Technical Appendix:
The DeepTree Layout and Rendering Engine

Florian Block, Michael S. Horn, Brenda Caldwell Phillips, Judy Diamond, E. Margaret Evans and Chia Shen,
Senior Member, IEEE

1 Introduction

This technical report provides details regarding the DeepTree layout and rendering engine, and is a supplement to [1].

2 Tree Layout Algorithm

The tree layout used by the DeepTree is shown in Fig. 2 (left). It is based on SpaceTree's [4] "continuously scaled tree" – using fixed progressive scaling of the nodes. The principle governing this layout is that all children are fully contained within the width of the parent. This "fractal" rule leads to an exponential decrease of bounding box width based on the node's level within the tree.

However, due to the size of our tree structure, which had 123 levels at its deepest point currently and will undoubtedly increase continuously, we ran into accuracy problems with the `Rect` structure on which the bounding boxes of our nodes were based. As we are continuously sub-dividing the available width of a node to accommodate its children, we are also continuously decreasing the accuracy of the floating point of our bounding box. If we assume a perfectly bifurcated tree, in which every node has exactly two children, we would exhaust our floating point accuracy after 52 levels (a double floating point allocates 52 levels to the fraction), preventing us to further subdivide space for contained nodes. Accuracy problems on the implementation levels should be a concern for all fractal algorithms (such as [2][1][1]), however, we could not find any reference to this problem in prior work.

Our solution was to implement a layout and rendering engine that is based on *relative* bounding boxes. In the following subsection, we briefly define the layout and the core components of the rendering engine. Note that low-level description is included here not only to allow a replication of our work, but also to illustrate, how feedback by our four stakeholders affected every low-level decision.

Let the tree be a set of nodes $T := \{N_S\}$ where S represents the number of nodes in the tree, a root $R \in N$, and each node N has an indexed set of children $C_N := \{c_i : i \in [0, |C_P| - 1]\}$. Let $P_N \in T$ be the parent of N, and I_N be the zero-aligned index of a node N within the children of its parent P_N, where $I_N \in [0, |C_P| - 1]$. Also, let C be a child node, P its parent, $v_N := (x_N, y_N)$ a vector defining the top left corner of a node N's relative bounding box, and w_N and h_N the width and height of a node N's relative bounding box. We can then recursively distribute the bounding box of C as follows:

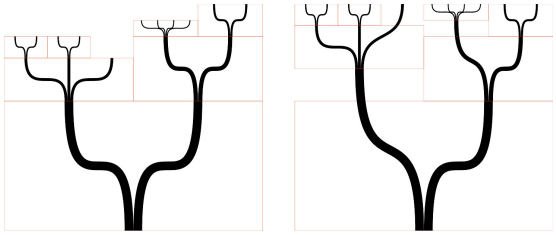


Fig. 2. Visualization of the DeepTree layout algorithm. Children are contained within the width of the parent node. (right) top aligned.

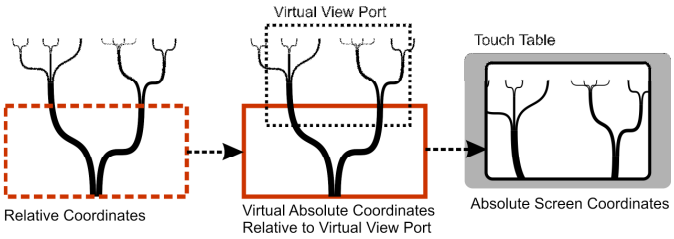
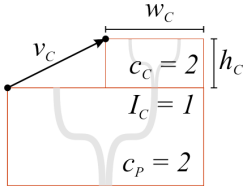


Fig. 3. Projecting relative coordinate system to absolute screen coordinate system.

$$w_C = \frac{1}{|C_P|}, \qquad h_C = \frac{w_C}{|C_C|},$$

$$v_C = \begin{pmatrix} I_C \cdot w_C \\ -h_C \end{pmatrix},$$

with $C \in T \vee C \neq R$



For every node except the root, the bounds are expressed relative to the top left corner of its parent's bounds, and as multipliers of the parent's bounds' width. This calculation has to be done top-down (from the root to the leaves), as the calculation of the child depends on the values calculated for the parent. As the dividend used to calculate w_C is always 1, the accuracy does not reduce, hence allowing unlimited depth.

The described calculation creates a distribution of node bounds as shown in Fig. 2, left. In a second pass we top-align the tree, so that every terminal node is on the same vertical level (cf. Fig. 2, right). We first calculate the distance of every node to its deepest (visually highest) nested child. Let d_N be the distance to the deepest nested child of N, D_n the set of all distances d_c, with $c \in C_N$. We can then recursively define d_N and the horizontal shift s_N as follows:

$$d_N = \begin{cases} h_N, if\ |C_N| = 0 \\ h_N + \max(D_N) \cdot h_N, if\ |C_N| > 0 \end{cases}; \quad s_N = d_N - \max(D_{P_N})$$

This calculation has to be done bottom-up, starting with the leaves, and then propagating d and s up to the root. We can now define our top-aligned bounding box with the offset $v'_N = (x'_N, y'_N)$, width w'_N and height h'_N of a node N as follows:

$$v'_N = v_{N+} \begin{pmatrix} 0 \\ s_C \end{pmatrix},\ w'_C = w_C,\ h'_C = h_C,$$

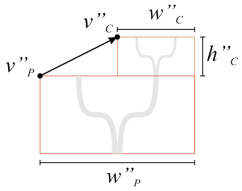
3 DeepTree Rendering Engine

In preparation for rendering we first translated our relative bounding boxes into absolute bounding boxes. Let V_R be the initial absolute offset , and W_R and H_R be the initial absolute width and height of the root R, $v''_N := (x''_N, y''_N)$ the vector defining the absolute offset of the node N's absolute bounding box, and w''_N and h''_N the width and height of a node N's absolute bounding box. We can then calculate the child's absolute bounding box based on its parent as follows:

$$w''_C = w''_P \cdot w'_C,\ h''_C = w''_C \cdot h'_C,$$

$$v''_C = \begin{pmatrix} x''_P + x'_C \cdot w''_P \\ y''_P + y'_C \cdot w''_P \end{pmatrix},$$

with $C \in T \vee C \neq R$



If we now define a virtual viewport in the same coordinate space (with offset $V := (X, Y)$, width V_W and height V_H) we can simply translate the absolute virtual bounding boxes into the screen coordinate system using standard methods. Fig. 3 illustrates the overall process. After setting the initial absolute virtual coordinates of the root (left), we calculate the absolute virtual coordinates of all visible nodes (middle) and then project the node bounds to the screen's coordinate space, and draw the nodes within their screen bounding boxes (right). At each render frame, the calculation of screen bounds is repeated, and done recursively starting from the root. We can stop the recursion when either of the two cut-off criteria are met: a) the width of a node's screen bounds has surpassed the size of a pixel; b) nodes that are horizontally outside of the current virtual viewport. Note that this optimization is not visible to the viewer, as the optimization is done outside the visible screen bounds, and below pixel-level. It also ensures a seamless transition between frames when zooming and panning.

We can seamlessly navigate through the tree by translating and/or scaling the virtual viewport at each frame, however, we apply two constraints. First, more detail can only be found in the very top of the tree – the canopy. Thus we always scale the viewport around the canopy, which causes the canopy to remain on the same vertical screen coordinate. Secondly, panning of the viewport is limited to the x-axis. These constraint had several benefits for us: 1) a portion of the canopy of the tree – the space in the tree where all the "life" is – would be always visible and at a consistent screen location, making it easy for visitors to keep it in focus, and use it as navigational aid; 2) it enabled a simple input gesture for manual navigation.

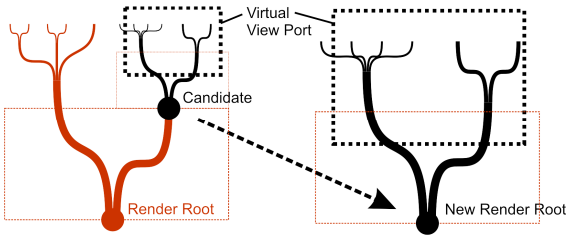


Fig. 4. Root transfer.

3.1 Root transfer

As we are zooming into the tree, and thus revealing nodes that are further and further away from R, we would again run into accuracy problem while recursively calculating the absolute bounds. Let $\overline{R}$ be the *render root* from which the recursive process starts on every frame, $\overline{V}$ be the absolute offset, $\overline{W}$ and $\overline{H}$ the absolute width and height of $\overline{R}$'s bounding box, and initially be $\overline{R} = R$. As we are continuously zooming into the tree structure, many nodes that were previously visible will move out of the viewport (cf. Fig. 4, left). At each render pass, out of all visible nodes, let R' be the visible node with the minimum absolute width w''_N, and $x''_N \leq X \wedge X + V_W \leq x''_N + w''_N$ (containing the whole viewport). Only descendants of R' can be visible. If R' is different from $\overline{R}$ at the end of each render pass, we call TransferRoot($R', v''_{R'}, w''_{R'}, h''_{R'}$):

```
void function TransferRoot(Node r, Vector v, float w, float h)
{
    R̄ = r; W̄ = w; H̄ = h;
    V̄ = (v.x − X   v.y − Y); X = 0; Y = 0;
    float reset = 1000 / V_W;
    W̄ *= reset; H̄ *= reset; V_W *= reset; V_H *= reset;
}
```

TransferRoot first sets *r* as the new $\overline{R}$ and *w* and *h* as its width and height (cf. Fig. 4, right). Then, it translates both viewport and $\overline{R}$'s new absolute offset, so that the viewport's offset is zero. Thirdly, it scales the width and height of both viewport as well as $\overline{W}$ and $\overline{H}$ so that the viewport gains a width of 1000. While this value is arbitrary, this step resets the accuracy of the underlying floating point values. This method maintains the accuracy of the size of $\overline{R}$, and the calculation of the currently visible children, respectively.

An equivalent process has to be started *before* every render pass. Based on the current $\overline{R}$, we can calculate the bounds of the parent by reversing the introduced calculation:

$$w''_P = \frac{w''_C}{w'_C},\ h''_P = w''_C \cdot h'_C,\quad v''_P = \begin{pmatrix} x''_C - x'_C \cdot w''_P \\ y''_C - y'_C \cdot w''_P \end{pmatrix}$$

We can then call TransferRoot($P_{\overline{R}}, w''_P, v''_P, w''_P,\ h''_P$). Note that a root transfer is not visible to the viewer, as it simply recalibrates the viewport around a new visible root, which stays in a fixed screen location before and after the transfer. This was essential to support G2. The described rendering engine allows us to render and seamlessly navigate trees with unlimited depth and size.

References

[1] Block, F., Horn, M. S., Phillips, B. C., Diamon, J., Evans, M. E., Shen, C. The DeepTree Exhibit: Visualizing the Tree of Life to Facilitate Informal Learning. Under submission to InfoVis 2012.

[2] Lamping, J. and Rao, R. The hyperbolic browser: A focus+ context technique for visualizing large hierarchies. *Journal of Visual Languages and Computing*, 7(1), pp. 33-55, 1996.

[3] Neumann, P., Carpendale, S., and Agarawala, A. Phyllotrees: Phyllotactic patterns for tree layout. In *Proc. EuroVis'06*, pp. 59-66, 2006.

[4] Plaisant, C.; Grosjean, J.; Bederson, B.B. SpaceTree: supporting exploration in large node link tree, design evolution and empirical evaluation. In *Proc. InfoVis 2002*, pp. 57- 64, 2002.

[5] Teoh, S.T. and Ma, K.-L.. RINGS: A technique for visualizing large hierarchies. In Proc. GD'02, pp. 51-73. *Lecture Notes in Computer Science*, Springer, 2002.