DOITrees Revisited: Scalable, Space-Constrained Visualization of Hierarchical Data

Jeffrey Heer [1,2]
[1]Group for User Interface Research
University of California, Berkeley
Berkeley, CA 94720-1776 USA
jheer@cs.berkeley.edu

Stuart K. Card [2]
[2]Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94301 USA
card@parc.com

ABSTRACT
This paper extends previous work on focus+context visualizations of tree-structured data, introducing an efficient, space-constrained, multi-focal tree layout algorithm ("TreeBlock") and techniques at both the system and interactive levels for dealing with scale. These contributions are realized in a new version of the Degree-Of-Interest Tree browser, supporting real-time interactive visualization and exploration of data sets containing on the order of a million nodes.

Categories and Subject Descriptors
H.5.2 [**Information Interfaces**]: Graphical User Interfaces. I.3.6 [**Methodology and Techniques**]: Interaction Techniques.

General Terms: Algorithms, Design, Human Factors.

Keywords: visualization, tree, layout, focus+context, scalability

1. INTRODUCTION
The visualization of tree structures has received a great deal of attention due to their wide applicability and algorithmic tractability. File systems, organization charts, and taxonomies are just a few commonly encountered examples. In addition, many other richer graph structures, such as web sites, family trees, and social networks, are amenable to tree-based layouts. As a result, the visualization of trees has been the topic of an immense body of research work, progressing from static images to dynamic, interactive visualizations of increasing scale (both [2] and [5] provide excellent reviews of this topic). Still, exponential increases in processing power, networking, and data storage have given rise to increasingly massive data sets. Meanwhile, display size and resolution have grown at much slower rates and people's perceptive capabilities remain more or less constant. Thus improved means for tree visualization and exploration are still very much desirable.

Recent projects have addressed concerns of scale. Previously, we have described Degree-Of-Interest Trees (DOITrees): interactive trees with animated transitions that fit within a bounded region of space and whose layout depends dynamically on the user's estimated degree-of-interest [2]. DOITrees use multiple focus+context techniques to achieve these goals: *logical filtering* of nodes, using the estimated degree-of-interest to determine which nodes to display; *geometric distortion*, changing node sizes to match the estimated interest; *semantic zooming* of content based on node size; and *aggregate representations* of elided subtrees. By employing scaling and overlapping of nodes, the original DOITrees ensured that trees stayed within a bounded space, promoting use as a component of larger applications. Original versions of DOITrees were limited, however, by a layout algorithm unsuited for handling multiple foci and by unoptimized DOI calculations, limiting scale.

Similar in spirit to DOITrees is Plaisant et al.'s SpaceTree [6], which uses logical filtering and aggregation of nodes, combined with animation and automated camera management, to visualize tree structures. The SpaceTree supports multiple foci, search, and filtering, but does not aggressively employ space constraints on the tree, at times requiring a great deal of manual panning.

In this paper we extend this previous work to create tree visualizations that better support data sets of increasing magnitude. The bulk of our contribution lies in an optimized, scalable approach to degree-of-interest (DOI) calculation, and in TreeBlock, an efficient, space-constrained multi-focal tree layout technique. Algorithms for each are presented. These contributions are instantiated in a new version of the DOITree browser, and leveraged by search, filtering, and authoring features facilitating the exploration of large tree structures. These make possible the rapid exploration of DOITrees perhaps three orders of magnitude larger than the original DOITrees. We first describe the system in use and then provide the details of its implementation.

2. DESCRIPTION
Our new variant of Degree-of-Interest Trees presents a top-down node-link representation of the tree, optimized for the display of largely textual data. Figure 1 shows a DOITree visualization of the Open Directory Project (http://dmoz.org), a volunteer-driven international directory of websites containing over 600,000 categories. Clicking on a node in the visualization causes it to become the new focus, immediately initiating a smooth, slow-in slow-out animation between tree configurations. Newly visible nodes flow out from their parents, while other previously visible nodes become hidden, returning to their parents and fading out to transparency, ultimately being replaced by an elision graphic indicating the size of the unexpanded subtree. To help users track the appearance of previously unseen information, newly visible nodes are initially highlighted. After the nodes reach their final positions this highlighting gently fades out.

The orientation of the tree is not fixed; data can be displayed top-down, bottom-up, left-to-right, or right-to-left. This orientation can be changed on-the-fly by a single keystroke, initiating an animated transition between configurations.

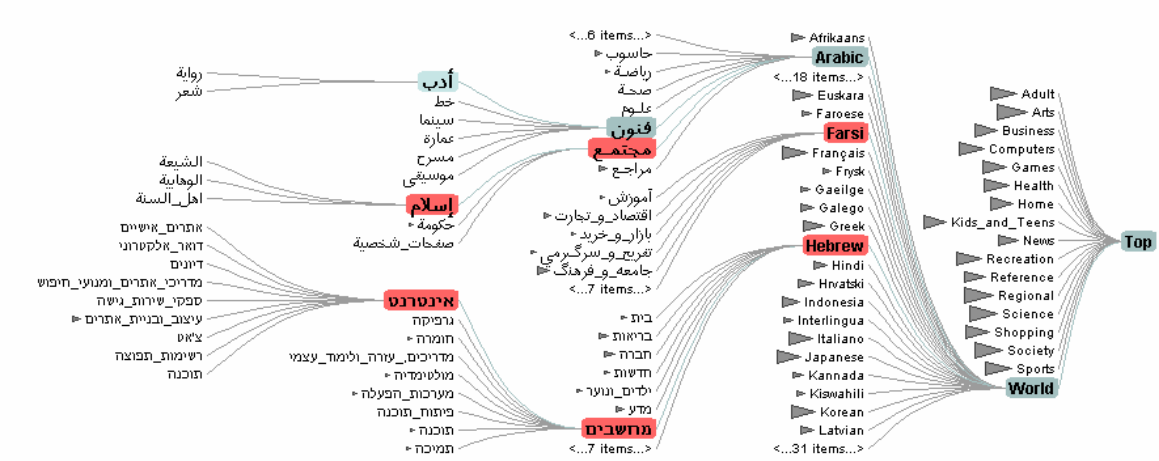

Figure 1 DOITree visualization of the Open Directory Project (http://dmoz.org). The tree contains over 600,000 nodes, laid out in a right-to-left orientation. Multiple foci have been selected and the various expanded branches are allocated as much space as possible given the display constraints.

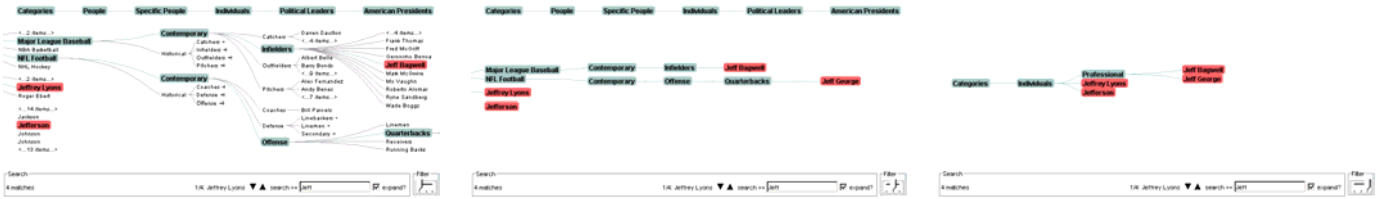

Figure 2 Search and Filtering in DOITrees. As the Filter slider on the bottom right of the display is moved further to the right, the DOI distribution becomes increasingly constrained, creating more compact views by reducing context. The tree moves from a full fisheye to just search results and their ancestors, to finally just search results and their least common ancestors.

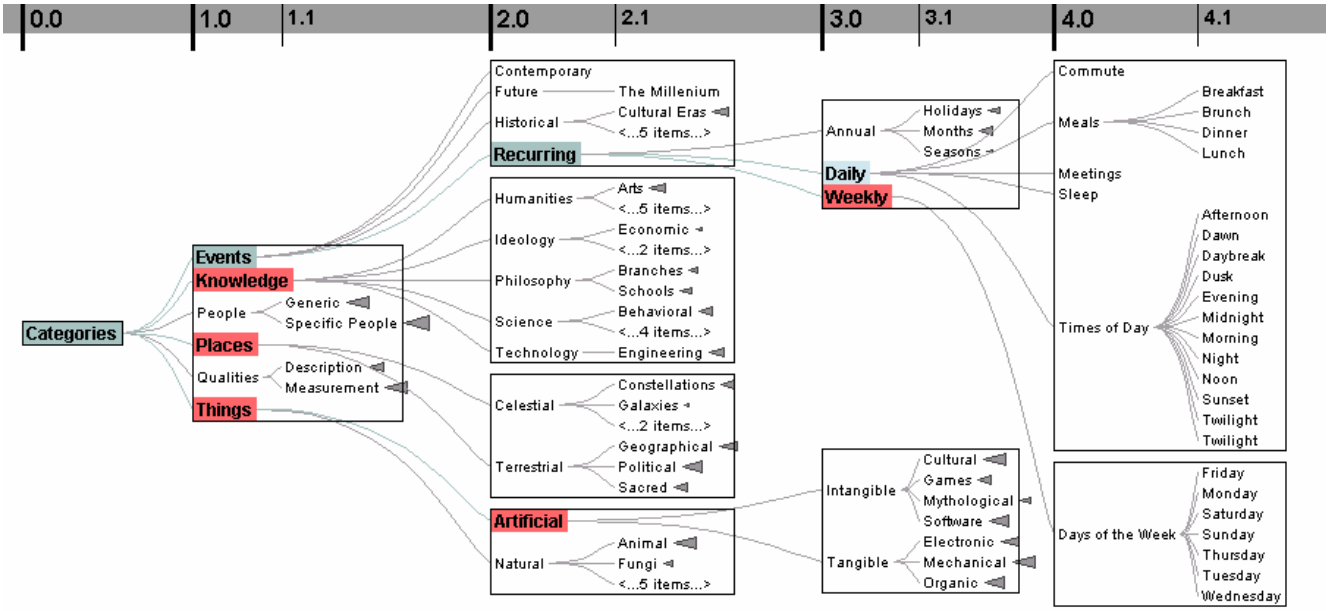

Figure 3 Block-level representation of a visualized tree. Siblings and lower-interest subtrees are grouped together in rigid blocks to simplify layout. Additionally, the tree is segregated into varying depth levels (indicated by the markers above the tree) to facilitate dynamic computation of space requirements.

To support comparison across tree branches, users can select additional focus nodes. This is currently achieved by right clicking a node, thereby assigning it maximal user interest. The underlying TreeBlock layout algorithm responds by expanding multiple tree branches as necessary, maximizing the allocated space for each expanded branch, subject to the space constraints of the display. Deeper tree paths (e.g., those rooted at "Arabic" and "Hebrew" in Figure 1) expand to use up available space underneath other, shorter tree paths (e.g., the one rooted at "Farsi"). In our experience the use of curved edges helps offset the disorientation that can arise following tree paths through the limited display space.

2.1 Space Constraints

Problems naturally arise, however, when the display space is not sufficient for displaying the current view of the tree. These problems can occur along both the breadth and depth dimensions.

When the tree breadth exceeds the display bounds a number of solutions may be applied. One approach is to use scrolling or panning. Another technique is to scale the tree, as done by both zooming interfaces and earlier versions of DOITrees [2]. Nodes can also be overlapped, as done in [2]. A common organization chart convention, incorporated by both DOITrees and SpaceTrees, is to layout nodes along multiple rows. Three-dimensional visualizations can also apply natural perspective, using rotation to selectively present sibling nodes as done in Cone Trees [7].

An additional mechanism used by our visualization is to aggregate nodes of lower interest. When expanded tree branches contain too many nodes to fit on the screen at once, the nodes of lowest estimated interest are automatically culled until the tree breadth fits within the display, replacing these nodes with an aggregate representation (as in the "World" category in Figure 1). Clicking this aggregate will redistribute the estimated interest between the nodes in the branch. This expands the aggregate, revealing hidden nodes while previously visible nodes are newly aggregated.

Space constraints also pose a problem along the depth dimension. As increasingly deeper levels of the tree are visualized, the tree may become too large for the display. Again, scrolling/panning and scaling techniques can be applied. In addition to tree scaling, DOITrees now support automatic panning of the tree view, centered on the most recent user-selected focus node. To provide tree context, the visualization presents a "bread-crumb" trail of ancestor nodes along the periphery of the visualization (see Figure 2). Clicking on such a node causes the view to translate and ancestors to "fall" from the bread-crumb trail into their proper positions.

2.2 Exploring Large Trees

In addition to the problem of space constraints, the massive scale of important data sets (e.g., web indices and biological taxonomies) presents additional challenges to the user. As we describe in the Method section, we utilize efficient Degree-of-Interest estimation and tree layout techniques to ensure real-time interaction with trees on the order of a million nodes. However, this still leaves users with the challenge of exploring incredibly large information structures. To help facilitate user interaction with such structures, DOITrees support search and advanced filtering options, similar to those found in the SpaceTree [6]. Users can additionally create their own trees of bookmarked nodes, building spatial indices of large data sets.

Typing a query into the search box causes matching nodes to light up in the visualization. Subtree aggregates light up to indicate branches containing search hits. A linear list of search results is available to the user; double clicking an entry in the results list takes the user to that section of the tree.

Additionally, users can have search results treated as foci of the display, causing all branches containing search hits to be expanded (see Figure 2). For searches with many matches this may result in an overwhelming tree. Search filtering can be used to prune the display, controlled by both the mouse scroll wheel and an on-screen slider. At the lowest pruning level a full fisheye distribution is used, at the medium level only focal nodes and their ancestors are visualized, while at the highest level only focal nodes and their least common ancestors are displayed. This last setting can be especially useful for collapsing visualizations in which the search results occur at varying depths of the tree. Using the mouse scroll wheel switches between filtering levels while staying within the context of browsing, enabling users to dive in and out of search results on-the-fly.

Finally, we suspect that many users may want to bookmark areas of the tree, either for comparison purposes or to facilitate revisiting. To support this across sessions, we allow users to create *index trees*, user-organized subsets of the larger data set. Users can click on a node in the larger tree and then drag it onto the index tree window, placing the nodes in the desired location in the tree. The index tree is a full DOITree visualization in its own right, with the additional feature that double clicking a node in the index tree causes the corresponding node in the original data set to become a focus node. This allows users to build their own task specific indices into larger data sets, and save them for later use.

3. METHOD

In this section we describe the techniques employed to implement the visualization just described. We present both a generally applicable caching approach for efficiently computing Degree-of-Interest estimates of relevant tree nodes, and TreeBlock, a novel space-constrained, multi-focal tree layout algorithm.

3.1 Degree-Of-Interest Calculation

At the core of our approach is the use of lightweight modeling of the user's interest to inform the layout and presentation of the tree. User interest is modeled using a Degree-Of-Interest (DOI) function [2, 3], which assigns a single number representing the estimated relative interest of the user to each node in the structure. This provides a simple yet powerful abstraction for determining which nodes will be visualized and guiding subsequent layout and visual presentation.

For general browsing, we currently employ a multi-focal version of Furnas' FISHEYE view [3] to compute the Degree-Of-Interest. This function assigns a maximal DOI to focus nodes and their parents, up to the root of the tree. Interest values for the remaining nodes then decrease linearly as a function of distance from the highest interest nodes. As nodes below a particular interest threshold will not be visualized, we exploit the convexity of the FISHEYE distribution [3] to stop the interest calculation routine at these "disinterest threshold" boundaries, thus bounding computation time to the number of visible nodes only. For non-convex DOI distributions consisting of convex subsets (e.g., the least common ancestors view above) calculation time is bounded by the number of visible nodes plus the number of nodes linking visible regions.

We have implemented this technique using a caching scheme that maintains visual representations only for those tree elements above

the disinterest threshold and thus considered sufficiently interesting for display. Tree nodes are added to the cache, if not already present, when their DOI is set. Tree links between cache entries are maintained separately from the original tree, allowing DOI to vary discontinuously across the tree. For each cache entry, a counter keeps track of the number of interaction cycles since the entry was last visited by a DOI function. When this count surpasses a threshold value (typically 1), the entry is removed from the cache. This process preserves the state of nodes that remain visible across transitions while freeing space as nodes become elided. Thus quite large trees can be visualized by displaying in full detail only a limited subset of the total structure at a given time.

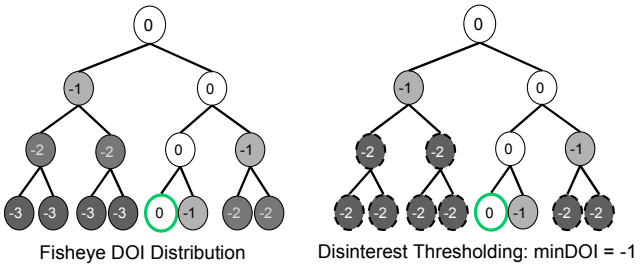

Figure 4 Fisheyes with and without optimized DOI calculation. Nodes with a dotted outline are not visited by the algorithm.

Using this formulation of DOI simplifies other aspects of our implementation. For instance, to visualize the distribution of search results in a bounded space, we can simply change the DOI function so that branches of the tree without any search hits are assigned interest levels below the visible threshold. Generalizing this approach, one can use DOI functions to visualize results for a wide range of dynamic queries. The modularity of the DOI function allows this to be done without modifying downstream components such as the layout algorithm.

3.2 Tree Layout

Once the DOI calculation is complete, we can proceed with the actual layout of the tree. In this section we describe TreeBlock, a new layout algorithm that achieves space-constrained, multi-focal layout in time nearly-linear to the number of visible nodes. The algorithm first makes a preliminary pass through the tree, computing the space required by an *unconstrained* layout while also segmenting the tree into higher-level blocks. This information is then used to constrain the layout of the tree so that it fits into the available space. Trees of excessive depth are handled by translating the view of the tree, while providing a "bread-crumb" trail of any parent nodes that are no longer visible. Trees of excessive breadth are handled by aggregating nodes of lower interest. A final pass through the tree then assigns nodes to their on-screen locations.

3.2.1 Tree Extents and Segmentation

The first phase of our layout algorithm computes the space taken by the tree in the absence of any display constraints, simultaneously segmenting the tree into a block structure to simplify subsequent layout calculation. The output of this phase is the breadth and depth, in pixels, of the unconstrained tree layout, and an aggregated block representation of the tree (see Figure 3).

The to-be-visualized subset of the tree is traversed in a depth-first fashion, and for each group of siblings the breadth and depth requirements are computed. Node sizes, which may be variable in both breadth and depth, are computed in constant time by querying rendering modules. Global data structures monitoring the depth requirements for each level of the tree are maintained, allowing depth requirements for particular levels of the tree to be dynamically computed across different tree branches.

Groups of siblings and subtrees containing only lower-interest nodes are grouped into blocks, which allow the aggregated nodes to be subsequently positioned as a single unit. This block structure is depicted in Figure 3. The blocks are indexed by their depth level in the tree, while the depths of subtrees are indexed by their sub-level, as depicted by the markers at the top of the figure. This segregation allows each depth level of the tree to be separately considered with respect to space constraints and positioning, in turn enabling a flexible handling of multiple foci.

3.2.2 Space Constraints

In the second phase of the layout algorithm, the computed tree breadth and depth values are compared against the bounds of the available display space. If the breadth and depth values from the unconstrained layout do not exceed the available space, nothing extra need be done. However, if the either of these values exceeds the bounds of the display, the tree size must be reduced.

3.2.2.1 Handling Excessive Breadth

In the case of excessive breadth, aggregation can be applied in addition to or in lieu of scaling. For a given depth level with excessive breadth, aggregation removes the nodes of lowest interest until the depth level fits within the display bounds. Removed nodes are then represented in aggregate (*e.g.*, the siblings of "Arts" under "Humanities" in Figure 3). Clicking on one of these aggregates causes the DOI values of the aggregated nodes to rise, which in turn causes the aggregate to expand, revealing a subset of elided nodes.

Aggregates are computed by sorting all the items in the given depth level by their DOI values, then sequentially removing the lowest-interest nodes and updating size calculations until the blocks will fit within the display bounds. Since the lowest-interest nodes may be dispersed throughout the given depth level, this requires some book-keeping of elided nodes, updating size calculations when adjacent nodes are marked for aggregation.

3.2.2.2 Handling Excessive Depth

In the case of excessive depth, the tree is too tall for its allotted space. One solution, applied in the contexts of both geometric scaling [2] and zooming [6], is to scale the tree to fit. Excessive scaling, however, eventually destroys the legibility of the tree. As a result, in some cases (e.g., largely textual data) scaling can be undesirable. One recourse is to introduce scrolling or panning [6].

The solution implemented in our current work is to automatically translate the view, centering the visualization on the most recent user-selected focus node. A bread-crumb trail of elided ancestors is used to maintain context and provide immediate access to higher levels of the tree. The translation component is determined by counting how many depth levels, starting from the root, must be removed to allow the user-selected focus and its children to fit into view. In addition, we add some extra logic to prevent the tree from translating back and forth as the user browses a group of siblings with descendants of shifting depth requirements. This prevents the jarring effect of the focus unduly bouncing around.

3.2.3 Position Assignment

In the final phase of the layout algorithm, nodes are assigned their screen co-ordinates. This pass iterates through each depth level from the root of the tree down, at each level first determining the position of the blocks and then laying out the nodes each block contains. Iterating through depth levels visits nodes in a breadth-first fashion, ensuring that parent nodes have their positions assigned before children blocks are considered.

To facilitate users' visual search, we would like children blocks centered underneath their parent nodes. Due to screen boundaries and space constraints, however, this is not always possible, so we would like to minimize the distance between parents and children. This could be achieved by minimizing the squared distance between parent nodes and the centers of their children blocks, subject to the constraint that all blocks fit within display bounds and do not overlap. This constitutes a quadratic programming optimization problem, which, while solvable using various optimization techniques, we address with an approximation that runs in time linear to the number of blocks.

We process the blocks within a depth level from the "outside-in," beginning with the blocks nearest the edges of the display and centering them under their parents without exceeding the display boundaries. We then compute the amount of space needed for the remaining blocks (which might be zero), and push the current blocks towards the display boundary as necessary to free the requisite space and ensure that they do not overlap. The distance each block is moved is proportional to its size, as larger blocks can move further and still be directly underneath (though not centered under) their parent. This process is then repeated recursively for the pair of blocks adjacent to the previous ones, until either all blocks have been positioned or there is only a single block left, which we simply place within the remaining space, aligning its center as close as possible to its parent.

Once the blocks for a given level have been positioned, the provided bounds are used to position the nodes and aggregates within the blocks by a depth-first traversal of the block content.

4. EVALUATION AND ANALYSIS

Both empirical and theoretical analyses confirm the efficiency of our algorithmic contributions. Figure 5 depicts a comparison of DOI calculation times for three approaches to the FISHEYE view: a naïve tree traversal (the typed used by the original DOITrees), a pruned tree traversal that updates only the tree rooted at the least common ancestor of current and previous foci [4], and our disinterest thresholding approach. The plots were generated by timing DOI calculations of random walks in uniform trees. The tests were performed on a 1GHz Pentium III IBM ThinkPad T23 with 256MB RAM. As shown by the graph, our approach scales logarithmically, while the others scale linearly, exceeding a 100ms threshold [1] around 30,000 nodes.

An informal asymptotic analysis also shows that the proposed TreeBlock algorithm has a nearly linear running time, bounded from above by $n \log n$, where n is the number of visible nodes prior to aggregation. Both the first and third passes made by the algorithm are clearly linear, as they involve a single walk through the tree and our node size computation and tree segmentation routines run in constant time. The only point at which non-linear complexity is introduced is during the second pass, if and when aggregation occurs, requiring the $n \log n$ operation of sorting the nodes in a given depth level. However, this rarely includes the whole visualized structure, making the common case less costly than the upper bound might imply. In actual usage the algorithm has performed admirably, as rendering bottlenecks will slow performance long before the time required by the layout algorithm becomes an issue.

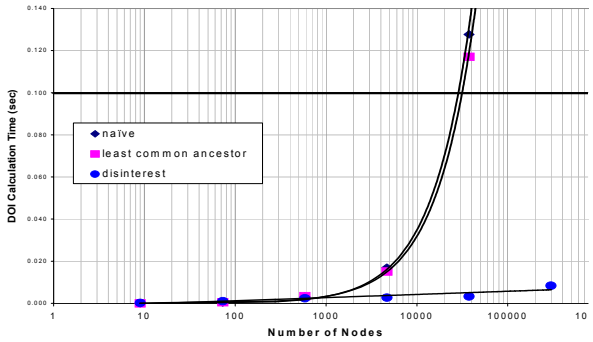

Figure 5 Comparison of DOI calculation times

5. CONCLUSION

In this paper we have extended previous work on focus+context visualizations, describing an optimized approach to DOI calculation and presenting TreeBlock, a new tree layout algorithm that supports the display of multi-focal trees within bounded space constraints. Together, these features allow interactive visualization at animation rates with tree structures on the order of million nodes while making the most of the available screen real estate. On top of these system features we have implemented techniques for supporting the exploration of large trees, including advanced search and filtering options and the ability to construct index trees, user-defined spatial indices into the larger data set. We are currently in the process of running a controlled user study to understand the impact of these techniques on user experience. In future work we intend to use the results of this evaluation to improve our design and visualize ever larger and varied structures.

6. REFERENCES

[1] Card, S.K., T.P. Moran, and A. Newell, *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum, 1983.

[2] Card, S.K. and D. Nation. Degree-of-Interest Trees: A Component of an Attention-Reactive User Interface. *Advanced Visual Interfaces*. July 2002.

[3] Furnas, G.W., The FISHEYE View: A New Look at Structured Files, in *Readings in Information Visualization: Using Vision to Think*, Morgan Kaufmann: San Francisco, 2001.

[4] Furnas, G.W. Generalized Fisheye Views. *CHI'86, Human Factors in Computing Systems* 1986.

[5] Herman, I., G. Melancon, and M.S. Marshall, Graph Visualization and Navigation in Information Visualization: A Survey. *IEEE Transactions on Visualization and Computer Graphics*, 2000. **6**: p. 24-43.

[6] Plaisant, C., J. Grosjean, and B. Bederson. Spacetree: Supporting Exploration in Large Node Link Tree, Design Evolution and Empirical Evaluation. *IEEE Symposium on Information Visualization*. October 2002.

[7] Robertson, G.G., J.D. Mackinlay, and S.K. Card. Cone Trees: Animated 3d Visualizations of Hierarchical Information. *CHI'91, Human Factors in Computing Systems* 1991.