International Journal of Advanced Intelligence
Volume 2, Number 1, pp.81-103, July, 2010.

AIA
International
Advanced
Information
Institute

Visualizing and Navigating Hierarchical Information on Mobile User Interfaces

Jie Hao[1], Chad Allen Gabrysch[1], Chunying Zhao[1], Qiaoming Zhu[2] and Kang Zhang[1,2]

[1] *Department of Computer Science,*
The University of Texas at Dallas
800 West Campbell Road, Richardson, TX 75080-3021, USA

[2] *School of Computer Science and Technology, Soochow University*
Suzhou, China 215006

{*jie.hao; cag016500; cxz051000; kzhang* }*@utdallas.edu;*
qmzhu@suda.edu.cn

Received (May 2010)
Revised (June 2010)

This paper presents a visualization approach called Radial Edgeless Tree (RELT) for visualizing and navigating hierarchical information on small screens. Major advantages of the RELT approach include: elegance recursive division of the display area, space-filling, maximum usage of screen estate, and clarity of the hierarchical structure. It offers the flexibility such that users can customize the hierarchy's root location and stylize the layout. The RELT drawing algorithm is adaptable and customizable for different application domains. We have implemented the RELT interface on the Google Android emulator. The paper presents the Android implementation, and then provides an analytical and empirical comparison of the Android implementation with a traditional cell phone interface in terms of their performances in navigating hierarchical information.

Keywords: Hierarchy visualization; Mobile interface; Tree drawing; Aesthetic layout; Stock market visualization.

1. Introduction

The past few years have witnessed rapid adoption of powerful and flexible handheld devices among consumers. While the hardware capabilities of these devices have dramatically increased, their screen size is limited by the total size of the device, and likely will not substantially increase beyond what is currently seen on the market. Users of these devices expect much of the functionality of larger, more powerful systems such as laptops and desktops, yet merely scaling down the information visualization methodologies of these larger devices to smaller screens can lead to problems in readability and navigation.

Much of the information a user wishes to visualize is arranged in a hierarchical structure, such as stock prices, menu interfaces, file systems, and so on. When visualized, a hierarchical structure can give the user an efficient way to determine the relationships among the distinct data elements and find the desired information

quickly. Thus, visualizing hierarchical information is vitally important for many applications and thus becoming a growing area of research. The proliferation of powerful mobile devices means this topic is not limited to users of desktop computers. Efficient ways to visualize hierarchical data on these mobile handheld devices have not been investigated as widely as visualization for larger screens.

Various approaches have been proposed for browsing the web information on small screens [1,2]. Some of these approaches involve innovative interaction techniques [3,4]. Although visualization techniques have been proposed for viewing tree structures on desktop screens, little progress has been made on maximizing the space usage for displaying trees on small screens. Effective and efficient navigation through hierarchical structures on limited viewing spaces has not been widely investigated.

Aiming at visualizing trees on small screens with space optimization, the authors proposed an approach, called Radial Edgeless Tree, or RELT, for visualizing and navigating hierarchical information on mobile devices [5]. RELT implements an area partitioning algorithm. By arranging a set of tree nodes as adjacent non-overlapped polygons in a radial manner, RELT maximally utilizes the display area while maintaining clarity of the tree structure.

The original RELT approach divides information objects into four groups which are assigned with rules that dictate the space allocation of the objects. The root is arranged at the top-left of the screen. For each node on the screen, its parent is placed on the upper left hand side, and its children are placed on the lower right hand side. The nodes at the same level approximately locate along circle whose center is at top left corner, as illustrated in Fig. 5 (a).

This paper presents a generalized version of RELT with two major enhancements:

Depending on the viewer's preference or the application requirements, 1) the root of the hierarchy can be placed anywhere on the screen; and 2) a given hierarchical structure can be visualized and navigated in either concentric or multi-centered mode (as explained in Section 3).

The rest of this paper is organized as follows. Section 2 reviews the current hierarchy visualization techniques. Section 3 presents the RELT generalization concepts and a corresponding layout algorithm. Section 4 describes the RELT implementation on the Google Android platform, and compares its performance with that of a traditional cell phone interface in terms of the efficiency in finding the needed information in a hierarchy. Section 5 summarizes the paper and discusses the remaining challenges for future research.

2. Related Work

Various approaches have been investigated for visualizing hierarchical information and can be generally classified into Connection and Space-filling categories. The Connection approaches draw the hierarchical information in a node-edge diagram.

Each information object is visualized as a node. An edge depicts the relationship between two connected nodes. All the connection approaches essentially focus on two major layout issues:

(i) Where to locate the nodes?
(ii) How to connect the related nodes?

The connection approaches have been intensively investigated. The method by Reigold and Tilford [6] is possibly one of the best-known techniques which simply locate child nodes below their common ancestors. An H-tree [7,8,9] positions child nodes in the vertical or horizontal direction of their common ancestors. Radial tree [7,8] technique creates concentric circles and then places nodes on those circles according to the nodes' depths in the hierarchy. The elegance of this technique is to keep adjacent branches from overlapping. Researchers have also developed hierarchy information visualization in 3D, such as Cone Trees [8,10,11], and Balloon Views [8,12].

A second class of hierarchy visualization is the space-filling approach. In these algorithms, efficient space usage is the primary goal, and they accomplish this by removing the edges that classify the connection-oriented approach and use the position of the nodes alone to convey hierarchical meaning. Treemaps illustrated in Fig. 1 (a) are an example of a space-filling method, in which the tree is divided into several nested rectangles and the nested enclosures define the parent-child relationship, and are shown in Fig. 1(a). Additionally, each rectangle's size and color may play a role in conveying information to the viewer, depending on the data and application involved [8,13,14]. The primary drawback to a treemap is the lack of clarity regarding the hierarchical structure. There is no clearly defined root location, and while color can be used to identify nodes of equivalent depth, the lack of a clearly defined structure makes understanding the visualization difficult for some users.

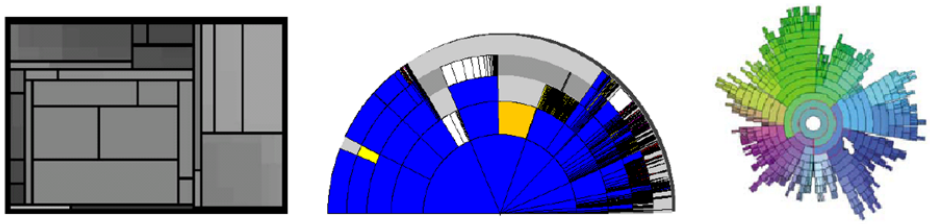


Fig. 1. Space filling methods.

Information Slices [15], shown in Fig. 1(b), use concentric semicircles to indicate depth and sibling relationship among nodes, with a wedge of the complete semicircle indicating a sub-tree under the root. The InterRing method, shown in Fig. 1(c), works in a similar manner, with full concentric circles instead of semicircles [16]. While both of these algorithms improve screen utilization by removing connecting edges, a substantial space usage issue still exists for these algorithms in that they are circular representations of the tree that must fit on screens which are almost universally rectangular.

The aforementioned approaches, when simply adapted to mobile interfaces, usually cannot show the information as effective as on desktop screens. The biggest limitation that makes these approaches unsuited for mobile interfaces is their loss of legibility or the overall sense of a hierarchy when reduced in size.

Most current mobile hierarchy information visualization methods were originally designed for desktop displays. Connection approaches are usually chosen to be adapted for mobile interfaces due to their desirable property of showing a clear structure. Magic Eye View and Rectangular View [17] in Fig. 2 and Fig. 3 are typical examples. However, the drawback, i.e. uneconomic space-usage, is as distinct as their advantage.

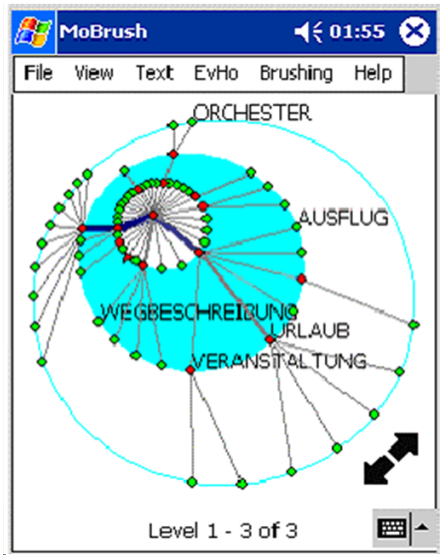


Fig. 2. Visualization of hierarchical structures using Magic Eye View on a mobile device.

Fig. 2 illustrates the layout using modified Magic Eye View [17] on mobile simulator. This method can visualize one thousand nodes which is ten times more than that by traditional drawing method [6].

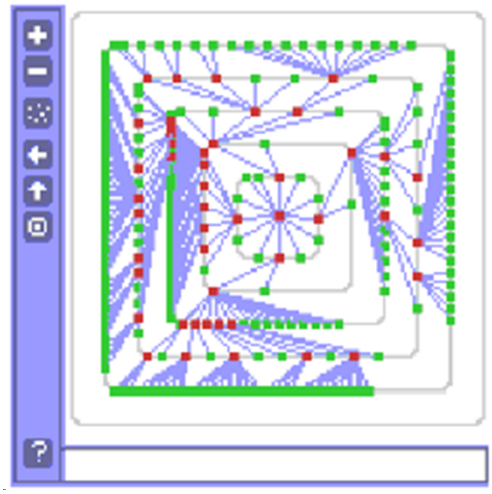


Fig. 3. Visualization of 332 nodes using rectangular view.

As illustrated in Fig. 3, Rectangular View [17] is also a modified hierarchy visualization method. This method is claimed to be able to handle hierarchy of 3389 nodes.

Space-filling methods, maximally utilizing the screen, however, have their own problems in the process of adaptation. For InterRing [16] and Information Slices [15], efficiency of space usage depends significantly on how balance the tree is. When certain branches have depths much larger than the other branches, space usage is compromised. Moreover, both methods have serious limitations on the clarity of labeling. Although Tree Maps fully utilize the display space, its hierarchical structure is, however, hidden behind the enclosure relationships.

In summary, none of the above hierarchy visualization methods can achieve both structural clarity and economic space usage on mobile devices. Next section describes the RELT methodology that aims at achieving both goals.

3. General Radial Edgeless Trees

This section first introduces the methodology behind the general RELT and the terminology, and then discusses the RELT layout algorithm in details.

3.1. *Methodology and terminology*

Fig. 4 illustrates a university's web site, where schools of "Management", "Engineering', and "Science" each have a few departments. Our original RELT algorithm [5] fixes the location of the root ("University") at the top-left corner of the screen as shown in Fig. 5(a). This fixed layout may not suit several applications. The generalized RELT allows the root to be displayed anywhere on the screen, depending on

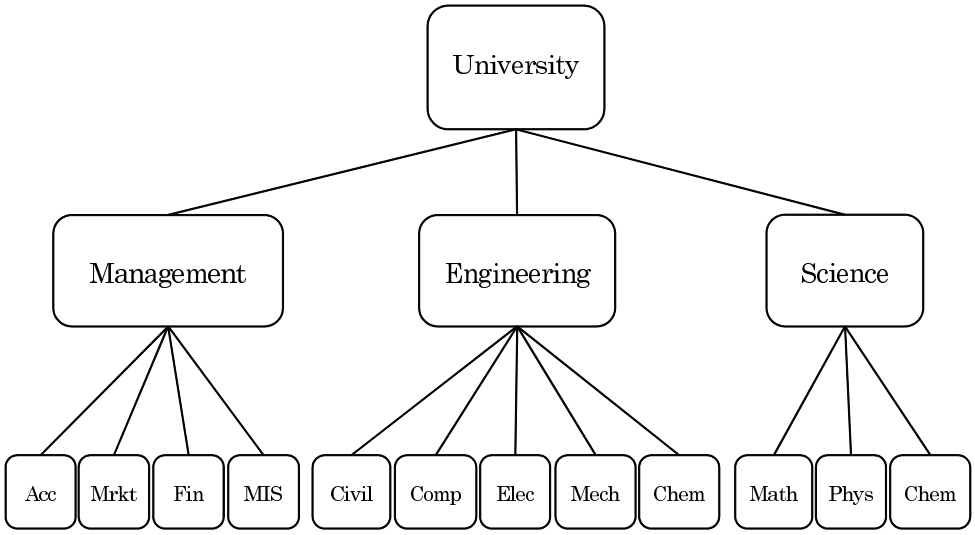


Fig. 4. A university web structure.

the user's preference and the requirements of application domains.
For example, Fig. 5 (b) illustrates a layout in which the root is located at the bottom-left corner.

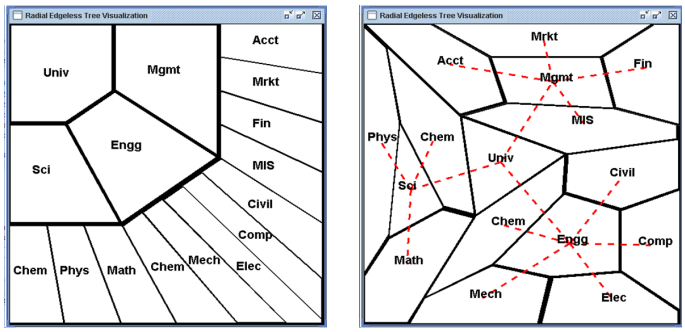


Fig. 5. University structure visualized as a rooted tree in RELT: (a) Layout with root at top-left (b) An alternative layout with root at bottom-left.

Fig. 6 (a) and (b) are two type of center-rooted layouts, where each level-2 node, i.e. a school, shares a border with the root, and each level-3 node shares a border with its level-2 parent. We will call this type of center-rooted layout as

concentric, or simply CC. As shown in Fig. 5 (a) with the university example, departments locate under the school they belong to, which in turn locates adjacent to the university.

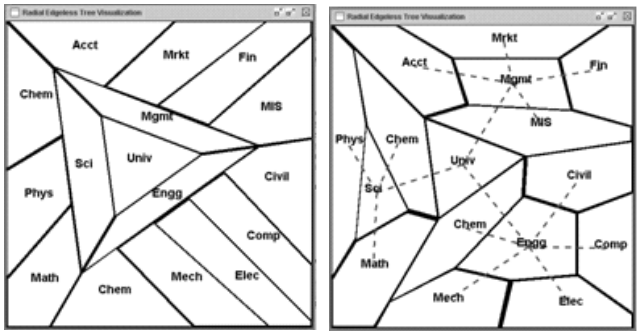


Fig. 6. Variations of center-rooted visualization: (a) Concentric layout (b) Multi-centric layout.

Fig. 6(b) illustrates another type of center-rooted layout which considers the sub-area of each child node as a center-rooted sub-tree. We will call this later type of layout as multi-centric, or simply MC. In this case, the schools form the sub-centers with their departments around them.

Given any of these layouts, the user is able to navigate through the hierarchy structure. For example, to view the detailed information on the Department of Computer Engineering, the user can select "Comp". By selecting "Comp" in Fig. 5 (a) that has the root at the top-left, we obtain the zoom-in view as in Fig. 7, where the entire display space is filled with the sub-tree rooted at "Comp". Therefore with a single user interaction, multiple levels of hierarchy (five levels in the university example) can be reached.

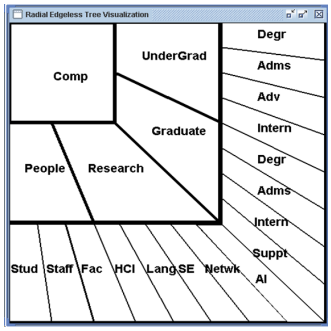


Fig. 7. A navigated view of two levels down the hierarchy.

How a node is selected depends on the interaction technique supported for navigation. A touch-screen interface (by either a finger or a stylus) would apparently suit the RELT methodology very well. A single touch on a node would serve the selection of the node. With a key or button based interaction support, direction buttons may be assigned to the moving directions within a predefined radial range, and/or up and down the hierarchical levels, plus a confirmation button, as we experimented on a cell phone emulator to be discussed in Section 5.

In general, RELT is highly scalable since the number of levels on a single display can be determined based on the application or maximized to the extent that the polygonal nodes have enough space for text labels. If color coding can easily discriminate and identify multiple groupings and levels of information, each tree node may be drawn by just a few colored pixels and thus maximize the size of the hierarchy being visualized.

In discussing the concept and terminology for generating the RELT layout, we will refer a tree node as a vertex that occupies a polygonal area in the drawing space. *A rooted tree* $T = (V, E, r)$ is a tree with a vertex r as the *root*. V is the vertex set and E is the edge set. One and only one path can exist from r to any other vertex. All $leaf$ vertices form a subset of V, denoted as $L \subset V$. For two directly connected vertices, the one closer to the root is the parent and the other is the child. In a typical node-edge type of graph, a directed edge $(v, u) \in E$ is used to express the parent-child relationship, where v is the parent and u is the child. In RELT, the relationship between a parent and its children is shown by their adjacency on a radial sector and/or positions relative to each other.

RELT classifies vertices into three types:

(i) $v = r : T(v)$ represents the entire tree.
(ii) $v \neq rANDv \notin L : T(v)$ represents the sub-tree rooted at v.
(iii) $v \neq L : T(v)$ refers to the leaf vertices.

If $v \notin L$, i.e. type 1 or 2, then the set of v's children can be expressed by:

$$CV = \{u|(v, u) \in E\} \tag{1}$$

CV_i is the i_{the} child of v. The display space is recursively divided into several non-overlapping polygonal areas each visually representing a vertex. Three types of "area" are defined:

(i) $WA(v)$ represents the entire area under vertex v.
(ii) $OA(v)$ is the area occupied by v itself.
(iii) $DA(v)$ is the area occupied by all of v's descendants.

3.2. *Rules and algorithm*

A RELT layout can be generated by applying the following four general rules:

(i) Every $T(v)$ has $WA(v)$ that is assigned to v by its parent. $WA(r)$ is the entire display area.

(ii) How $OA(v)$ is determined inside $WA(v)$ depends on the layout type (LT, described below) specified by the user.
(iii) If v is a non-leaf vertex, it distributes its area $DA(v)$ to its children.
(iv) The area size of a leaf vertex is proportional to one of the vertex's properties, such as the weight as discussed later.

A RELT layout can be calculated by function RELT(V, r, LT) in which V is the vertex set, r specifies the root vertex, and LT indicates the layout type. The user-provided LT parameter determines whether the hierarchy is viewed concentric, i.e. the CC layout, or children surrounding their parents that form multiple subcenters, i.e. the MC layout, as introduced in Section 3.1. The RELT algorithm is presented in pseudo code below.

```
Algorithm RELT
Input: vertex set V, root r, layout type LT
Begin
Set root location // User clicks on screen to decide root location
DFS               // Depth first search to traverse the tree
if vertex v is not yet processed then
        if vertex is a non-leaf
           OA(v) = createOA(v, LT);
   // calculate the area occupied by v
           if v has more than one child
              DA(v) = WA(v) - OA(v)
              WA(CV) = DF(DA(v))
              // v distributes DA(v) to its children
              else      // v has only one child
                   DA(v) = WA(v)
                   // DA(v) belongs to the only child
              else         // leaf vertex
                   WA(v) = OA(v) = DA(v)
End
```

Each vertex v can be assigned a corresponding weight $w(v)$ which is computed by a weight function WF:

$$w(v) = WF(w(CV_1), w(CV_2) \dots w(CV_n)) \tag{2}$$

Eq. (2) basically says that, the value of $w(v)$ depends on the overall weight of v's children. Different weight functions may be defined to meet different application requirements (see Section 4.2 for the weight function used to visualize the stock market). This paper uses the following simple weight function as an example:

(i) If $v \in L, w(v) = WF(1) = 1$;

(ii) Otherwise, assuming the given v has n children, then
$w(v) = WF(1, w(C_{V_1}), w(C_{V_2}) \dots w(C_{V_n}))$
$= 1 + \sum_{i=1}^{n} w(C_{V_i})$

A leaf vertex v has no children, therefore needs not distribute subareas. Only its own area $OA(v)$ needs to be calculated. For a non-leaf vertex v, after $OA(v)$ being constructed, $DA(v)$ is distributed to its children. The distribution is determined by a distribution function $DF()$. Here, we propose $DF()$ that partitions the distribution area based on the size of $DA(v)$ and the children's weights:

$$DF(DA(v), w(C_{V_1}), w(C_{V_2}) \dots w(C_{V_n})) \rightarrow WA(C_V) \tag{3}$$

Function $createOA(v, LT)$ creates the node v's own area, depending on the layout type LT. The following pseudocode describes $createOA(v, LT)$ for the concentric layout type. Due to the space limit, the function description for the multi-centric layout is omitted here.

```
Function createOA(v,LT)
NLevel = the number of levels in the tree
If v = r has m children (m >1)
   // rootP is the root location determined by the user
   // startP is a point on the boundary of the screen
   // In our implementation, we fix startP = (0,0)
   movingP = startP
   draw a line scanL from rootP to movingP
   no = 0
   record scanL as Lno
   while (no < m)
      movingP along boundary clockwise, accumulate area by scanL
        If accumulated area is to the right child depending on WF()
            no = no +1
            record scanL as Lno
   while (no >0)
      calculate Pno
      // distance from rootP to Pno is 1/NLevel of Lno
      no = no -1
   while (no < m)
      link Pno and P(no.-1 mod m) to form OA (r)
      no = no +1
If v = r has one child
   draw OA (v, LT) as a polygon
If v is non-root and non-leaf
```

```
    // vParent is parent of v, vLevel is level of v
    // commonL is the line shared by WA(v) and OA(vParent)
    // point1 and point2 are two end points of commonL
    point1' = point1
    point2' = point2
    // boundary1 is boundary of WA(v) containing point1.
    // boundary2 is boundary of WA(v) containing point2.
    move point1' and point2' along boundary1 and boundary2
    if distance from point1' to point1 is 1/vLevel of the
       length of boundary1, then stop (similarly for point2')
    connect point1' and point2' to form stopL
    // OA(v) is the area between commonL and stopL.
end
```

We now discuss the complexity of the RELT algorithm. As discussed in Section 3.1, a given tree $T = (V, E, r)$ has n vertices which are divided into three types. RELT applies depth first search to traverse the tree. Table 1 illustrates the vertex types and their corresponding operations used in the RELT algorithm.

All the operations in the right column can be computed in linear time. Therefore the overall time complexity is $O(n)$, where n is the number of vertices.

In summary, RELT recursively partitions the display area into a set of non-overlapping polygons. Because every part of screen is utilized, economic screen estate is achieved. Parent-child relationships are explicitly represented by adjacent relationships, and thus the structural clarity is maintained.

3.3. *Labeling*

Labeling in the general RELT algorithm is straightforward. The center of a node polygon is used as the center of the text label, so that the text is positioned in an efficient manner[5,18]. In certain situations, particularly when the polygon size is small or skewed in a vertical rather than horizontal fashion, this method produces inadequate results as the text would overlap onto other polygons. A simple im-

Table 1. Vertex types and corresponding operations.

Vertex Type	Operation
$v = r$	*createOA*(v, LT)
	DA(v) = WA(v) - OA(v)
	DF (DA(v)) $\rightarrow$ WA(CV)
$v \neq r\ AND\ v \notin L$	*createOA*(v, LT)
	DA(v) = WA(v)
$v \in L$	WA(v) = OA(v) = DA(v)

provement to this is to rotate the text to match the longest side of the polygon. This produces much improved results, but is sometimes inconsistent because it relies on the polygon shape which in turn relies on the overall structure of the tree; this leads in some cases to adjacent polygon labels having drastically different angles.

The general RELT algorithm positioned labels by finding the longest edge of a polygon and aligning the text to it. The proposed solution preserves consistency, and when combined with additional measures ensure that the text stays reasonably within the containing polygon. It is described in pseudocode as follows:

```
Algorithm AlignLabel
    Input: Vertex v, Label l
    Begin
       P = Centroid of given vertex v
       T = Degree offset from the root point of the RELT Display
       F = AdjustedFontSize()
       DrawLabel(P, T, F)
    End
```

This implementation differs from the original RELT method that chose the longest side of the polygon to align the label with. This method aligns the label with the degree offset from the root position, creating a more consistent layout than the previous version. The *AdjustedFontSize()* function enables the labels to grow or shrink depending on the following equation, and calculates the font size as follows:

$$FontSize = 5 + (\frac{10.0}{\log TotalVisibleNodes}) \tag{4}$$

Eq.(4) enlarges or shrinks the text based on how many visible nodes exist in the tree.

4. Android Implementation and Evaluation

4.1. *Development environment*

The implementation of the RELT algorithm was done on the Google Android platform, a mobile platform developed by Google specifically targeted at the smart phone market. Google has included support for touch screen and network capable devices; these features combined with Android's open source nature make it ideal for developing an RELT implementation. The development environment consists of the Android SDK and optionally an Eclipse plug-in for simplified development, both of which are used in this implementation. As shown in Fig. 8, the emulator provides a touch screen mobile display and a keyboard for input.

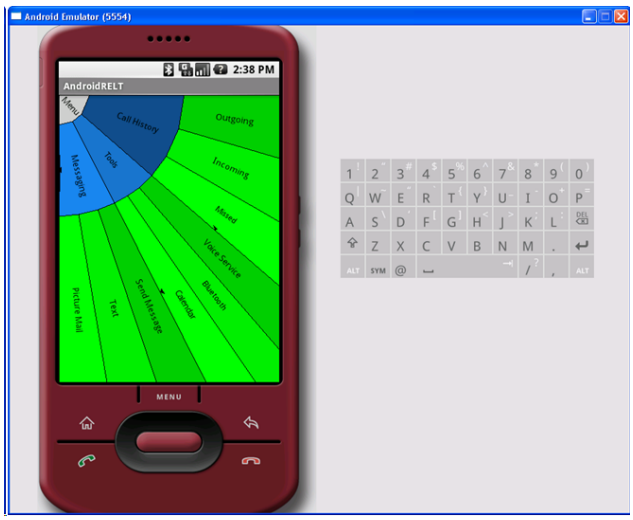


Fig. 8. Android emulator with sample RELT output.

4.2. *Some implementation issues*

Forced Square Perspective. The forced square perspective means that the rectangular viewing space is forced into a square. This feature may be useful in applications needing additional screen space to display other information such as node details, search fields, and more. While this extra space has not yet been utilized in the current RELT implementation, it remains available for any extensions for the algorithm.

Keyboard Navigation. The display parameters are accessible to the user via a section of shortcut keys (detailed key assignments are omitted here due to the space limit). These keys allow the user to increase or decrease the number of children displayed, the depth displayed, the root location, and the ability to toggle the screen between a forced square perspective and the full screen perspective.

Automatic Screen Resizing. Our implementation does not assume a particular screen size, and thus will accommodate any Android device the RELT runs on. Additionally, the orientation of the screen does not matter, as the RELT screen can be drawn on landscape, portrait, or square displays. Obviously this has limits, as a very small screen will not have sufficient space to display enough information with any hierarchical algorithm, including the RELT algorithm. Fig. 9 illustrates this principle by showing the RELT algorithm running on the emulator using a smaller phone size.

Fig. 9. RELT on a smaller device.

User Configuration Options. The Android implementation was designed for flexibility. A large number of settings are stored in a configuration file, such as the number of children and starting display depth. This allows a range of testing scenarios for user preferences. The primary configuration options allow the user or developer to set the number of starting visible children and depth, enabling customization to the targeted screen size.

4.3. *Analytical study*

In order to evaluate whether RELT provides a more efficient interface for navigation, this section presents an experimental comparison of our implementation on the Google Android emulator with the traditional cell phone interfaces, as shown in Fig. 10 that is modeled on Sprint PCS Vision Phone®. The Sprint interface has 9 top level service categories. There are totally 43 second level sub-categories under the 9 services, and each second level sub-category may include 0 to 6 third level items or sub-sub-categories. A third level sub-sub-category may include a few items at the fourth level. The last level is the 5th level. The entire hierarchical structure can be represented as a tree that has about 150-250 nodes. Some of the nodes at levels 2 through 5 are leaves, which should be the cell phone functions that the user wishes to arrive at in the shortest time, or with the fewest user interactions.

Here we explain the inherent difference between these the RELT and traditional

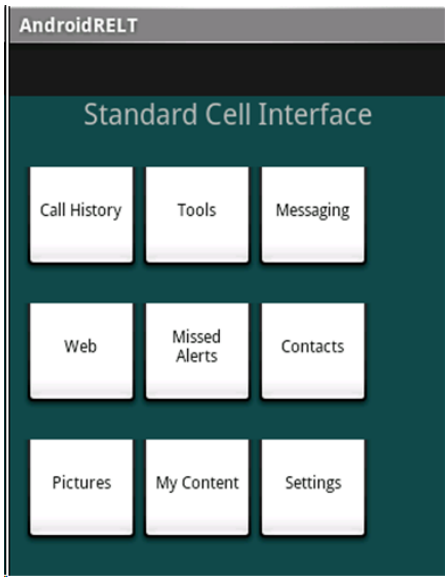


Fig. 10. Traditional cell phone interface.

cell phone interfaces. To ease the comparison, we model the problem with preconditions and assumptions.

(i) The example hierarchy can be represented as a full k-ary tree by applying traditional hierarchical drawing method, as illustrated in Fig. 11. Assume the hierarchy of n levels.
(ii) The number of levels that can be visualized effectively on one screen using RELT is limited at 3 ($\leq n$). In reality, users are allowed to adjust the number of levels shown on the screen.

Compared with the single level layout as on a traditional layout, RELT displays a multi-level layout on one screen and thus extends the viewing scope. This benefits the user in two aspects. First, by showing multiple levels, a RELT layout records a part of the navigation path (while traditional layout like Sprint shows only a point on the path). This helps the user to maintain his/her mental map during navigation. Second, RELT shows more nodes on one screen. Take the k-ary hierarchy as an example, up to $(k^3 - 1)/2$ can be shown on one screen. This would reduce the potential of selecting wrong nodes and also reduce the number of nodes leading to the target node.

As Fig. 13 illustrates, RELT offers enhanced navigation ability by reducing the size of the hierarchy. The hierarchy size reduction is achieved by increasing the amount of information on the screen that the user can see at a glance.

Table 2 illustrates the number of touches, with a touch-screen interface, on the intermediate nodes that the user has to traverse to find the target node in the

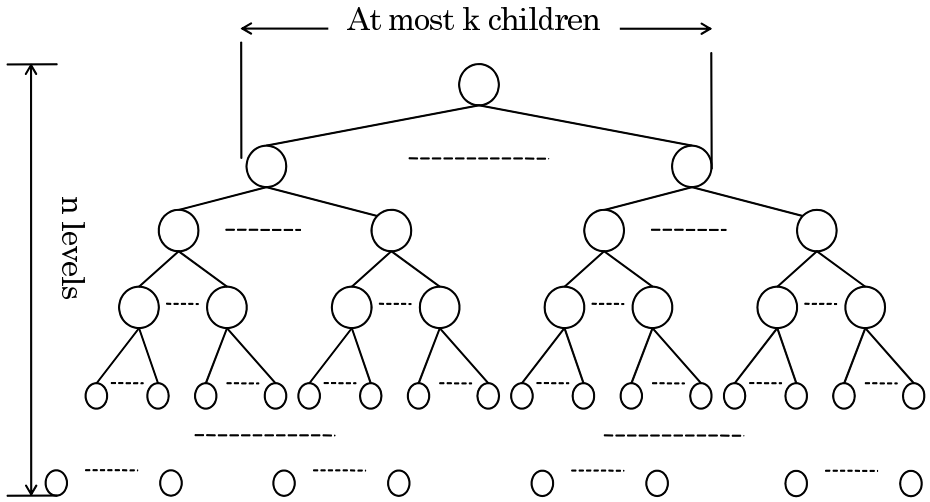


Fig. 11. An example hierarchy where a node has a maximum of k children.

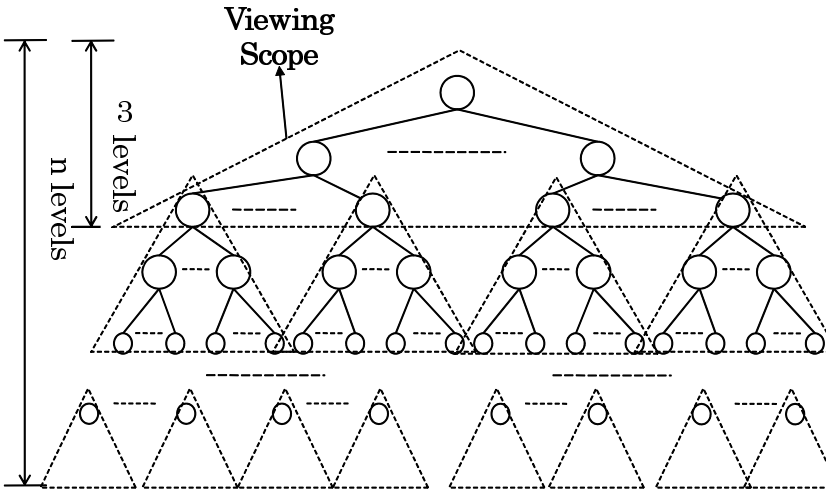


Fig. 12. Observation window illustration.

hierarchy. Table 3 shows the number of clicks for a button-based user interface. Assume that the target node locates at the i^{th} level and there are totally j nodes in the first i levels. In the tables, "Explicit" means that the user knows where the target node locates. "Implicit" includes the worst and best cases when the user

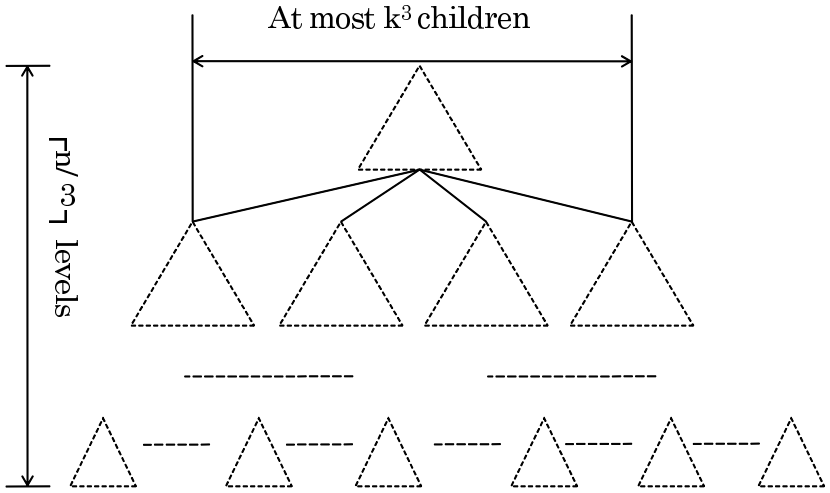


Fig. 13. Hierarchy reduction - the tree in Fig. 11 shrinks to $\ulcorner n/3 \urcorner$ levels of k^3-ary tree.

Table 2. Number of touches (for intermediate nodes) on a touch screen interface.

# of Screen Touches	Explicit	Implicit (worst)	Implicit (best)
Sprint	i-1	j-1	i-1
RELT	0	0	0

Table 3. Number of button pushes on a button-based interface.

# of Button Pushes	Explicit	Implicit (worst)	Implicit (best)
Sprint	i-1	j-1	i-1
RELT	i-1	i-1	i-1

has no idea about the target node's location. The navigation approaches of Table 2 and Table 3 are touch-based and button-based respectively.

Table 2 shows that, using Sprint to navigate, the user has to touch i-1 times (i.e. traverse i-1 intermediate nodes) to arrive at the target node in the best case. The user needs only one touch to arrive at the target (i.e. traverse 0 intermediate node) in RELT.

Table 3 shows that instead of jumping from the root node to the target node as on a touch screen, the user has to push buttons to arrive at the target node level

by level. Using RELT, the user has to traverse i-1 intermediate nodes. Cascading menus work equally well for both interfaces in the "Explicit" and best "Implicit" cases.

4.4. *Empirical Study*

We also conducted an empirical study to compare the actual user performance on the two interfaces. To prevent possibility of bias, the RELT interface was implemented without weight functions, thus both interfaces have identical hierarchical structures with regards to left and right sibling relationships. The user study was ported to a HTC G1 cellular phone, the only phone implementing the Android platform at the time of this experiment, and users were given a brief explanation of both interfaces before starting the experiment.

The study was conducted among 15 volunteer students as test subjects, who were given a set of nine tasks, listed below, corresponding to a single option on both interfaces:

(i) Find if you called Mike on March 23;
(ii) Find whether David has called you on March 10;
(iii) Find the Settings of the Receiver Volume of the Speaker;
(iv) Go to Stopwatch Lap2;
(v) Find the Settings of the Power-off Tone;
(vi) Launch Tetris game;
(vii) View the Picture Album;
(viii) Find the Settings of the Messaging Signature; and
(ix) Find the Directory Services.

These tasks were completed by each subject in both the RELT interface and a traditional cell phone interface, with the starting interface alternated amongst the subjects to prevent familiarity bias in the second test. Logging was programmed into the test application to ensure consistent time recording. Table 4 shows the average times of these subjects performed on the tasks.

These results indicate that tasks 1, 2, 6, and 7 have reduced average navigation time with the RELT interface, and tasks 3, 4, 5, 8 and 9 show increased average navigation time. This is consistent with our hypothesis that using the RELT navigation system decreases the navigation time to tasks on the front or near the front of the navigation screen, at the expense of those tasks farther away. Tasks 1, 2, 6 and 7 are all accessible from the RELT's initial screen options or within 3 navigation clicks of the initial screen. The remaining tasks are farther away, requiring more navigation to reach in terms of both breadth and depth.

The other interesting finding from this study is the average time reduction for the RELT interface as the tasks proceed. For task 3, the RELT interface average time is over 7.5 times greater than its traditional interface counterpart. However, by tasks 8 and 9 this time is closer to between 2 and 3 times greater, indicating a

Table 4. Average task time in seconds.

Tasks	Traditional	RELT
1	44.007	36.854
2	22.086	13.244
3	12.681	91.563
4	17.579	84.900
5	14.710	83.112
6	36.463	33.200
7	15.682	14.994
8	16.334	34.282
9	22.542	56.447

learning curve is taking effect and reducing the amount of time needed to navigate to distant options on the RELT interface.

We designed a questionnaire that includes space for comments and familiarity with cell phone interfaces. Of the 15 sample subjects, only one did not have experience with a cell phone, and of these 14 most had used an interface similar to the traditional one that was tested. Of these 14, all ranked current cell phone interfaces as easy or acceptable but could be improved. The freeform comments from the subjects touched on a variety of issues. Many mentioned the learning curve of the RELT interface, and how it must be overcome to navigate efficiently to options not on the starting screen. Others were confused by too many options on the screen at once, but some commented that having multiple levels of information present at once gave them a better idea of whether they were in the correct location.

In summary, this user study has reinforced our perceptions of the RELT interface in regards to navigation and ease of use. A predominant goal of the interface is to decrease the amount of time needed to navigate to frequently used locations, sometimes at the expense of those locations rarely used. The average time to navigate to those options on the initial RELT screen were roughly equal or lower than their equivalents on the traditional cell phone screen, while those options hidden on the RELT screen tended to fare worse in terms of average time. Additionally, this user study must be viewed with the information that the majority of participants had experience with one interface, the traditional one, and not the RELT interface. This could indicate the reduced average times for the last RELT tasks due to developing familiarity with the interface, and thus more in depth testing could indicate whether the learning curve is causing the larger RELT average times.

5. Conclusions and Future Work

The ubiquitous data management and processing using mobile devices create a new era for information visualization, and also multiple challenges for effective user interface design. This paper has presented the RELT approach for visualizing hier-

archical information on mobile interfaces. Two issues, economical space usage and clear hierarchical structure have been effectively addressed in RELT. The algorithm has been implemented to generate all the RELT figures in this paper. It has a linear time complexity and can be easily adapted to suit various applications. We have previously demonstrated this adaptability by visualizing the stock market performance, where the size of each leaf vertex is made a function of the corresponding company's capitalization [18].

The RELT display and interactivity are also ported to the Google Android emulator platform in order for us to have a meaningful comparison with some existing cell phone interfaces. Having implemented real-world hierarchical information on the Android emulated RELT and traditional interfaces, we have obtained valuable user experience data. The study is of course only a preliminary experiment. More empirical evidences need to be established before general conclusions can be made.

Our immediate future work is to conduct comprehensive usability study with varied navigation schemes, in comparison with existing popular cell phone interfaces.

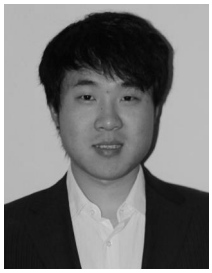
As shown in all the RELT visualizations, effective labeling of meaningful texts inside the almost arbitrarily shaped polygonal vertices (nodes) with a minimal computational cost can be a technical challenge. We have enhanced the previous simple method and obtained satisfactory labeling effects as demonstrated in our Android implementation. A related issue is how to determine (usually maximize) the font size when labeling the nodes without cluttering the screen. Another challenge, or potentially limitation, for the RELT approach is its handling of extremely imbalanced hierarchical structures. These issues will be addressed in our future research.

References

1. Y. Arase, T. Hara, T. Uemukai, and S. Nishio. *OPA Browser: A Web Browser for Cellular Phone Users*, In Proc. of ACM UIST'07, Newport, USA, pp. 71-80, 2007.
2. P. Baudisch, X. Xie, C. Wang and W-Y. Ma. *Collapse-to-Zoom: Viewing Web Pages on Small Screen Devices by Interactively Removing Irrelevant Content*, In Proc. of ACM UIST'04, Santa Fe, USA, pp. 91-94.
3. L.R. Rabiner and B.H. Juang. *Summarizing Personal Web Browsing Sessions*, In Proc. of ACM UIST'06, Montreux, Switzerland, pp. 115-124, 2006.
4. J.O. Wobbrock, J. Forlizzi, SE. Hudson and B.A. Myers. *"WebThumb" Interaction Techniques for Small-Screen Browsers*, In Proc. of ACM UIST'02, Paris, France, pp. 205-208. 2002.
5. J. Hao and K. Zhang. *RELT: Visualizing Trees on Mobile Devices*, In Proc. of 9th International Conference on Visual Information Systems (VISUAL'07), Shanghai, China, **LNCS 4781**, Springer, pp. 344-357, 2007.
6. E.M. Reingold and J.S. Tilford. *Tidier Drawing of Trees*, IEEE Trans. on Software Engineering, **7**(2), pp. 223-228, 1981.
7. P. Eades. *Drawing Free Trees*, Bulletin of the Institute for Combinatorics and Its Applications, pp. 10-36, 1992.
8. I. Herman, G. Melançon and M.S. Marshall. *Graph Visualization in Information Visualization: a Survey*, IEEE Trans. on Visualization and Computer Graphics, pp. 24-44, 2000.
9. Y. Shiloach. *Arrangements of Planar Graphs on the Planar Lattices*, PhD Thesis, Weizmann Institute of Science, Rehovot, Israel, 1976.
10. G.G. Robertson, J. D. Mackinlay and S. K. Card. *Cone Trees: Animated 3D Visualizations of Hierarchical Information*, In Proc. of the ACM CHI 91 Human Factors in Computing Systems Conference, pp. 189– 194, April 28 - June 5, 1991.

11. G.G. Robertson, S.K. Card and J.D. Mackinlay. *Information Visualization Using 3D Interactive Animation*, Communication of the ACM, Vol. 36, No. 4, pp. 57-71, 1993.
12. C. S Jeong and A. Pang. *Reconfigurable Disc Trees for Visualizing Large Hierarchical Information Space*, In Proc. of the IEEE Symposium on Information Visualization, IEEE CS Press, pp. 19-25, 1998.
13. B. Johnson and B. Shneiderman. *Tree-maps: A Space-filling Approach to the Visualization of Hierarchical Information Structures*, In Proc. of the 1991 IEEE Visualization, IEEE, Piscataway, NJ, pp. 284-291, 1991.
14. B. Shneiderman. *Treemaps for Space-Constrained Visualization of Hierarchies*, last updated April 28, 2006. http://www.cs.umd.edu/hcil/treemap-history/index.shtml.
15. K. Andrews and H. Heidegger. *Information Slices: Visualising and Exploring Large Hierarchy Using Cascading, Semi-circular Discs*, In Proc. of IEEE Symposium on Information Visualization, pp. 9-12, 1998.
16. J. Yang, M.O. Ward and E.A. Rundensteiner. *InterRing: An Interactive Tool for Visually Navigating and Manipulating Hierarchical Structures*, In Proc. of the IEEE Symposium on Information Visualization, pp. 77 -84, 2002.
17. B. Karstens, M. Kreuseler and H. Schumann. *Visualization of Complex Structures on Mobile Handhelds*, Proc. International Workshop on Mobile Computing, 2003 - wwwicg.informatik.uni-rostock.de..
18. J. Hao, K. Zhang, C.A. Gabrysch and Q.M. Zhu. *Managing Hierarchical Information on Small Screens*, Proc. Joint International Conferences on Asia-Pacific Web Conference (APWeb) and Web-Age Information Management (WAIM), LNCS, Springer, pp.429-441, 2009.

Jie Hao

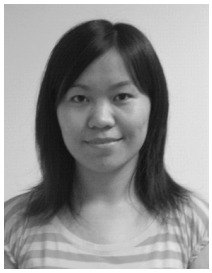


He received the B.S. Degree in Applied Mathematics from Beijing Jiaotong University, Beijing, China in 2005.He recivied his M.S. and Ph.D. Degree in Computer Science at the University of Texas at Dallas, U.S.A in 2008 and 2010. His research interest mainly focuses on information visualization.

Chad Allen Gabrysch

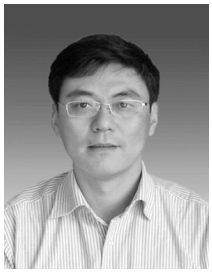
He was an M.S. student of Computer Science at the University of Texas at Dallas while conducting this work. He obtained his B.S. in Computer Science from the University of Texas at Dallas in 2007.

Chunying Zhao



She received the B.E. and M.E. Degrees in Computer Engineering from Nankai University, Tianjin, China in 2002 and 2005, respectively. She is currently a Ph.D. candidate in the Department of Computer Science at the University of Texas at Dallas. Her research interests include software visualization, program comprehension, and reverse engineering, visual languages.

Qiaoming Zhu



He is Director of Information Technology, Professor of School of Computer Science and Technology at the Soochow University, China. He received his B.Sc. in Physics from Soochow University in China in 1984, and Ph.D. in Computer Science from Soochow University in 2006. Dr. Zhu's current research interests include Chinese information processing, embedded systems and software engineering, and has published over 30 papers in international conferences and journals in these areas. His research has been funded by the Chinese 863 High-Tech Research Program, Chinese NSF and MoE. Dr. Zhu is on the Editorial Board of Journal of Chinese Information Processing. His home page is at www.zhhz.suda.edu.cn/ qmzhu/.

Kang Zhang

He is Professor and Director of Visual Computing Lab, Department of Computer Science at the University of Texas at Dallas. He is also an Adjunct Professor of the UT-Dallas Computer Engineering Program and GIS Program. He received his B.Eng. in Computer Engineering from University of Electronic Science and Technology of China in 1982, and Ph.D. from the University of Brighton, UK, in 1990. Prior to joining UT-Dallas, he held academic positions in the UK, Australia, and China. Dr. Zhang's current research interests include information visualization, visual languages, aesthetic computing, and software engineering; and has published over 180 papers in these areas. He has authored and edited five books. His research has been funded by the UK SERC, Australian Research Council, Sun Microsystems, Texas State, US NSF, US NIH, and US Department of Education. He has been the General Chair and Program Chair of several major international conferences. Dr Zhang is on the Editorial Boards of Journal of Visual Languages and Computing, International Journal of Software Engineering and Knowledge Engineering, and International Journal of Advanced Intelligence. His home page is at www.utdallas.edu/ kzhang.