses an amount of space relative to the weight of the item being represented in the hierarchy. Figure 2.4 (b) shows a

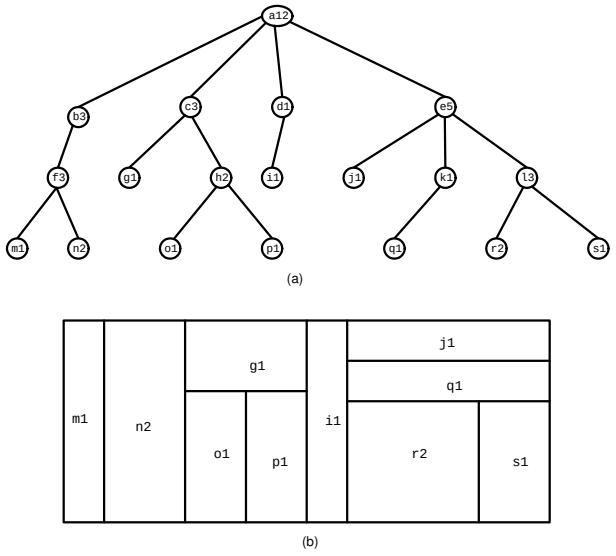


Figure 2.4: Tree structure (a) and its corresponding TreeMap visualization (b). The letters are the labels of nodes, the numbers following the letters are the weights of nodes.

TreeMap visualization which is constructed from the tree in figure 2.4 (a). TreeMaps

give an integrated view of the entire hierarchy and thereby simplify the amount of interaction required to locate items of interest in large hierarchies. In most cases, TreeMaps will help users quickly locate the requested nodes and with a glance allow them to get related information. For instance, users can quickly locate the largest or smallest element in a hierarchy, such as the largest or smallest file if the TreeMap represented a file system.

TreeMaps are well-suited for revealing global patterns in the data such as large pockets of empty space on a disk drive. However, the standard browsing mechanisms provided for inspecting the data can be complex, in particular as the size of the hierarchy grows larger. The method utilized by the TreeMap for browsing data is through drilling down into (moving down) the hierarchy or rolling up (moving up) to find nodes of interest. This interaction approach is very common and has been widely established by file and directory explorers provided in most current operating systems. The typical user interaction for locating a node consists of clicking the parent directory (or subtree) in which might reside a node of interest. The subtree fills the entire space and the user can recursively select subtrees until reaching their final location or node (Figure 2.5). This form of interaction is analogous to zooming into a region of interest with each step of the zoom operation being a subtree in the hierarchy. In general, and particularly in the context of this thesis, content browsing refers to the task of locating specific content, such as a photograph or

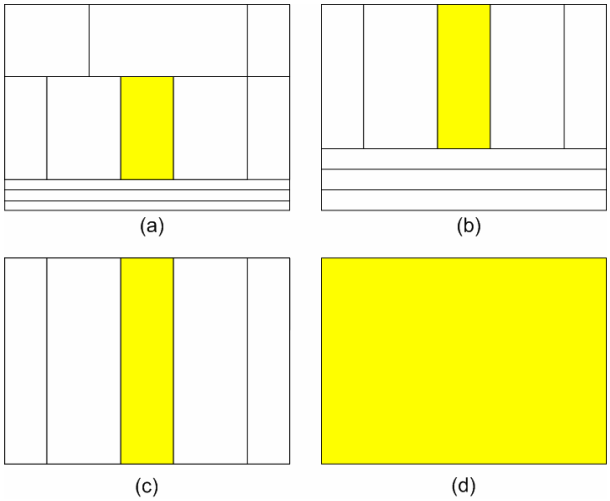


Figure 2.5: TreeMap uses a drill-down approach to open successive layers of a hierarchy. Selecting a parent node opens it and presents the subtree in the entire view. A total number of three drill-down operations are required to inspect the node highlighted above.

document. This is analogous to flipping through pages of a catalog of products or a phone directory.

Several problems arise with the standard navigation scheme of drilling down and rolling up. Users can spend a significant amount of time browsing for specific items in hierarchies using such an interaction. In space-filling visualizations the only visual cue available to the user for locating a node are its visual attributes, such as color and size of node. This reduces the user's ability to quickly find elements unless they can adequately match the object sought after to its visual mappings. Another drawback with the drill-down approach is the number of unnecessary "trips" a user

may take to reach the file adequately. In the drill-down approach, traversing each successive layer requires abandoning the previous view. This can typically lead to disorientation during navigation and reduce the amount of context available for the task. The lack of context in browsing can negatively impact performance as the user has to internally reorient and reestablish relations between views to determine the group or cluster to which an element belongs to.

As an enhancement to the TreeMap, Turo and Johnson [24] use animation to present relative changes over a sequence of time. In their research the animation is used to show the change of node weight such as an increase in stock price over a fixed period of time. However, their animation is not continuous and works in discrete steps. They also use animation to exaggerate the property of nodes. An exponential function is applied to the nodes such that large nodes become very large and small nodes shrink even further. The rationale for doing this is to show more clearly the presence of large items in the hierarchy. Their technique does not scale well to browsing or searching tasks. Furthermore, the distortion is simply created by scaling the size of all the objects.

The navigation problems inherent in the TreeMap representation, as discussed above, will be addressed in this thesis by introducing a new distortion technique for the TreeMap. The next few sections discuss the research leading to the techniques

developed here.

CushionMap

CushionMap [25] extends and enhances the appearance of standard TreeMap. Wijk and Wetering [25] were interested in revealing the structure of the hierarchy in the TreeMap particularly for the balanced tree. If each parent has the same number of children with the same size, the standard TreeMap becomes a regular grid. In this case, users can hardly figure out the ancestor of a node. CushionMap uses shading to show the hierarchy structure. Figure 2.6 shows a CushionMap, which displays

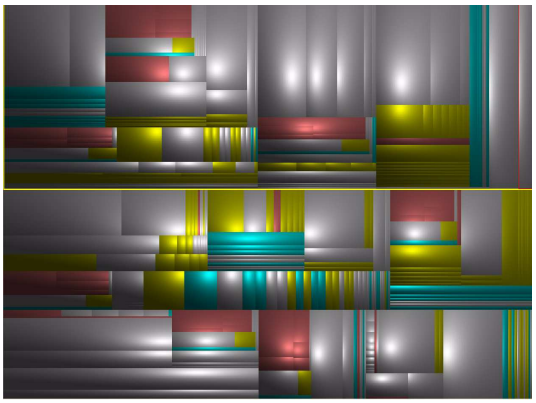


Figure 2.6: Cushion TreeMap [25]

the surface in illuminated shades. In the CushionMap, the parent-child relationship is much clearly presented than in the standard TreeMap. However, even very small nodes in the cushion map are difficult to extract visually. Furthermore, Cush-

ionMap make use of the same drill-down technique as the TreeMap and therefore relationships between elements in a hierarchy can be difficult to assimilate.

Sunburst

Stasko and Zhang [23] designed a radial space-filling visualization method called Sunburst (Figure 2.7). Sunburst uses a radial layout to represent hierarchical structures, with elements represented as parts of concentric circles. As shown in figure

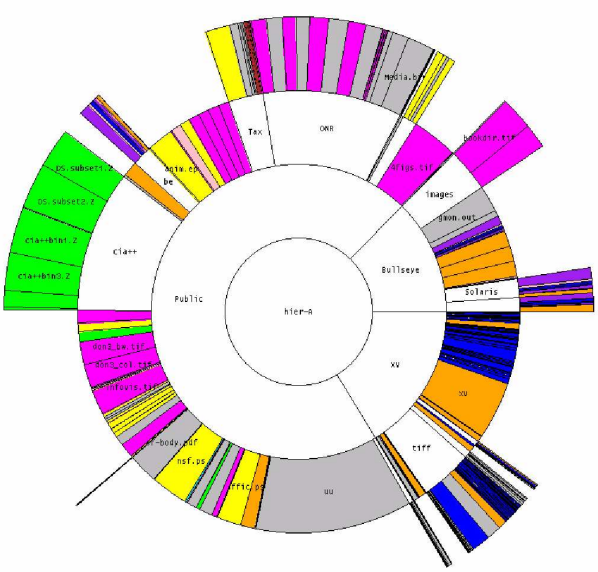


Figure 2.7: Sunburst visualization [23]

2.7, the root of the hierarchy is placed in the center of concentric circles. Other elements are arranged around the center according to the level order in the hierarchy, and the deepest level is furthest away from the center. The angular wedge

taken by an item corresponds to the weight of the node (size of a file, importance, etc.) and the color of an item corresponds to the type of a node. The authors of Sunburst compared Sunburst to TreeMap. Their evaluation shows that users perform similarly with both tools on small hierarchies.

In general space-filling techniques are well suited at displaying large quantities of data in a limited display space. However, a high level of interaction is necessary for providing the contents of the data being represented. As revealed in the images presented in this section on space-filling visualizations, small items are difficult to see as they are given a size proportional to the size or weight of all the elements in the hierarchy. To identify and present details of regions that are not clearly visible or that occupy a small amount of the display space, additional presentation techniques have been developed. One such technique is the focus+context method.

2.1.3 Focus+Context

Focus+Context techniques have been designed for allowing the user to see details that are in focus while maintaining context information of the global view. Generally in focus+context, the size of the items in focus increases to present enough details, and the size of the non-focus items decreases but is still visible to show the relevant information. Focus+context methods have been applied to the display of

graphs [8], trees [18], and tabular data [17].

Fisheye views [8] are a specific case of focus+context techniques. They provide a balance of local details and global structure information. By using distortion, fisheye views increase the size of local space to present more detailed information. On the other hand, fisheye views decrease the size of non-focal items that constitute the context. Fisheye views are used in information visualization to display large information structures, and to facilitate users' attention to local details. Focus+Context techniques assist users in viewing peripheral information while maintaining their focus on the elements of interest. Examples of visualizations that have adopted fisheye views are the Table Lens [17], Sunburst [23], and Information-Slices [1]. Table Lens [17] is an example of fisheye views applied to a tabular rectangular structure. Sunburst [23] and Information-Slices [1] are examples of hierarchical space-filling visualizations using fisheye views. The TreeMap representation being a rectangular space-filling representation can take advantage of the implementation of fisheye views discussed here.

Table Lens

Table Lens is a technique [17] that was designed for visualizing large tables. Using focus+context or fisheye techniques, Table Lens modifies the layout of a table by dividing the table into a focal area and context area. In the focal area, Table Lens

uses distortion to increase the space without bending any row and column. Cells divide the focal space to the appropriate size to show enough detail information from the table. In the context area, cells divide the space equally to maintain the structural information and only shows part of the content in the cells. In Table Lens, users can define more than one focal area to obtain detail information from different parts of the table. Figure 2.8 shows an example of Table Lens. Focus (on the left side of the red line) is shown in clear detail. Context (on the right side of the red line) is shown in a demagnified manner.

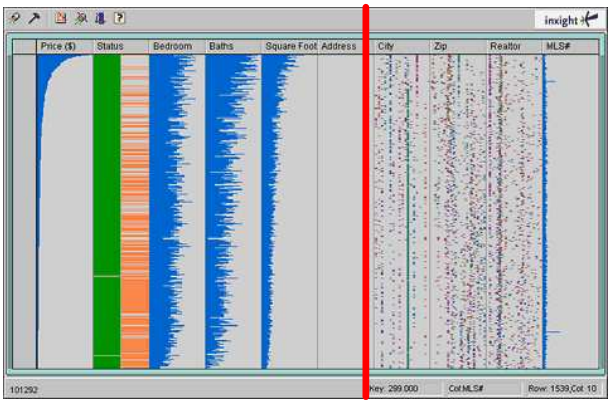


Figure 2.8: Table Lens from inxight.com. The left side of the red line is the focal area, the right side of the red line is the context area.

Focus+Context in the Sunburst

Stasko and Zhang [23] designed three focus+context displays for the Sunburst: angular detail, detail outside and detail inside (Figure 2.9). These techniques allow

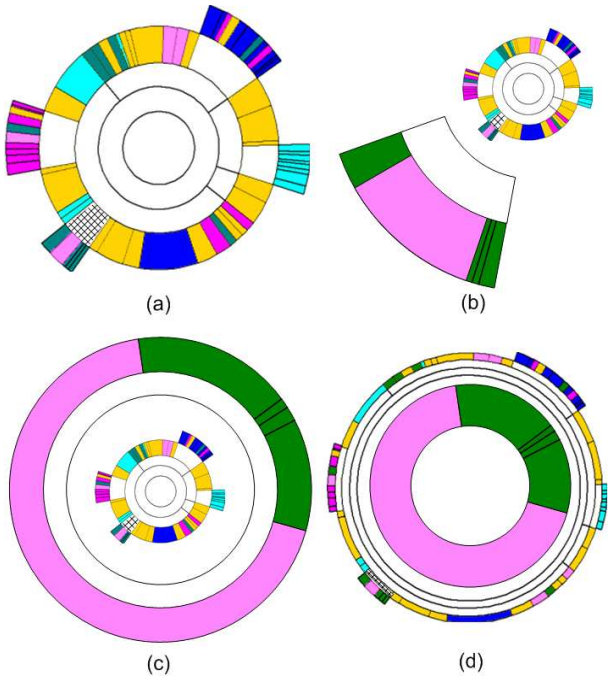


Figure 2.9: Sunburst (a) has been developed with three different focus+context methods: (b) angular detail, (c) detail outside and (d) detail inside. These techniques are well-suited for browsing the structure of a hierarchy [23].

users to interact with subtrees in the hierarchy by facilitating the view of small items in detail while providing context of the entire hierarchical structure. Each technique takes advantage of smooth animated transitions between the views to help users maintain their orientation during navigation tasks. While each of these

techniques is well suited for showing detail and overview of the hierarchy structure, they are not created with the intention of viewing the content of nodes within hierarchies.

Information-Slices

Another example of focus+context for space-filling visualizations is the Information-Slices technique [1] for displaying the details of substructures within a hierarchy (Figure 2.10). As the user clicks on a node, its representative sub-structure

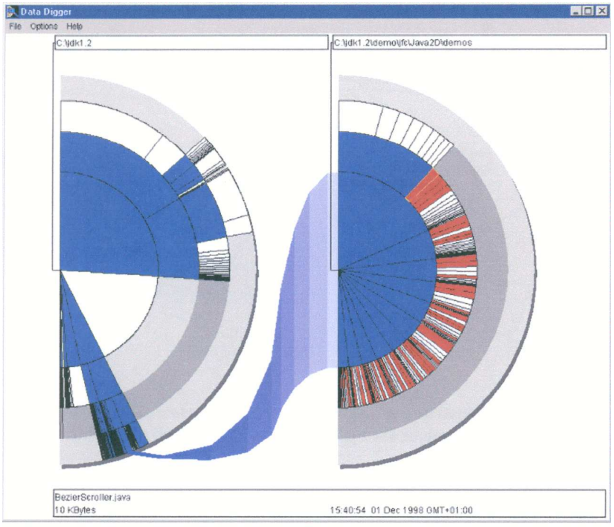


Figure 2.10: Information-Slices [9].

is opened using a semi-circle presentation and linked by an arc. As in the focus+context techniques for the Sunburst, the semi-circle is designed primarily to

facilitate detail viewing of substructures within a hierarchy.

In both the visualizations described above, focus is created by "fanning-out" and enlarging the subtree of interest. Context is provided by presenting the original hierarchy in the same view and by visually depicting the relation between the subtree and its location in the original hierarchy. Radial space-filling visualizations have the added advantage of being capable of showing the hierarchical structure more explicitly than other space-filling representations such as the TreeMap. Therefore replicating the same type of focus+context techniques onto the TreeMap may not lead to a visually comprehensible presentation. In addition, the use of fisheye views in the TreeMap will facilitate content browsing, which has not been implemented in the space-filling techniques discussed above.

2.2 Interaction

Presentation methods and interaction techniques are highly complementary in visualization tools. Interaction allows users to find specific information of interest that is being presented. Without interaction, users cannot take full advantage of the presentation. Interactions come in many forms which include browsing and searching. In this thesis, the techniques that have been developed for browsing and searching in the TreeMap will be discussed later. A review of the research on

browsing, zooming, and searching techniques is discussed in the sections below.

2.2.1 Browsing Techniques Applied to the TreeMap

Several interaction techniques have been developed for browsing information content in the TreeMap structure. Brushing is a technique that uses highlights to isolate important items in a dataset. Quantum TreeMap is a technique that improves the classical TreeMap, and Photomesa is an application that helps users browse and organize photos. All of them are used for browsing information content in the TreeMap.

Brushing

Brushing for example, is an effective technique for browsing information and performing exploratory data analysis. In particular, brushing is a technique which assists the user in selectively isolating subsets of data for exploration and inspection. Fua et al. [7] designed an interactive structure-based brushing technique which can be used to perform selection in hierarchical datasets. Structure-based brushing was applied to TreeMaps to assist users in selecting clusters of interest using a structure-based coloring [7]. Using this tool, users can specify regions of interest based on the location or depth of the clusters of interest in the hierarchy. By dy-

namically selecting a bounding region in the structure-based brush, corresponding elements in the TreeMap get highlighted. Using such a technique clusters of interest (based on the variable being mapped in the display) can be selectively inspected. This technique however, does not display the content of nodes in the TreeMap.

Quantum TreeMap

The primary variation of the TreeMap that has facilitated direct browsing of hierarchical content is the Quantum TreeMap [5]. Quantum TreeMaps were designed to facilitate browsing of entities in a hierarchy that consist of 'quantum' or indivisible size, such as images. An integral component of the quantum TreeMap is an ordered layout algorithm which maximizes the amount of space available for the nodes in the hierarchy by rearranging the display based on the size of the elements.

Photomesa

Photomesa (Figure 2.11) [2], an application based on the Quantum TreeMap, displays a thumbnail of all images in a directory. The basic mechanism for browsing images or other content is achieved by hovering or zooming into thumbnails of interest. Smooth animation between different endpoints in the zooming operation facilitates context viewing. However, the objects zoomed into overlap the Quantum TreeMap and occlude parts of the display region. Furthermore, to navigate

Figure 2.11: PhotoMesa [2]

or browse from one region to another the user needs to roll out (move out from the current viewing space) and drill back (move into, which includes panning or scrolling) into the area of interest. Another characteristic of the Quantum TreeMap is that the underlying hierarchical structure of the information is collapsed onto a flattened view to facilitate the task of browsing. The motivation to flatten the hierarchical structure is based on the assumption that users are typically interested in groups of items and not the inter-structural relationships between these.

In the techniques described above, drilling down into levels of the hierarchy is the prevalent form of navigation. Brushing provides a mechanism for selecting and filtering items of interest based on its visual property (such as color mapping) or depth in hierarchy. In the case of the Quantum TreeMaps, hierarchical structures

with elements of heterogeneous sizes and content may not be easily browse-able. An implicit requirement for the technique I introduce is to allow users to investigate node content in the tree without any transformation on the underlying hierarchical structure. In addition, a requirement for browsing content in the TreeMap would be to allow access to content in the structure without the need of traversing all the layers in the hierarchy. The solution to this restriction was derived from the results on the work in *continuous semantic zooming*, described next.

2.2.2 Continuous Semantic Zooming

The work conducted in this thesis is primarily inspired by the concept of continuous semantic zooming (CSZ) developed by Schaffer et al [20]. This technique is characterized by two distinct but interrelated components: continuous zooming [4] and presentations of semantic content at various stages of the zoom operation. CSZ manages a 2D display by recursively breaking it up into smaller areas. A region of interest becomes the focus and as the continuous zoom is applied, successive layers of a display "open up" (Figure 2.12). At each level of the operation the technique enhances continuity through smooth transitions between views and maintains location constraints to reduce the user's sense of spatial disorientation. The amount of detail shown in parts of the display is controlled by pruning the display and

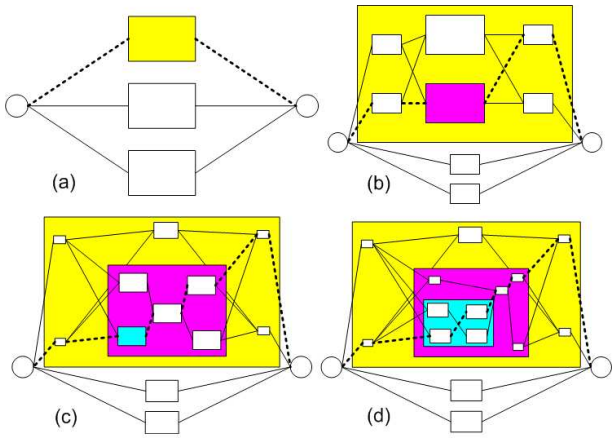


Figure 2.12: Continuous Semantic Zooming applied to a clustered network. Details of each cluster are visible based on the level of zooming employed. Progressing from (a) to (d) reveals more details as smooth transitions are created between views [20].

presenting items of non interest in summary form. A study comparing continuous semantic zooming to drill-down (full zoom), on a network of hierarchical clusters, shows that users can navigate and perform tasks related to node-link diagrams more efficiently with CSZ than with the traditional approach [20].

Continuous semantic zooming has been applied to information structures other than topological graphs. Datelens [3] employs continuous semantic zooming to reveal varying degrees of content in tabular structures in a smooth and continuous manner. The distortion in Datelens is linear and is applied to the cells of interest in a grid. As the level of distortion increases semantic information is revealed based on

the size of the region available for the display. An evaluation comparing Datelens to common calendar based interactions reveals that continuous semantic zooming enhances content browsing in tabular structures [3]. The distortion algorithm used in Datelens cannot be directly applied to the TreeMap as the alignment of cells in the TreeMap is not symmetrical. Furthermore, the TreeMap uses a hierarchical and not a tabular structure.

The approach I have implemented is similar in concept to the continuous semantic zoom: smooth transition between zoom levels is applied and content visibility is increased as the nodes enlarge.

2.2.3 Search and Search Results Visualization

Another form of interaction is searching. Search techniques are widely used in many areas of interaction such as in file system search, Internet search, etc. Generally, search includes three steps: users submit search keywords to the system, the system searches the data base, and finally search results are presented to the users. In most cases, the search results are presented in plain text to the users. This requires scanning and scrolling the results. Furthermore, users have difficulty in knowing the priority of one result and the relationship between two search results. Some techniques such as Motion Queries [26], WaveLens [16], and Lighthouse [14], have

been developed to solve these problems.

Motion Queries

C. Ware and R. Bobrow [26] describe an interactive technique which supports visual queries on graphs containing up to a few thousand nodes. In a medium sized node-link graph which contains more than 20 or 30 nodes (figure 2.13), a main

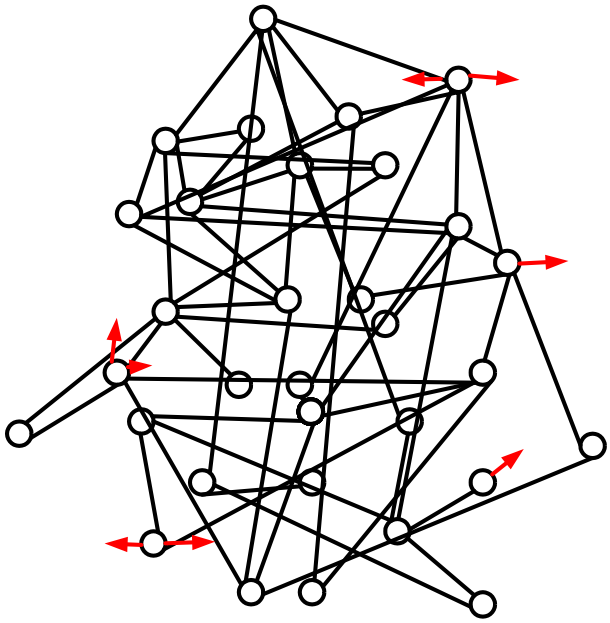


Figure 2.13: Medium sized node-link graph. Motion of the nodes are indicated by the arrows.

problem [26] is that the nodes and links in the graph become visually incomprehensible. Based on this kind of graph, the targets of query actions are a group of nodes which link each other. Therefore, how to represent the query results in a

clearly visible way is the major objective of the work in [26].

In Ware and Bobrow's solution, simple motion is used to highlight the query results. To achieve their purpose, they demonstrate five methods which are static highlighting and four motion highlighting methods (circular, jolt, crawl, and expanding nodes). In three experiments, they record response time and accuracy of the participants in locating query results using the five methods. The results suggest that querying large graphs, motion highlighting is more efficient and more accurate than static highlighting. In this research I extend the results of Ware and Bobrow to showing search results in the TreeMap. In particular I develop algorithms to use simple motion by distorting the nodes that are relevant to the search results.

WaveLens

With most Internet search engines, search results are displayed as a linear list of text. In most case, if the searching engine can return more than just a few lines of each page which is found, users will find the particular web page easier. Typically, users want to view enough quantities of results to compare each result in the limited screen space. To view linear lists, scrolling is the fundamental interaction technique available. However, scrolling can be ineffective for very large lists. Paek et al. [16] designed a system referred to as the WaveLens to address these problems by using

fish eyes representations.

WaveLens shows each search result in just a few numbers of lines, similar to a normal search engine list. However, this list is compressed such that scrolling is kept to a minimum. When the mouse cursor moves and hovers over one of the result items, a fish eye lens is applied vertically in the page, and this result becomes the focus of the fish eye lens. For the focused result, WaveLens fetches more samples from the target web page and shows them on the screen and compresses other result items to save the screen space. In addition, WaveLens magnifies the item of interest by using a large font, and minimizes the furthest result item from the focused item using a small font. Moving the mouse cursor over another result will change the focus and magnify the newly selected information instead.

Paek et al. compared the WaveLens technique with normal static search results list. Their studies show that with WaveLens, participants complete search tasks more quickly and more accurately [16]. In my research, I will also apply some of the concepts used in the Wavelens system, in particular the idea of distorting the display to show items of relevance in the search result set.

Lighthouse

Lighthouse [14] (Figure 2.14) is an interface for a Web-based information retrieval

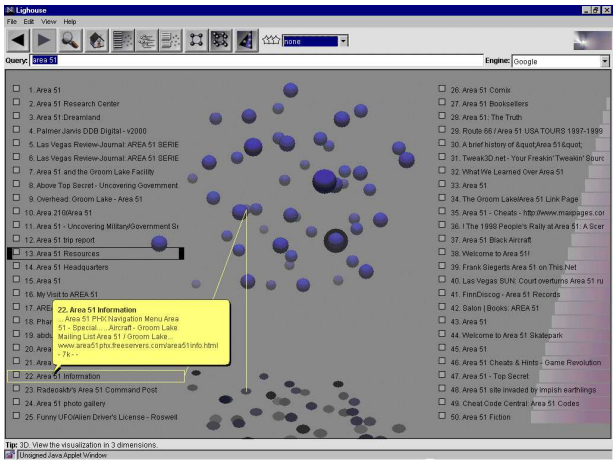


Figure 2.14: LightHouse

system. In normal search engines, result items are in a list of text and sorted by the similarity index to the query terms (usually no more than 10 items per page). Normally, users open the link of each result item and judge whether the web page is the correct target. If not, the user will go back to the result items list and open another. For most users, the necessary sequence of actions are tedious and unproductive, therefore they often stop looking after browsing the first set of results without achieving their expected goal. Furthermore, search engines do not show the similarity levels between two or more result sets. Consequently, users may repeatedly open pages that are similar in content.

Lighthouse is designed to address these problems. In lighthouse, the first 50 ranked results are listed to the left and right side of the screen. To save space, only

the title of each result is shown. Up to 50 spheres are used and are floating in the middle of the screen, which correspond to the search results. The structure of the spheres are grouped and the position of each sphere represents the ranks and the relationships of the search results. The spheres with higher ranks are closer to the users and have larger sizes (can cover spheres with lower ranks). Two spheres that are very similar to each other will be located near one another. While the mouse cursor hovers upon a sphere, a popup window which contains a brief introduction of the target page will appear. Thus, users can decide to click the sphere to view the target page or view other spheres. Studies have shown that users are more successful with Lighthouse than with normal text listing methods.

The concept of simple motion as used by Ware and Bobrow [26], the fisheye method used in Wave Lens [16], and the use of relevance order as demonstrated in Lighthouse [14] have been adopted for displaying search results in the TreeMap, as discussed in Chapters 3 and 4.

Chapter 3

Algorithms

This chapter describes the distortion algorithm I have implemented. The distortion algorithm modifies the sizes of the blocks for each node of interest (target node) and its neighbours dynamically. The distortion algorithm is composed of computing neighbours, changing node size, and distorting nodes. I applied the distortion algorithm on browsing single node (uni-distortion) and representing multiple search results (multi-distortion).

3.1 Uni-Distortion Technique

To facilitate rapid browsing of node contents within a space-filling representation of hierarchies, such as the TreeMap, I designed algorithms that apply the

focus+context and continuous zooming techniques described in Chapter 2. This technique is significantly different than the current drill-down approach provided in the TreeMap representation. The behavior of the algorithm is depicted in figure 3.1 below. The content of the node appears as the node of interest grows. The

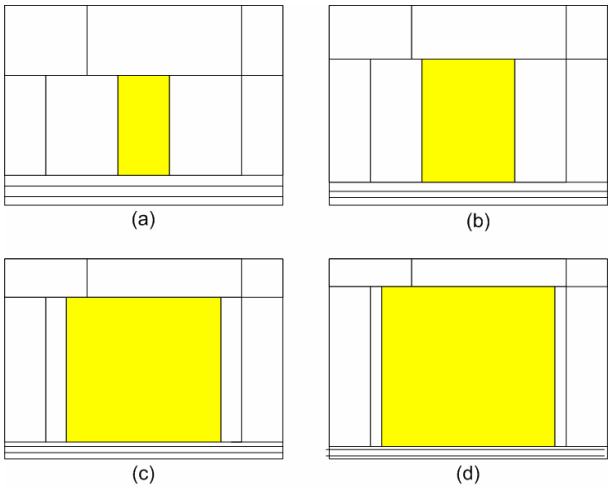


Figure 3.1: Distortion can be applied to nodes of interest. As the node expands, content data is revealed. From rest (a) to full expansion of a selected node (d).

expansion is triggered as the user selects a node with the cursor and until the node remains selected, i.e. the user clicks down with the mouse and does not release it. This is analogous to the continuous zooming technique which expands nodes and magnifies regions of interest. The major difference here being that in this technique the magnification of a node causes other nodes to contract or decrease in size. The concept of showing node content as it increases is similar to the idea of semantic

zooming [20] where additional information is provided as the user magnifies the object. This is different than regular zooming which simply magnifies the geometry of the object. In the following sections I describe the various algorithms developed for achieving the distortion effect. I first describe the data structures that were necessary for the algorithms.

3.2 Data Structure

Each displayable item in the TreeMap is represented as a node in the tree. The algorithm for displaying a TreeMap is well-known and described in [21]. Each node must contain enough information so that it can be redrawn as needed. In addition, each node must contain additional information, which will enable detection of neighbours and proper distortion of nodes.

We define a node to contain the following information:

- *size (weight)*: indicates the size of a node. For a data file, this may be the size of a file. For a directory, this may be the cumulative size of all the files contained within it.

- *width, height*: the width and height of the node when it is drawn on the screen. These values are computed based on its size and the size of its parent

node.

- *amount*: the increment of distortion for each step.

- *orientation*: this field determines whether the node is displayed horizontally or vertically with respect to its siblings in the TreeMap. Nodes at the same level have the same orientation, and the orientation alternates between levels of the TreeMap.

- *parent*: this field is a pointer to the parent node.

- *prevSibling, nextSibling*: these are pointers to the node's previous and next siblings. A node's previous sibling is the sibling that is drawn to the left of it, if the orientation is horizontal and is the sibling that is drawn above it, if the orientation is vertical. A node's next sibling is the sibling that is drawn to the right of it, if the orientation is horizontal and is the sibling that is drawn below it, if the orientation is vertical.

- *children*: this is an array of pointers to each of its children.

- *leftNeighbour*: this pointer is set to its left sibling (in the TreeMap) which contains the node of interest, if this is the "right-neighbour" of the node of interest. Otherwise it is set to NULL. The *rightNeighbour, topNeighbour, bottomNeighbour* member variables are defined in a similar manner. This

variable is described in more detail in the next section.

In addition to the data structure for a node, there are several other important data items in the distortion algorithm. The root of the tree is denoted by *ROOT*. The constant *MIN_SIZE* will denote the smallest width or height that a node can shrink to. The constant *MAX_SIZE* denotes the largest width or height that a node can grow to. Finally, we have four global variables *gLeftNeighbour, gRightNeighbour, gTopNeighbour, gBottomNeighbour* which denote the left, right, top and bottom neighbours of the node of interest. These four variables represent the nodes that will shrink to make room for the node of interest.

3.3 Computing Neighbours

When a node A has been selected as the "node of interest", that is, the node that is to be distorted, the first step of the distortion process is to compute the neighbours of A. A *neighbour* of a node A is any other node B such that:

1. A and B have overlapping borders in the TreeMap,
2. B is not an ancestor of A, and
3. B is as near the root node as possible.

Suppose A has a *left-neighbour* B; that is, B is a neighbour of A where B's right border and A's left border intersect. Furthermore, suppose C is some other node where C's right border and A's left border intersect and C is not an ancestor of A. Then, by the above definition, it can easily be seen C must be an ancestor of B. This is similarly true for the *right-neighbour, top-neighbour and bottom-neighbour* of A, which are defined similarly as left-neighbour. Thus, using this definition, any node A has at most one *left-neighbour, right-neighbour, top-neighbour, and bottom-neighbour*.

Suppose that B is the left-neighbour of A, it is useful to keep track of the ancestor C of A such that B and C are at the same level in the tree, which is stored in *B.rightNeighbour*. Clearly, B and C are siblings. This information is useful when applying the distortion algorithm, which is explained below. It is easy to see that C must be *B.nextSibling*. Note that it is possible for A and B to be at the same level, in which case C is A, as *B.nextSibling* is A. Similarly, we can define the right-neighbour, top-neighbour and bottom-neighbour. It should be noted that the four neighbours of a node A must be distinct and it is possible for node A to not have a neighbour.

3.3.1 Compute Left Neighbour

Algorithm 1 computes the left-neighbour of a node A. Since the left-neighbour of A is to the left of A, we check to see if *A.PrevSibling* is *NULL* only when the orientation is **horizontal**.

Algorithm 1 ComputeLeftNeighbour(A)

```
if A ≠ ROOT then
  if A.orientation = HORIZ and A.PrevSibling ≠ NULL then
    LeftNeighbour ← A.PrevSibling
    LeftNeighbour.RightNeighbour ← A
    return(LeftNeighbour)
  else
    return(ComputeLeftNeighbour(A.parent))
  end if
else
  LeftNeighbour ← NULL
  return(LeftNeighbour)
end if
```

When *ComputeLeftNeighbour* is called with the "node of interest", say A, the algorithm will determine its left-neighbour B such that A's left border intersects B's right border. The routine will return B, and *B.RightNeighbour* is set to *B.NextSibling*, which is A, if A and B are at the same level, or else an ancestor of A.

3.3.2 Compute Right Neighbour

Algorithm 2 computes the right-neighbour of a node A. Since the right-neighbour of A is to the right of A, we check to see if *A.NextSibling* is *NULL* only when the orientation is **horizontal**.

Algorithm 2 ComputeRightNeighbour(A)

```
if A ≠ ROOT then
  if A.orientation = HORIZ and A.NextSibling ≠ NULL then
    RightNeighbour ← A.NextSibling
    LeftNeighbour.RightNeighbour ← A
    return(RightNeighbour)
  else
    return(ComputeRightNeighbour(A.parent))
  end if
else
  RightNeighbour ← NULL
  return(RightNeighbour)
end if
```

When *ComputeRightNeighbour* is called with the node A, the algorithm will determine its right-neighbour B such that A's right border intersects B's left border. The routine will return B, and *B.LeftNeighbour* is set to *B.PrevSibling*, which is A, if A and B are at the same level, or else an ancestor of A.

3.3.3 Compute Top Neighbour

Algorithm 3 computes the top-neighbour of a node A. Since the top-neighbour of A is to the left of A in the tree structure, we check to see if *A.PrevSibling* is *NULL* only when the orientation is **vertical**.

Algorithm 3 ComputeTopNeighbour(A)

```
if A ≠ ROOT then
  if A.orientation = VERT and A.PrevSibling ≠ NULL then
    TopNeighbour ← A.PrevSibling
    TopNeighbour.BottomNeighbour ← A
    return(TopNeighbour)
  else
    return(ComputeTopNeighbour(A.parent))
  end if
else
  TopNeighbour ← NULL
  return(TopNeighbour)
end if
```

When *ComputeTopNeighbour* is called with the node A, the algorithm will determine its top-neighbour B such that A's top border intersects B's bottom border. The routine will return B, and *B.BottomNeighbour* is set to *B.NextSibling*, which is A, if A and B are at the same level, or else an ancestor of A.

3.3.4 Compute Bottom Neighbour

Algorithm 4 computes the bottom-neighbour of a node A. Since the bottom-neighbour of A is to the right of A in the tree structure, we check to see if *A.NextSibling* is *NULL* only when the orientation is **vertical**.

Algorithm 4 ComputeBottomNeighbour(A)

```
if A ≠ ROOT then
  if A.orientation = VERT and A.NextSibling ≠ NULL then
    BottomNeighbour ← A.NextSibling
    BottomNeighbour.TopNeighbour ← A
    return(BottomNeighbour)
  else
    return(ComputeTopNeighbour(A.parent))
  end if
else
  BottomNeighbour ← NULL
  return(BottomNeighbour)
end if
```

When *ComputeTopNeighbour* is called with the node A, the algorithm will determine its bottom-neighbour B such that A's bottom border intersects B's top border. The routine will return B, and *B.TopNeighbour* is set to *B.PrevSibling*, which is A, if A and B are at the same level, or else an ancestor of A.

3.4 Changing Node Size

When the size of a node is modified, this needs to be propagated to all its children. All children nodes change their size according to the original proportion. The recursive propagation algorithm is described in algorithm 5.

Algorithm 5 ChangeNodeSize(A, δ)

$n \leftarrow$ number of children of A
$oldsize \leftarrow A.size$
$A.size \leftarrow A.size + \delta$
for $i = 0$ to $n - 1$ **do**
  ChangeNodeSize($A.children[i], (oldsize + \delta) * A.children[i].size/oldsize$)
end for

3.5 Distorting Neighbours

The following algorithm decreases the size of the neighbors on each side of the target node.

3.5.1 Distort Left Neighbour

Algorithm 6 decreases the size of the left-neighbour of the node of interest. This left-neighbour is given by *gLeftNeighbour* and initially computed by *ComputeLeft-Neighbour*. The algorithm decreases the size of the left-neighbour by *amount* and

increases *gLeftNeighbour.RightNeighbour* by the same amount. This ensures that the overall size of the TreeMap is not changed, and the node of interest's size, which is a descendent of *gLeftNeighbour.RightNeighbour*, is increased. If node A is in-

Algorithm 6 DistortLeft(*amount*)

```
global gLeftNeighbour {Executes on iteration of the distortion algorithm}
if gLeftNeighbour ≠ NULL then
  if gLeftNeighbour.width > MIN_WIDTH then
    ChangeNodeSize(gLeftNeighbour, −amount)
    ChangeNodeSize(gLeftNeighbour.RightNeighbour, amount)
  else
    if gLeftNeighbour.PrevSibling ≠ NULL then
      temp ← gLeftNeighbour.RightNeighbour
      gLeftNeighbour ← gLeftNeighbour.PrevSibling
      gLeftNeighbour.RightNeighbour ← temp
    else
      gLeftNeighbour ← ComputeLeftNeighbour(gLeftNeighbour)
    end if
  end if
end if
```

fluenced by algorithm 6 such that its width is less than *MIN_WIDTH* (since the orientation of the node is **horizontal**), then propagation is applied. This involves computing the left-neighbour of node A. There are two possible scenarios. If A has a sibling to the left of it, then this sibling is now the left neighbour of the node of

interest. Otherwise, we need to compute the left neighbour of this node A using the *ComputeLeftNeighbour* method.

3.5.2 Distort Right Neighbour

Algorithm 7 decreases the size of the right-neighbour of the node of interest.

Algorithm 7 DistortRight($amount$)

```
global gRightNeighbour {Executes on iteration of the distortion algorithm}
if gRightNeighbour ≠ NULL then
  if gRightNeighbour.width > MIN_WIDTH then
    ChangeNodeSize(gRightNeighbour, −amount)
    ChangeNodeSize(gRightNeighbour.LeftNeighbour, amount)
  else
    if gRightNeighbour.NextSibling ≠ NULL then
      temp ← gRightNeighbour.LeftNeighbour
      gRightNeighbour ← gRightNeighbour.NextSibling
      gRightNeighbour.LeftNeighbour ← temp
    else
      gRightNeighbour ← ComputeRightNeighbour(gRightNeighbour)
    end if
  end if
end if
```

This right-neighbour is given by *gRightNeighbour* and initially computed by *ComputeRightNeighbour*. The algorithm decreases the size of the right-neighbour by

amount and increases *gRightNeighbour.LeftNeighbour* by the same amount. This ensures that the overall size of the TreeMap is not changed, and the node of interest's size, which is a descendent of *gRightNeighbour.LeftNeighbour*, is increased.

If node A is influenced by algorithm 7 such that its width is less than *MIN_WIDTH* (since the orientation of the node is **horizontal**), then propagation is applied. This involves computing the right-neighbour of node A. There are two possible scenarios. If A has a sibling to the right of it, then this sibling is now the right neighbour of the node of interest. Otherwise, we need to compute the right neighbour of the node A using the *ComputeRightNeighbour* method.

3.5.3 Distort Top Neighbour

Algorithm 8 decreases the size of the top-neighbour of the node of interest. This top-neighbour is given by *gTopNeighbour* and initially computed by *ComputeTopNeighbour*. The algorithm decreases the size of the top-neighbour by *amount* and increases *gTopNeighbour.BottomNeighbour* by the same amount. This ensures that the overall size of the TreeMap is not changed, and the node of interest's size, which is a descendent of *gTopNeighbour.BottomNeighbour*, is increased.

If node A is influenced by algorithm 8 such that its width is less than *MIN_HEIGHT* (since the orientation of the node is **vertical**), then propagation is applied. This

involves computing the top-neighbour of node A. There are two possible scenarios. If A has a sibling to the left of it, then this sibling is now the top neighbour of the node of interest. Otherwise, we need to compute the top neighbour of this node A using the *ComputeTopNeighbour* method.

Algorithm 8 DistortTop(*amount*)

```
global gTopNeighbour {Executes on iteration of the distortion algorithm}
if gTopNeighbour ≠ NULL then
  if gTopNeighbour.height > MIN_HEIGHT then
    ChangeNodeSize(gTopNeighbour, −amount)
    ChangeNodeSize(gTopNeighbour.BottomNeighbour, amount)
  else
    if gTopNeighbour.PrevSibling ≠ NULL then
      temp ← gTopNeighbour.BottomNeighbour
      gTopNeighbour ← gTopNeighbour.PrevSibling
      gTopNeighbour.BottomNeighbour ← temp
    else
      gTopNeighbour ← ComputeTopNeighbour(gTopNeighbour)
    end if
  end if
end if
```

3.5.4 Distort Bottom Neighbour

Algorithm 9 decreases the size of the bottom-neighbour of the node of interest. This bottom-neighbour is given by *gBottomNeighbour* and initially computed by *ComputeBottomNeighbour*. The algorithm decreases the size of the bottom-neighbour

Algorithm 9 DistortBottom(*amount*)

```
global gBottomNeighbour {Executes on iteration of the distortion algorithm}
if gBottomNeighbour ≠ NULL then
  if gBottomNeighbour.height > MIN_HEIGHT then
    ChangeNodeSize(gBottomNeighbour, −amount)
    ChangeNodeSize(gBottomNeighbour.TopNeighbour, amount)
  else
    if gBottomNeighbour.NextSibling ≠ NULL then
      temp ← gBottomNeighbour.TopNeighbour
      gBottomNeighbour ← gBottomNeighbour.NextSibling
      gBottomNeighbour.TopNeighbour ← temp
    else
      gBottomNeighbour ← ComputeBottomNeighbour(gBottomNeighbour)
    end if
  end if
end if
```

by *amount* and increases *gBottomNeighbour.TopNeighbour* by the same amount. This ensures that the overall size of the TreeMap is not changed, and the node of interest's size, which is a descendent of *gBottomNeighbour.TopNeighbour*, is in-

creased.

If node A is influenced by algorithm 9 such that its width is less than *MIN_HEIGHT* (since the orientation of the node is **vertical**), then propagation is applied. This involves computing the bottom-neighbour of node A. There are two possible scenarios. If A has a sibling to the right of it, then this sibling is now the bottom neighbour of the node of interest. Otherwise, we need to compute the bottom neighbour of this node A using the *ComputeBottomNeighbour* method.

3.6 Uni-Distortion Algorithm

The distortion algorithm increases the size of a node of interest while shrinking its neighbours. While the user clicks on the node of interest. The node grows as the mouse selection is maintained and returns to its original size upon release of the mouse selection. The contents are revealed gradually as the node grows in size. Instead of presenting only a subset of the tree during the exploration operation (as is the case with the drill-down), in this approach the user can continuously select items from any location in the hierarchy and inspect their contents. Traversing layers of the hierarchy is thereby removed.

Algorithm 10 describes the final distortion algorithm. It begins by determining the neighbours of the node of interest, A. Then it decreases each of the sizes of

these neighbours of A, while increasing the size of A which provides the distortion effect. Then the entire TreeMap is redrawn. This process is repeated until an external event stops the distortion process, or the node of interest being increased has reached a fixed maximum width or height.

Algorithm 10 DistortAlgorithm(A)

```
global gLeftNeighbour, gRightNeighbour
global gTopNeighbour, gBottomNeighbour
gLeftNeighbour ← ComputeLeftNeighbour(A)
gRightNeighbour ← ComputeRightNeighbour(A)
gTopNeighbour ← ComputeTopNeighbour(A)
gBottomNeighbour ← ComputeBottomNeighbour(A)
while DISTORTING = true and A.width < MAX_SIZE and A.height < MAX_SIZE do
    DistortLeft(amount)
    DistortRight(amount)
    DistortTop(amount)
    DistortBottom(amount)
    RedrawTreeMap(ROOT)
    Sleep(sleep_interval)
end while
```

Figure 3.2 below shows the effect of clicking and releasing a node. The content appears as the node is being opened and the distortion appears gradually. The next section describes the multi-distortion algorithm used for showing search results.

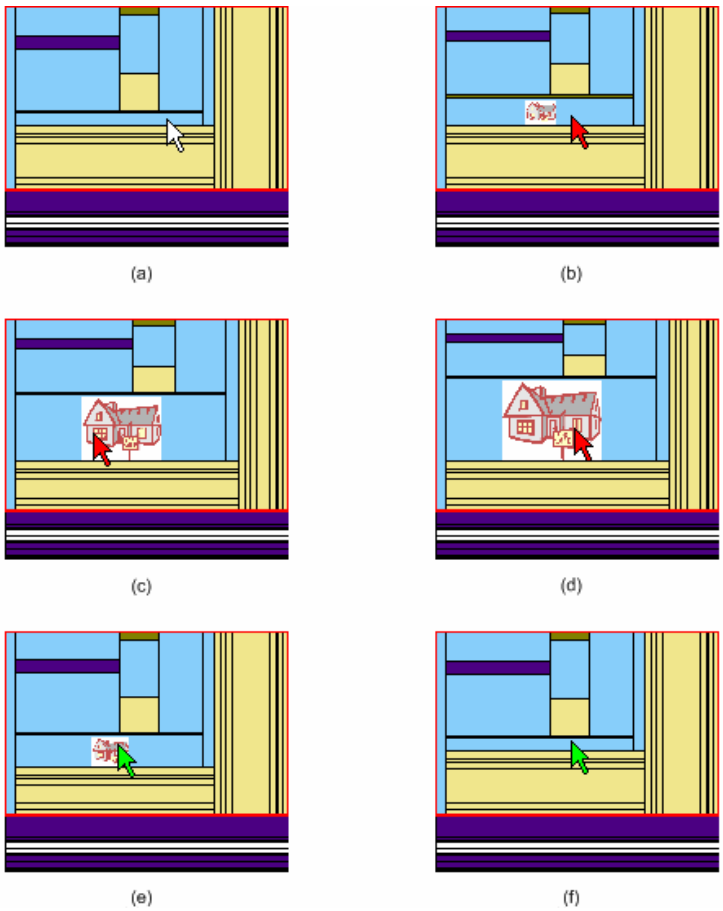


Figure 3.2: Implementation results of the uni-distortion algorithm. The red arrow indicates selection (b)-(d) and the green arrow indicates release of the mouse button (e) and (f).

3.7 Multi-Distortion Technique

The distortion of the nodes in the TreeMap can also be applied to the representation of search results. For instance, after viewing the TreeMap representation, users may typically be interested in searching for items with specific content (for example, files with certain keywords). Typically this type of interaction will result in obtaining a set of result items with each item having a degree of importance. In this case, the distortion technique described earlier will be applied to multiple nodes simultaneously. The new algorithm is a special case of the general algorithm described earlier.

Some major differences between the multi-distortion technique and the uni-distortion technique are as follows. In the multiple distortion algorithm, a target node cannot become a neighbour of other target nodes, and one node cannot become a neighbour of two target nodes. Furthermore, additional mappings will be used in the multiple distortion algorithm. The distortion method has one primary attribute: the amount of distortion (amplitude). This attribute can be used to distinguish the levels of significance for each result. Therefore, significance is assigned to the amplitude of the distortion. For instance, a distortion with a large amplitude can imply more important content than a distortion with a small amplitude.

The behavior of the multi-distortion algorithm is depicted in figure 3.3. Figure

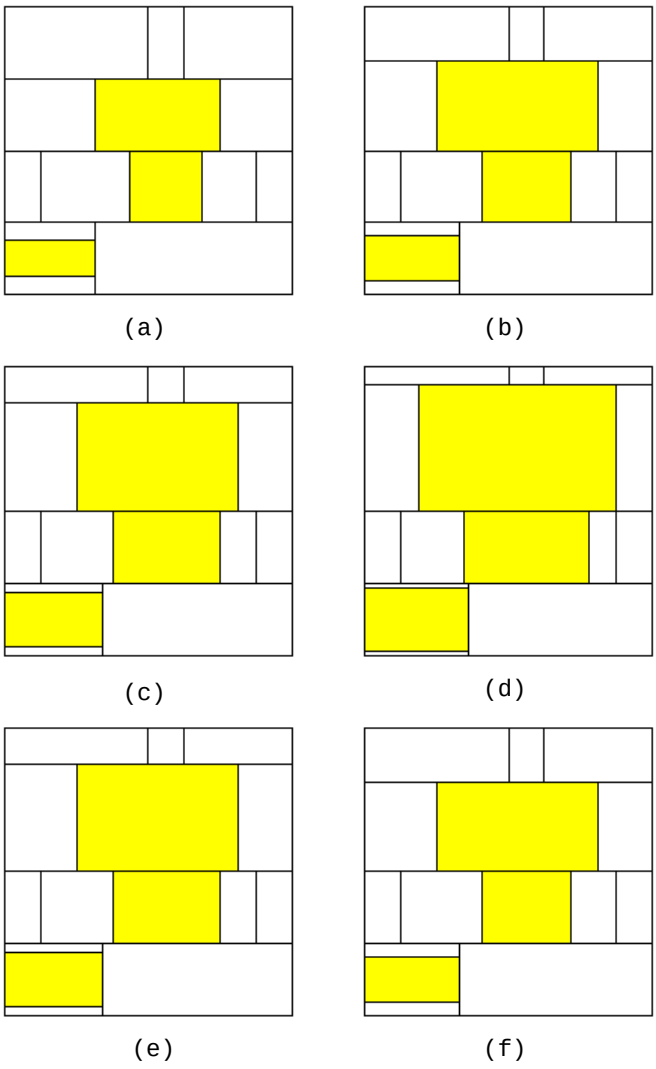


Figure 3.3: The behavior of the multi-distortion algorithm. Figure (a) is the original state, figure (b)-(d) indicate the size of each node increased, figure (e)-(f) indicate the size of each node decreased. After state (f), the state of TreeMap goes back to (a).

3.3 (a) is the initial state of the TreeMap. The highlighted nodes are the search result nodes which will get distorted. Figure 3.3 (b) to (d) shows that the size of the nodes increased due to the distortion. The amplitude of the distortion indicates the priority of the node. Larger amplitude represents higher priority. Figure 3.3 (e) to (f) show that the size of nodes decrease because of the distortion. After state (f), the TreeMap returns back to state (a).

3.8 Multi-Distortion Algorithm

Algorithm 11 describes the distortion of one node in the multi-distortion technique. It starts by determining the neighbours of the node of result, A. If the neighbours are not other result nodes or neighbours of other result nodes, it decreases the size of these nodes. The size of A increase at the same time to provide the distortion effect. Then the entire TreeMap is redrawn. This process is repeated until an external event stops the distortion process or the node has reached a fixed maximum width or height.

Algorithm 12 describes the final multi-distortion algorithm. $A_1, A_2, ..., A_n$ are search result nodes which should be distorted. Algorithm 11 is applied one by one on these result nodes to generate multiple distortions.

Algorithm 11 OneNodeinMultiDistortAlgorithm(A)

global $gLeftNeighbour, gRightNeighbour$

global $gTopNeighbour, gBottomNeighbour$

bool isResult {Whether a node is a search result}

bool isNeighbour {Whether a node is a neighbour of another search result}

$gLeftNeighbour \leftarrow$ ComputeLeftNeighbour(A)

$gRightNeighbour \leftarrow$ ComputeRightNeighbour(A)

$gTopNeighbour \leftarrow$ ComputeTopNeighbour(A)

$gBottomNeighbour \leftarrow$ ComputeBottomNeighbour(A)

while $DISTORTING = true$ and $A.width < MAX\_SIZE$ and $A.height < MAX\_SIZE$ **do**

  if not $gLeftNeighbour.isResult$ and not $gLeftNeighbour.isNeighbour$ **then**

    DistortLeft($amount$)

  end if

  if not $gRightNeighbour.isResult$ and not $gRightNeighbour.isNeighbour$ **then**

    DistortRight($amount$)

  end if

  if not $gTopNeighbour.isResult$ and not $gTopNeighbour.isNeighbour$ **then**

    DistortTop($amount$)

  end if

  if not $gBottomNeighbour.isResult$ and not $gBottomNeighbour.isNeighbour$ **then**

    DistortBottom($amount$)

  end if

  RedrawTreeMap($ROOT$)

  Sleep($sleep\_interval$)

end while

Algorithm 12 MultiDistortAlgorithm($A_1, A_2, ..., A_n$)

OneNodeinMultiDistortAlgorithm(A_1)

OneNodeinMultiDistortAlgorithm(A_2)

...

OneNodeinMultiDistortAlgorithm(A_n)

Figure 3.4 shows the implementation results of multi-distortion algorithm. Figure 3.4 (a) shows the initial state of the TreeMap. Figure 3.4 (b) shows the TreeMap being distorted. After the search keywords submitted, all result nodes distort in different amplitudes according to the relevance to the search keywords. According to the amplitude of the distortion, users can give an order of each search result. As shown in figure 3.4 (c), the labels on the nodes indicate the importance level of each node. The node with a label "1" has the highest importance level. After all search results are identified, the TreeMap goes back to (a).

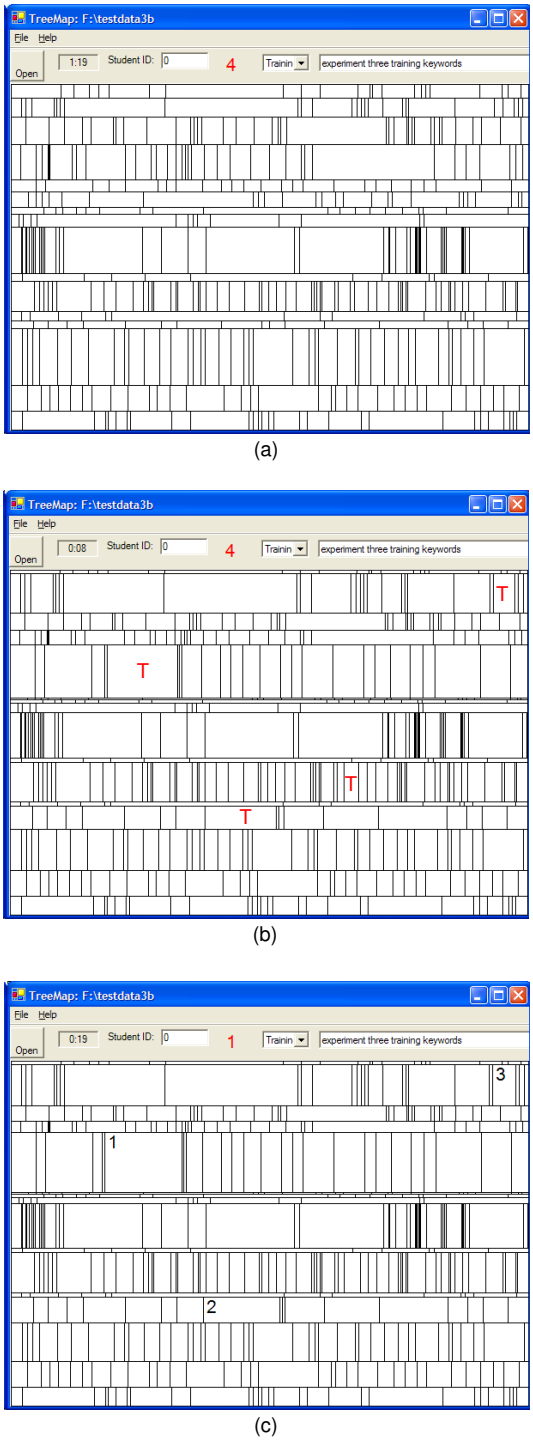


Figure 3.4: Implementation results of the multi-distortion algorithm. (a) is the initial state. (b) shows the TreeMap is distorting (nodes with a "T" inside are target nodes). In figure (c), the labels on the nodes indicate the importance level. After all search results are identified, the TreeMap goes back to initial state.

Chapter 4

Evaluation

Three experiments were designed to evaluate the effectiveness of the distortion technique. Experiment 1 evaluate the effectiveness of the distortion technique in browsing. Experiment 2 evaluates the effectiveness of the distortion technique in browsing with context. Experiment 3 evaluates the effectiveness of the distortion technique in representing search results. The following results were anticipated:

4.1 Hypotheses

From the results of the research discussed in Chapter 2, I have formulated the following hypotheses.

- Hypothesis 1: Overall, users will locate the contents of interest faster via the distortion technique versus the drill-down technique.

- Hypothesis 2: In deep hierarchical structures, users will locate the targets faster via the distortion technique versus the drill-down technique.

- Hypothesis 3: In the distortion technique, performance will not differ between deep and wide hierarchies.

- Hypothesis 4: The distortion technique will allow users to maintain relationships among various areas of a TreeMap more efficiently than the drill-down technique.

- Hypothesis 5: With small results set (number of results ≤ 5), users will identify all search results faster via the distortion technique versus the highlight technique.

- Hypothesis 6: With small results set (number of results ≤ 5), users will identify all search results more accurately via the distortion technique versus the highlight technique.

- Hypothesis 7: With large results set ($5 <$ number of results ≤ 10) using the highlight technique, users can identify all search results faster than with the distortion technique.

- Hypothesis 8: With large results set ($5 <$ number of results ≤ 10) using the highlight technique, users can identify all search results more accurately than with the distortion technique.

4.2 Experiment One - Browsing

Experiment 1 was designed to compare the Drill-Down method (conventional browsing approach) to the Distortion method for locating specific content (pictures) in the TreeMap. In this experiment subjects were required to locate specific images, but alternatively I could have asked subjects to locate text file or any other type of data.

4.2.1 Method

Subjects

Twenty undergraduate students participated in the experiment and were assigned to one of the two conditions: Distortion first or Drill-Down first. Subjects were volunteers from a computer science course in human-computer interaction. All were familiar with the concept of file and directory structures and had reasonable experience performing standard file management routines. None had any previous

experience with the TreeMap although they were familiar with space-filling concepts and the TreeMap representation.

Materials

Two different types of hierarchy were used for the experiment: deep and wide. The deep hierarchy was constructed using six levels, with a maximum of three sub-directories per node. The wide hierarchy was created with a depth of three levels, and each node contained a maximum of six sub-directories. Both types of hierarchies, deep and wide, contained thirty different pictures each and more than three hundred files of various other types. To reduce learning effects, I used two sets of hierarchies (Set A and Set B) which were created with the same hierarchical structures but entirely different images and files. Half the participants started the experiment with the Drill-Down method and the other half started using the Distortion method. After completing the tasks in one set of hierarchies with one method, the participants switched onto the other set of hierarchies with the other method. All tasks in the experiment required that subjects locate a specific picture in the TreeMap.

Participants performed the experiment on a 17" monitor with a 1024×768 resolution. The prototype ran over Windows XP. The task was described to them before they began running the trials.

Procedure

Before starting the experiment, each subject got familiarized with both browsing techniques. The experiment started when the participant indicated that he or she was comfortable using the tool and its interface.

In each task, I randomly chose one picture as a target picture from all thirty pictures in a hierarchy, and displayed the target picture to the participant in a window outside of the TreeMap. Half of the image files used as targets were small and occupied only a small fraction of space on the display. The other half of the image files are bigger. The subject was required to browse the hierarchy until he/she located the target image in the TreeMap. The target image was available throughout the task. Figure 4.1 shows the interface used in experiment 1.

Each participant performed 3 trials with wide hierarchies and 3 trials with deep hierarchies in the following sequence W1, D1, W2, D2, W3, and D3, where W represents the wide hierarchies and D represents the deep hierarchies. The participant was free to end the trial if they could not locate the specified picture. A time limit was not imposed for this task. I recorded whether the participant located the correct target, whether the participant withdrew from the task, and the time to execute the task in all conditions. In summary, the whole experiment involved: 20 participants $\times$ 2 main conditions $\times$ 2 types of hierarchies $\times$ 3 trials =

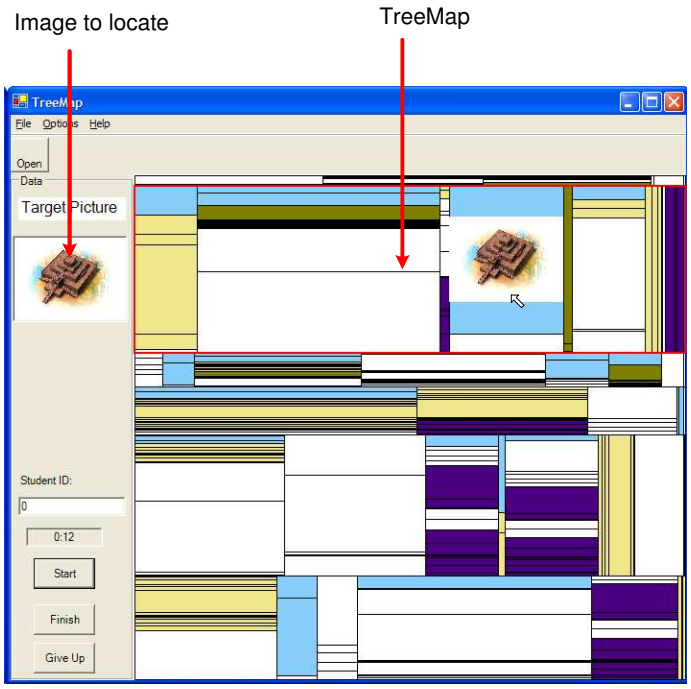


Figure 4.1: Interface used in experiment 1. When users click a node, the content of the node is shown (if the node is a picture file). Users are required to browse the TreeMap until he/she finds the target picture.

240 trials in total.

4.2.2 Results and Discussion

To test the first three hypotheses, I measured subjects' performance on the given task with respect to time until completion. I recorded the average response for locating the target. Of 240 trials to locate the target, 13 attempts were incomplete.

Of these 13 attempts to locate the target, 9 attempts resulted in giving up on the task. All 13 incomplete/give-up results were excluded from the data analysis.

The results are summarized in Table 4.1 (The detailed result of the analysis is in Appendix II). Average completion times were not consistent with the normality assumptions in both datasets (distortion or drill-down). The analysis was therefore performed on the log transform of the recorded performance times. The time to locate target data were analyzed by means of a 2 × 2 (Type of Method × Hierarchical Structure) one-way analysis of variance (ANOVA), with both Type of Method (Drill-Down vs. Distortion) and Hierarchical Structure (Deep vs. Wide) serving as repeated measures. An alpha level of 0.05 was used for all statistical tests. Type of Method was found to be significant ($F(1, 19) = 50.70$, $p < 0.001$) with the Distortion method group's mean task time (28.79 sec) being faster than the Drill-Down method group's (56.06 sec). The main effect for Hierarchical Structure was not statistically significant ($F(1, 19)=1.74$,$p = 0.20$). However, a significant interaction effect was found between Type of Method and Hierarchical Structure, $F(1, 19) = 5.10$, $p = 0.036$.

In conjunction with the means, it is clear that participants completed the task faster overall with the Distortion method vs. the Drill-Down method, regardless of conditions of Hierarchical Structure; this supports the first hypothesis. The

	Wide	Deep
Distortion	31.06 (12.68) sec	26.51 (13.02) sec
Drill-Down	47.63 (20.97) sec	64.48 (29.69) sec

Table 4.1: Average completion times for wide and deep hierarchies with both methods (standard deviations are in parentheses).

significant interaction tells us that the effect of Type of Method depends on the level of Hierarchical Structure. The simple effect of Method for the wide hierarchical structure clearly indicates that a faster mean task time is achieved via the Distortion method than the Drill-Down method (31.06 seconds vs. 47.63 seconds). The simple effect of Method for the deep hierarchy is more pronounced (26.51 seconds vs. 64.48 seconds). The 95% confidence intervals for the simple effects of Method at both levels of Hierarchical Structure are significantly different. Taken together with the significant main effect found for Type of Method, there is strong support for the second hypothesis, i.e. the Distortion Method will be faster than the Drill-Down Method, especially in deep trees.

A close observation of mean completion times for the distortion technique reveals that on average subjects are faster with deep hierarchies than with wide hierarchies. A paired sample T-Test shows this difference in means to be non-significant ($T(1,19)=1.412$, $p=0.174$), supporting hypothesis 3, i.e. subjects will

perform equally well on deep and wide hierarchies with the distortion technique.

We did not observe any differences in mean completion times for targets occupying a fraction of the display space. However, in certain cases we observed that participants ignored potential targets if these occupied a small amount of space. Less than 3.75% of the total number of trials consisted of participants missing small targets. In hierarchies with over thousand nodes this could potentially affect the performance of the distortion technique. A possible solution to alleviate this problem would consist of combining the drill-down to open a node (which increases the reserved space) followed by distortions.

The results of the first experiment suggest that the distortion technique is a better alternative than the conventional drill-down interaction used for browsing content in TreeMaps. However the first experiment does not test whether distortion facilitates browsing for content within some pre-specified context. Context can serve as an aid to browsing tasks by allowing the user to view the content of data in neighboring cells. The second experiment, described below, is designed to test whether context around a node can be examined more efficiently with the distortion approach.

4.3 Experiment Two - Browsing with Context

Focus+context techniques are designed with the aim to facilitate navigation or browsing of elements by presenting in the same view a cluster of items defined as the context. Experiment 2 was designed to compare the Drill-Down method to the Distortion method for locating an object within a pre-specified context. Context is defined as being a set of images spatially and hierarchically related in a certain configuration. As shown in Figure 4.2, the context for the target (e) is made up of

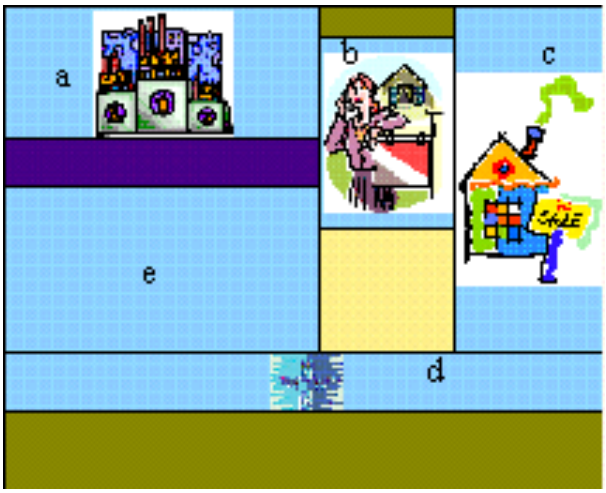


Figure 4.2: Context of image e (image not shown to participant) consists of images a, b, c, d, and four other files. Image a is a sibling of the target e, image b is in the sibling subtree of the target's parent, and image c is a sibling of the target's parent. Image d is a sibling to the subtree containing a, b, c, and e.

four images (a, b, c and d) and four other files. We hypothesize that participants will locate objects quicker and with fewer errors using the Distortion method over the Drill-down method in a task requiring context (Hypothesis 4).

4.3.1 Method

Subjects

Twelve graduate students participated in the experiment and were assigned to one of the two conditions: Distortion first or Drill-Down first. Subjects had a bachelor degree in either computer science or computer engineering. All were familiar with the concept of file and directory structures and had reasonable experience performing standard file management routines. None had any previous experience using the TreeMap and were not familiar with space-filling concepts.

Materials

As in experiment 1, two different types of hierarchy were used for the experiment: deep and wide. The deep hierarchy was constructed using six levels, with a maximum of three sub-directories per node. The wide hierarchy was created with a depth of three levels, and each node contained a maximum of six sub-directories.

Participants performed the experiment on a 17" monitor with resolution 1024×768

and ran the prototype over Windows XP. The task was described to them before they began the trials.

Procedure

Before starting the experiment, each subject got familiarized with both browsing methods. Once each participant indicated that he or she was comfortable using the tool and its interface, the experiment started.

The task, defined as the context browsing task, consisted of locating a picture within a preconfigured context. The target image was not shown to the user as we wanted the subject to identify the target based on its neighbouring images and their interrelations. In this experiment, context is defined by the spatial arrangement and structural relation of objects with respect to the target. Figure 6 is a sample context for node (e) used in the experiment. This task is similar in concept to the sub-structure identification task defined in [11]. By defining such a task, subjects would need to visually maintain the relative positions and relationships between files while browsing for the target image. Figure 4.3 shows the interface used in experiment 2.

Each participant performed the task with three different deep hierarchies and three different wide hierarchies using both methods. Each hierarchy contained

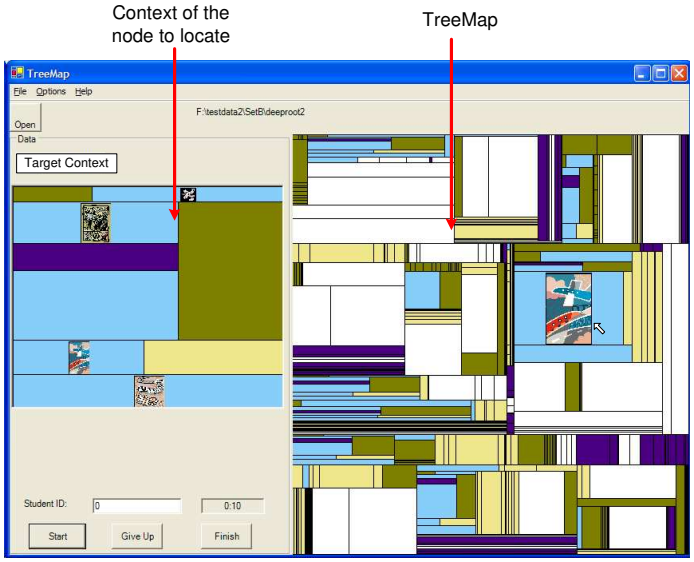


Figure 4.3: Interface used in experiment 2. When users click a node, the content of the node is shown (if the node is a picture file). Users are required to browse the TreeMap until he/she find the target picture in the specified context.

seven unique contexts, ten pictures and more than three hundred other types of files. In each task, we randomly chose one context from a hierarchy, and displayed the context to the participant in a separate window outside of the TreeMap.

Two sets of hierarchies (A and B) were used for reducing learning effects. They were constructed with similar structure but different sets of images. Half the participants started the experiment with the Drill-Down method and the other half started using the Distortion method. After completing the tasks in one set of hierarchies with one method, the participants switched onto the other set of hierarchies

with the other method. The participant was given the choice to withdraw from the task when he or she could not locate the target picture. For each task the subject was given a maximum time limit of 120 seconds.

I recorded whether the participant located the correct target, whether the participant withdrew from the task, whether the participant exceed the time limit, and the time to execute the task in all cases. In summary, the whole experiment involved: 12 participants × 2 main conditions × 2 types of hierarchies × 3 trials = 144 trials in total.

4.3.2 Results and Discussion

Context was held constant across all conditions. The time to locate target data was analyzed by means of a 2 × 2 (Type of Method × Hierarchical Structure) univariate analysis of variance (ANOVA), with both Type of Method (Drill Down vs. Distortion) and Hierarchical Structure (3 Nodes Wide vs. 6 Nodes Deep) serving as repeated measures.

The results are summarized in Table 4.2 (The detailed result of the analysis is in Appendix II). An alpha level of 0.05 was used for all statistical tests. The main effect of Type of Method was found to be significant, $F(1, 11) = 22.69$, $p = 0.01$, with the Distortion method group's mean task time (37.70 seconds)

	Wide	Deep
Distortion	34.73 (8.38) sec	40.66 (11.16) sec
Drill-Down	53.05 (16.55) sec	59.6 (23.09) sec

Table 4.2: Average completion times for Wide and Deep hierarchies with both Methods (standard deviations are in parentheses).

being faster than the Drill-Down method group's (56.33 seconds). The main effect of Hierarchical Structure was not statistically significant, $F(1, 11) = 1.06$, $p = 0.33$. Finally, an interaction effect was not found between Type of Method and Hierarchical Structure, $F(1, 11) = 0.007$, $p = 0.935$.

Out of 144 trials, 12 timeouts were recorded over 6 participants. All 12 timeouts were observed when subjects were interacting with the Drill-Down technique. From the 12 timeouts, 10 were reported on the Deep hierarchy. In addition to the 12 timeouts, 3 out of the 144 trials were giveups. All three trials were on the distortion technique. Similarly, only 2 out of 144 trials were recorded as incorrectly found, one on the Distortion and the other on the Drill-Down technique.

These results support the hypothesis in that participants will perform better in a context browsing task with the Distortion method than with the Drill-Down approach. The context browsing task assesses the participants' ability to maintain relations between elements in the structure. In the Distortion approach this is fa-

cilitated as the user, upon opening nodes, can inspect them and decide whether the appropriate relations exist. In the Drill-Down approach participants are required to drill-down and roll-up over several iterations to assess the existence of such relationships. In the Drill-Down approached we observed that in many cases nodes that were previously visited would be visited over again to confirm their content. We believe this is one factor that caused the degradation in performance with the Drill-Down technique.

Results from experiments 1 and 2 provide evidence that distortion can be used to efficiently browse node content in hierarchies visualized as space-filling representations. Experiment 3 was designed to test the effectiveness of distortion as a method for identifying search results. For distortion to be effective, users will have to locate search results quicker and also be able to identify which results are more relevant. Both criteria were tested in the experiment described next.

4.4 Experiment Three - Search Results Representation

Experiment 3 was designed to compare the Distortion technique to the conventional method of showing search results (highlight) to indicate the level of importance of

multiple search results in the TreeMap. Importance level can be defined by how often search keywords occur in a node, or whether all search keywords occur in a node, etc. Importance levels in the distortion technique is mapped onto the amplitude of distortion, ie. the larger the amplitude, the higher the importance. In the highlight approach importance level is indicated by the level of saturation. The more saturated the node, the higher the importance.

4.4.1 Method

Subjects

Twelve graduate students participated in this experiment. Half of the subjects were assigned to one condition: Distortion first, and half of them were assigned to the other condition: Highlight first. Subjects were from the computer science department and engineering department and were familiar with the concept of searching in windows file systems and searching on the Internet. All subjects were familiar with the highlight technique and the TreeMap, but none had any experience using distortion to represent search results in the TreeMap.

Materials

One hierarchy containing one thousand files was used for this experiment. Two different types of search keywords were used for the experiment: long and short. Using the short search keywords, the search would generate 3, 4, or 5 search results (small set). Using the long search keywords, the search would generate 6, 8, or 10 search results (large set). Six short search keywords and six long search keywords were used in the experiment. To reduce learning effects, I used two sets of search keywords (Set A and Set B) which would generate the same numbers of search results but in entirely different positions. Half the subjects started the experiment with the Distortion method and the other half started with the Highlight method. After completing the tasks using one set of keywords, the subjects switched to use the other set of keywords with the other method.

This experiment consisted of two types of representations: distortion and highlight. In the distortion method, all search results were animated in and out using distortions until the subjects identified all results or the subjects withdrew from the task. The amplitude of the distortion represented the importance level of each result. A result with larger amplitude has a higher importance level. In the highlight method, all search results were filled by a color. Subjects were required to identify all the results or could withdraw from the task. The saturation of the color

represented the importance level of the result. A result with higher saturation has a higher importance level.

Participants performed the experiment on a 19" monitor with resolution 1024×768 and ran the prototype over Windows XP. The task was described to them before they began the trails.

Procedure

Before starting the experiment, each subject got familiarized with both representations. Once each participant indicated that he or she was comfortable using the tool and was familiar with the interface, the experiment started.

Each participant performed the tasks with six different short keywords and six different long keywords using both methods. The 12 trials were executed in the following sequence S1, S2, S3, ..., S6, L1, L2, L3, ..., and L6, where S represents the small sets of search results and L represents the large sets of search results. No time limit was set for this task, and the subjects were free to finish the trial if they could not identify all search results. Figure 4.4 shows the interface used in experiment 3.

I recorded the time to execute the task, the number of results identified by the subjects, and whether the subject identified the correct importance level for each

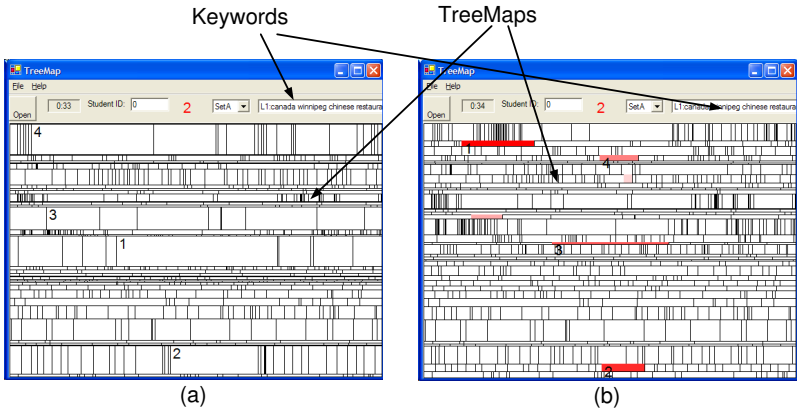


Figure 4.4: Interface used in experiment 3. When users click a node, an number is shown as the order of the users' click (if the node is a search result). Users are required to identify all search results. (a) is using the distortion method, (b) is using the highlight method.

search result in all conditions. In summary, the whole experiment involved: 12 subjects × 2 main conditions × 2 type of search result sets × 6 trials = 288 trials in total.

4.4.2 Results and Discussion

To test hypotheses 5, 6, 7, and 8 mentioned at the beginning of this section, I recorded the time it took the participants to select the items that are in the search set. The participants selected the items in order of importance (from most important to least important). Accuracy in selecting the order of importance was also

	Small Set	Large Set
Distortion	5.01 (1.47) sec	7.06 (2.83) sec
Highlight	2.43 (0.66) sec	3.30 (1.11) sec

Table 4.3: Average completion times for small and large result sets with both methods (standard deviations are in parentheses).

recorded, i.e. to get a perfect score the participant would need to select the items from most important to least important in the correct order.

Time to locate search items and accuracy in determining the correct order of importance were analyzed using a repeated measure univariate ANOVA and a paired-sample T Test (The detailed result of the analysis is in Appendix II). The results are summarized in Tables 4.3 and 4.4. An alpha level of 0.05 was used for all statistical tests. The main effect of representation type on time to complete the task and the main effect of representation type on accuracy were significant, $F(1, 11) = 44.61$, $p < 0.001$ and $F(1, 11) = 19.25$, $p = 0.001$ respectively. Participants performed significantly faster and more accurately when the search results were highlighted than when they presented using distortion.

The main effect of the size of the search result on time to complete the task was significant, $F(1, 11) = 11.68$, $p < 0.001$. Participants performed significantly faster when they were presented with a small set of search results (less than five items in

	Small Set	Large Set
Distortion	4.48%	7.42%
Highlight	3.49%	2.65%

Table 4.4: Average error rate for small and large result set with both methods.

the result set) than when they were presented a large set of results. However, the main effect of the size of the search result on accuracy was not significant $F(1, 11) = 3.63$, $p = 0.083$.

The participants completed the tasks faster when the search results where highlighted vs. distorted in a small result set ($T(1, 11)=6.88$, $p < 0.001$). This does not support Hypothesis 5 (H5). However, The analysis also suggests that participants were not less accurate in identifying the importance order of items in the result set with the distortion technique than with the highlight representation in the small result set ($T(1, 11)=1.16$, $p = 0.27$). This does not support with Hypothesis 6 (H6). In the large result set participants performed the task significantly faster and were more accurate when the results were highlighted than when they were distorted ($T(1, 11)=5.27$, $p < 0.001$ and $T(1, 11)=7.06$, $p < 0.001$, respectively). This is a support of hypotheses 7 and 8.

In general I observe that distorting multiple nodes simultaneously does not provide any benefit to locating items in the TreeMap. The motivation in using an-

imated distortions was to allow small nodes, that are not clearly visible, to become visible. However, the results do not indicate a clear advantage. One possible reason for poor performance with the animated distortion is the amount of distortion that results from this technique. Furthermore, having nodes distort at different levels of amplitude may not enhance focus and user concentration. The distortion might only be beneficial when one node or a few nodes (less than 3 nodes) need to be in focus.

Chapter 5

Conclusion and Future Work

Interaction is necessary for leveraging the power of visualization systems. Without interaction the visualization may only convey partially the information being represented. As the structures being visualized grow in size, the interaction with the visualization becomes more complex. A common representation technique for displaying large hierarchies is the TreeMap. In this visualization the hierarchy occupies the entire display space and parent-child relationships are presented using nested relationships. The conventional method of interacting with or browsing the TreeMap consists of drilling-down and rolling-up through successive layers of the hierarchy. The disadvantage of drill-down and roll-up operations is the amount of time it takes the user to reach leaf nodes of the hierarchy, where content typically

resides. Furthermore, drill-down and roll-up operations do not allow the user to maintain context of the entire hierarchy.

In this thesis I introduce a new interaction technique that facilitates browsing of elements represented in the common space-filling representation known as the TreeMap. This interaction technique is based on continuous zooming techniques [20] and uses distortion to allow users to inspect the content of nodes without opening successive layers of the hierarchy. I implemented the algorithms for single node and multiple node distortions (uni-distortion and multi-distortion, respectively). The uni-distortion technique assists in the task of browsing, in which the user is locating specific content and is interested in viewing nodes one at a time. The multi-distortion technique simultaneously distorts several nodes and produces an animated effect. It was anticipated that such a technique would assist users in locating different items in the hierarchical structure such as when performing a search. The conventional TreeMap algorithm was used as the starting point for drawing the hierarchy, which I extended to implement the distortion algorithms.

Three experiments were conducted to evaluate the effectiveness of the distortion techniques for the TreeMap. In the first experiment, the distortion technique was compared to the drill-down approach which is the conventional method of browsing the TreeMap. The results of the first experiment suggest that subjects are faster

at browsing and locating objects of interest in the distortion technique. The effect is more pronounced when the hierarchy is several layers deep. The main reason that users perform better at the distortion technique is due to the fact that in the drill-down approach the user has to drill-down and roll-up during several iterations until the node is found. In the distortion technique, the user has access to leaf nodes from the top-most view of the hierarchy. As anticipated, the results show that performance with the distortion technique is unaffected by the depth of the hierarchy. However, when nodes are very small, the distortion technique may not facilitate the browsing task. The results of this experiment show that a small percentage of nodes were not inspected in the distortion technique due to their limited display size. A solution to this drawback could be to combine both he drill-down and the distortion technique into one interaction mode, where the drill-down is first applied and then the distortion, once the node becomes clearly visible.

In the second experiment, participants were required to locate a sub-structure of the hierarchy. The sub-structure was identifiable by the number of nodes it contained, their relative parent-child and sibling-sibling relationships, and by their content. By browsing the hierarchy to identify the appropriate nodes and their contents, users were required to locate one node within the sub-structure. This task, referred to as the context browsing task, was designed to evaluate the effectiveness of the distortion technique for its ability in assisting users to maintain context while

browsing the hierarchy. The results show that participants were quicker in locating the objects with the distortion method than with the drill-down approach. The results of the first two experiments corroborate with those of [20] in suggesting that interactive techniques employing variations of continuous semantic zooming operations are an enhancement to full zoom approaches (such as drill-down) under certain conditions.

Finally, the third experiment evaluated the effectiveness of the multi-distortion technique for assisting users to identify search result sets and the level of importance for result items in search results. The results of this experiment suggest that the multi-distortion technique is not as effective as simple highlighting for showing search result sets. This is primarily due to the high level of distractibility created by the animations from the multi-distortion technique. An alternative to displaying the distortions simultaneously would have been to show them in series starting from the most important to the least important. This would have allowed users to locate nodes that would have not been visible otherwise but still provide a fair indication of items in the tree.

5.1 Contributions

This thesis offers several contributions to the area of information visualization. These include:

- the uni-distortion and multi-distortion algorithms. Researchers in the past have indicated having difficulty designing the distortion algorithms on the TreeMap due to the level of instability that could result from distributing the weights of nodes in the hierarchy [24] The algorithms designed in this thesis are highly stable and robust under user interaction.

- the second contribution is the series of evaluation and their results. Typically, very few systems in information visualization undergo the process of controlled experiments. In this case, the technique could have been easily evaluated since the TreeMap was created with conventional browsing techniques which assisted as controls for the experiment.

- a final contribution is the context browsing task for testing the interaction's capacity for allowing focus+context browsing.

5.2 Future Work

I conclude by proposing several lines of research and applications of the distortion technique for future endeavors. The first line of work could investigate the application of the distortion technique to current file browsing systems. Recent work has shown progress in this area [15], and the distortion technique may prove beneficial in such systems. Prototypes are necessary in order to establish the best mode of interaction for such systems.

The TreeMap can be universally used for browsing many forms of hierarchies and on different platforms. One platform that can take advantage of the TreeMap and the distortion methods introduced in this thesis is handheld systems, such as PDAs. Due to the physical characteristics and limitations of handheld devices, PDAs do not have large display spaces. For multi-tasking systems where users have to switch between windows frequently, the distortion technique may provide a good alternative. Multiple windows could reside in the TreeMap and users can simply distort the nodes for opening those windows most required. For example, in figure 5.1, users of the PDA need to compare two pictures and read a comment of one picture simultaneously. In this case, the users can click on the picture to enlarge it while reading the comments of the picture. The user also can enlarge these two pictures to compare them in detail. The users can also open more files if

Figure 5.1: Distortion of TreeMap on a PDA. (a) shows the initial state, in which a user can get previews of pictures. In (b), the user opens a picture to get detail information. In (c), the user can read the description of the picture without switching window or running another application. In (d), the user can open another picture to compare them. The user can switch quickly between pictures and descriptions.

they need to execute more tasks.

Finally, the distortion technique may be applicable for viewing web pages that are classified hierarchically. The central question then would consist of determining whether the distortion technique, which allows focus+context interaction facilitate web browsing while reducing the amount of disorientation that typically results from browsing multiple pages in a given browsing session.

All of the above lines of research would necessitate further implementation on the distortion technique and well-designed studies to evaluate the effectiveness of the resulting systems.

Bibliography

[1] K. Andrews and H. Heidegger. Information slices: Visualising and exploring large hierarchies using cascading, semi-circular discs. In *INFOVIS*, pages 9–12, 1998.

[2] B. B. Bederson. Photomesa: A zoomable image browser using quantum treemaps and bubblemaps. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 71–80, 2001.

[3] B. B. Bederson, A. Clamage, M. Czerwinski, and G. G. Robertson. Datelens: A fisheye calendar interface for PDAs. *ACM Transactions on Computer-Human Interaction*, 11(1):90–119, 2004.

[4] B. B. Bederson, J. D. Hollan, K. Perlin, J. Meyer, D. Bacon, and G. W. Furnas. Pad++: A zoomable graphical sketchpad for exploring alternate interface physics. *Journal of Visual Languages and Computing*, 7(1):3–32, 1996.

[5] B. B. Bederson, B. Shneiderman, and M. Wattenberg. Ordered and quantum treemaps: Making effective use of 2d space to display hierarchies. *ACM Transactions on Graphics*, 21(4):833–854, 2002.

[6] S. K. Card, J. D. Mackinlay, and B. Shneiderman. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers, San Francisco, CA, 1999.

[7] Y. Fua, M. O. Ward, and E. A. Rundensteiner. Structure-based brushes: A mechanism for navigating hierarchically organized data and information spaces. *IEEE Transactions on Visual Computing and Graphics*, 6(2):150–159, 2000.

[8] G. W. Furnas. Generalized fisheye views. In *Proceedings of CHI'86*, pages 16–23, 1986.

[9] P. P. Irani. *Geon diagrams: a perception based method for visualizing structured information*. PhD thesis, University of New Brunswick, 2002.

[10] P. P. Irani, D. Slonowsky, and P. Shajahan. The effect of shading in extracting structure from space-filling visualizations. In *Information Visualization, Eighth International Conference*, pages 209–216, 2004.

[11] P. P. Irani and C. Ware. Diagramming information structures using 3D perceptual primitives. *ACM Transactions on Computer-Human Interaction*, 10(1):1–19, 2003.

[12] B. Johnson and B. Shneiderman. Treemaps: A space-filling approach to the visualization of hierarchical information. In *IEEE Visualization '91 Conference*, pages 284–291, San Diego, Ca, October 1991.

[13] J. Lamping and R. Rao. Laying out and visualizing large trees using a hyperbolic space. In *ACM Symposium on User Interface Software and Technology*, pages 13–14, 1994.

[14] A. Leuski and J. Allan. Lighthouse: Showing the way to relevant information. In *INFOVIS*, pages 125–130, 2000.

[15] M. J. McGuffin, G. Davison, and R. Balakrishnan. Expand-ahead: A space-filling strategy for browsing trees. *INFOVIS*, pages 119–126, 2004.

[16] T. Paek, S. Dumais, and R. Logan. Wavelens: a new view onto internet search results. In *CHI 2004*, pages 727–734, Vienna, Austria, April 2004.

[17] R. Rao and S. K. Card. Table lens: Merging graphical and symbolic representations in an interactive focus plus context visualization for tabular information. In *Proceedings of CHI'94*, pages 318–322, 1994.

[18] G. G. Robertson, S. K. Card, and J. D. Mackinlay. Information visualization using 3D interactive animation. *Communications of the ACM*, 36(4):56–71, 1993.

[19] G.G. Robertson, J.D. MacKinlay, and S.K. Card. Cone trees: Animated 3d visualizations of hierarchical information. In *Proceedings of CHI'91*, pages 189–194, 1991.

[20] D. Schaffer, Z. Zuo, S. Greenberg, L. Bartram, J. Dill, S. Dubs, and M. Roseman. Navigating hierarchically clustered networks through fisheye and full-zoom methods. *ACM Transactions on Computer.-Human Interaction*, 3(2):162–188, 1996.

[21] B. Shneiderman. Tree visualization with treemaps: a 2d space-filling approach. *ACM Transactions on Graphics*, 11(1):92–99, 1990.

[22] R. Spence. *Information Visualization.* ACM Press, Pearson Education Limited, 1st edition, 2001.

[23] J. T. Stasko and E. Zhang. Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In *INFOVIS*, 2000.

[24] D. Turo and B. Johnson. Improving the visualization of hierarchies with TreeMaps: design issues and experimentation. In *Proceedings of the 3rd conference on Visualization '92*, pages 124–131, 1992.

[25] J. J. van Wijk and H. van de Wetering. Cushion treemaps: Visualization of hierarchical information. In *INFOVIS*, pages 73–78, San Francisco, 1999.

[26] C. Ware and R. Bobrow. Motion to support rapid interactive queries on node-link diagrams. *ACM Transactions on Applied Perception*, 1(1):3–18, 2004.

Appendixes

Appendix I TreeMap Algorithm [12]

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

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


FindPath(point thePoint)
{   if node encloses thePoint then
    foreach child of thisNode do {
        path = FindPath(thePoint);
        if (path != NULL) then
            return(InsertInList(thisNode, path));               Add child to path
    }
    return (NULL);                                              Start path, thePoint is in this node, but not in any of its children
}
```

Appendix II Analysis Results of Experiments 1, 2, and 3 in SPSS

Experiment 1 - Repated Measures Output

Within-Subjects Factors

Measure: MEASURE_1

tech	struct	Dependent Variable
1	1	distortion_ wide
	2	distortion_ deep
2	1	drilldown_ wide
	2	drilldown_ deep

Descriptive Statistics

	Mean	Std. Deviation	N
distortion_wide	31.0657	12.67520	20
distortion_deep	26.5104	13.02087	20
drilldown_wide	47.6323	20.96507	20
drilldown_deep	64.4811	29.68697	20

Tests of Within-Subjects Effects

Measure: MEASURE_1

Source		Type III Sum of Squares	df	Mean Square	F	Sig.
tech	Sphericity Assumed	14871.549	1	14871.549	50.697	.000
	Greenhouse-Geisser	14871.549	1.000	14871.549	50.697	.000
	Huynh-Feldt	14871.549	1.000	14871.549	50.697	.000
	Lower-bound	14871.549	1.000	14871.549	50.697	.000
Error(tech)	Sphericity Assumed	5573.463	19	293.340		
	Greenhouse-Geisser	5573.463	19.000	293.340		
	Huynh-Feldt	5573.463	19.000	293.340		
	Lower-bound	5573.463	19.000	293.340		
struct	Sphericity Assumed	755.646	1	755.646	1.740	.203
	Greenhouse-Geisser	755.646	1.000	755.646	1.740	.203
	Huynh-Feldt	755.646	1.000	755.646	1.740	.203
	Lower-bound	755.646	1.000	755.646	1.740	.203
Error(struct)	Sphericity Assumed	8253.573	19	434.399		
	Greenhouse-Geisser	8253.573	19.000	434.399		
	Huynh-Feldt	8253.573	19.000	434.399		
	Lower-bound	8253.573	19.000	434.399		
tech * struct	Sphericity Assumed	2290.682	1	2290.682	5.096	.036
	Greenhouse-Geisser	2290.682	1.000	2290.682	5.096	.036
	Huynh-Feldt	2290.682	1.000	2290.682	5.096	.036
	Lower-bound	2290.682	1.000	2290.682	5.096	.036
Error(tech*struct)	Sphericity Assumed	8540.580	19	449.504		
	Greenhouse-Geisser	8540.580	19.000	449.504		
	Huynh-Feldt	8540.580	19.000	449.504		
	Lower-bound	8540.580	19.000	449.504		

General Linear Model

Within-Subjects Factors

Measure: MEASURE_1

tech	size	Dependent Variable
1	1	dist_small
	2	dist_large
2	1	high_small
	2	high_large

Descriptive Statistics

	Mean	Std. Deviation	N
dist_small	5.0099	1.46884	12
dist_large	7.0649	2.83065	12
high_small	2.4342	.65572	12
high_large	3.3009	1.10656	12

Tests of Within-Subjects Effects

Measure: MEASURE_1

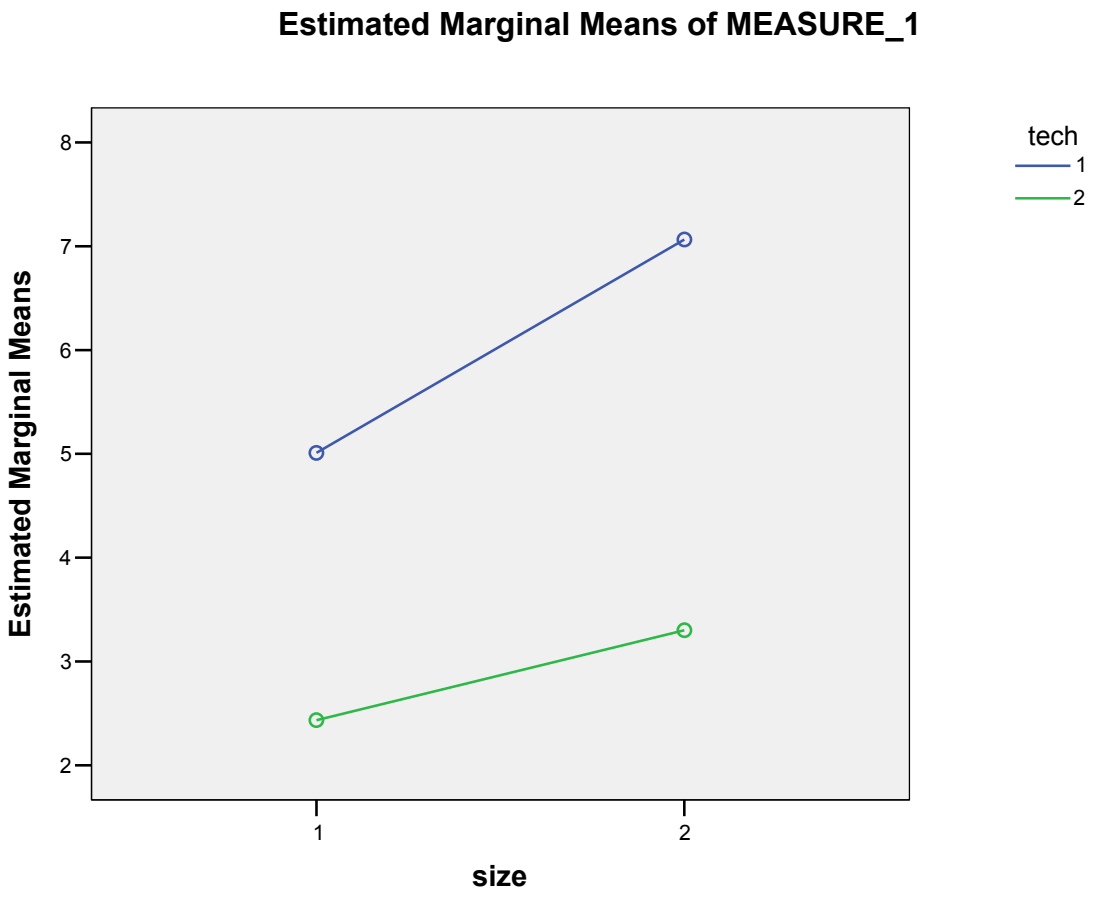
Source		Type III Sum of Squares	df	Mean Square	F	Sig.
tech	Sphericity Assumed	120.577	1	120.577	44.610	.000
	Greenhouse-Geisser	120.577	1.000	120.577	44.610	.000
	Huynh-Feldt	120.577	1.000	120.577	44.610	.000
	Lower-bound	120.577	1.000	120.577	44.610	.000
Error(tech)	Sphericity Assumed	29.732	11	2.703		
	Greenhouse-Geisser	29.732	11.000	2.703		
	Huynh-Feldt	29.732	11.000	2.703		
	Lower-bound	29.732	11.000	2.703		
size	Sphericity Assumed	25.609	1	25.609	11.682	.006
	Greenhouse-Geisser	25.609	1.000	25.609	11.682	.006
	Huynh-Feldt	25.609	1.000	25.609	11.682	.006
	Lower-bound	25.609	1.000	25.609	11.682	.006
Error(size)	Sphericity Assumed	24.114	11	2.192		
	Greenhouse-Geisser	24.114	11.000	2.192		
	Huynh-Feldt	24.114	11.000	2.192		
	Lower-bound	24.114	11.000	2.192		
tech * size	Sphericity Assumed	4.236	1	4.236	3.531	.087
	Greenhouse-Geisser	4.236	1.000	4.236	3.531	.087
	Huynh-Feldt	4.236	1.000	4.236	3.531	.087
	Lower-bound	4.236	1.000	4.236	3.531	.087
Error(tech*size)	Sphericity Assumed	13.196	11	1.200		
	Greenhouse-Geisser	13.196	11.000	1.200		
	Huynh-Feldt	13.196	11.000	1.200		
	Lower-bound	13.196	11.000	1.200		

Tests of Within-Subjects Contrasts

Measure: MEASURE_1

Source	tech	size	Type III Sum of Squares	df	Mean Square	F	Sig.
tech	Linear		120.577	1	120.577	44.610	.000
Error(tech)	Linear		29.732	11	2.703		
size		Linear	25.609	1	25.609	11.682	.006
Error(size)		Linear	24.114	11	2.192		
tech * size	Linear	Linear	4.236	1	4.236	3.531	.087
Error(tech*size)	Linear	Linear	13.196	11	1.200		

Profile Plots

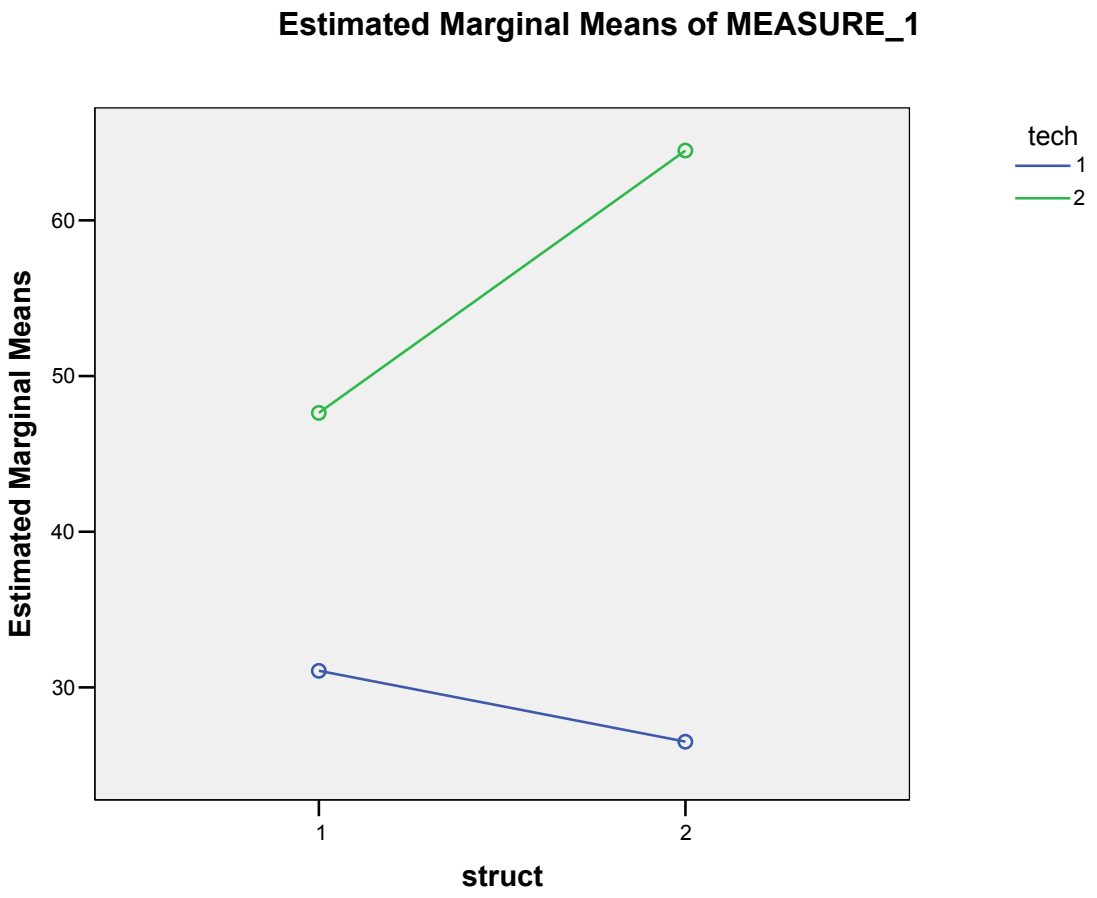


Tests of Within-Subjects Contrasts

Measure: MEASURE_1

Source	tech	struct	Type III Sum of Squares	df	Mean Square	F	Sig.
tech	Linear		14871.549	1	14871.549	50.697	.000
Error(tech)	Linear		5573.463	19	293.340		
struct		Linear	755.646	1	755.646	1.740	.203
Error(struct)		Linear	8253.573	19	434.399		
tech * struct	Linear	Linear	2290.682	1	2290.682	5.096	.036
Error(tech*struct)	Linear	Linear	8540.580	19	449.504		

Profile Plots



T-Test

Paired Samples Statistics

		Mean	N	Std. Deviation	Std. Error Mean
Pair 1	distortion_wide	31.6107	20	13.34529	2.98410
	distortion_deep	26.6458	20	12.94921	2.89553

Paired Samples Correlations

		N	Correlation	Sig.
Pair 1	distortion_wide & distortion_deep	20	.285	.223

Paired Samples Test

		Paired Differences					t
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper	
Pair 1	distortion_wide - distortion_deep	4.96483	15.72102	3.51533	-2.39283	12.32250	1.412

Paired Samples Test

		df	Sig. (2-tailed)
Pair 1	distortion_wide - distortion_deep	19	.174

Experiment 2 - Repeated Measures Output

Within-Subjects Factors

Measure: MEASURE_1

tech	struct	Dependent Variable
1	1	dist_wide
	2	dist_deep
2	1	dd_wide
	2	dd_deep

Descriptive Statistics

	Mean	Std. Deviation	N
dist_wide	34.7348	8.38170	12
dist_deep	40.6642	11.15545	12
dd_wide	53.0511	16.55274	12
dd_deep	59.6018	23.08924	12

Tests of Within-Subjects Effects

Measure: MEASURE_1

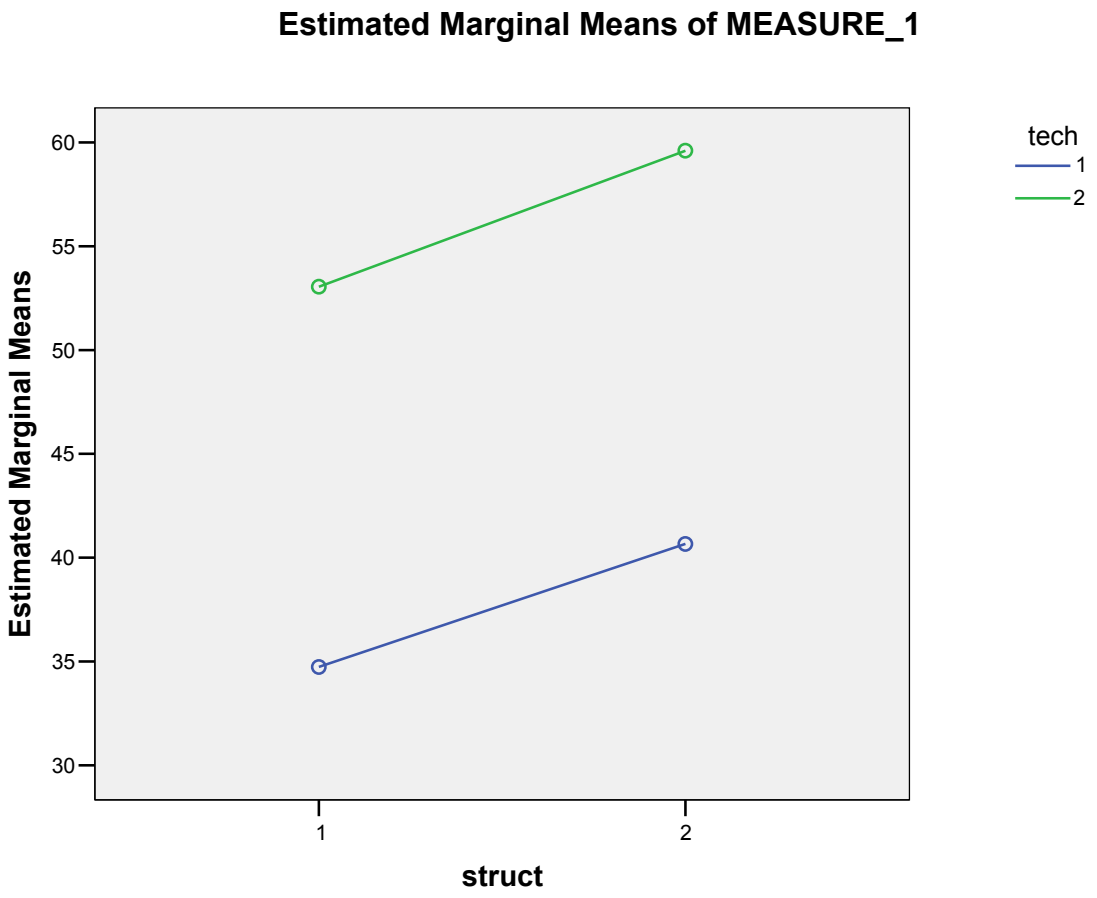
Source		Type III Sum of Squares	df	Mean Square	F	Sig.
tech	Sphericity Assumed	4163.568	1	4163.568	22.689	.001
	Greenhouse-Geisser	4163.568	1.000	4163.568	22.689	.001
	Huynh-Feldt	4163.568	1.000	4163.568	22.689	.001
	Lower-bound	4163.568	1.000	4163.568	22.689	.001
Error(tech)	Sphericity Assumed	2018.560	11	183.505		
	Greenhouse-Geisser	2018.560	11.000	183.505		
	Huynh-Feldt	2018.560	11.000	183.505		
	Lower-bound	2018.560	11.000	183.505		
struct	Sphericity Assumed	467.263	1	467.263	1.062	.325
	Greenhouse-Geisser	467.263	1.000	467.263	1.062	.325
	Huynh-Feldt	467.263	1.000	467.263	1.062	.325
	Lower-bound	467.263	1.000	467.263	1.062	.325
Error(struct)	Sphericity Assumed	4838.107	11	439.828		
	Greenhouse-Geisser	4838.107	11.000	439.828		
	Huynh-Feldt	4838.107	11.000	439.828		
	Lower-bound	4838.107	11.000	439.828		
tech * struct	Sphericity Assumed	1.158	1	1.158	.007	.935
	Greenhouse-Geisser	1.158	1.000	1.158	.007	.935
	Huynh-Feldt	1.158	1.000	1.158	.007	.935
	Lower-bound	1.158	1.000	1.158	.007	.935
Error(tech*struct)	Sphericity Assumed	1830.850	11	166.441		
	Greenhouse-Geisser	1830.850	11.000	166.441		
	Huynh-Feldt	1830.850	11.000	166.441		
	Lower-bound	1830.850	11.000	166.441		

Tests of Within-Subjects Contrasts

Measure: MEASURE_1

Source	tech	struct	Type III Sum of Squares	df	Mean Square	F	Sig.
tech	Linear		4163.568	1	4163.568	22.689	.001
Error(tech)	Linear		2018.560	11	183.505		
struct		Linear	467.263	1	467.263	1.062	.325
Error(struct)		Linear	4838.107	11	439.828		
tech * struct	Linear	Linear	1.158	1	1.158	.007	.935
Error(tech*struct)	Linear	Linear	1830.850	11	166.441		

Profile Plots



Experiment 3 - General Linear Model

Within-Subjects Factors

Measure: MEASURE_1

tech	size	Dependent Variable
1	1	dist_small_error
	2	dist_large_error
2	1	high_small_error
	2	high_large_error

Descriptive Statistics

	Mean	Std. Deviation	N
dist_small_error	.0448	.02403	12
dist_large_error	.0742	.01548	12
high_small_error	.0349	.02476	12
high_large_error	.0265	.01846	12

Tests of Within-Subjects Effects

Measure: MEASURE_1

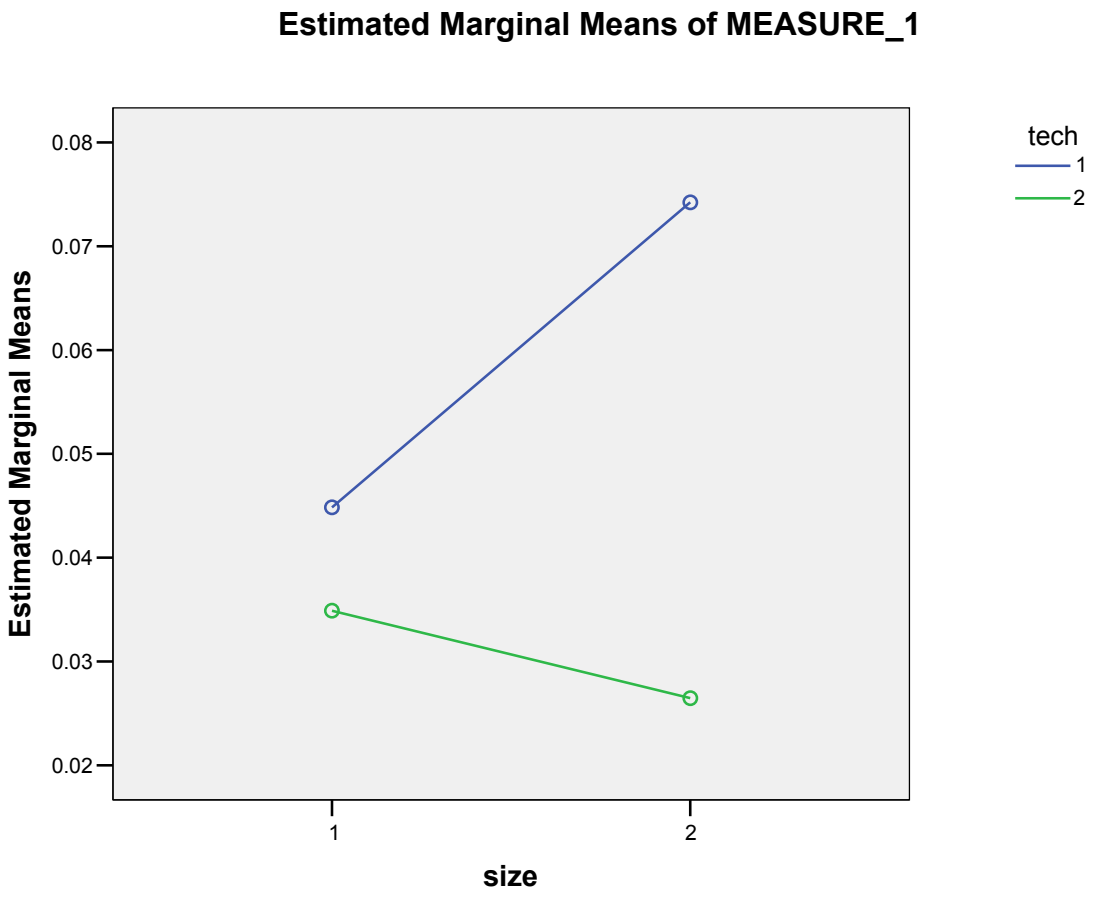
Source		Type III Sum of Squares	df	Mean Square	F	Sig.
tech	Sphericity Assumed	.010	1	.010	19.250	.001
	Greenhouse-Geisser	.010	1.000	.010	19.250	.001
	Huynh-Feldt	.010	1.000	.010	19.250	.001
	Lower-bound	.010	1.000	.010	19.250	.001
Error(tech)	Sphericity Assumed	.006	11	.001		
	Greenhouse-Geisser	.006	11.000	.001		
	Huynh-Feldt	.006	11.000	.001		
	Lower-bound	.006	11.000	.001		
size	Sphericity Assumed	.001	1	.001	3.628	.083
	Greenhouse-Geisser	.001	1.000	.001	3.628	.083
	Huynh-Feldt	.001	1.000	.001	3.628	.083
	Lower-bound	.001	1.000	.001	3.628	.083
Error(size)	Sphericity Assumed	.004	11	.000		
	Greenhouse-Geisser	.004	11.000	.000		
	Huynh-Feldt	.004	11.000	.000		
	Lower-bound	.004	11.000	.000		
tech * size	Sphericity Assumed	.004	1	.004	21.932	.001
	Greenhouse-Geisser	.004	1.000	.004	21.932	.001
	Huynh-Feldt	.004	1.000	.004	21.932	.001
	Lower-bound	.004	1.000	.004	21.932	.001
Error(tech*size)	Sphericity Assumed	.002	11	.000		
	Greenhouse-Geisser	.002	11.000	.000		
	Huynh-Feldt	.002	11.000	.000		
	Lower-bound	.002	11.000	.000		

Tests of Within-Subjects Contrasts

Measure: MEASURE_1

Source	tech	size	Type III Sum of Squares	df	Mean Square	F	Sig.
tech	Linear		.010	1	.010	19.250	.001
Error(tech)	Linear		.006	11	.001		
size		Linear	.001	1	.001	3.628	.083
Error(size)		Linear	.004	11	.000		
tech * size	Linear	Linear	.004	1	.004	21.932	.001
Error(tech*size)	Linear	Linear	.002	11	.000		

Profile Plots



T-Test

Paired Samples Statistics

		Mean	N	Std. Deviation	Std. Error Mean
Pair 1	dist_small_time	5.0099	12	1.46884	.42402
	high_small_time	2.4342	12	.65572	.18929
Pair 2	dist_large_time	7.0649	12	2.83065	.81714
	high_large_time	3.3009	12	1.10656	.31944
Pair 3	dist_small_err	.0448	12	.02403	.00694
	high_small_err	.0349	12	.02476	.00715
Pair 4	dist_large_err	.0742	12	.01548	.00447
	high_large_err	.0265	12	.01846	.00533

Paired Samples Correlations

		N	Correlation	Sig.
Pair 1	dist_small_time & high_small_time	12	.469	.124
Pair 2	dist_large_time & high_large_time	12	.497	.100
Pair 3	dist_small_err & high_small_err	12	.260	.414
Pair 4	dist_large_err & high_large_err	12	.055	.864

Paired Samples Test

		Paired Differences					t
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference		
					Lower	Upper	
Pair 1	dist_small_time - high_small_time	2.57572	1.29759	.37458	1.75127	3.40016	6.876
Pair 2	dist_large_time - high_large_time	3.76403	2.47415	.71423	2.19203	5.33603	5.270
Pair 3	dist_small_err - high_small_err	.00995	.02968	.00857	-.00890	.02881	1.162
Pair 4	dist_large_err - high_large_err	.04776	.02342	.00676	.03288	.06265	7.064

Paired Samples Test

		df	Sig. (2-tailed)
Pair 1	dist_small_time - high_small_time	11	.000
Pair 2	dist_large_time - high_large_time	11	.000
Pair 3	dist_small_err - high_small_err	11	.270
Pair 4	dist_large_err - high_large_err	11	.000