Journal of Visual Languages and Computing 31 (2015) 48–69

Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

journal homepage: www.elsevier.com/locate/jvlc

A mobile interface for navigating hierarchical information space ☆

Abhishek P. Chhetri [a,*], Kang Zhang [b,c], Eakta Jain [c,d]

[a] Computer Engineering Program, Erik Jonsson School of Engineering and Computer Science, University of Texas at Dallas, Richardson, TX 65080-3021, USA
[b] School of Software Engineering, Tianjin University, Tianjin, China
[c] Department of Computer Science, University of Texas at Dallas, Richardson, TX 65080-3021, USA
[d] Texas Instruments, Dallas, TX, USA

ARTICLE INFO

Article history:
Received 2 June 2015
Accepted 5 October 2015
Available online 22 October 2015

General terms:
Algorithms
Design
Human factors

Keywords:
Hierarchy visualization
Mobile devices
Navigation
RELT (Radial Edgeless Tree)
User interface

ABSTRACT

This paper presents ERELT (Enhanced Radial Edgeless Tree), a tree visualization approach on modern mobile devices. ERELT is designed to offer a clear visualization of any tree structure with intuitive interaction. Such visualization can assist users in interacting with a hierarchical structure such as a media collection, file system, etc.

In the ERELT visualization, a subset of the tree is displayed at a time. The displayed tree size depends on the maximum number of tree elements that can be put on the screen while maintaining clarity. Users can quickly navigate to the hidden parts of the tree through touch-based gestures. We have conducted a user study to evaluate this visualization for a music collection. The study results show that this approach reduces the time and effort in navigating tree structures for exploration and search tasks.

1. Introduction

Current market trends show strong rise in smartphones and tablets. Smartphones can now match the processing capabilities of laptop/PCs from a few years ago with a fraction of the power usage. With emails, contacts, documents, pictures and music all stored in the cloud, one no longer needs to sit in front of a personal computer to access data. There are, however, still many challenges in mobile computing, such as smaller screens and lack of separate input devices such as keyboard.

Although the screen resolution in mobile devices has been increasing, in terms of screen size they are still much smaller than laptops/PC monitors. This makes it difficult to present tabular and hierarchical structures in mobile device when a large proportion of application data are hierarchical in nature. For example, a file system is a hierarchical structure, and a file list within a folder is usually displayed in tabular format. A multimedia collection such as music, pictures, videos, etc. may exist in hierarchical structures, and is usually displayed in tabular form in laptops/PCs. Fig. 1 shows music hierarchy presented as a table in a PC media player.

Apart from the presentation issues, how we interact with these data structures on mobile devices is also challenging. Modern mobile devices are mostly equipped with touch screens, and soft keyboards. Any keyboards or buttons displayed on screen take space, which is already at a premium. Thus, it is necessary to come up with intuitive

☆ This paper has been recommended for acceptance by Gennaro Costagliola.

* Correspondence to: 25 Grant Cir, Richardson, TX 75081 USA.
E-mail addresses: achhetri@utdallas.edu (A.P. Chhetri), kzhang@utdallas.edu (K. Zhang), eakta.jain@gmail.com (E. Jain).

http://dx.doi.org/10.1016/j.jvlc.2015.10.002
1045-926X/

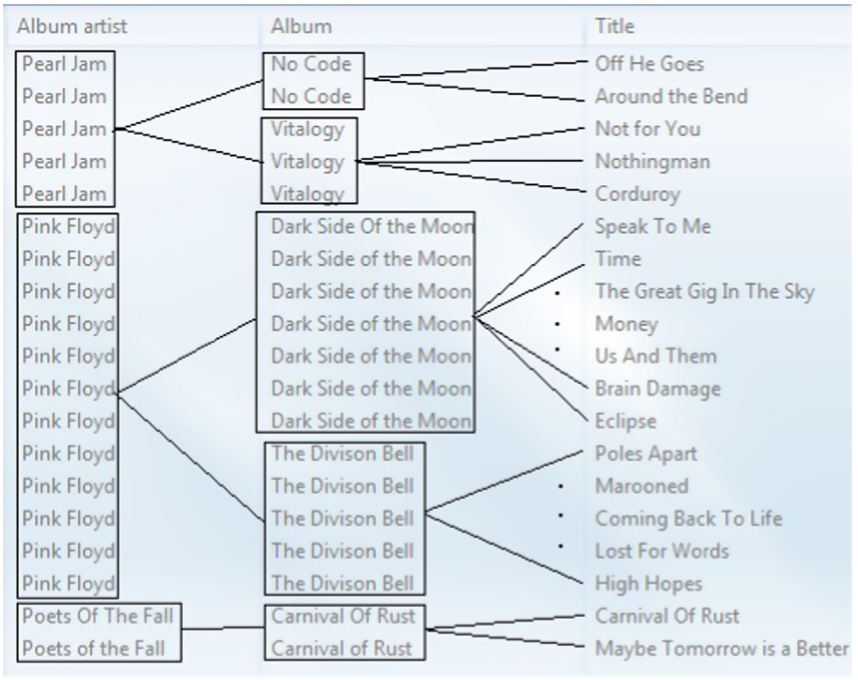

Fig. 1. Tabular display of artist/album/track in a media player (edited to show hierarchical division).

methods of interaction without sacrificing screen area for input. Most hierarchical structures are represented by lists in mobile devices, as shown in Fig. 2. Lists offer fast interaction but can only display one level of hierarchy at a time. This paper presents a technique for visualizing and navigating hierarchical structures on mobile devices that focuses on following two issues:

1. Maximal utilization of screen area to display hierarchical structures.
2. Intuitive interaction to allow rapid navigation and exploration of the structures.

The research aims to utilize the screen estate to display maximum possible information, without sacrificing clarity. The objective is to use touch technology in most modern smartphones to implement gesture based commands that are intuitive and mimic real-world object interactions. The paper presents further enhancement over our earlier prototype [1], and a user study using media player application utilizing ERELT. Our user study shows that ERELT supports faster exploration of tree structures than traditional list based interfaces.

To date, considerable research has been done in the areas of information visualization and human-computer interaction (HCI). While recent advances in visualization techniques for hierarchical structures have been promising, little previous work has focused on utilizing visualization as UI elements. Our contribution is a hierarchy visualization technique for small screens with a practical approach for user interaction. We extensively evaluate the ease of user interaction with the proposed ERELT visualization through a user study. Our results show that it takes significantly less time and fewer number of touches to perform exploration and search tasks. In some cases, the number of touches is reduced by nearly 50%. This suggests that ERELT is a more appropriate interface for users for interacting with hierarchical data than the traditional list interface.

The next section reviews the previous research on the subject, including our own, and explore their shortcomings and our latest improvements. Section 3 describes our visualization approach. Section 4 focuses on navigation and interaction details, followed by the evaluation of the research through user tests. Finally, Section 5 presents conclusion and ideas for future work.

2. Background and related work

2.1. Hierarchy visualization

2.1.1. Implicit vs. explicit visualization

Hierarchy visualization techniques can broadly be classified into two types – implicit and explicit [2]. Explicit visualization techniques display the edges between the connected vertices of the hierarchy. In contrast, implicit techniques, also known as enclosure techniques, show hierarchical relations through shape, location and area of vertices [3].

Explicit visualization techniques are simpler to understand as each relation between nodes is explicitly displayed through links. However, the edges require larger area to draw and much display area around edges and between nodes is unused. There has been some research on explicit visualization for mobile devices, such as Magic-Eye view [5] and Space Manager [6].

Implicit visualization techniques vary in area utilization depending on the scheme to show parent–child relation between nodes. They generally take up less area because they do not use connecting edges between the nodes. Space efficiency is crucial for visualization of hierarchy in a small screen [4]. We focus mainly on implicit hierarchy visualization since they better cover the display area than explicit techniques [2]. In case of implicit layouts, the relationship between two adjacent nodes can be represented through inclusion, adjacency or overlap [3], as shown in Fig. 3. Techniques based on inclusion and adjacency are widely used for hierarchy visualization whereas overlap techniques are not commonly found.

2.1.2. *Inclusion based implicit visualization*

One of the most commonly used implicit visualization techniques is the Tree-Map [7,8]. A Tree-Map uses inclusion to show parent–child relations between nodes. It maximizes area utilization by enclosing child nodes in their parent node. As shown in Fig. 4, the tree-map is good at displaying many leaf nodes in a small space but cannot clearly show parent–child relationships. Nodes can be shaped like thin lines for large trees, making individual nodes less distinct.

The Squarified Treemaps [9] and the Cushion Treemaps [10], shown in Fig. 5, attempted to solve the problem of node shape in Tree Maps by forcing the nodes to be as square as possible. This makes the individual nodes more distinct. Moreover, the Cushion Treemaps displays individual nodes with gradient shades to give them a 3D look.

There are other variations of Tree-Maps which have non-rectangular nodes. Generalized Treemaps [11] applied tree map concept to pie (circular) and pyramid (triangular) shapes. Voronoi Treemap [12] represents nodes using arbitrary polygons. Jigsaw Map [13] uses a complex algorithm to sub-divide the parent node into child nodes, where nodes take the shape of complex jigsaw puzzle pieces.

In space-filling inclusive visualization techniques, like tree maps, leaf nodes take up all the space. Thus, intermediate nodes are hard to identify. Circular Partitions [14] uses boundary thickness to differentiate levels of the

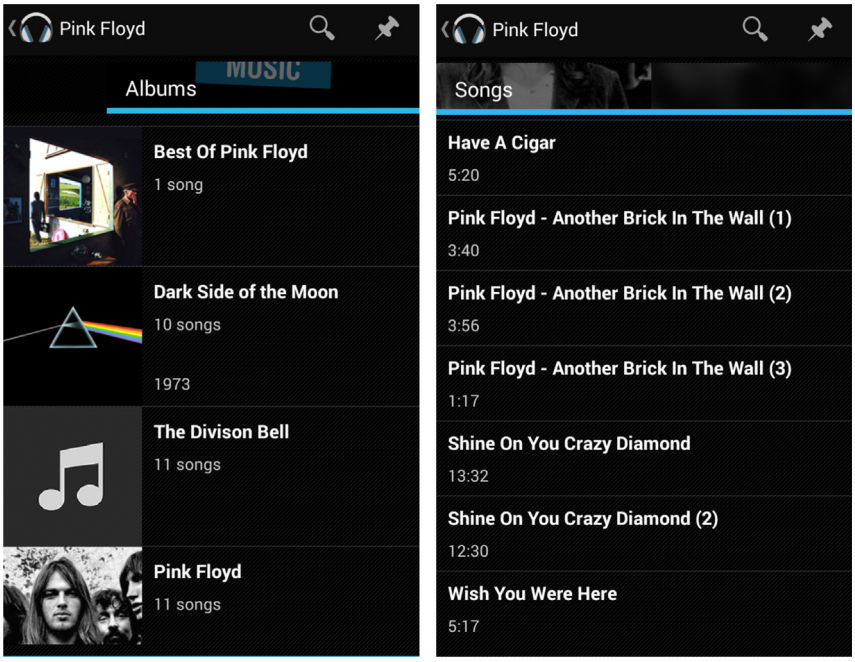

Fig. 2. Android's list view showing albums list and songs list.

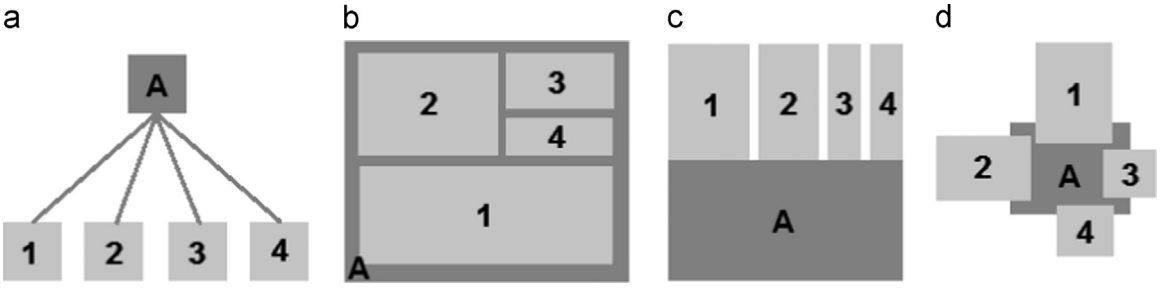

Fig. 3. (a) Explicit layout, (b) Implicit layout by inclusion, (c) Implicit layout by adjacency, (d) Implicit layout by overlap.

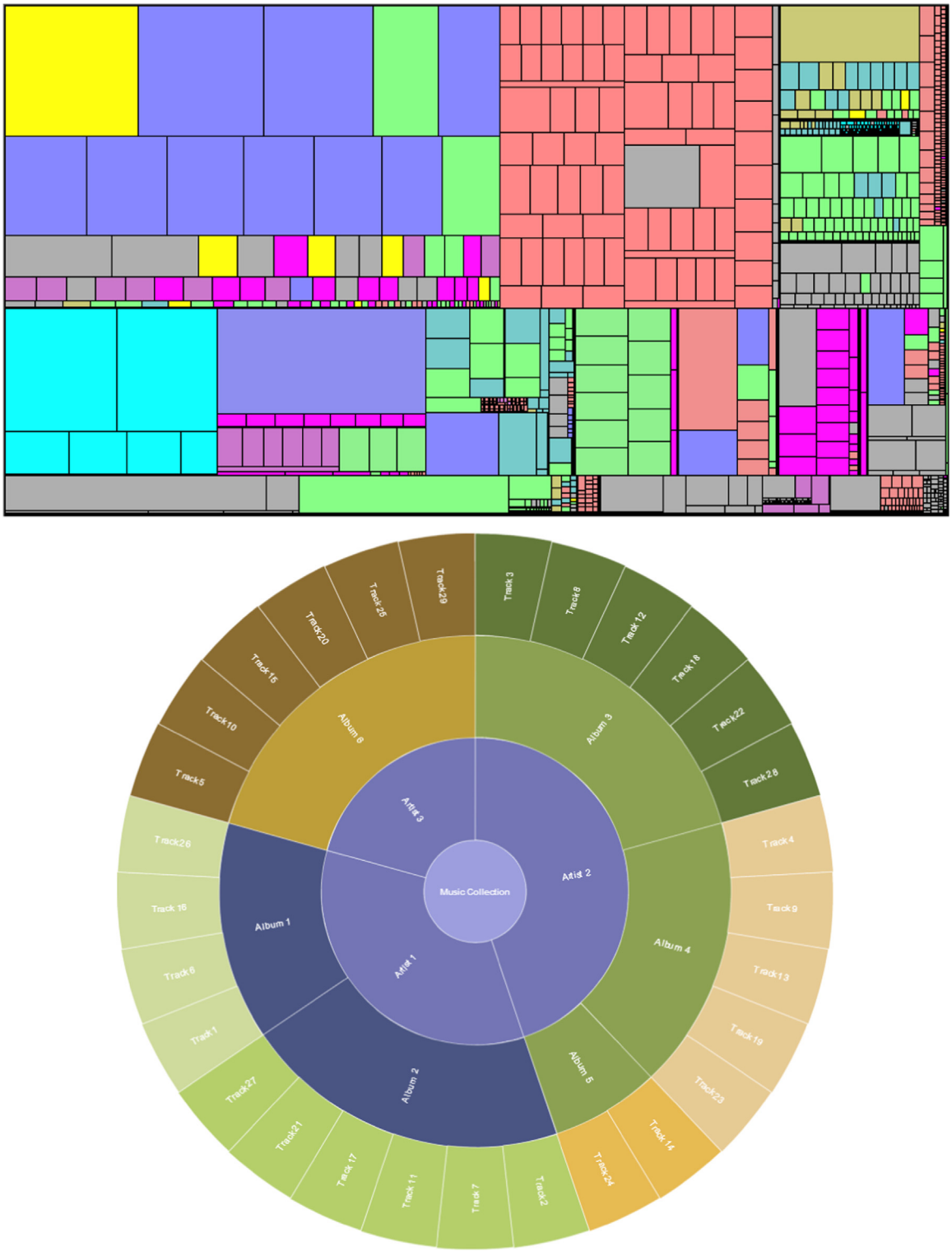

Fig. 4. Tree-Map (top), Sunburst (bottom).

hierarchy. Voronoi Treemap [12] uses colors to indicate sub-division of intermediate nodes.

Inclusive techniques that use node packing instead of sub-division provide better representation of intermediate levels. Packing is a technique where each node has a distinct boundary, and child nodes are placed inside their parent node instead of being creating by dividing the parent node. Pebble Map [15] and CropCircles [16] have

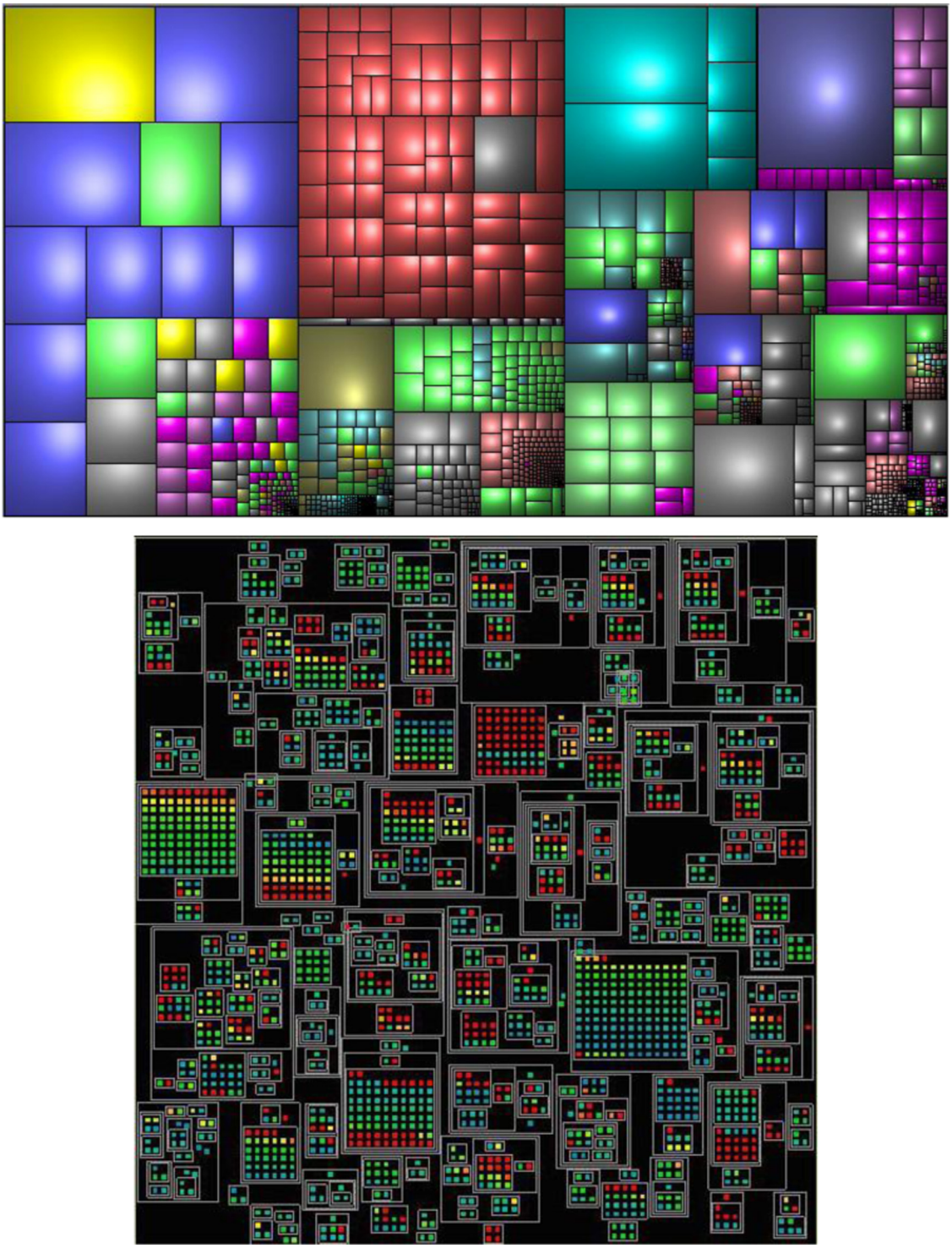

Fig. 5. Sub-division in Cushion Treemap (top), Node packing in Data Jewelry Box (bottom) [Image Source: http://www.research.ibm.com/trl/projects/webvis/jewel1/index_e.htm].

circular nodes whereas Ellimap [17] has elliptical nodes packed within parent nodes. Data Jewelry-Box [18] and Quantum Treemap [19] are slight variation on traditional Tree-Maps, where nodes are rectangular but instead of being sub-divided, they are just placed in the parent nodes leaving space in between. Fig. 5 shows Data Jewelry-Box visualization for partial structure of a very large web-site. The Data-Jewelry-Box techniques attempts to better represent intermediate nodes at the cost of maximum area usage. Space lost in this node packing technique depends

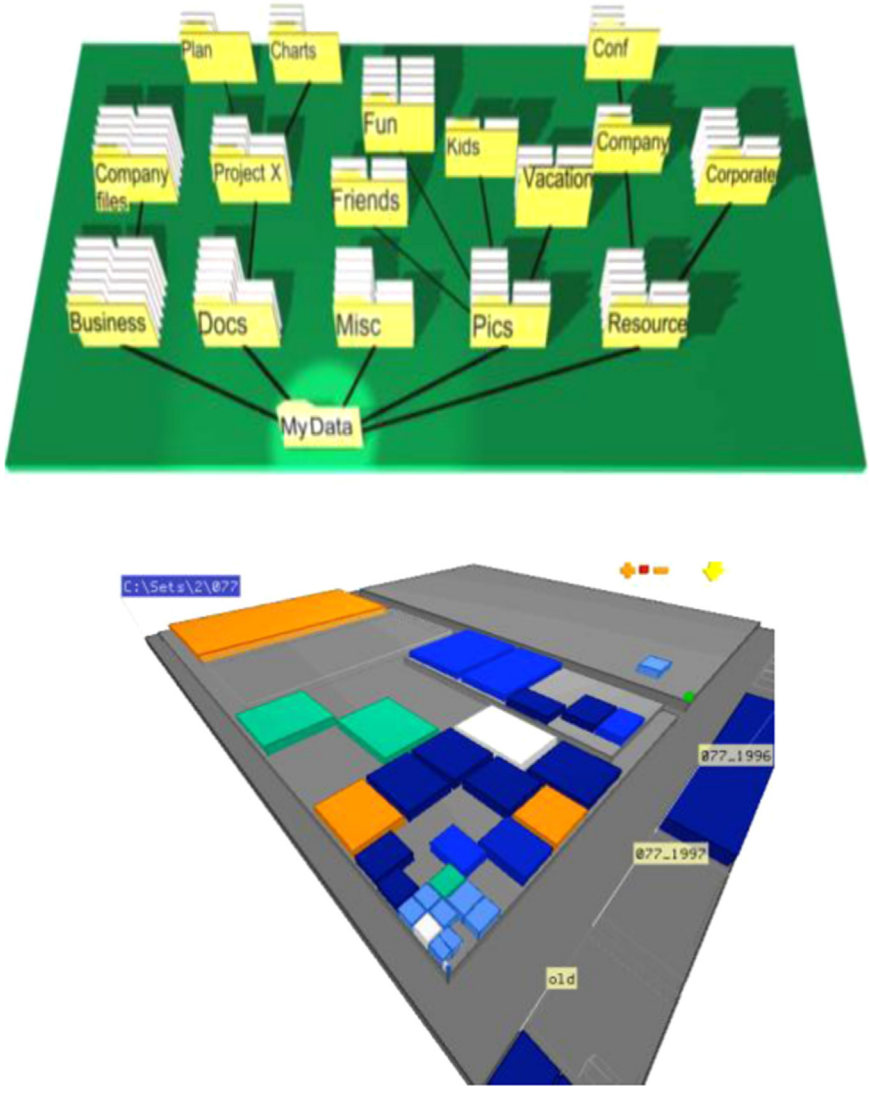

Fig. 6. 3D hierarchy visualization in space manager (top), Step Tree (bottom).

on the shape of the nodes and the structure of the hierarchy.

Another technique clearly indicating hierarchical levels in an inclusive technique is to add a third dimension. 3D Nested Treemap [20] has a simple representation of a treemap and there is a space between the two levels of the tree in 3D space. 2½-D Treemaps [21] have more complex structure where rectangular 2D nodes extrude onto a point in the third dimension forming pyramids. 3D Icicle [22] and Step Tree [23] shown in Fig. 6 are not completely space-filling, but have hierarchical levels with certain widths in the 3D space. With varied widths, the height of the node shows the level. Information Pyramids [24] is a similar visualization where 3D nodes are placed on the 2D surfaces of parent nodes. 3D Circle Packing [25] has cylindrical nodes on top of each other showing each intermediate node clearly in 3D space. Nested columns technique [26] also has circular nodes but the nodes are displayed as 3D circular walls, with child nodes inside the walls of their parent node. Lifted Treemap [27] has a slightly different 3D representation, where part of the tree map rising vertically from a node. It aims mainly at reducing cluttering in a large hierarchy.

As shown in Figs. 6, 3D visualization can often look aesthetically pleasant. However, it is computationally intensive, has occlusion problems and may not be space efficient. Smallman et al. [28] finds 2D display better for any task that requires precise judgment because they were able to provide more information clearly in a 2D display. When we view a 3D visualization on a 2D display, we are viewing the 2D projection of the 3D layout. Because of this the third dimension is unavailable to the user and a lot of information is lost.

Inclusion based implicit visualization shares parent node space with child nodes. We did not choose inclusion based implicit visualization for our system because these techniques do not allocate large enough space for intermediate nodes compared to the leaf nodes. As ERELT was

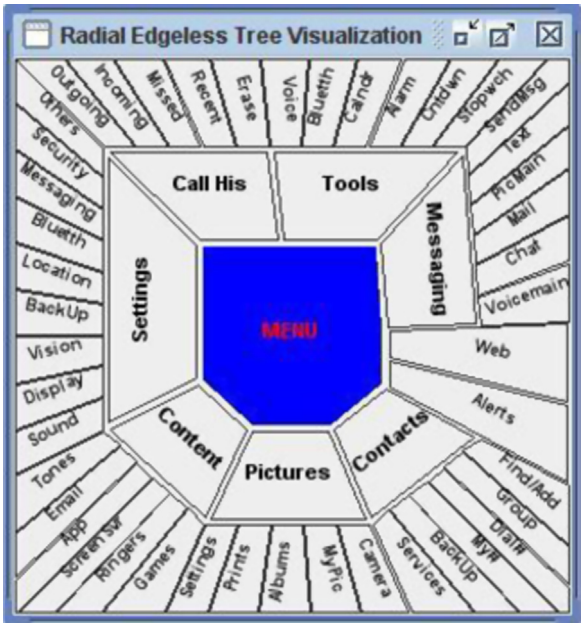

Fig. 7. Radial Edgeless Tree (RELT).

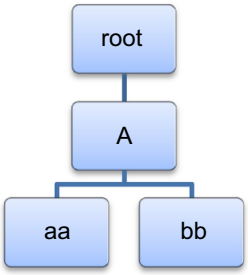

Fig. 8. A simple example where RELT fails to create tree.

designed to serve as mobile device user interface, it needs distinct space for the intermediate nodes not impeded by leaf node space.

2.1.3. Adjacency based implicit visualization

The intermediate node relations are more effectively displayed in implicit layout created through adjacency instead of enclosure [3]. Here, child nodes are drawn adjacent to one or more sides of their parent node. In such layouts tree nodes may extend parallel to the axes or extend radially outwards from a point in the root. Sunburst [29] and InterRing [30] are examples of this type of layout, where the root is at the center and the child nodes extend radially outwards. The deeper the tree level, the more area it occupies. This works well because trees usually have more nodes at deeper levels. Polar Treemap [26] and Pie-Tree [31] are also radial adjacency layouts with the area bounded by a circle. Aggregate Tree Map [32] has a non-circular radial layout. Cushioned Icicle Plot [33] is an example of adjacency layout which does not extend radially, but instead extends in one linear direction. 3D Beamtree [34] is a different type of implicit layout, where a node is a cylindrical beam attached to its parent beam to show parent–child relation.

The implicit, adjacency layouts discussed above are able to show intermediate levels of a hierarchy, independent of the shape of the display area, and thus do not fully utilize the screen space. For example, a Sunburst layout drawn at the center of a rectangular screen of a smartphone may leave large spaces at the two ends while fully occupying shorter width of the screen. Also, some of these and other radial division algorithms, such as InterRing [30], are designed for data visualization without many provisions for user interaction or exploration.

Radial Edgeless Tree (RELT) [35–37] is another adjacency-implicit layout technique. It combines the best features of inclusive and adjacency algorithms. Like the space filling inclusive layout, RELT divides the given display area into several vertices to maximize the area utilization. Like the adjacency layout, it clearly shows intermediate nodes. Fig. 7 shows a mobile phone menu in RELT layout.

Our tests on RELT reveal some particular cases where it does not produce good visualization. For example, if the second level of a tree has only one or two child nodes then the area of the root node would totally collapse or be very small. Since the shape of any node depends on radial lines dividing area between its children, RELT sometimes produce strange shaped polygon nodes. This case is illustrated in Figs. 8 and 9.

Visualization is easy for a viewer to understand when the spatial arrangement reveals the conceptual organization of the information [38]. Since RELT sacrifices a consistent structure and arrangement of nodes to best utilize the given space, it loses the instant clarity that layouts like Sunburst or Data Jewelry Box provides. Shape inconsistency in RELT also forces the node labels to fall in different directions hindering readability.

Adjacency based implicit visualization worked better for our system because it separates intermediate nodes space from leaf nodes. However, there are differences between our technique and most of these. Firstly, we put huge emphasis on shape of display area in determining visualization layout shape, whereas techniques like Sunburst [29] and InterRing [30] have fixed shape. Secondly, our technique needs to scale up by hiding data because we cannot increase in size, whereas many of the techniques displayed here can increase in size to display as much data as required.

2.2. Visualization as User Interface (UI)

Visualization techniques are on the one hand used for communicating the structure of data (e.g. Tree-Maps and Sunburst), and on the other hand, for interacting with the data as well (e.g. Space Manager, RELT). In mobile interfaces, lists are used to present hierarchical structures like menus, files and media library, displaying nodes under a single parent at a time. The list based UI offers linear search and thus is inefficient. Researchers have explored varied and unique solutions to the problem.

Wavelet Menu [39] is a hierarchical menu based on radial implicit visualization technique. The menu appears

on a circular ring around a contact point on the screen. The contact finger can then be moved to the target option in the menu. Once on the target option, the sub-option under the target option appears as another ring just outside the current menu ring. This process can be continued to navigate further into the menu hierarchy. Space Manager [6], already discussed, is a 3D, explicit hierarchical layout designed as mobile UI. Another UI system, Flow [40] slides in menu options from one side of the screen and slides out of the other based on context. This method is not based on hierarchical visualization, but can be used to explore hierarchical structures.

Not all types of hierarchical structures are well suited for mobile devices. Narrow and deep hierarchies have been proven to be better for mobile screen than broad and short hierarchy in terms of user performance and preference [41]. Narrow hierarchy means lesser number of nodes under each parent and since most traditional UI displays one level of nodes at a time, narrow hierarchy presents uses with lesser number of option to search through.

For a hierarchy visualization to be used as UI on a small screen there has to be a limit to the number of nodes it can contain. Users need to be able to distinguish between visualization elements and interact with any element with relative ease. Generally, the individual visual elements can be made distinct using variations in color (hue), brightness (luminance), size, and/or orientation of the elements. These variations can be applied to the border of the element or the whole. There is a limitation on the minimum size and viewing angle required to differentiate two visualization elements. Healey et al. [42] specify visual angle and resolution limit on these factors based on their user tests. However, these values are still much smaller than the area of an element required for touch based interaction and label display.

The limit on the number of visual elements on the screen implies that at any time the visualization displays a subset of nodes within the data hierarchy. The system needs to provide the users with both the focus on one subset of the hierarchy and overall contextual awareness of the current position in the data space. One way to provide such focus+context awareness is through fish-eye zooming [43]. Another approach is to display visual indicators in the focused area that presents contextual information with objects outside of the display area [44].

The primary means of interacting with mobile devices these days is through touch based interface. Most of the popular mobile devices are also equipped with motion and orientation sensors like accelerometer and gyroscope. Huber et al. [45] explored touch based and motion based input to design efficient user interfaces for mobile applications. Ruiz et al. [46] introduced DoubleFlip system to prevent false positives due to normal motion of mobile device. Even in case of touch based input, systems like Micro-Roll [47] explore how to detect simple rolling of a finger on the touch-screen instead of motion from one point to another.

Visualization and UI are often evaluated by running tests in labs using an eye-tracker. However, usability testing in mobile devices is more complex because these devices are designed for mobility. Thus a static user testing in a lab environment may not affectively reflect most usage scenarios [48]. Moreover, running eye-track test on a small screen is difficult and may not produce accurate result, even though some attempts have been made to apply eye-tracker testing on small screens [49]. Instead, we evaluate our visualization using comparative feedback from test subjects.

2.3. Enhanced RELT (ERELT)

ERELT [1] was designed to overcome the shortcomings of RELT and provide a dynamic visualization on a mobile UI for navigating hierarchical structures. ERELT produces more consistent shape and orientation for the nodes and node labels.

In ERELT the node shape depends on a set of proportionate rectangles and radial division lines, to ascertain that the node shapes are always in a proper form. This allows node labels to be written in a consistent manner yielding high readability.

ERELT provides mechanism to navigate through larger tree while displaying only a part of it. Unlike RELT, ERELT also maintains regular border between different levels of the tree which makes it easier to follow the structure

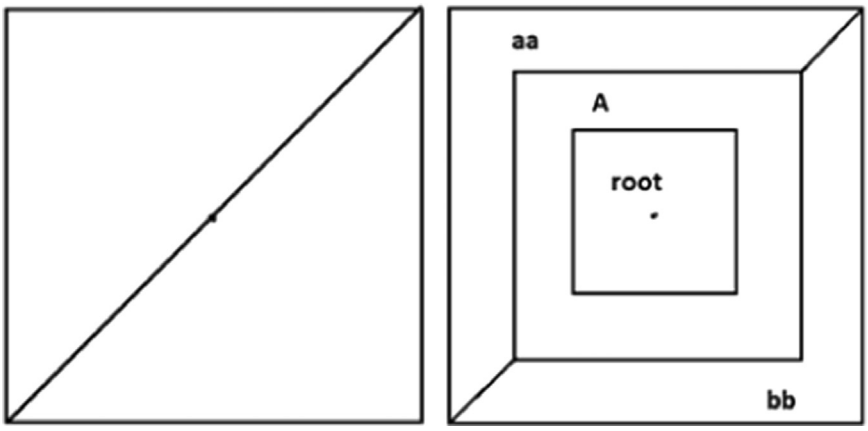

Fig. 9. (Left) Enhanced RELT (ERELT) Tree for the hierarchy in Fig. 8. (Right) The Same Tree generated by RELT algorithm.

when navigating between different parts. ERELT algorithm handles some of the special cases where RELT fails. One such case is shown in Figs. 8 and 9.

Based on our experience and user feedback from the initial prototype, we have made several improvements to RELT. These changes affected both the performance and presentation. We added color effects to indicate different node groups and to provide context when users scroll through various nodes in the tree. Visual effects are added to make tree elements appear like interactive buttons rather than plain drawings on the screen. To account for the performance cost of enhanced graphics, the visualization algorithm has since been completely re-designed to make it much faster. These changes are discussed in the next section.

3. Design

The main idea behind ERELT is to display a hierarchical structure on a small screen where users can view multiple levels of hierarchy in a single display and interact with it. This technique divides the screen proportionally into various levels. These divisions form multiple levels of rectangles one inside another. The innermost rectangle contains the root. The area between the innermost rectangle and the next rectangle contains the level under root, and so on. This implies that the deeper levels of the tree, which have more number of nodes, are represented by larger rectangles. Radial lines starting at root separate the nodes within a level. The space between two adjacent radial lines gets wider farther from the root, which is ideal because there are usually more nodes at levels farther away from the root. Fig. 10 shows a sample music library and Fig. 11 shows its ERELT representation.

Unlike our previous prototype [1], the root has been fixed at the top-left corner of the screen. Although it is more intuitive to place the root at the center of the layout, our tests showed that placing root at the center made the central region more crowded as drawn in ERELT prototype [1] in Fig. 12.

Having the root at the center and nodes populating at all sides of the root creates wider and shorter nodes. Our labeling scheme, however, works well with narrower and longer nodes. Though we adopt corner-rooted layout for this user study, we can easily generate center-rooted layout, which is more intuitive and better suited for wider screens, e.g. tablets and PCs.

Apart from layout design choices, user interactions and implementation issues should also be considered. The authors of InterRing [30] had attempted to establish a set of functionality that a good hierarchy visualization and navigation system should provide. For example, users should be able to select any particular node and interact with it. Different nodes or levels should disappear or reappear based on selection to preserve memory and screen area. There should be a mechanism for users to access hidden nodes by panning or rotating the tree view. Karstens et al. [5] proposed another set of considerations for tree visualization aiming at efficient implementation. They suggest limiting the number of tree elements drawn on the screen at a time. Visualization in 2D graphics is much faster than in 3D. Also the paper encourages the use of simple graphical primitives like straight lines instead of curves.

Additional design elements such as animation are also known to drastically change the user experience [50]. Correct use of animation can make GUI more intuitive and the delay in application response appears shorter.

3.1. Layout

The display area is divided into a number of polygons, each representing a node in the hierarchy. Two important issues are considered:

- The division should consider the size and the shape of the display area to maximize the space utilization.
- The division should distribute the area such that each node is given a minimum space to fill the label.

Briefly, the layout algorithms works by first dividing the display area into different levels of the tree, then dividing each level into different nodes in the level. The layout algorithm is a recursive algorithm, where at each iteration, the root of a given sub-tree is separated from rest of the sub-tree and then displayed on the screen. The algorithm is then repeated for each sub-tree under the root.

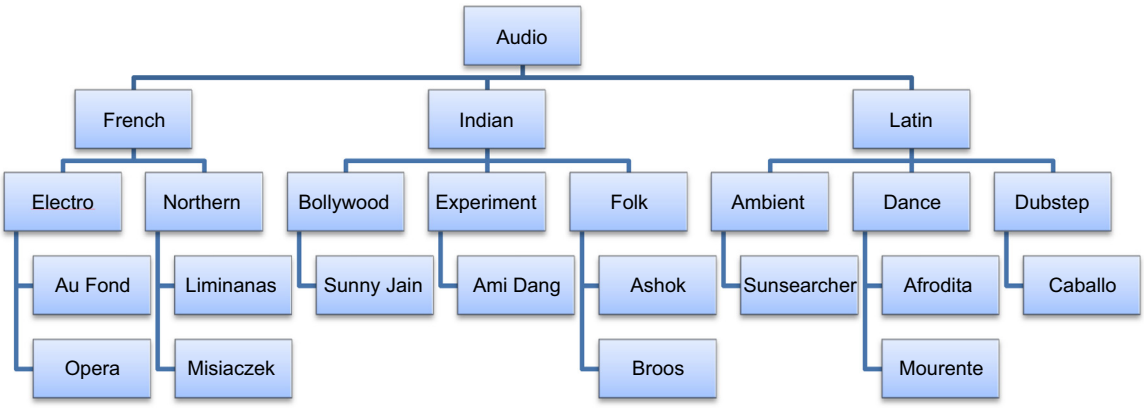

Fig. 10. Sample music library segment (language, genre and artist levels).

To facilitate accurate description of our approach and algorithms, we define various parameters as the following:

N = any given node
$\mathbf{N_{parent}}$ = parent of the given node
$\mathbf{Rect_L}$ = virtual rectangle at level **L**
$\mathbf{Rect_{L+1}}$ = virtual rectangle at level **L**+**1**
$\mathbf{Line_{start}}$ = radial line forming the starting boundary of N
$\mathbf{Line_{end}}$ = radial line forming the ending boundary of N
$\mathbf{W_N}$ = weight of N
$\mathbf{W_{parent}}$ = weight of N_{parent}
$\mathbf{W_{cumulativeN}}$ = cumulative weight of all the sibling nodes processed before **N**, under $\mathbf{N_{parent}}$

Before starting the division algorithm, a setup process is performed:

```
// Weight of the node is the factor that determines the relative size of
   the nodes
void setWeight (Node node) {
   if (node is Leaf_Node) {
      // Set leaf node weight as 1 for equal leaf node sizing
      node.weight=1;
   } else {
      // Set parent node weight as sum of child node weights
      for each (Node n in node.childNodes) {
         setWeight (n);
         node.weight += n.weight;
      }
   }
}
```

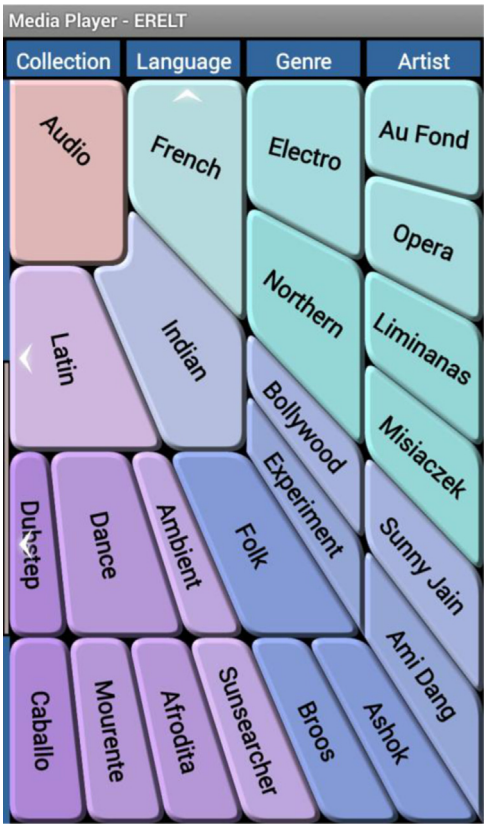

Fig. 11. ERELT visualization of the structure in Fig. 10.

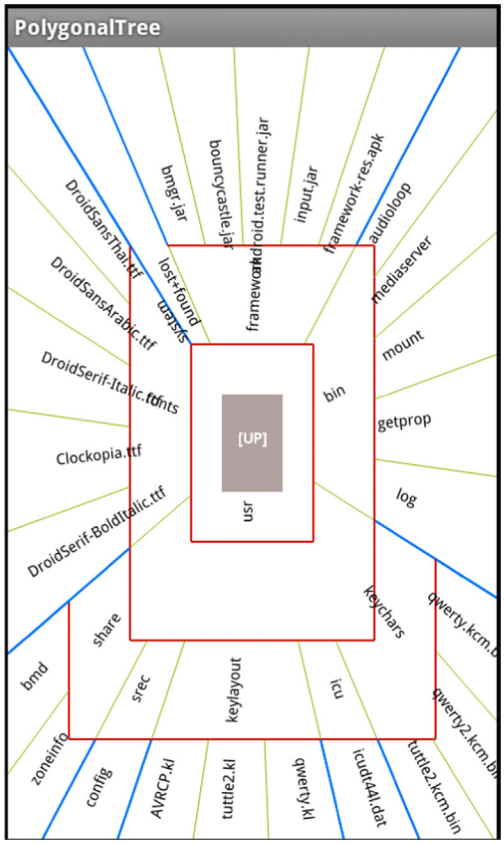

Fig. 12. Center-rooted Layout in an earlier prototype: less space for labeling in mid-area.

A node's weight determines the relative size of the node. In the algorithm above the weight of all the leaf nodes is set 1. This generates tree having leaf nodes of similar sizes. Setting the leaf nodes' weight based on any other property will result in a display tree with area distribution that reflects that property. E.g. ERELT structure can be used to display file structure in a cloud drive like Dropbox, where leaf nodes' weight can be size of the file the leaf node is representing. Here, the relative sizes of the nodes with represent sizes of files and folders in the cloud drive.

As illustrated in Fig. 13, we start by selecting the top-left corner as the root for the tree. This choice of root location is arbitrarily made for the current user study and not essential for the algorithm to work. The algorithm works for the root at any location in the display area.

We divide the screen into H virtual rectangles, where H is the total height of the tree measured by the number of levels. For large trees, H can be limited to a maximum threshold value. Handling large trees will be discussed in details later. These rectangles are "virtual" because they are not yet drawn onto the screen but only exist as variables in the algorithm. This definition of "virtual" will apply to any lines or rectangles. The H virtual rectangles are created in the following steps:

1. Connect four corners of the display area to the root center (currently top-left corner) by four virtual lines. These four lines will be referred to as “sector lines”.
2. Divide the four lines into H equal segments (one for each level).
3. Form virtual rectangles by connecting four division points at each level.

Each virtual rectangle has the same aspect ratio as that of the display area. It represents one level of the tree, with the size proportional to the level of the tree. The four sections of the display area divided in Step 1 by the sector lines will be referred to as *sectors*. Virtual rectangles for corner-rooted layout for a tree of 4 levels are shown in Fig. 13. This layout helps utilize the entire screen space.

For any given node, the division algorithm listed below calculates the area and draws each node as a polygonal button. Drawing nodes onto the screen is handled by an algorithm that uses radial lines to divide the node area into smaller sections to represent child nodes. This algorithm runs recursively for each node and its child nodes until the entire tree is plotted.

The following function is called to recursively draw every node:

```
// Pass the Root Node to the following function to recursively draw the
   entire tree
void plotTree (Node N) {
   drawNode(N);
   if (N is not Leaf_Node)
      for each (Node child in N.childNodes)
         plotTree (child);
}
// Calculate and draw boundary lines for the given node
void drawNode (Node N) {
   if (N is Root_Node){
      // Innermost rectangle as the root
      plotNodeOnAreaEnclosedByShapes (Rect1);
   } else {
      Get WN, Wparent, WcumulativeN;
      // Divide the area under Nparent radially into Wparent parts with
   equal areas,
      // and get dividing lines
      Lines[]=divideArea (Nparent, Wparent);
      // Select the starting and ending radial lines for the current node
      // based on cumulative weight of previous siblings and its own
   weight
      Linestart=Lines[WcumulativeN];
      Lineend=Lines[WcumulativeN + WN];
      // Draw the polygonal node as the area between RectL, RectL+1,
   Linestart & Lineend
      plotNodeOnAreaEnclosedByShapes (RectL, RectL+1, Linestart,
   Lineend);
   }
}
// Divide the area under nodeArea radially into nodeWeight parts with
   equal area,
// and compute dividing lines
Lines[] divideArea (nodeArea, nodeWeight) {
   radialLines=[]; // empty array
   partArea=nodeArea/nodeWeight;
   halfArea=nodeArea/2;
   area=partArea;
   count=1;
   while (count <=nodeWeight) {
         if (area <halfArea)
            angle=arctan(2*area/(width*width));
      else
            angle=PI/2 - arctan(2*(wholeArea - area)/
   (height*height));
      radialLine[count]=line(start=center, angle=angle);
      area +=partArea;
      count++;
   }
   return radialLines;
}
```

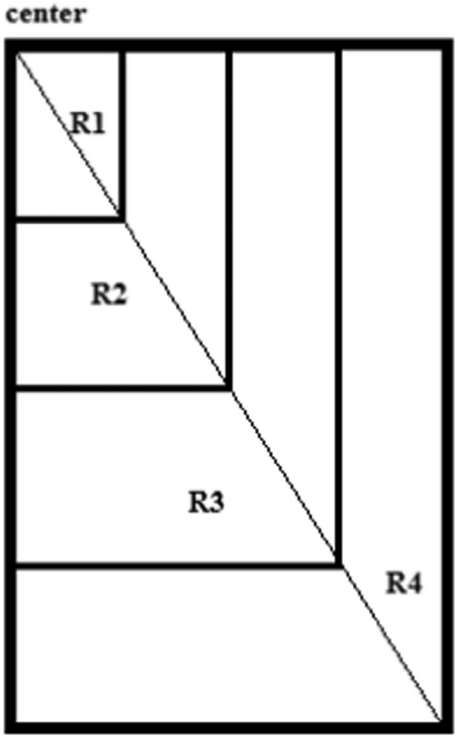

Fig. 13. Virtual rectangles for corner-rooted layout for a tree of four levels.

3.2. Scalability

In theory the layout algorithm works for a tree of any size. However, for applications that require labels on all the nodes, drawing a large number of nodes on a single screen makes labels unreadable and the visualization loses clarity. To handle such situations the algorithm enforces some limitations.

1. Level threshold is the total number of levels displayed on the screen at a time. For most popular smartphone screens (between four and five inches), we found four to be the proper level threshold. For a smartphone screen less than four inches, the threshold value of three is better for readability. Larger devices (5–6 in.) can properly display trees with level threshold of five or six.

2. Branching threshold is the total number of child nodes displayed under each parent at a time. The threshold value of three was found to be good for general cases. For smaller trees however, the value of four works just as well.

After the two thresholds are set, the tree structure is pruned using these values and only the sub-tree within the threshold is displayed. The remaining part of the tree is considered hidden. To display any part of the tree, a node in that part is selected as the root and the corresponding sub-tree is displayed.

Wherever the branching threshold is applied, arrow indicators are used to show sibling nodes hidden to the left or right of the displayed nodes, as shown in Fig. 14.

3.3. Complexity analysis and comparison with RELT

The computation time for ERELT does not depend on the size of the tree because the algorithm draws only a sub-tree at a time. Thus, the level threshold and the branching threshold values determine the time complexity.

For a tree wbith N nodes,
Level threshold = **d**
Branching threshold = **b**
Maximum nodes to be drawn = $\mathbf{b^d} - 1$
Thus,
Computational complexity = $\mathbf{O(b^d)}$
For a given device, **b** and **d** do not depend on the size of the tree and can be set as constants (typically around 3 to 5), therefore,
Computational complexity = **O(1)**

Since RELT [35–37] only describes method to draw the complete tree, following assumptions are made for comparison:

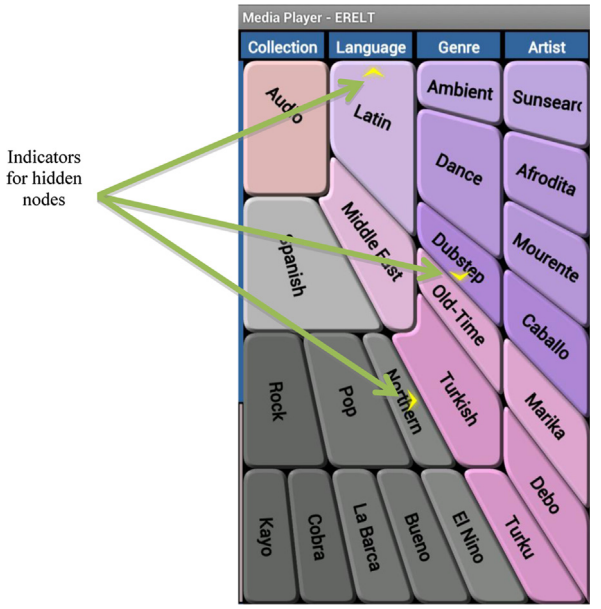

Fig. 14. The arrow markers indicating direction of nodes hidden due to branching threshold.

1. Level threshold is larger than or equal to the tree height.
2. Branching threshold is larger than or equal to the largest possible number of child nodes of a single node.

The RELT algorithm operates by fixing one radial line around a node fixed and moving the other radial line along the display area border, one pixel at a time. At each step, the area within the two radial lines is calculated. The process continues until the area obtained is equal to the required area for the node. The number of calculations to obtain the area for a node depends on number of pixels traversed along the border. Thus, even when the tree size remains constant, the computation cost increases with the increase in the size of the display.

The ERELT algorithm computes the display area by calculating the required angle at the center between the two radial lines. Given the required area value, this angle can be directly calculated using trigonometric functions. Thus, the computation does not depend on the display size, but only on the number of nodes to be displayed.

For a clear and readable layout the number of nodes to be displayed at any level must be smaller than the pixels along the border. Thus the computation in ERELT is much cheaper than that in RELT for displaying trees of the same size.

3.4. Coloring

Since the number of nodes increases away from the root, it becomes difficult to distinguish two nodes of different parents and two nodes of the same parent at deeper levels. To avoid confusion and provide a clear structure at the first glance, the algorithm uses a dynamic coloring scheme that allocates different colors to the nodes based on their "nearness" to each other. Here, nearness between two nodes at the same level refers to how far the tree needs to be traversed in the direction from the root to find a common ancestor node for the two nodes. If there are three nodes side by side with two of the nodes sharing the same parent, then the two sibling nodes have less color variation than the third node.

We use the HSL color space as shown in Fig. 15. to distribute colors to nodes. HSL has three components – Hue (color), Saturation (Intensity of color), and Lightness (Amount of black or white in the color). The algorithm assigns the nodes with the same color as their parent node and then changes hue, saturation or lightness based on level and position.

Coloring Algorithm:
// $\Delta\mathbf{h}$, $\Delta\mathbf{s}$ *and* $\Delta\mathbf{l}$ *are constant increments for hue, saturation and lightness values for adjacent nodes*
// *and* **N** *is the* **n**th *child of its parent*
setColor (Node N) {
  if (N is Root_Node) {
    // *set a predefined color for root*
    N.color = HSL (h, s, l);
  else if (N.level = Leaf_Node)
    N.color = N.parent.color;
  else if (N.level = Level_2)
    N.color = HSL ($h_{parent} + n^*\Delta h$, s_{parent}, l_{parent});
  else if (N.level = Level_3)
    N.color = HSL (h_{parent}, $s_{parent} + n^*\Delta s$, l_{parent});
  else if (N.level = Level_4)
    N.color = HSL (h_{parent}, s_{parent}, $l_{parent} + n^*\Delta l$);

}

In the coloring algorithm, changing colors of leaf nodes of the same parent is unnecessary. A leaf node inherits its parent node's color and thus will always be different from the leaf nodes under other nodes. The algorithm is designed to display for up to five levels. As discussed above, four levels are appropriate for displaying readable labels.

3.5. Labeling

The ERELT layout is designed with narrow and long nodes. This allows a node to be labeled along the length of the node. The consistent label style makes it easy for users to scan through the tree. For each node, the line bisecting the two radial lines forming the node is calculated and the label is placed along this line.

4. Navigation

The ERELT handles large trees by only displaying a subtree to maintain clarity and readability. Thus it is essential to have an effective interaction mechanism that allows users to navigate the structure and display the desired nodes. Our design supports touch-screen interactions available on most smartphones, with intuitive gestures triggering visualization and interactions.

Since the entire tree layout is an image drawn by the algorithm, for any user interaction on the tree layout, the interaction point in the image is obtained. Then the tree element corresponding to that point is identified.

There are several ways to detect location of a point relative to graphics objects. Point-Inside-Polygon Tests [51] are computationally expensive. Since the ERELT layout has regular patterns (i.e. structures formed by parts of radial lines and parts of proportionate rectangles), good performance could be achieved by taking advantage of these patterns as described in the following algorithm.

Our algorithm first finds the radial line connecting touch location and the center of the layout, and calculates the angle of the radial line. This angle gives the sector where the touch location lies. As shown in Figs. 16

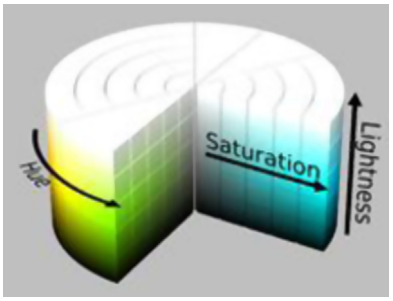

Fig. 15. HSL color-space.

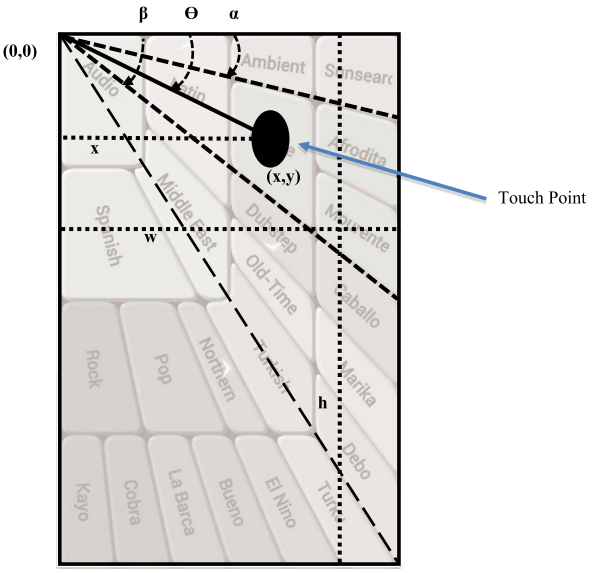

Fig. 17. Touch location node detection for top-right sector.

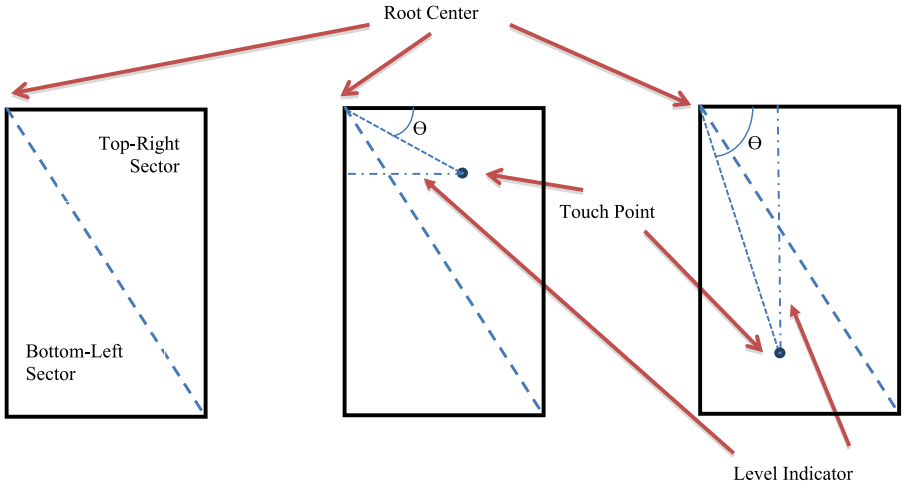

Fig. 16. Sectors, center angle (Θ) and level indicator.

and 17, for the cornered-root layout there can be two sectors – top-right and bottom-left. The algorithm and the definition of various values used in the algorithm are described below. Various virtual lines and elements of this algorithm are illustrated in Fig. 17.

Touch Location Detection:

```
x=x-coordinate of touch point
y=y-coordinate of touch point
H=height of the display sub-tree
w=width of the display area
h=height of the display area
// Find the node in the display tree where the touch
Node findTappedNode () {
  // Find the level of the sub-tree where the touch point lies
  if (top-right sector) {
    targetLevel=1 + (integer) (x / (w/H));
  } else {
    targetLevel=1 + (integer) (y / (h/H));
  }
  // Recursively search for the touch point in display sub-tree, starting
  from the Root Node
  return searchForPointInSubTree (Root_Node);
}
// Recursive function to search for the target node in given sub-tree
Node searchForPointInSubTree (Node N) {
  // If current node depth is the same as target level of touch, the
  tapped-node is found
  if (targetLevel=N.level)
    return N;
  // Find the child node with start and end angle containing the angle
  of touch
  foreach (Node child in Node.children) {
    α=child.startAngle;
    β=child.endAngle;
    if (Θ >= α && Θ <= β) {
      return searchForPointInSubTree (child);
      }
  }
}
```

This algorithm requires fewer comparisons and computations than the Point-Inside-Polygon Test.

The user's navigational gestures/actions are interpreted as *Tap* or *Scroll* gestures. Tap actions allow users to navigate to the tree levels hidden in the current view due to the level threshold. Scrolling gestures allow users to navigate to the sibling nodes hidden in the current view due to the branching threshold.

4.1. Tap

There are three types of tap interactions:

1. Whenever the user taps on a node in the tree, that node becomes the new root of the tree and a new layout will be drawn. Depending on the application, if the tapped node is actionable then appropriate actions can be taken instead of setting the node as root. One example is opening the selected file where tree is a file directory and tapped node as a leaf node represents a file.
2. Tapping within the root node triggers an "Up" command. If the current root is not the root of the main tree then tapping on it sets the parent of the current root as the new root and the new tree layout is drawn.

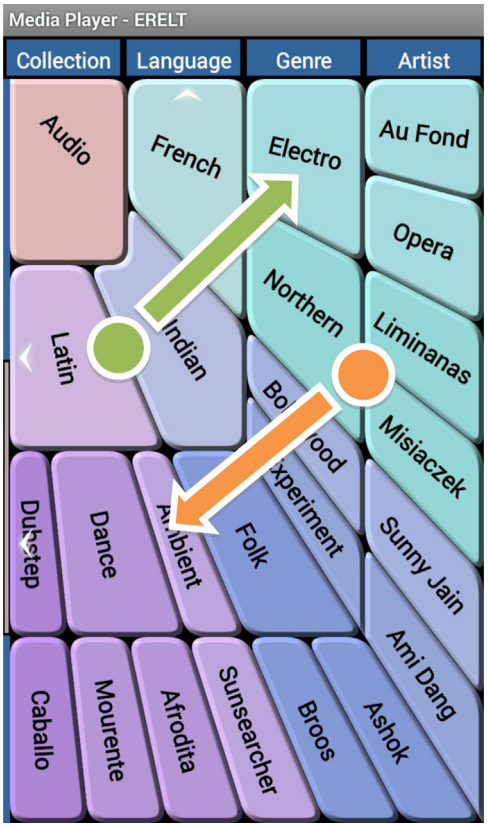

Fig. 18. Left/right scrolling.

3. Tapping on the back button of the device triggers a "Back" command. ERELT maintains a limited history of user interactions. The back command loads the previous sub-tree into the display. This allows users to quickly move into and out of several nodes, multiple levels down the tree.

4.2. Scroll (Drag)

A scrolling gesture illustrated in Fig. 18 occurs when the user touches the screen with one finger and drags the finger over a distance to a different point in the display, before raising the finger from the screen. Depending on the initial and final points of touch and the angle of the drag relative to the root, the gesture can be interpreted as:

1. **Left/Right Scroll:** This gesture starts at one of the nodes whose sibling nodes have been hidden due to the branching threshold. The drag and the arrow moves to the opposite direction of the hidden nodes to simulate the action of dragging the nodes to give room to display the hidden nodes. This gesture results in left/right scrolling of the nodes under the initial point of touch.
2. **Drag-Navigation:** This UI mechanism is similar to the scroll-bars on a table or list based interface. There is a thin scroll bar which can be moved by dragging the finger along left edge of the display screen. This allows users to quickly scroll through child nodes of current root node. During this interaction a hint box appears on

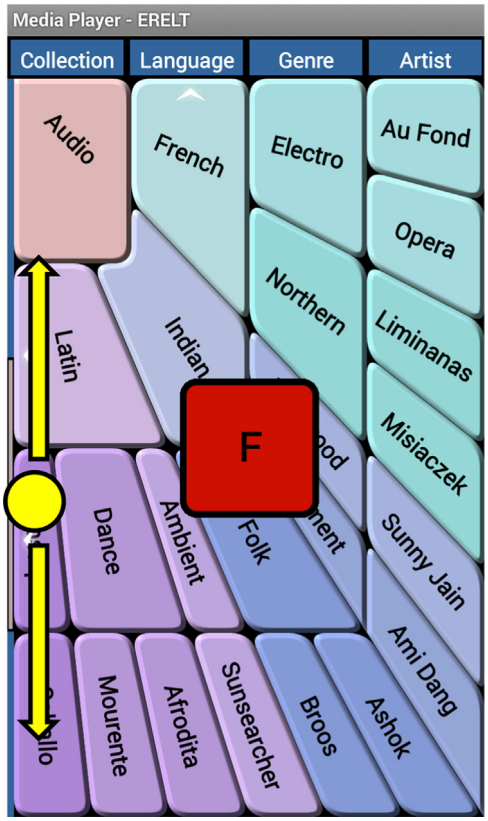

Fig. 19. Text hint during drag navigation.

the screen displaying the initial letter of current child node name displayed under the root. Fig. 19 shows such a hint being displayed during drag-navigation scrolling.

3. **Up:** From the initial touch location, the dragged finger moves away from the root. This gesture simulates pulling the tree structure down so that new root is revealed. It results in the "Up" command which sets the current root's parent as the new root.
4. **Back:** From the initial touch location the dragged finger moves in a direction towards the root. This gesture results in "Back" command, which loads the previous root as the current root. This action does not directly resemble any real world action.

4.3. *Flick motion*

Flick motion is the motion gesture based input sent to the system. Flick gesture occurs when the user holding the mobile device quickly jerks his/her hands to the right or left. Flicking the phone to right scrolls the layout to display first child under the current root, while flicking the phone to the left scrolls the layout to display last child under the current root node.

5. Implementation and optimization

ERELT is implemented on an Android platform and tested in Froyo (2.2), Gingerbread (2.3), Ice-Cream Sandwich (4.0) and Jelly-Bean (4.1) versions of Android OS. The application is written in Java to run in Dalvik VM in Android. We used Eclipse IDE with Android SDK for development. It was tested to run in Android simulator (Android 2.2, 2.3.3, 4.0 and 4.1), Motorola DroidX (Android 2.3.3) and Samsung GS3 (Android 4.1) during the development.

For prototyping and developer testing, ERELT has been built for two different Android applications: A File-system Explorer and a Media Player. These test applications also have traditional list based layout for comparison with the tree layout. An ERELT application uses Android 2D graphics functions to draw tree layout and animations onto a bitmap object. The image is then rendered on an Image View controller and presented to the user.

During the design and development of ERELT, one of the major issues is the performance. It is critical for user interactions to be at least comparable to Android's default List view. This was a challenge because unlike List view, ERELT has to process multiple levels of a tree, performs geometric and trigonometric calculations to create the layout, detects touches, and calls graphics routines to render the tree. Several optimization techniques are used to improve performance and user experience.

1. A major optimization potential is in the way Android 2D graphics functions are called. Android 2D graphics lets users access pixels of a graphics image stored in the video memory by using getPixel and setPixel functions. Working with a large image and calling these functions with each of the large number of pixels slow down the processing time considerably. A better alternative is to load the entire image into an accessible memory block from the video memory. The memory data can be directly modified to perform graphics operations on the image. Finally, we create a new image from the modified memory block of the image.
2. We have minimized the number of floating point calculations (specially multiplications) by pre-calculating values wherever possible. In some cases multiplications operations are replaced by addition/subtraction or shift operations.
3. Fixing the root of the tree at one location helps immensely, because it allows some of the layout computations to be done during the application startup and then used later wherever required. In some cases, certain computations could be performed beforehand, with the results directly used in the code.
4. To quickly detect the touch location, radial line angles and sector line angles are stored in memory for fast comparison.
5. A major effect on user experience is animated transitions. Multiple threads are used to achieve the effect. Whenever the user issues a command for drawing a new layout, the time taken can be utilized to animate the transition in the main thread. The layout preparation was done in a separate thread. The animation includes simple visual effects, such as scaling and rotation of the current image, such that they do not add to the processing time. Though the animation does nothing to reduce the reaction time of the system, it

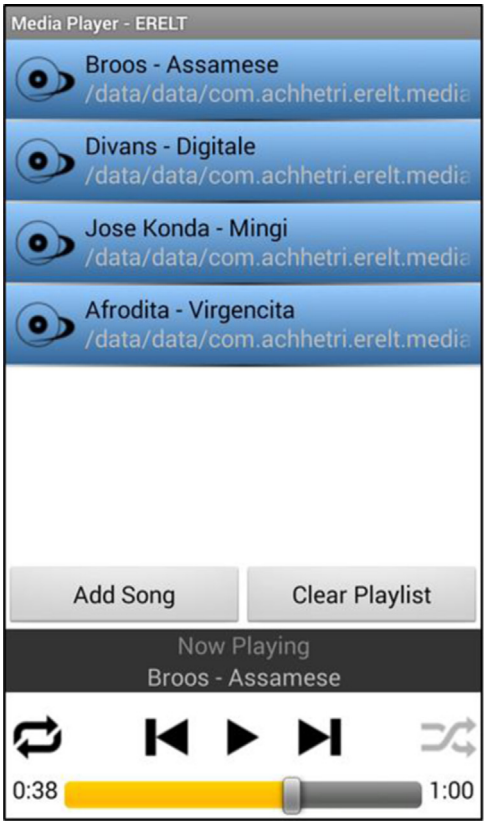

Fig. 20. Media control interface of the media player application.

however gives users some sense of actions before the next layout is ready. This prevents users from feeling anxious in waiting for the application to respond to their input, thus making the experience instantaneous.

6. Evaluation

6.1. Usability study

In order to evaluate whether ERELT provides an efficient interface for navigation, a usability study was performed to compare ERELT against a traditional list based interface for accessing hierarchical structures. The objective of the study was to compare effort and time taken to perform a set of tasks on ERELT and List views.

Since physical effort could be estimated by the number of interactions required [52], the number of touches required to perform any task was used to measure the effort [53]. In addition, mental or cognitive effort also affects the user experience [52]. However, the measurement of cognitive effort requires complex supervised techniques, such as eye-tracking, which is beyond the scope of this study.

6.2. User profile

A total of 38 subjects participated in the study. Most subjects were volunteers from a population of undergraduate and graduate students in the Computer Science, and the Arts and Technology programs at the University of Texas at Dallas, USA and School of Software Engineering at Tianjin University in China, between the ages of 18–30. About three-fourth of the subjects were completely new to the ERELT interface. The remaining quarter had seen the visualization technique but had not used it. Almost all the subjects used their own Android smartphones for the test, thus, were assumed to be familiar with the Android OS and the device.

6.3. Setup

A media player application was chosen for the usability study, where the media library has a hierarchical structure, composed of various levels such as language, genre, artist, albums and tracks. We chose this application as a test case because 63% of smartphone users use mobile phones as a music player [54]. It is the fifth highest use after GPS/navigation, social media, local search and news reading. A smartphone as media player often stores a large collection of music, and there is a need to manage this collection in more user friendly way [55]. An Android application was created that allows subjects to navigate the music library in ERELT interface to find the track they want to listen to. Fig. 20 shows the media control interface of the application. Pressing the “Add Song” button in the control interface opens up the music library in ERELT. The level threshold of four and branching threshold of three were chosen. “Up” and “Back” gestures were disabled in the usability study to reduce the complexity of the learning curve for the test subjects. Subjects can click on the root node for “Up” functionality, and use Android’s “Back” button for “Back” functionality. The application also includes an option for navigating the library through a list-based interface as shown in Fig. 21 for comparison.

All the subjects performed the same test on the same music library. Instead of retrieving media files from the subject’s own smartphone, the application was created with built-in music library of 8 languages, 22 genre, 38 artists, 49 albums and 138 tracks. Including the complete media files with this library structure would have made the application huge in size. Thus, we only included short (1 min) snippets of about 24 tracks, and associated all of 138 tracks in the library listing with one of these 24 tracks. The music library was based on a collection of music pieces from the real international artists providing their music freely over the Internet. The advantage of using independent international artists from various parts of the world was that there was little chance that some subjects were more familiar with the music library than others before the test. This initial library was then altered to change the structure and names of various elements so that it would be better suited for creating a variety of tests. For example, some of branches were removed from one place and added to another to create sections of the tree that were much denser than the rest of the tree. Another reason for this renaming and restructuring was to create multiple segments of the tree that had similar structures but differently named elements. This allowed us to create comparable tests for List view and Tree view (using ERELT) that looked different on names but similar on the structure.

The test included timed tasks which were designed in pairs, one for list and one for tree. They looked different based on names of elements in the tasks but required navigating similar structures in the hierarchy. This variation allows any subject to take the two tests without any learning effects. To remove bias, half of the subjects performed the list view test first followed by the tree view test, and the other half performed the tests in reverse order. The tasks involved in the test are detailed in the next section.

The application stores logs of user interaction events with timestamps. The logs can be manually or automatically uploaded to a data collection server. Each subject enters an ID into the application, which helps to organize the log files in the data server.

6.4. Test design

The user tasks include a paper based test and application test, consisting of four components:

6.4.1. Test for basic understanding

The first part of the test provided the subjects with a simple ERELT diagram shown in Fig. 22 containing four levels and asked them to answer five basic questions. They were asked to identify **Morchang**, **Recordings** and **Ashok** as artist, album or track nodes. Then they were asked to name artist and album for **Assamese** track and artist for **Field** album. This test checks for basic understanding of the ERELT layout before starting on application test. If a subject could not complete this part correctly, he/she was considered not understanding the ERELT structure, and the test data was not included for the evaluation.

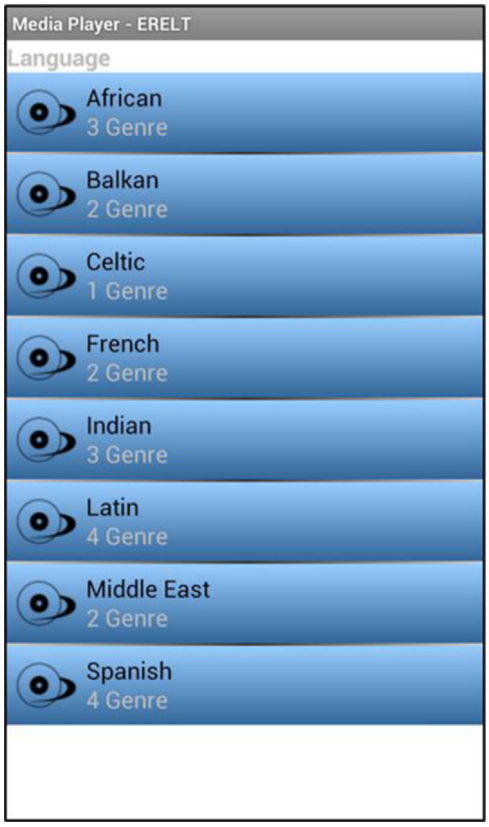

Fig. 21. Media library in list view.

6.4.2. Familiarization with the music library

The second part of the test has five questions on the music library. This would require subjects to browse the music library and perform tasks, such as counting albums with certain names, albums under certain artists, languages with certain genre, etc. This part of test is not timed but analyzed for general correctness. The tasks are designed to allow subjects to explore the application, its features and the music library, and thus much longer compared to the timed tasks in the next section. The subjects were not graded on their test results. They familiarize themselves with the music library before performing the timed-tasks. This step is necessary because the smartphone users are somewhat familiar with the media collection in their own smartphones.

6.4.3. Timed tasks

The third part consists of fix tasks that ask subjects to find and add various tracks to the media player playlist. In Tasks 1 & 2, the entire path from the root to the track was given. Tasks 3, 4 and 5 involved searching various parts of the hierarchy for a certain genre or album and adding multiple tracks to the playlist. Finally, Task 6 involved adding multiple tracks from a given path in the library. The time and the number of touches used to complete these tasks were stored in the log files. A list of timed tasks is shown below.

Timed Tasks Sets:

Task Set A

1. Add the track "**Etelemelo**" from the album "**Nzambe**" by the artist "**Jose Konda**" in the "**Afrobeat**" genre of the "**African**" music.
2. Add the track "**Paga**" from the album "**Elegante**" by the artist "**Konsum**" in the "**Electro**" genre of the "**Spanish**" music.
3. Add any two tracks from the "**Electro**" genre (can be from the same artist or album).
4. Add any two tracks from the "**Northern**" genre of any two different languages.
5. Add three tracks from three different "**Singles**" albums.
6. Add all tracks from the "**Rock**" genre of the "**Spanish**" language.

Task Set B

1. Add the track "**Digitale**" from the album "**Pas de tigre**" by the artist "**Divans**" in the "**Afrobeat**" genre of the "**African**" music.
2. Add the track "**Radium**" from the album "**Elegante**" by the artist "**Konsum**" in the "**Electro**" genre of the "**Spanish**" music.
3. Add any two tracks from the "**Dance**" genre (can be from the same artist or album).
4. Add any two tracks from the "**Folk**" genre of any two different languages.
5. Add three tracks from three different "**Live**" albums.
6. Add all tracks from the "**Dance**" genre of the "**Latin**" language.

6.4.4. Feedback

The final part of the test was a general survey that asks subjects their opinion on ease and quickness of using ERELT vs. traditional list layout. In addition, we collected

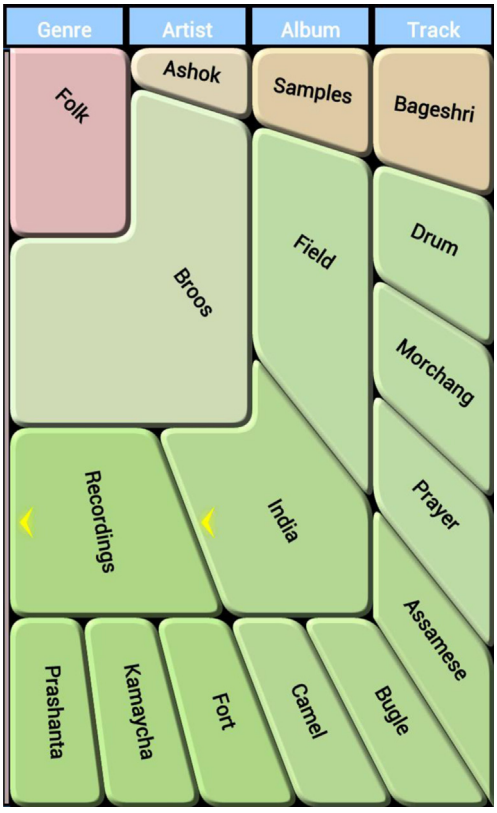

Fig. 22. Tree layout for basic understanding.

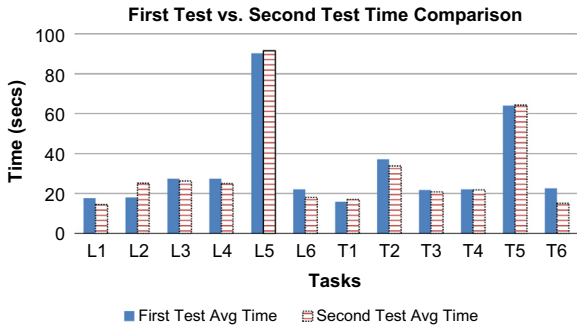

Fig. 23. Time comparison between the first and the second testers showed no significant difference.

their input on UI mechanism, missing features, suggestions on changes and enhancements, etc. The subjects expressed their perceived difficulty in using the tree layout on a scale of 1 to 10 in increasing order of difficulty. Then they were asked to choose which layout felt quicker to work with. The survey also allowed the subjects to comment if they thought some options or buttons were missing from the interface and any features they found to be counter-intuitive. These comments and ratings were only collected as a guide for future improvements. These were not used as validation of the visualization technique.

Through the series of timed tasks we aim to test for the hypothesis that there is a significant difference in time taken and touches required between the list based interface and tree based interface.

6.5. Results

Of 38 subjects, the results of 26 are used. 12 results were rejected for one or more of the following reasons: incomplete test, incorrect actions, and distracted during test. A subject is considered distracted if he/she took long pauses when performing the tasks. This is measured by checking the time spent in the middle of a task without any action. The mean time between actions during the entire test was measured to be approximately 2 and 3.5 seconds for list and tree layouts respectively. A subject who spent more than 20 seconds on the same screen without any action was removed from the analysis in the final result.

Out of the 26 valid test data, 12 subjects took list-first test and 14 took tree-first test. Comparing the average time and the average number of touches for the same tests done first and second did not, however, show that second test has any advantage over the first. This check verifies that the counter-balancing was effective and the subjects who did the tree layout test after the list layout test did not complete it faster or with fewer touches than those who did the tree layout test before the list layout test. The comparisons in time and the number of touches are shown in Figs. 23 and 24 respectively.

Comparing the average number of touches required to complete each of six different tasks between list and tree shows a significant improvement in ERELT based layout, as shown in Fig. 25. The mean values of number of touches and time for each task are shown in Table 1. In all but Task 2, there was a reduction of about 40% or higher in number of touches required when subjects switched from list based interface to ERELT interface. This anomalous result for Task 2 is discussed in Section 6.7.

Comparing the average time taken by subjects to complete each of the six tasks in list and tree showed two different results for two types of tasks, i.e. direct path and exploratory as shown in Fig. 26. This is discussed in detail in Section 6.7.

As shown in Table 1, average time per touch is clearly longer for the tree layout than for the list layout. However, the tree layout makes up for this difference by requiring less number of touches to complete the tasks, hence reducing the total time for a task.

6.6. T-test

We ran two-tailed paired *T*-tests to measure the significance of these results, as shown in Table 2.

The difference in touch count is statistically significant for all the tasks. In time comparison, however, the differences in Task 1, Task 4 and Task 6 are insignificant.

6.7. Interpretation

The average touches and time spent can be interpreted on the basis of tasks as the following:

1. **Tasks 1 and 2 (Direct path task):**

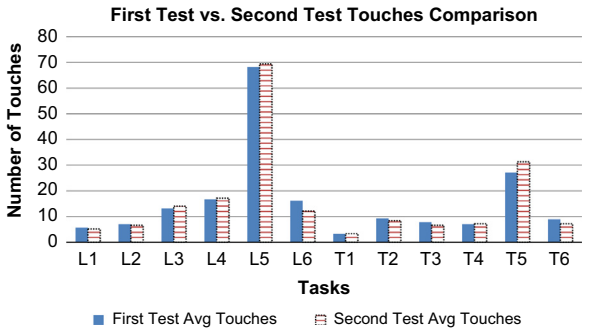

Fig. 24. Touches comparison between the first and the second testers showed no significant difference.

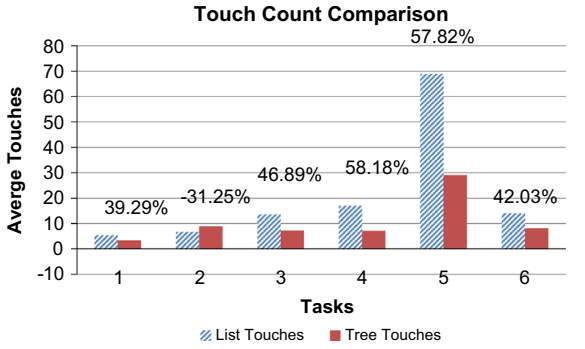

Fig. 25. Touch count comparison: percentage values show ERELT improvement over list layout.

Table 1
Results in time and difficulty.

	List	Tree
Average time per touch	2.16 s	3.4 s
• Task 1	3.1 s	5.7 s
• Task 2	3.2 s	4 s
• Task 3	2.2 s	3 s
• Task 4	1.6 s	3.2 s
• Task 5	1.4 s	2.3 s
• Task 6	1.5 s	2.4 s
Feedback: which felt faster?	4	22
Feedback: difficulty rating [1–10]		
• 1, 9, 10	–	0
• 2, 6	–	2
• 3, 5	–	5
• 4	–	10

In these tasks, the subjects were given the full paths to their targets. Here, ERELT did not have any advantage in time because subjects could just as quickly select paths to their targets in the list. Using ERELT in Task 1 gave subjects a significant advantage in the number of touches because the path was partially visible in the first screen. However, in Task 2, the path to the target was initially hidden due to the branching threshold. Thus the subjects had to scroll to the hidden nodes before a direct path to the target was found and therefore required more touches. Task 2 was the worst in both time and touches for ERELT. Based on subjects' feedback, they had difficulty with the arrow

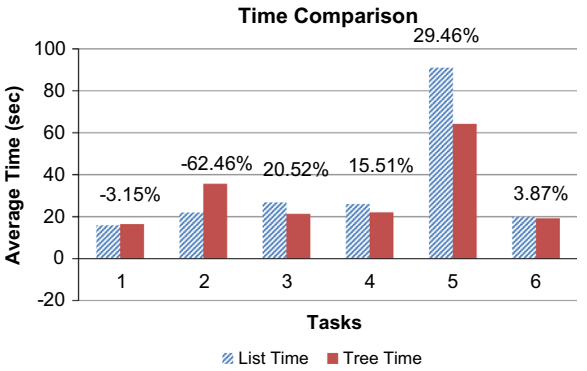

Fig. 26. Time comparison: percentage values show ERELT improvement over list layout.

indication for the hidden nodes, which may be the reason for this.

2. **Tasks 3, 4 and 5 (Exploration/Search tasks):**
The subjects were asked to search for tracks in certain genres, albums, etc, in Tasks 3, 4 and 5. ERELT interface was much faster in these tasks because it allowed subjects to quickly move in and out of various depths of the media library tree. ERELT also displays multiple branches at the same time thus making it easy to search for a tree node. ERELT showed more than 45% improvement in number of touches and 15-30% improvement in time taken to complete these tasks, over the list layout. However, the 15% time difference in Task 4 was found to be statistically insignificant from the *T*-test. Unlike Task 3, Task 4 requires subjects to navigate back and forth between multiple branches. Thus, this caused large variance in the subject's performance affecting statistical significance.

3. **Task 6 (Hybrid):**

This task asked subjects to add multiple tracks from a certain genre of a certain language. The first part of the task for the subjects to reach the genre node is similar to direct path tasks. The second part of the task for the subjects to reach various artist/albums in the genre node to add various tracks is similar to exploration task. Since ERELT allows subjects to jump multiple levels up and down, the number of touches was much fewer (10.6 touches per user per task versus 20.9 touches for the list view). Since list view interfaces are advantageous when the path is known, (especially due to subjects' familiarity with list views), ERELT did not show significant improvement in time.

Table 1 and Fig. 27 show that the average time per touch decreased for each subsequent task in both cases. This could be because of the subjects' getting more familiar with the layouts or with the music library.

On average the result shows that ERELT has definite advantage over list-based interfaces. The fewer touches mean less physical effort [18]. Except in the cases of direct and defined path, the ERELT gives comparable or better time performance. We conclude that the ERELT provides a

Table 2
Mean time and touches for each task with two-tailed *T*-test results.

Task Type	Task	Tree Mean	Tree SD	List Mean	List SD	*p*-value	Statistical significance
Touch comparison							
Direct path	**Task 1**	**3.269**	1.485	5.385	1.134	$p < 0.05$	Yes
	Task 2	8.885	2.405	**6.769**	1.986	$p < 0.05$	Yes
Exploration and search	**Task 3**	**7.230**	3.191	13.615	5.123	$p < 0.05$	Yes
	Task 4	**7.0769**	3.740	16.923	5.003	$p < 0.05$	Yes
	Task 5	**29.038**	12.379	68.846	25.768	$p < 0.05$	Yes
Hybrid	**Task 6**	**8.115**	2.805	14	7.065	$p < 0.05$	Yes
Time comparison (s)							
Direct path	**Task 1**	16.385	9.109	**15.885**	8.001	$p = 0.4040 > 0.05$	No
	Task 2	35.615	21.401	**21.923**	25.817	$p < 0.05$	Yes
Exploration and search	**Task 3**	**21.308**	11.206	26.808	11.795	$p < 0.05$	Yes
	Task 4	**22**	14.291	26.038	10.208	$p = 0.0658 > 0.05$	No
	Task 5	**64.192**	26.046	91	31.614	$p < 0.05$	Yes
Hybrid	**Task 6**	**19.115**	11.039	19.885	9.035	$p = 0.3930 > 0.05$	No

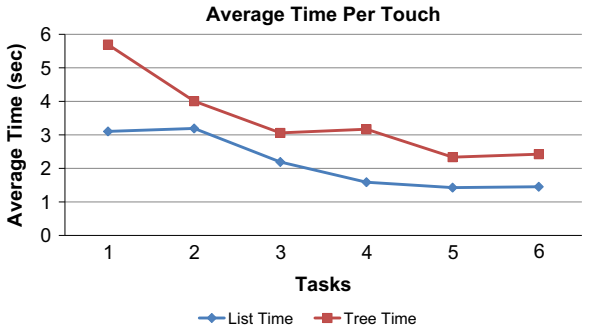

Fig. 27. Average time per touch for each task.

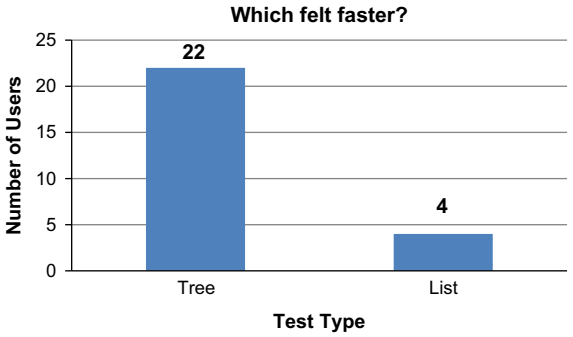

Fig. 28. Most subjects said using ERELT Felt Faster.

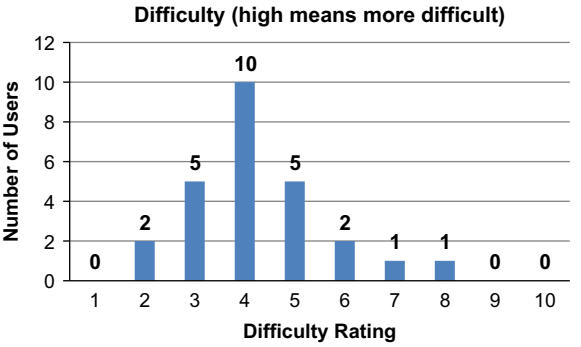

Fig. 29. Most subjects found difficulty in using ERELT to be less than average.

better user interface for exploration/searching tasks. Qualitatively, about 85% of the subjects said they felt tree layout was quicker to work with, as shown in Fig. 28. As shown in Fig. 29, most user ratings for the ERELT based interface in a difficulty scale of 1 to 10 are on the lower end of the scale, indicating relative ease in the user experience.

7. Conclusions

The paper has presented ERELT, a tree visualization technique, designed for displaying and navigating small screens on smartphones. ERELT not only focuses on visual presentation of hierarchical structures, but also contributes on exploration and navigation of such structures through intuitive user interactions, unlike most other tree visualization techniques. The paper addresses two main issues, optimal space usage and rapid navigation of hierarchical structures. Through a usability study we have found ERELT to be a faster and less tiring alternative to the traditional list based interface for exploring hierarchical information.

Our immediate future work is to further improve the design based on the feedback from the usability study. We will experiment with motion-based gestures for user interactions. During the usability study with Android mobile application, the test subjects were asked to provide subjective feedback on the new interface. Most common complaint was on hidden-node navigation. In the test version of the application, only the hidden nodes under the root node could be explored using right-left swipe. This needs to be changed to allow swiping of any node currently displayed on the screen. Time and number of touches for the Task 2 was much larger for the tree view compared to the list view, possibly because of high number of hidden nodes. So we need better system to convey to the users where the hidden nodes are. Making the

hidden node indicator more prominent or dynamic could improve this. There were also few navigation mechanisms disabled in the test application in order to reduce the complexity for a first time user. We plan to conduct another round of usability tests with modifications and full features included, using a different application, generating different kind of trees. This will again be compared against similar list based interface. Another direction is to explore the capability and usage of ERELT in larger screens such as desktop and laptop PCs, and for visualizing patterns and trends in big data applications.

References

[1] A.P. Chhetri, K. Zhang, Modified RELT for Display and Navigation of Large Hierarchy on Handheld Touch-Screen Devices, In: IEEE/ACIS 11th International Conference on Computer, Shanghai, China, 2012.

[2] H.-J. Schulz, H. Schumann, Visualizing graphs – a generalized view, Inf. Vis. 9 (2) (2010) 115–140.

[3] H.-J. Schulz, S. Hadlak, H. Schumann, The design space of implicit hierarchy visualization: a survey, IEEE Trans. Vis. Comput. Graph. 17 (4) (2011) 393–411.

[4] M.J. McGuffin, J.M. Robert, Quantifying the space-efficiency of 2D graphical representations of trees, Inf. Vis. 9 (2) (2010) 115–140.

[5] B. Karstens, M. Kreuseler, H. Schumann, Visualization of complex structures on mobile handhelds, In: Proceedings of International Workshop on Mobile Computing, 2003.

[6] T. Hakala, J. Lehikoinen, and A. Aaltonen, Spatial interactive visualization on small screen, in: Proceedings of the 7th International Conference on Human Computer Interaction with Mobile Devices & Services - MobileHCI '05, 2005, p. 137.

[7] B. Johnson and B. Shneiderman, Tree-Maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures, in: Proc. IEEE Conf. Visualization (Visualization '91), 1991, pp. 284–291.

[8] B. Shneiderman, Tree visualization with Tree-Maps: 2-d spacefilling approach, ACM Trans. Graph. 11 (1) (1992) 92–99.

[9] M. Bruls, K. Huizing, and J. Van Wijk, Squarified treemaps, In: Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization, 1999, pp. 33–42.

[10] J.J. van Wijk and H. van de Wetering, Cushion Treemaps: Visualization of Hierarchical Information, In: Proc. IEEE Symp. Information Visualization (InfoVis '99), 1999, pp. 73–78.

[11] R. Vliegen, J.J. van Wijk, E.J. van der Linden, Visualizing business data with generalized Treemaps, IEEE Trans. Vis. Comput. Graph. 12 (5) (2006) 789–796.

[12] M. Balzer and O. Deussen, Voronoi Treemaps, In: Proc. IEEE Symp. Information Visualization (InfoVis '05), 2005, pp. 49–56.

[13] M. Wattenberg, A Note on Space-Filling Visualizations and Space-Filling Curves, in Proc. IEEE Symp. Information Visualization (InfoVis '05), 2005, pp. 181–185.

[14] K. Onak and A. Sidiropoulos, Circular Partitions with Applications to Visualization and Embeddings, in Proc. ACM Symp. Computational Geometry (SCG '08), 2008, pp. 28–37.

[15] K. Wetzel, Pebbles – Using Circular Treemaps to Visualize Disk Usage, ⟨http://lip.sourceforge.net/ctreemap.html⟩, 2003. [Online].

[16] T.D. Wang and B. Parsia, Cropcircles: topology sensitive visualization of OWL class hierarchies, In: Proc. Int'l Semantic Web Conf. (ISWC '06), 2006, pp. 695–708.

[17] B. Otjacques, M. Noirhomme, X. Gobert, P. Collin, and F. Feltz, Visualizing the activity of a web-based collaborative platform, In: Proc. Int'l Conf. Information Visualisation (IV '07), 2007, pp. 251–256.

[18] Y. Kajinaga, T. Itoh, Y. Ikehata, Y. Yamaguchi, Data Jewelry-Box: A Graphics Showcase for Large-Scale Hierarchical Data Visualization, IBM Research. Technical Report RT0427, 2002.

[19] B.B. Bederson, "PhotoMesa: A Zoomable Image Browser Using Quantum Treemaps and Bubblemaps," in *Proc. ACM Symp. User Interface Software and Technology (UIST '01)*, pp. 71-80, 2001.

[20] N. Churcher, L. Keown, and W. Irwin, Virtual Worlds for Software Visualisation, in Proc. Software Visualisation Workshop (SoftVis '99), 1999, pp. 9–16.

[21] D. Turo and B. Johnson, Improving the Visualization of Hierarchies with Treemaps: Design Issues and Experimentation, In: Proc. IEEE Conf. Visualization (Visualization '92), 1992, pp. 124–131.

[22] J.B. Kruskal, J.M. Landwehr, Icicle plot: better displays for hierarchical clustering, Am. Stat. 37 (2) (1983) 162–168.

[23] T. Bladh, D.A. Carr, and J. Scholl, Extending Tree-Maps to Three Dimensions: A Comparative Study, in: Proc. Asia-Pacific Conf. Computer-Human Interaction (APCHI '04), 2004, pp. 50–59.

[24] K. Andrews, J. Wolte, and M. Pichler, Information Pyramids: A New Approach to Visualising Large Hierarchies, In: Proc. IEEE Conf. Visualization (Visualization '97), 1997, pp. 49–52.

[25] W. Wang, H. Wang, G. Dai, and H. Wang, Visualization of Large Hierarchical Data by Circle Packing, In: Proc. SIGCHI Conf. Human Factors in Computing Systems (CHI '06), 2006, pp. 517–520.

[26] B.S. Johnson. Treemaps: Visualizing Hierarchical and Categorical Data, Univ. of Maryland. PhD dissertation, 1993.

[27] A. Chaudhuri and H.W. Shen, A Self-Adaptive Treemap-Based Technique for Visualizing Hierarchical Data in 3D, In: Proc. IEEE Pacific Visualization Symp. (PacificVis '09), 2009, pp. 105–112.

[28] H.S. Smallman, M.S. John, M.B. Cowen, Availability in 2D and 3D displays, Comput. Graph. Appl. 21 (5) (2001) 51–57.

[29] J. Stasko and E. Zhang, Focus+Context Display and Navigation Techniques for Enhancing Radial, Space-Filling Hierarchy Visualizations, in: Proc. IEEE Symp. Information Visualization (InfoVis '00), 2000, pp. 57–65.

[30] J. Yang, M.O. Ward, and E.A. Rundensteiner, Interring: An Interactive Tool for Visually Navigating and Manipulating Hierarchical Structures, In: Proc. IEEE Symp. Information Visualization (InfoVis '02), 2002, pp. 77–84.

[31] A. Dix, R. Beale, and A. Wood, Architectures to Make Simple Visualisations Using Simple Systems, In: Proc. Int'l Working Conf. Advanced Visual Interfaces (AVI '00), 2000, pp. 51–60.

[32] M. Chuah, Dynamic Aggregation with Circular Visual Designs, In: Proc. IEEE Symp. Information Visualization (InfoVis '98), 1998, pp. 35–43.

[33] F. Chevalier, D. Auber, and A. Telea, Structural Analysis and Visualization of C++ Code Evolution Using Syntax Trees, In: Proc. Int'l Workshop Principles of Software Evolution (IWPSE '07), 2007, pp. 90–97.

[34] F. van Ham and J.J. van Wijk, Beamtrees: Compact Visualization of Large Hierarchies, In: Proc. IEEE Symp. Information Visualization (InfoVis '02), 2002, pp. 93–100.

[35] J. Hao, K. Zhang and M. L. Huang, RELT – Visualizing Trees on Mobile Devices, In: Proc. Int'l Conf. Visual Information Systems (VISUAL '07), 2007, pp. 344–357.

[36] J. Hao, C.A. Gabrysch, C. Zhao, Q. Zhu, K. Zhang, Visualizing and navigating hierarchical information on mobile user interfaces, Int. J. Adv. Intell. 2 (1) (2010) 81–103.

[37] J. Hao, K. Zhang and C. A. Gabrysch, Managing Hierarchical Information on Small Screens, in: Proc. Joint Int'l Conf. Advances in Data and Web Management (APWeb/WAIM '09), 2009, pp. 429–441.

[38] M. Bugajska, Framework for Spatial Visual Design of Abstract Information, in: Proc. Int'l Conf. Information Visualisation (IV '05), 2005, pp. 713–723.

[39] J. Francone, G. Bailly, L. Nigay, and E. Lecolinet, Wavelet menus: a stacking metaphor for adapting marking menus to mobile devices, In: Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '09), 2009, pp. 2–5.

[40] J. Bergman and J. Vainio, Interacting with the flow, in: Proceedings of the 12th international conference on Human computer interaction with mobile devices and services - MobileHCI '10, 2010, p. 249.

[41] A. Geven, R. Sefelin, and M. Tscheligi, Depth and breadth away from the desktop: the optimal information hierarchy for mobile use, In: Proceedings of the 8th conference on Human-computer interaction with mobile devices and services (MobileHCI '06), 2006, pp. 157–164.

[42] C.G. Healey, A.P. Sawant, On the limits of resolution and visual angle in visualization, ACM Trans. Appl. Percept. 9 (4) (2012) 1–21.

[43] I. Herman, G. Melancon, M.S. Marshall, Graph visualization and navigation in information visualization: a survey, IEEE Trans. Vis. Comput. Graph. 6 (1) (2000) 24–43.

[44] S. Burigat, L. Chittaro, On the effectiveness of Overview+Detail visualization on mobile devices, Person. Ubiquitous Comput. 17 (2) (2011) 371–385.

[45] J. Huber, J. Steimle, and M. Mühlhäuser, Toward more efficient user interfaces for mobile video browsing: an in-depth exploration of the design space, In: Proceedings of the international conference on Multimedia (MM '10), 2010, pp. 341–350.

[46] J. Ruiz and Y. Li, DoubleFlip: a motion gesture delimiter for mobile interaction, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11), 2011, pp. 1–4.

[47] A. Roudaut, E. Lecolinet, and Y. Guiard, MicroRolls: expanding touch-screen input vocabulary by distinguishing rolls vs. slides of the thumb, In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09), 2009, pp. 927–936.
[48] H. Duh, G. Tan, and V. Chen, Usability evaluation for mobile device: a comparison of laboratory and field tests, In: Proceedings of the 8th conference on Human-computer interaction with mobile devices and services (MobileHCI '06), 2006, pp. 181–186.
[49] C. Cuadrat Seix, M. S. Veloso, and J. J. R. Soler, Towards the validation of a method for quantitative mobile usability testing based on desktop eyetracking, in: Proceedings of the 13th International Conference on Interacción Persona-Ordenador - INTERACCION '12, 2012, pp. 1–8.
[50] C. Gonzalez, Does Animation in User Interfaces Improve Decision Making?, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '96), New York, 1996.
[51] E. Haines, Point in Polygon Strategies Graphics Gems, Morgan Kaufmann, 1994, pp. 24–46.
[52] C.J. Mueller, D.E. Tamir, O.V. Komogortsev, L. Feldman, Using Designer's Effort for User Interface Evaluation, IEEE International Conference on Systems, Man and Cybernetics, 2009.
[53] L. Feldman, C. J. Mueller, D. Tamir and O. V. Komogortsev, Usability Testing with Total-Effort Metrics, in: 3rd International Symposium on Empirical Software Engineering and Measurement, 15–16 Oct, 2009.
[54] Adobe, Adobe Mobile Experience Survey: What Users Want from Media, Finance, Travel & Shopping, October 2010. [Online].
[55] O. Goussevskaia, M. Kuhn, and R. Wattenhofer, Exploring music collections on mobile devices, In: Proceedings of the 10th international conference on Human computer interaction with mobile devices and services (MobileHCI '08), 2008, p. 359.