Point-Based Tree Representation: A new Approach for Large Hierarchies

Hans-Jörg Schulz* University of Rostock
Steffen Hadlak† University of Rostock
Heidrun Schumann‡ University of Rostock

ABSTRACT

Space-filling layout techniques for tree representations are frequently used when the available screen space is small or the data set is large. In this paper, we propose a new approach to space-filling tree representations, which uses mechanisms from the point-based rendering paradigm. Additionally, helpful interaction techniques that tie in with our layout are presented. We will relate our new technique to established space-filling techniques along the lines of a newly developed classification and also evaluate it numerically using the measures of the *Ink-Paper-Ratio* and *overplotted%*.

Keywords: Tree visualization, space-filling layout, point-based rendering.

Index Terms: I.3.8 [Computing Methodologies]: Computer Graphics—Applications.

1 INTRODUCTION

Large hierarchies occur frequently in real world applications, among which are the life sciences and engineering. Yet, their graphical representation is problematic, as the tree might be heavily unbalanced, extremely wide but relatively shallow, or quite narrow but very deep. The Treemap [9] and its successors are often used techniques for their representation, because they scale well above all known node-link-representations. This is due to the fact that they are space-filling techniques, meaning that they utilize the available screen space entirely. While many node-link-techniques try to optimize their usage of the available screen space, none of them can be called "space-filling" in the very sense of the word. With this paper, we want to close this gap and present a first approach to a space-filling node-link overview visualization for large graphs.

The idea behind those techniques that call themselves space-filling lies in the fact that they use the available space itself as the matter that is formed by the layout algorithm to represent the given tree. Conversely, node-link-representations rather see the tree with its nodes and edges as the object that needs to be shaped to fit the available space. The latter is of course hard to do in a manner that yields a truly space-filling representation, as the node-representing points usually leave some blank space in between them.

To solve this problem for a node-link-layout, we propose a tessellation of the available space that arranges the nodes in a space-filling way. This tessellation is taken from the point-based rendering paradigm, where similar problems occur. An example of how our visualization technique treats a large real world hierarchy is shown in Fig. 1. The depicted tree is the categorization hierarchy of the Open Directory Project dmoz.org with more than 750.000 nodes. This categorization is used by many sites throughout the internet for classifying websites (e.g. by Google Directory) or news articles (e.g. by ScienceDaily).

*e-mail: hjschulz@informatik.uni-rostock.de
†e-mail: hadlak@informatik.uni-rostock.de
‡e-mail: schumann@informatik.uni-rostock.de

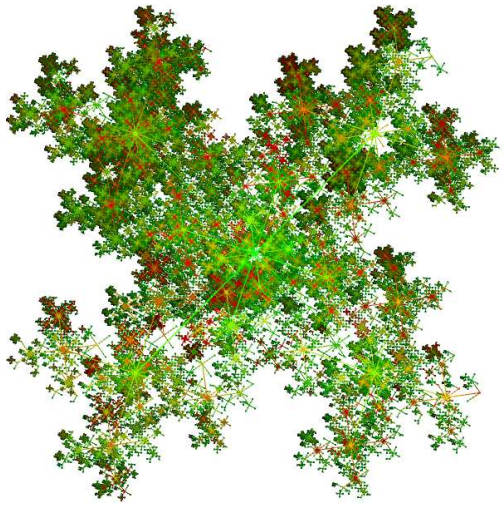


Figure 1: Point-based visualization of the DMOZ classification hierarchy. It contains 754403 nodes of which are 576818 leaves. (dmoz.org snapshot from 03-SEP-2008)

It can be seen that our technique still leaves parts of the screen space empty, if the tree is unbalanced or partially very narrow. This is actually a very helpful property of an overview, as it allows to grasp certain tree characteristics at a glance. Yet overall, nodes are always positioned in between existing nodes in an attempt to avoid overlap, creating a space-filling pattern according to the point-based tessellation. Our method, its ties to point-based rendering, and some useful interaction techniques that go along with our layout are presented in Section 2. Its key concepts are illustrated with examples from the DMOZ data set shown in Fig. 1. Then, we conceptually and numerically relate the new point-based layout technique to existing techniques. For the conceptual comparison, we introduce a classification scheme in Section 3, which generalizes and categorizes space-filling techniques according to their layout properties. The numerical comparison is then done with two established space-filling techniques in Section 4. At the end, we give a short conclusion and sketch ideas for future work in Section 5.

2 A POINT-BASED APPROACH TO TREE VISUALIZATION

When large amounts of data come into play, visualization designers look for techniques that make the best use of the available screen space. Space-filling layout techniques achieve this goal for the special case of hierarchical structures. Yet, in the past, space-filling techniques have been set equal to implicit tree layouts. It seemed that only implicit techniques with their 2-dimensional graphics primitives are able to fully fill the available screen space. Hence, explicit techniques, which try to maximize their usage of the screen space, usually call themselves "space-optimized" or "space-efficient". The layout presented here is a first attempt to fill the gap in between space-filling, implicit techniques and space-optimized, explicit techniques. It is designed as an explicit technique that is truly space-filling in the sense of the following definition:

Definition: A tree visualization is called *space-filling*, iff

$$\text{Ink-Paper-Ratio} = \frac{|\text{used pixels}|}{|\text{available pixels}|} = 1$$

and

$$|\text{available pixels}| \ll |\text{nodes to display}|$$

The first condition of this definition formulates the usual understanding of the term "space-filling", namely that every available pixel is used. The measure of the Ink-Paper-Ratio [4] is basically the quotient of Tufte's Data Density and his Data-Ink-Ratio [19]:

$$\text{Data Density} = \frac{|\text{nodes to display}|}{|\text{available pixels}|}$$

$$\text{Data-Ink-Ratio} = \frac{|\text{nodes to display}|}{|\text{used pixels}|}$$

The second condition of the definition makes sure that the first property does not simply hold because of massive overplotting, as many techniques become virtually space-filling if they are used with too much data on a small screen space.

2.1 Inspiration

The idea for our proposed layout method stems from the well-established area of Point-Based Graphics [7]. By point-based methods, triangular graphics primitives, of which 3-dimensional renderings mostly consist, are replaced by point primitives. This makes sense, as today's high-resolution-models consist of millions of triangles, which cover only a relatively small screen area and often share the same pixels. This results in a computational overhead, which is usually not worth the result. Especially so, as the same result can be computed with less effort by using point primitives instead, which are easier to render. Yet, their usage brings along a new problem: where triangles describe a surface by design, points need to be carefully arranged to guarantee that no holes or gaps are left in between. This effect is called *undersampling* and an example is shown on the left side of Fig. 2, where the points are sampled regularly and leave gaps in steep regions. The right side of Fig. 2 shows a closed surface instead, which is generated by the more advanced $\sqrt{5}$-sampling [16]. This improvement is due to the adaptive nature of this method, as it samples more points in otherwise undersampled areas and thus achieves complete coverage with only as many points as needed.

Interestingly, a space-filling, explicit tree layout technique would need to solve the very same problem to cover the entire screen space without leaving gaps. So, even though the $\sqrt{5}$-sampling method has been developed for a different context, our tree layout uses it to position the tree's nodes in a space-filling way.

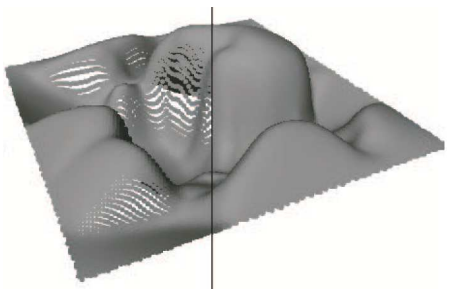


Figure 2: A point-based rendering of a surface with undersampling effects (left) and $\sqrt{5}$-sampling results (right) – taken from [16]

2.2 The Basic Layout Technique

The $\sqrt{5}$-sampling uses a hierarchical sample scheme for positioning points at possibly undersampled, blank spots around other points. For each step of this technique, the starting grid will be refined by a rotation of approx. 27° and a reduction of $1/\sqrt{5}$ of the distance between two adjacent grid points. Around every undersampled point, four new points will be inserted at the nearest position in the current grid. Fig. 3 illustrates the steps of this algorithm. This shows nicely how the overall density increases with every recursion step and how gaps are filled in between the points.

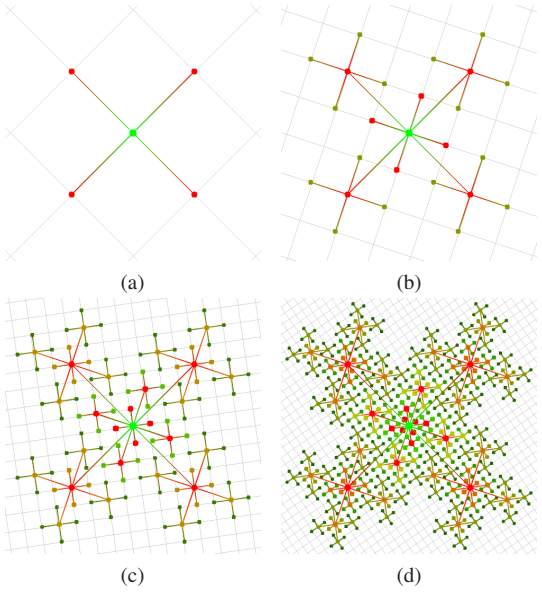


Figure 3: Four recursion steps of the $\sqrt{5}$-sampling method.

In Fig. 3, additional lines where included that are not part of the original $\sqrt{5}$-sampling method. These lines (edges) between the points (nodes) already hint at a possibility to map a tree structure onto the resulting point positions. While the $\sqrt{5}$-sampling is originally adapted according to the surface properties, our technique uses it to adapt to the tree characteristics. This is done in the same recursive, step-wise manner as the $\sqrt{5}$-sampling method itself:

(a) Starting with the root in the center of the available screen space, the root's first 4 children will be arranged around it.

(b) In the next step, the next 4 children of the root and the first 4 children of the previously laid out nodes are positioned by a rotation and scaling according to the $\sqrt{5}$-sampling.

(c) Then, the same procedure is repeated to layout the next 4 children of the root node, the next 4 children of the nodes laid out in step (a), and the first 4 children of the nodes that have been added in step (b).

(d) This last step of Fig. 3 adds another 4 children around the root node, as well as 4 children to all nodes from steps (a)-(c).

This procedure is repeated until all nodes of the tree are positioned. Since not all siblings of a node can be positioned in the same step and thus on the same level, a green-to-red color scale is used to visualize the number of siblings as an indication of a subtree's width. In the example from Fig. 3, this can be observed for the 16 red children of the root node: while the first four of them could be placed very prominently, the following steps can assign

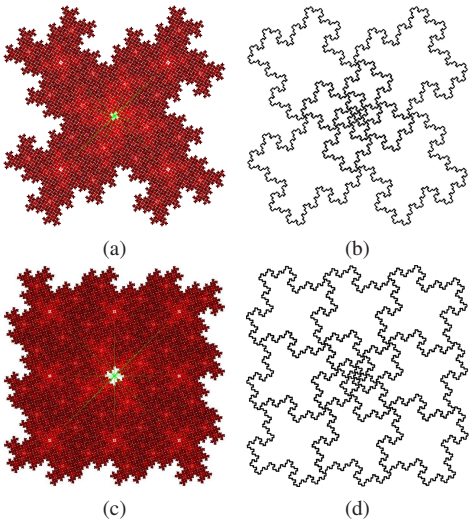


Figure 4: Layout adjustment with 8 (c+d) instead of 4 (a+b) children positioned around the root node.

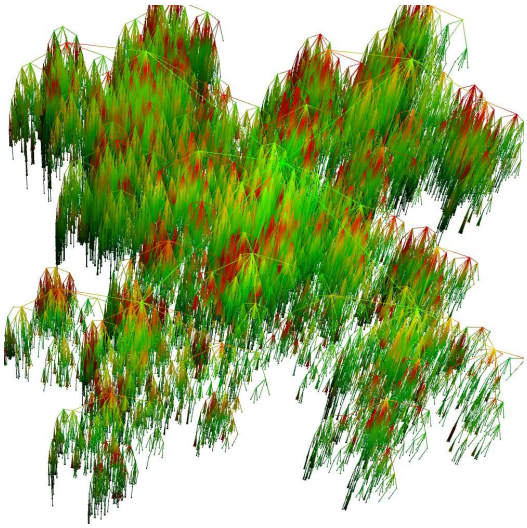


Figure 5: Improving readability by encoding the hierarchy levels on the z-axis in a 3D representation of the tree from Fig. 1.

only decreasing areas to the remaining children. So, even though the last few of them do not really stick out anymore, it can be seen from the red color assigned to all of them that they have a lot of siblings and that the tree is quite wide at the first level.

Also, the level of a node within the tree is mapped to the brightness of this color. This is done for the same reason as above: siblings are not necessarily assigned the same amount of screen space, but should still be distinguishable from lower levels. For example in Fig. 4a, it can be observed by the brightness that the area around the root in the middle of the representation contains nodes of the same level as the centers of the surrounding four areas.

So apart from the outer border, the described incremental refinement of the layout allows to use indeed every single pixel of the available space and hence to be space-filling in this sense. Yet, as this layout method predefines all possible node positions independently of the characteristics of the tree itself, the filling degree is tree dependent. This is not considered a limiting factor, though, as the areas that remain empty carry a lot of information about a tree's width and balancing – an advantage over other established space-filling techniques, which use the available space completely at the cost of this structural information. We even use the different degrees of filling to determine whether or not to add edges to a certain area. Only when the representation is sparse enough, edges are drawn. This reduces the visual clutter on the dense parts of the tree visualization, and still provides information on the linkage of sparsely scattered nodes in other areas. In our implementation, the threshold on the amount of filling, which determines whether to draw edges or not, can be interactively changed to a value that suits the user's task. Besides this, our technique comes with four more benefits by design due to its fixed positioning:

- It **can be laid out in** $O(n)$ in a DFS or BFS manner. The node positions for a full tree of a certain width and depth can even be precomputed and then be used as a layout template for different instances of actual trees.

- It **preserves the orientation** even if parts of the tree are altered, deleted, or added. This is because local changes to the hierarchy will only result in local changes to the layout.

- It allows for an **easy comparison** of large trees. Because the difference between the layout of similar subtrees will primarily be the scaling, it is easy for a user to find recurring patterns.

- It is possible to **determine an optimal starting distance** between the root and its first four children from the given display resolution by reverse calculation of the rotation and scaling to maximize the usage of the available space. As a rule of thumb, a third of the given dimension works well. This means for a screen space of 600×600, the distance between the nodes in the initial grid should be 200.

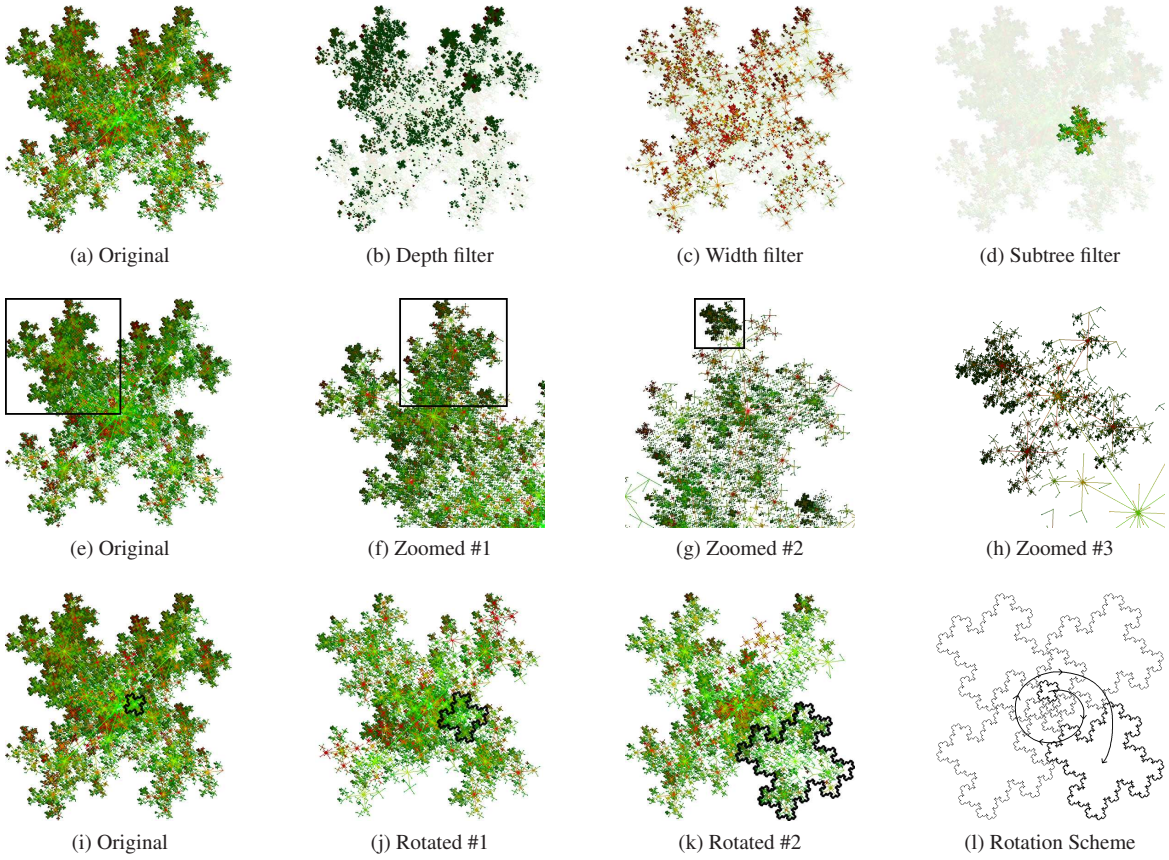
As it can be seen in Fig. 4a, the rather rugged outline of the area filled by a full tree leads to relatively large unoccupied areas at the border. For a square screen space, the full tree occupies about 60% of it. Since the available screen space is usually of rectangular shape, its utilization can further be improved by the adjustment shown in Fig. 4c and 4d. Here, four additional children of the root were laid out in the first step. Due to the interleaving structure, their subtrees integrate seamlessly with the overall layout. This adjustment increases the utilization of the screen space to about 80%.

To enhance the readability of our layout, we provide a 3-dimensional extension as an alternative display style, into which the 2-dimensional orthogonal top-down view can be tilted interactively. It improves particularly the perception of the depth of the hierarchy as it assigns a z-coordinate to each node according to its hierarchy level. This approach is illustrated in Fig. 5.

It should be noted that even though we provide a lot of facilities to improve the readability of the layout (color coding, 3D extension, etc.) it still requires some training to read it properly. One simply cannot squeeze as many nodes as possible on the screen and at the same time expect this representation to be well spaced out and instantaneous to grasp for the untrained eye. So, the interaction techniques described in the following allow to manipulate the representation. They help in building up an understanding of the data beyond the initial layout by providing mechanisms for exploration.

2.3 Interaction Techniques

The layout technique as described above is designed in such a way that it communicates an overall impression of a tree's characteristics, but also serves as a good basis for further exploration of lower levels of the hierarchy. So, while our initial point-based layout serves as an overview to start from, the following techniques enhance it with suitable methods of interaction along the lines of the *Visual Information Seeking Mantra "Overview first, zoom and filter, then details-on-demand"* [15]:



(a) Original (b) Depth filter (c) Width filter (d) Subtree filter

(e) Original (f) Zoomed #1 (g) Zoomed #2 (h) Zoomed #3

(i) Original (j) Rotated #1 (k) Rotated #2 (l) Rotation Scheme

Figure 6: Different filtering techniques (a-d), subsequent zooming (e-h), and rotation (i-l) of subtrees reveal more details.

Filtering: For our layout, we propose three filtering approaches, which allow users to locate nodes of interest and mask out the rest of the tree as it is shown in Fig. 6. The first filter shows only the nodes of selected hierarchy levels in the layout. As this filter can be adjusted dynamically, the user gains insight about the growth of the tree and its depth. Fig. 6b shows for example only hierarchy levels 7 and below of the DMOZ hierarchy. The second filter accounts for the number of siblings. An application of this filter is illustrated in Fig. 6c, where only nodes with more than 30 siblings are drawn. Both filters can be used in conjunction, for example to locate where the tree becomes deep and wide. The third filter can be used to target a specific subtree, masking out all nodes that are not part of it. This is depicted in Fig. 6d, where only the subtree corresponding to the top-level DMOZ category "Business" is shown.

Zooming: This functionality is used to scale a desired region of the layout to the size of the available screen space. Depending on the properties of the tree (its depth and width), the dense space-filling layout will eventually break up and show the individual nodes. Also, since zooming-in results in a sparser layout, the degree of filling of the now zoomed-in region is recomputed and if needed, additional edges are drawn. This effect can be observed in Fig. 6e-6h, where it was zoomed into the subtree "World/Deutsch/Regional/Europa" of the DMOZ hierarchy from Fig. 1. To identify regions of interest in the overview that may be worthwhile to zoom in, we also provide a GraphSplat [20] as a linked view to indicate particularly dense regions in the layout. The GraphSplat for the example from Fig. 1 can be seen in Fig. 7. Such views have first been suggested in [3] for the visualization of millions of items. In such large data sets, cluttering artifacts and overplotting occur often and obscure the view on the actual number of data items in a certain region. This property is shared by many space-filling techniques as they try to produce dense representations, which may easily lead to overlapping nodes. Our technique is no exception to that and so, we found it helpful to get additional insight on where large numbers of nodes can be found.

Details-on-Demand: It is an inherent property of our layout that nodes first positioned have the most space for displaying their children. We make use of this property by sorting children by the number of nodes below them. By laying them out first, the largest subtrees are thus also allotted the most screen space. Yet, if a subsequently positioned node and the subtree rooted in it are of particular interest, a reorganization of the nodes allows to enlarge this subtree by placing it on one of the outermost positions. For reducing the cognitive effort of comparing the original visualization with the reordered result, we use a rotation of the node list as a traceable reorganization method. The rotation can be performed in both directions for a selected subtree and its siblings. The rotation scheme is illustrated in Fig. 6l and Fig. 6i-6k show the rotation in action. Here the DMOZ category "Adult" is being rotated outwards within the hierarchy shown in Fig. 1, so that more and more details become visible, as more space becomes available for the layout of the subtree of interest.

In combination, all these techniques play together well and allow for different paths of exploration, while the first overall impression of the tree and its balancing can be directly derived from the point-based layout. As an overview visualization should, it conveys the most important features of the tree and at the same time invites the user through its interaction techniques to take a closer look.

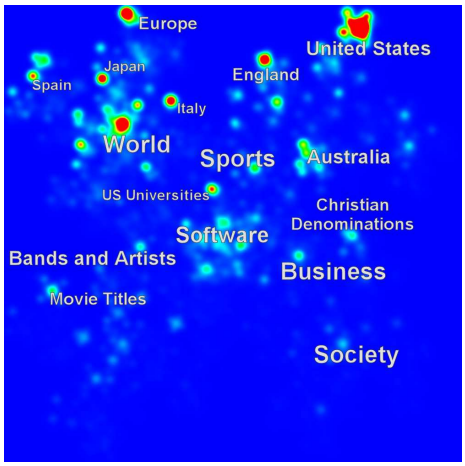


Figure 7: This GraphSplat shows areas of high density in Fig. 1. The nodes within the DMOZ hierarchy that correspond to the areas of high density have been sought out and used to annotate the GraphSplat with the font sizes reflecting the hierarchy levels. It can be seen that most of the dense spots correspond to regional subtrees.

3 Classification

When relating our technique to other space-filling techniques, we realized that they have quite different characteristics, which divide their spectrum into several categories. This is not surprising, as the term "space-filling" appears with different notions in the literature. We present a more general view on space-filling techniques that is able to encompass all the different characteristics and to adequately relate our's as well as the existing techniques with each other.

To achieve this, we propose to broaden the scope of the space-filling paradigm defined in Section 2 by introducing two basic relaxations. The first regards space-optimized, explicit node-link layouts. Apparently, these are not space-filling in the defined sense, but we found it noteworthy that actually many of the explicit layout methods can easily be transformed into equivalent implicit, and thus space-filling techniques. This is possible because many of them rely on a partitioning (top-down) or packing (bottom-up) of the available space. But instead of using the resulting areas themselves as node representations, as implicit techniques do, explicit techniques use them to put nodes (points) inside. If these techniques are modified so that instead of the points, the respective areas of the screen space are shown, one obtains an equivalent implicit technique. This principle is exemplified in Fig. 8. So, the first relaxation of the definition is to consider explicit layouts to be space-filling, too, if such a transformation from an explicit to an implicit version is possible and only the implicit version satisfies the definition from Section 2.

The second relaxation concerns the influence of screen space and tree characteristics on the space-filling property of a layout. It is introduced because some techniques are space-filling under all circumstances, while others require certain constraints to be met:

I. **Specific aspect ratio:** Some techniques require a certain aspect ratio to be truly space-filling – e.g. an aspect ratio of 1 for PieTrees [13].

II. **Certain shape:** Other techniques may cope well with different aspect ratios, but can only fill the available screen region if it is of a certain shape – e.g. a rectangular shape for Treemaps.

III. **Unlimited screen space:** As a number of techniques grow outwards from the root node, no upper bound for their final size can be fixed in beforehand – e.g. for Sunburst [17].

IV. **Particular data characteristics:** Many techniques fill out the entire available screen space only if the given tree fulfills certain properties – e.g. a uniform height for Icicle Plots [10].

So, the second relaxation is to consider a layout as space-filling, even if the conditions of the definition are only met under any of the constraints listed above. The examples for the different constraints show that this is very much in tune with the term "space-filling", as it is commonly used.

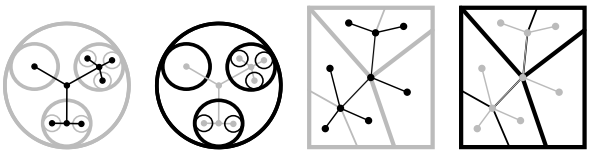
We used the above four constraints to classify the different existing space-filling techniques. Using relaxations of the space-filling definition distinguishes our classification from previous characterizations like [22], which rather added additional constraints on top of the space-filling property, e.g. being order preserving or handling structural changes smoothly, to create a "perfect" space-filling layout. With our relaxations a visualization designer who wants to use a specific space-filling technique can just look up which constraints his visualization tool must fulfill to create a truly space-filling tree representation to start with before adding even more criteria on top of that.

So in theory, a space-filling visualization technique might be constrained by any of the 16 possible combinations of the four points mentioned above. Yet, it is an interesting observation that when trying to match up techniques from the literature with the constraints above, only four of these combinations appear in practice. These four combinations are:

1. No constraints at all. Only very few techniques are able to completely fill an arbitrarily sized screen space of any desired aspect ratio and shape for any imaginable tree. Examples for these rare cases are the Voronoi Treemaps [1] and the Space-Optimized Tree Visualization [12] shown in Fig. 8c.

2. Constrained by the shape. These techniques can handle it all: arbitrary trees, different aspect ratios and an upper bound on the available screen space. Just the shape of it has to be right, which usually means "rectangular". Since rectangular screen regions are the most common in today's GUIs, this constraint is not a limiting factor in practice. Besides the standard Treemap, other examples for this category include Radial Edgeless Trees [8] and the Draw Tree Algorithm [5].

3. Constrained by aspect ratio and shape. In this case, not only the shape of the screen space is restricted, but also its aspect ratio. This group of techniques consists mainly of radial approaches like the circular Treemap [21], RINGS [18], the Balloon Drawings from Fig. 8, and the aforementioned PieTrees. Since they all base on a circular shape, an aspect ratio of 1 is a necessity. Otherwise the techniques would degenerate into elliptic shapes, which are often considered to be techniques on their own right – e.g. ellimaps [14].



(a) Explicit (b) Implicit (c) Explicit (d) Implicit

Figure 8: (a) The Balloon Drawings [11] or Bubble Tree Drawings [6] are explicit techniques by design. (b) Yet, if instead of the nodes their bounding circles are used, it becomes a circular Treemap technique. (c) The Space-Optimized Tree Visualization [12] has been published as an explicit technique as well. (d) The derived version uses the underlying polygonal space-division in a Treemap-manner to yield an equivalent implicit layout.

4. Constrained in every aspect. This category contains all techniques that pose constraints on every aspect (shape, aspect ratio, screen space, and tree characteristics) to become truly space-filling. These are basically all the techniques which grow outwards, like Sunburst or Icicle Plots. As their shape, aspect ratio and required screen space depends on the tree itself, it is a matter of the data whether these techniques really fill out the space. In the cases of Sunburst and Icicle plots, the tree must have a uniform depth to gain a complete coverage of a circular resp. rectangular space. Additionally, for Icicle Plots, the ratio of the tree's width and height determines the aspect ratio of its icicle representation.

Interestingly, as discussed in Section 2, our approach falls into neither of these four categories. Instead, so far it seems to be the only member of the following 5th category:

5. Constrained by shape, aspect ratio, and tree characteristics. This class of techniques requires a certain shape of a certain aspect ratio, which is only filled up to the last pixel, if the tree's width and/or height adhere to certain principles – e.g. being a multiple of 4. Yet, it is still possible to set an upper bound on the size, which will not be exceeded by the visual representation.

These five categories can be schematically depicted as shown in Fig. 9 to give an impression of the landscape of currently existing space-filling techniques and the gaps in between them. For example, Fig. 9 shows that the constraint I (aspect ratio) seems to be a stricter version of constraint II (shape): while there exist many techniques that require a certain shape, but can handle arbitrary aspect ratios, there is no known technique that does the opposite – is able to fill an area of arbitrary shape but needs a fixed aspect ratio. So, a fixed shape is a necessary condition for a fixed aspect ratio – at least until a technique is developed to fill this gap.

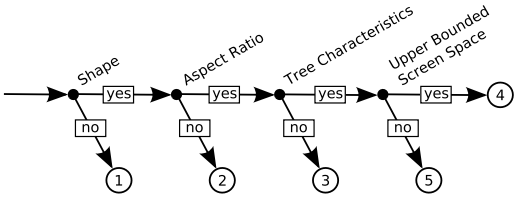


Figure 9: The five categories of space-filling techniques and what they are constrained by.

4 Evaluation

This section evaluates the basic, unrefined layout of our point-based technique. This means that we do not use any of the enhancements or layout adjustments that are described in Section 2.2: not the adaptive edge drawing nor the placement of 8 instead of 4 children at the first level. We found this just fair, as we used for instance the Space-Optimized Tree layout also in its basic and not its computationally more intensive alternative layout scheme [12].

For the evaluation, we took it as an indicator for the goodness of a layout how well it distributes the nodes over the available screen space and thus hopefully minimizes overplotting. This seemed reasonable, as overplotting is a problem of any dense, space-filling node-link-layout. To numerically quantify screen utilization and overplotting artifacts, we used the Ink-Paper-Ratio that was mentioned in Section 2, as well as *overplotted%* from [2]:

$$\text{overplotted\%} = 100 \times \frac{|\text{overplotted pixels}|}{|\text{used pixels}|}$$

Besides our point-based layout, we investigated RINGS and the Space-Optimized Tree Visualization with regard to the above measures. We chose these two techniques basically because they are explicit techniques, too, but fall into different categories of our classification – both of which being less constrained than our own layout.

To compare the different techniques, we fixed the node size to 1 pixel and used a fixed, quadratic screen space of 600×600, as RINGS and our layout require a rectangular shape with an aspect ratio of 1. This section details the comparison for an artificial test set of hierarchies and for the DMOZ example hierarchy from Fig. 1.

4.1 Evaluating the Test Set

As an optimal space-filling layout could place 360.000 1-pixel nodes on a 600×600 screen-space, we constructed three different full trees with nearly as much nodes. These trees are fully balanced, so that each non-leaf node has the same amount of children. The chosen width-depth-combinations and their overall sizes are listed in Table 1. The computed measures for the different layouts and trees are shown in Fig. 10 and some visual results in Fig. 11 and 12.

test case	depth	width	# of nodes	# of leaves thereof
A	6	8	299.593	262.144
B	7	6	335.923	279.936
C	9	4	349.525	262.144

Table 1: Sizes of the three evaluated width-depth-combinations.

Ink-Paper-Ratio: The Ink-Paper-Ratio shows the utilization of the available screen space. A higher value means a better usage of the space and is thus preferable. First of all, it is noteworthy that all three techniques behave differently with respect to increasing depth and decreasing width. While the Ink-Paper-Ratio is steadily rising for the Space-Optimized Tree layouts, it stays about the same for our point-based technique. The latter is interesting, as it can be seen in the GraphSplats of Fig. 12 that for all of the three different trees, the distribution of nodes on the screen space is changing dramatically. As for RINGS, in terms of the Ink-Paper-Ratio, it would behave similarly to our layout, if it were not for a special case in the RINGS layout for trees with exactly 6 children. This case is depicted in Fig. 11b, and it can be seen that the layout allows the sixth subtree to occupy the entire middle of the layout, which leads to the better ratio for this case. Furthermore, the left diagram in Fig. 10 shows that the maximal Ink-Paper-Ratio is still below 50% for all of the test cases, which is already quite good for an explicit layout. Also, it is not surprising that the Space-Optimized Tree layout achieved the best utilization for all of the trees as it is the one with the most adaptable of the three layouts examined. Yet, its adaptability, which tries to distribute the nodes evenly and thus to maximize

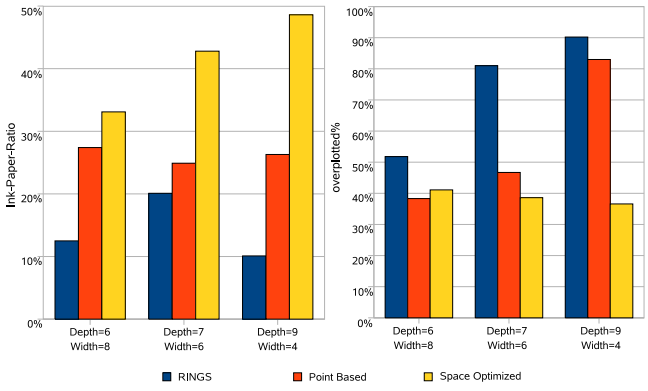
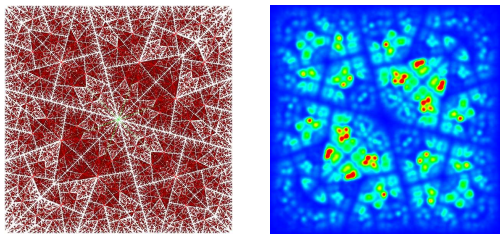
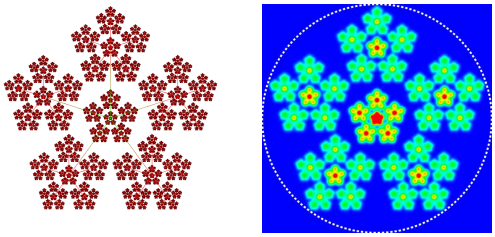


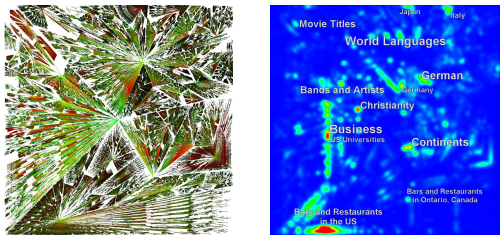
Figure 10: The Ink-Paper-Ratios and overplotted%-values for the 3 trees from Table 1.



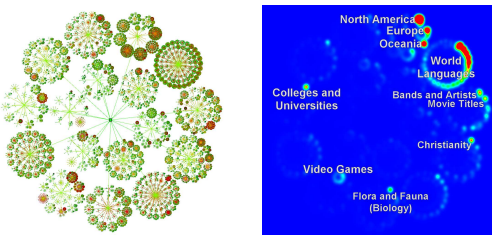
(a) Space-Optimized Layout for Test Case B



(b) RINGS Layout for Test Case B



(c) Space-Optimized Layout for the DMOZ Hierarchy



(d) RINGS Layout for the DMOZ Hierarchy

Figure 11: The Space-Optimized layout and the RINGS layout with their respective GraphSplats showing test case B and the DMOZ hierarchy from Fig. 1. The dotted circle in the GraphSplat of (b) illustrates the circular area in which the RINGS-representation is laid out, accounting for the wasted space at the bottom of the quadratic screen space.

the screen usage, leads to the effect that the overall characteristics of the tree are evened out, which makes it hardly possible to relate the density of one subtree to another. This is where techniques like the point-based layout or RINGS have their strengths. Here, tree characteristics are preserved through the layout process at the expense of a lower screen utilization, resulting in blank spaces that are not occupied. That the Ink-Paper-Ratios of RINGS are below the ones of the point-based layout is mostly due to its circular approach, which leaves a lot of whitespace by design. This is not necessarily a drawback, as it allows to clearly distinguish the individual subtrees from each other, whereas the point-based and the Space-Optimized approach achieve a tighter packing and thus a better Ink-Paper-Ratio, but at the expense of the subtrees' distinguishability.

overplotted%: The overplotted% values give the relative amount of overplotted pixels, which is in this case equivalent to visual clutter. Hence, a lower value means less clutter and is therefore desirable. It can be observed in the diagrams that the overplotted% values of our point-based and the RINGS layout are both rising with increasing depth and decreasing width. Both show also a distinct jump from about 50% to about 80%. Only this jump occurs earlier for RINGS than for the point-based approach. This can also be observed from Fig. 12b to 12c. Interestingly, the overplotted% values for the Space-Optimized Tree layout are even slightly falling with increasing depth and decreasing width. We believe this is due to the fact that this technique is very much influenced by tree width, because it partitions the available space according to the number of children. When the number of children is large, it produces many narrow partitions, which make the layout at the next level even harder. Here again, like in the Ink-Paper-Ratio diagram, our point-based technique is sandwiched in between the other two techniques. The only exception is test case A, where point-based performed slightly better than the Space-Optimized layout. This overall distribution is related to the Ink-Paper-Ratio, since the use of more pixels has an effect on the degree of overplotting.

4.2 Evaluating the DMOZ Hierarchy

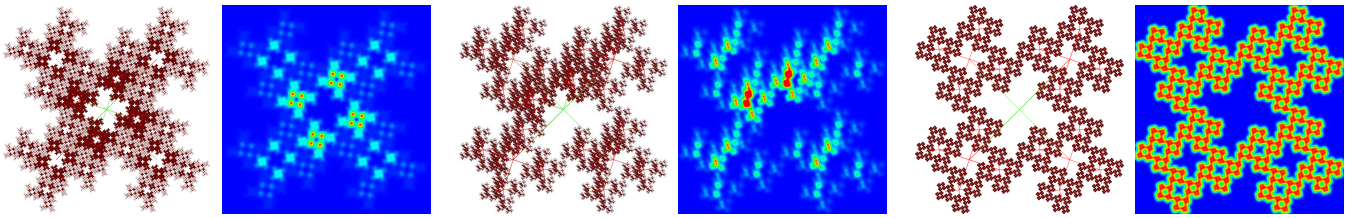
Visualizing the DMOZ hierarchy with RINGS and the Space-Optimized Tree Layout yields Fig. 11c and 11d . The numerical values for overplotted% and the Ink-Paper-Ratio reflect our findings from above, so that again our point-based layout is in between the other two. Yet, the comparison of the actual layouts hints at some interesting properties. For example, while RINGS has its advantage definitely on the aesthetical side, clutter tends to occur frequently already at higher levels of the hierarchy. This can be observed from the uniformly large font in the GraphSplat in Fig. 11d, as the hierarchy level is mapped onto the font size. The varying font sizes in the GraphSplats in Fig. 7 and Fig. 11c show a much more diverse occurrence of cluttering artifacts for the point-based and the Space-Optimized layout. So, RINGS seems to be perfect to give a nice-looking high-level overview of a large tree like the DMOZ hierarchy. Contrary to that, the other two techniques allow a first glance that reaches further downward the hierarchy, as they do not restrict themselves to small circular patterns. Yet, while the Space-Optimized layout tries to minimize clutter by prioritizing space utilization above everything else, its algorithm actually produces more clutter than even RINGS. This can be seen in Fig. 11c and is due to the already mentioned very narrow areas that are allocated in case of a wide subtree. This also produces the alignment of the dense regions that can be observed in the GraphSplat, as the midpoints of these narrow areas, which are used as roots for the individual subtrees, are all lying side by side. Of course, the point-based layout cannot prevent clutter from occurring at all. But its algorithm distributes it nicely and thus minimizes it, as it is less likely to accumulate in one region.

5 Conclusion and Future Work

We presented an overview technique for large trees that maximizes the usage of screen space and preserves tree characteristics.

Using a sampling mechanisms from the field of point-based graphics, we developed our new point-based layout technique, which fulfills the above criteria, as it makes good use of the available screen space without extensive overplotting and also conveys the basic shape and other features of the tree. As an overview technique, it allows the user to recognize the width and the depth of a tree through color coding and the distribution within the available screen space. Interaction techniques (filter, zoom, rotate) allow to selectively drill down for further exploration of the tree.

We related this new layout to existing ones on a conceptual

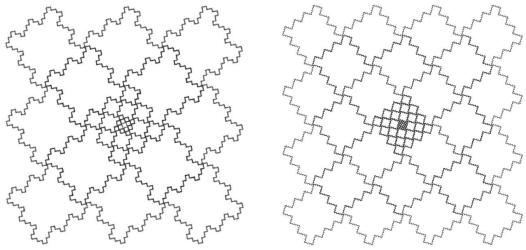


(a) Point-based Layout for Test Case A (b) Point-based Layout for Test Case B (c) Point-based Layout for Test Case C

Figure 12: The point-based layouts of the test cases and their respective GraphSplats.

level through a newly introduced classification of space-filling techniques. In addition, we compared our technique with two other space-filling techniques for different large trees. For this, we chose a numerical approach and measured screen utilization and visual cluttering. As a result, the introduced point-based layout positioned itself between the other two examined techniques, which confirms that we seem to have found a good compromise between filling the entire screen area and preserving the tree characteristics.

For the future, we are planning to extend the spectrum of point-based tree layout techniques with other possible sampling methods as shown in Fig. 13. Because they tile the screen space into more subregions, the two depicted layout schemes are expected to produce better results on wide trees than the $\sqrt{5}$-sampling that was used in this paper. Having a number of different sampling algorithms allows us to choose the one that is best fit for a given tree.



(a) $\sqrt{13}$-Sampling (b) $\sqrt{25}$-Sampling

Figure 13: Two other possible arrangements for space-filling, point-based layouts.

Once such an adjustable sampling mechanism has been investigated and included into our layout, the next important step would be the evaluation of our technique in the form of a user study. In this paper, we have already presented a numerical evaluation with respect to different tree characteristics, namely depth and width. As the comparison presented in this paper has shown, all the compared techniques have their advantages for different types of trees. But it is also the task at hand that determines whether a certain representation is useful or not for a tree of a certain size. So, the dimensions that are still missing are different hierarchy sizes and different exploration tasks. We plan to conduct this user study with large hierarchies from biological data and biological domain experts within our interdisciplinary graduate school *dIEM oSiRiS*. This will enable us to answer the important question from which tree size upwards such a dense representation as the point-based layout is appropriate, and for which kind of exploration tasks.

Acknowledgements

This work was supported in part by the DFG graduate school *dIEM oSiRiS*.

References

[1] M. Balzer and O. Deussen. Voronoi treemaps. In *Proc. of IEEE InfoVis 2005*, pages 49–56, 2005.

[2] G. Ellis and A. Dix. The plot, the clutter, the sampling and its lens: Occlusion measures for automatic clutter reduction. In *Proc. of AVI 2006*, pages 266–269, 2006.

[3] J.-D. Fekete and C. Plaisant. Interactive information visualization of a million items. In *Proc. of IEEE InfoVis 2002*, pages 117–124, 2002.

[4] A. U. Frank and S. Timpf. Multiple representations for cartographic objects in a multi-scale tree – an intelligent graphical zoom. *Computers and Graphics*, 18(6):823–829, 1994.

[5] A. Garg and A. Rusu. Straight-line drawings of general trees with linear area and arbitrary aspect ratio. In *Proc. of ICCSA 2003*, pages 876–885, 2003.

[6] S. Grivet, D. Auber, J.-P. Domenger, and G. Melancon. Bubble tree drawing algorithm. In *Proc. of ICCVG 2004*, pages 633–641, 2004.

[7] M. Gross and H. Pfister, editors. *Point-Based Graphics*. Morgan Kaufmann Publishers, 2007.

[8] J. Hao and K. Zhang. A mobile interface for hierarchical information visualization and navigation. In *Proc. of IEEE ISCE 2007*, 2007.

[9] B. Johnson and B. Shneiderman. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In *Proc. of IEEE Vis 1991*, pages 284–291, 1991.

[10] J. B. Kruskal and J. Landwehr. Icicle plots: Better displays for hierarchical clustering. *The American Statistician*, 37(2):162–168, 1983.

[11] C.-C. Lin and H.-C. Yen. On balloon drawings of rooted trees. *Journal of Graph Algorithms and Applications*, 11(2):431–452, 2007.

[12] Q. V. Nguyen and M. L. Huang. Space-optimized tree: a connection+enclosure approach for the visualization of large hierarchies. *Information Visualization*, 2(1):3–15, 2003.

[13] R. O'Donnell, A. Dix, and L. J. Ball. Exploring the pietree for representing numerical hierarchical data. In *Proc. of HCI 2006*, pages 239–254, 2006.

[14] B. Otjacques, M. Noirhomme, X. Gobert, P. Collin, and F. Feltz. Visualizing the activity of a web-based collaborative platform. In *Proc. of IEEE IV 2007*, pages 251–256, 2007.

[15] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proc. of IEEE Visual Languages 1996*, pages 336–343, 1996.

[16] M. Stamminger and G. Drettakis. Interactive sampling and rendering for complex and procedural geometry. In *Proc. of EG Workshop on Rendering Techniques 2001*, pages 151–162, 2001.

[17] J. T. Stasko and E. Zhang. Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In *Proc. of IEEE InfoVis 2000*, pages 57–65, 2000.

[18] S. T. Teoh and K.-L. Ma. Rings: A technique for visualizing large hierarchies. In *Proc. of Graph Drawing 2002*, pages 268–275, 2002.

[19] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 2nd edition, 2001.

[20] R. van Liere and W. de Leeuw. Graphsplatting: Visualizing graphs as continuous fields. *IEEE TVCG*, 9(2):206–212, 2003.

[21] W. Wang, H. Wang, G. Dai, and H. Wang. Visualization of large hierarchical data by circle packing. In *Proc. of ACM SIGCHI 2006*, pages 517–520, 2006.

[22] M. Wattenberg. A note on space-filling visualizations and space-filling curves. In *Proc. of IEEE InfoVis 2005*, pages 181–185, 2005.