# Simple Algorithms for Orthogonal Upward Drawings of Binary and Ternary Trees

Sung Kwon KIM*
Department of Computer Science
Kyungsung University
Pusan 608-736
Korea
ksk@csd.kyungsung.ac.kr
fax: +82-51-628-6059

## 1 Introduction

A drawing of a tree $T$ maps each vertex of $T$ to a distinct point in the plane and each edge $(u, v)$ of $T$ to a chain of line segments with endpoints $u$ and $v$. A *planar orthogonal upward grid* drawing of a tree is a drawing satisfying the following conditions:

- *(Planar)* No two edges intersect.
- *(Orthogonal)* Each constituent line segment of an edge is either horizontal or vertical.
- *(Upward)* Each edge is vertically monotone; so no vertex is allowed to locate above its parent, though they may lie on the same horizontal line.
- *(Grid)* Each vertex or bend has integer coordinate.

Throughout the paper, all drawings are *planar orthogonal upward grid drawings* unless otherwise specified, so *planar orthogonal upward grid* will be omitted hereafter.

We summarize our results:

1. For a binary tree with $n$ vertices, one can compute a drawing with $O(n \log\log n)$ area, $O(n/\log n)$ bends and at most four bends per edge. (see Section 2.) A drawing with the same area and number of bends can also be computed by the algorithm of [3], but an edge may have $\Theta(\log n)$ bends.
2. For a ternary tree with $n$ vertices, one can compute a drawing with area $O(n \log n)$, and this is optimal. (see Section 3.)

## 2 Drawings of binary trees with four bends per edge

For a binary tree with $n$ vertices, Garg, Goodrich, and Tamassia [3] compute a drawing with $O(\log n)$ width, $O(n \log\log n/\log n)$ height, $O(n \log\log n)$ area, and $O(n/\log n)$ bends. In this section we will present a simple drawing algorithm with the same bounds. In both the drawings of Garg, Goodrich, and Tamassia, and our drawings, only $O(n/\log n)$ edges have bends. Difference is that in our drawings an edge has at most four bends, while in their drawings an edge may have $\Theta(\log n)$ bends.

Our algorithm works in a similar way as the one of Garg, Goodrich, and Tamassia [3]. Given a binary tree $T$ with $n$ vertices, we first partition $T$ into $O(n/\log n)$ partial trees,[1] each of

*Supported in part by Korea Science and Engineering Foundation, No. 94-1400-04-01-3.

[1] A partial tree of $T$ is a subgraph of $T$, and a sub-

size $O(\log n)$, by deleting $O(n/\log n)$ edges. The partial trees are drawn as in [3], and their drawings are vertically stacked. The deleted edges are drawn around the stack of drawings.

For partitioning, Garg, Goodrich, and Tamassia use separators of a binary tree $T$. Find a separator of $T$ and remove it to partition $T$ into two partial trees. (An edge of $T$ is called a *separator* if its removal leaves two partial trees, each having at least $\frac{1}{3}n$ and at most $\frac{2}{3}n$ vertices.) Recursively repeat this "find and remove separators" procedure $O(n/\log n)$ times until all the partial trees of $T$ left have size $\Theta(\log n)$.

We use a different partitioning method, which is a variation of the one used for parallel tree contraction by Gazit, Miller, and Teng [4], [5, section 3.3.5]. The partitioning method was first used by Fujiwara *et al.* [2].

1. For each vertex $v$ of $T$, compute $size(v)$, the number of descendants of $v$ in $T$, and
2. delete the edge $(v, parent(v))$ if $\lfloor \frac{size(v)}{\log n} \rfloor > \lfloor \frac{size(u)}{\log n} \rfloor$, and $\lfloor \frac{size(v)}{\log n} \rfloor > \lfloor \frac{size(w)}{\log n} \rfloor$, where $u$ and $w$ are the children of $v$.

It is easy to see that this deletes $O(n/\log n)$ edges and leaves $O(n/\log n)$ partial trees, each of size $O(\log n)$. We call these partial trees *fragments*. Each fragment has at most $2\log n$ vertices, and may have only one vertex. A fragment is *trivial* if it has only one vertex.

A *fragment tree* $FT$ of $T$ is a tree whose vertices are the fragments obtained above and whose edges are the deleted edges. Then $FT$ is a binary tree. This is true because the least common ancestor edge of any two deleted edges is also deleted. So, each fragment $F$ is associated with at most three deleted edges, one from its parent fragment (if exists), and two to its two child fragments (if exist). The root of $F$ is adjacent to the deleted edge from the parent fragment of $F$. The vertices of $F$ that are adjacent to the deleted edges to its child fragments are called the *connection* vertices. A trivial fragment always has two child fragments (easy to prove this); so its only vertex is not only its root but also its connection vertex. If a non-trivial fragment has two child fragments, then it has two distinct connection vertices.

Consider a binary tree $T$ with $m$ vertices. A planar *straight-line orthogonal* upward grid drawing of $T$ with width $O(m)$ and height $O(\log m)$ can be constructed [1, 3]. In this drawing, each edge is either a horizontal or a vertical line segment. A binary tree $T$ is *left-heavy* if for each vertex $v$ of $T$ with left child $u$ and right child $w$, $size(u) \geq size(w)$. A *right-heavy* binary tree is similarly defined. Assume that $T$ is left-heavy. If it is not, transform $T$ into a left-heavy tree.

1. Traverse the vertices of $T$ in reverse preorder, i.e., visit the root, then its right subtree and then its left subtree.
2. Initially, visit the root and set $x(root) = y(root) = 0$, where $x(v)$ and $y(v)$ are the $x$- and $y$-coordinates of vertex $v$, respectively, in the output drawing.
3. Visit vertex $v$ with parent $u$.
   (a) If $v$ is the right child of $u$, then set $x(v) = x(u)$ and $y(v) = y(u) - 1$.
   (b) Otherwise, set $x(v) = x(w) - 1$ and $y(v) = y(u)$, where $w$ is the vertex visited immediately before $v$

See Figure 1 for a left-heavy binary tree and its output drawing by the above algorithm.

Let $\Delta_T$ be the drawing of $T$ output by the above algorithm. The root of $T$ is at the upper-right corner of $\Delta_T$ and has coordinate $(0, 0)$. The $x$- and $y$-coordinates of the vertices in $\Delta_T$ are non-positive. Each *right edge* (an edge between a vertex and its right child) is vertical and of length one. Each *left edge* (an edge between a vertex and its left child) is horizontal and of length $\geq 1$.

A binary tree is *almost left-heavy* if all but exactly one vertex satisfy left-heaviness, i.e., if there exists exactly one vertex whose right subtree is heavier than its left subtree. Note that

tree of $T$ is a partial tree of $T$ rooted at a vertex $v$ and including all descendants of $v$.

$\Delta_T$ for an almost left-heavy $T$ still has width $O(m)$ and height $O(\log m)$.

An important property of $\Delta_T$ for a left-heavy or almost left-heavy $T$ is that every non-root vertex of degree one or two is not *obstructed*. That is, a downward ray from a non-root vertex $v$ of degree one or two in $\Delta_T$ does not intersect $\Delta_T$ except at $v$. Since the connection vertices of $T$ are of degree one or two, they are not obstructed. Another important property of $\Delta_T$ is that the non-root vertices of degree one or two appear in $\Delta_T$ from right to left according to the reverse preorder sequence of the vertices of $T$.

Transform each fragment $F$ of $FT$ into a left-heavy fragment, and obtain $\Delta_F$ for each (left-heavy) fragment $F$. Transform $FT$ into a right-heavy tree. (Right-heaviness here is in terms of the number of fragments.) Consider a fragment $F$ with two child fragments $F_1$ and $F_2$. Assume that $F_1$ (resp., $F_2$) is the left (resp., right) child fragment of $F$ in (right-heavy) $FT$. Let $c_i$ be the connection vertex of $F$ that is adjacent to $F_i$ for $i = 1, 2$. If $c_1 = c_2$ (in this case $F$ is trivial) or $c_1$ is to the left of $c_2$ (by comparing their $x$-coordinates), nothing is done. Otherwise, locate the least common ancestor $c$ of $c_1$ and $c_2$ in $F$ and switch the children of $c$. Then $F$ becomes almost left-heavy for $c$ is the only vertex that violates left-heaviness. In $\Delta_F$ for this almost left-heavy $F$, $c_1$ is to the left of $c_2$. So, we will assume that in each $\Delta_F$, the connection vertex of $F$ to its left child fragment is to the left of the connection vertex to its right child fragment. In Figure 2(b), a right-heavy fragment tree of the fragment tree in Figure 2(a) is shown. Each fragment $F$ is drawn as its $\Delta_F$.

Vertically stack the $\Delta_F$'s one above the other, sorted from top to bottom according to the preorder sequence of the fragments $F$ of (right-heavy) $FT$. See Figure 3. The $\Delta_F$'s are right-justified so that the roots of $\Delta_F$'s lie on the same vertical line. Note that the stack of $\Delta_F$'s has width $O(\log n)$ and height $O(n \log\log n / \log n)$ as each $\Delta_F$ has width $O(\log n)$ and height $O(\log\log n)$, and there are $O(n/\log n)$ $\Delta_F$'s stacked vertically.

The deleted edges will be drawn between $\Delta_F$'s and on the right of the stack as in Figure 3. Let $k$ be the number of fragments in $FT$, i.e., the number of $\Delta_F$'s in the stack. Then, $k = O(n/\log n)$ and there are at most $2k$ deleted edges. We will show that $2(k-1)$ extra horizontal tracks (two per between every consecutive $\Delta_F$'s) and $\lceil \log k \rceil$ extra vertical tracks on the right of the stack are sufficient to draw these $2k$ deleted edges. Our final drawing will have width $O(\log n) + \lceil \log k \rceil = O(\log n) + \log(\frac{n}{\log n}) = O(\log n)$ and height $O(n \log\log n / \log n) + 2(k-1) = O(n \log\log n / \log n) + O(n/\log n) = O(n \log\log n / \log n)$.

Note that a fragment is immediately above its left child fragment in the stack as the fragments are sorted according to their preorder sequence. Figure 4 shows how the deleted edges from the connection vertices of a fragment $F$ to its left and right child fragments $L$ and $R$ are drawn.

(a) $F$ is a trivial fragment. Then, $F$ has two child fragments. Let $v$ be the only vertex of the fragment. Its left edge is drawn as a vertical segment for the root of $L$ is directly below $v$. For the right edge $(v, r)$, starting at $v$, go rightward to vertical track $i$ ($i$ will be determined later), go downward along track $i$ to the horizontal track just above $R$, go leftward along the track to the point just above $r$ and go downward to $r$.

(b) $F$ is a non-trivial fragment and has two child fragments. For the left edge $(c_1, \ell)$, starting at connection vertex $c_1$, go downward to the *bottom* horizontal track (remember that there are two extra horizontal tracks between two consecutive $\Delta_F$'s), go rightward along the track to the point just above $\ell$ to reach downward to $\ell$. For the right edge $(c_2, r)$, starting at $c_2$, go downward to the *top* horizontal track, go rightward to vertical track $i$, and the remaining part is the same as the right edge in (a).

(c) $F$ is a non-trivial fragment and has only one child fragment. The child fragment is its right child fragment. The drawing of this right edge is similar to that of the left edge in (b). If the connection vertex of $F$ is at the lower-right corner of $\Delta_F$, then a vertical segment is sufficient.

As mentioned before, there are $\lceil \log k \rceil$ extra

vertical tracks on the right of the stack, numbered $1, 2, \ldots, \lceil \log k \rceil$ from right to left. Consider the right-heavy $FT$. Only right edges of $FT$ use extra vertical tracks. We will assign a track number to each right edge of $FT$ as follows:

1. Find the rightmost path $P$ of $FT$ and assign track 1 to each edge of $P$.
2. Removing the vertices and edges of $P$ from $FT$ leaves several subtrees of $FT$. Find the rightmost path of each of these subtrees and assign track 2 to each edge of these rightmost paths.
3. Repeat this "find and remove rightmost paths" procedure until all right edges are removed.

Because $FT$ is right-heavy and of size $k$, the maximum integer assigned is at most $\lceil \log k \rceil$. We say that a right edge of $FT$ *spans* a $\Delta_F$ if in the stack of $\Delta_F$'s one endpoint of the edge is above $\Delta_F$ and the other is below $\Delta_F$. Since the $\Delta_F$'s in the stack are sorted according to the preorder sequence of the fragments of $FT$, it is easy to see that no two right edges spanning the same $\Delta_F$ are assigned the same track number.

In our drawings only $O(n/\log n)$ edges have bends and each bended edge has at most four bends, so the number of bends is $O(n/\log n)$.

# 3 Drawings of ternary trees with optimal area

We will first show that there is a ternary tree $T(n)$ with $n$ vertices which requires $\Omega(n \log n)$ area in any drawing. (See Figure 5.) $T(n)$ has a chain with $n/4$ vertices. Each vertex $v$ of the chain has left and right children, which are leaves, and the middle child of $v$, except the bottommost one, is the vertex just below $v$ in the chain. The middle subtree of the bottommost vertex of the chain is a complete binary tree with $n/4$ vertices. In any drawing of $T(n)$, the complete binary tree requires $\Omega(\log n)$ width (as shown in [3]), and the chain and the leaves attached to it require $\Omega(n)$ height (easy to prove this). Thus any drawing of $T(n)$ requires $\Omega(n \log n)$ area.

An $O(n \log n)$ area algorithm for drawing a ternary tree $T$ with $n$ vertices will be given. (Our algorithm works in a similar way as the one for order-preserving upward polyline drawings of bounded-degree trees in [3].) Our drawing is *order-preserving*, i.e., in our drawing the left edge of a vertex of $T$ is to the left of its middle edge, which is in turn to the left of its right edge. Let $v$ be the root of $T$. Let $T_1, T_2$, and $T_3$ be the left, middle, and right subtrees of $v$, respectively. Recursively draw subtrees $T_i$ for $i = 1, 2, 3$. Vertically stack the drawings of the subtrees as in Figure 6 so that the subtree with the most vertices is on the bottom and the other two subtrees are in the order of their indices. Place $v$ on top of the stack of the drawings, and draw the edges between $v$ and its children in order-preserving way. Figure 6(b), (c), and (d) show how to draw the edges between $v$ and its children in the cases in which $T_3, T_2$, and $T_1$, respectively, have the most vertices among the subtrees. In case $v$ has less than three children, a similar drawing can be done.

Let $H(n)$ and $W(n)$ be the height and width of the drawing of a ternary tree with $n$ vertices output by the algorithm above. We have

$$H(n) \leq H(n_1) + H(n_2) + H(n_3) + 6,$$

where $n_i$ is the number of vertices of $T_i$ for $i = 1, 2, 3$. At most six extra horizontal tracks are needed to draw between $v$ and its children in Figure 6(b-d). Since $H(1) = 1$ and $n_1 + n_2 + n_3 = n - 1$, $H(n) = \Theta(n)$.

Let $n' = \max\{n_1, n_2, n_3\}$ and $n''$ be the second largest in $\{n_1, n_2, n_3\}$. Then,

$$W(n) \leq \max\{W(n'), W(n'') + 3\},$$

as at most three extra vertical tracks are needed to draw the edges between $v$ and its children in Figure 6(b-d). Since $n' \leq n - 1$ and $n'' \leq \frac{1}{2}(n-1)$,

$$W(n) \leq \max\{W(n-1), W((n-1)/2) + 3\}.$$

Since $W(1) = 1$, an easy induction proves $W(n) \leq \log n + 3$ for all $n \geq 1$.

Note that our drawing algorithm for ternary trees can be used to draw binary trees in order-preserving way to match the lower bound $\Omega(n \log n)$ of the order-preserving drawing of binary trees [3].

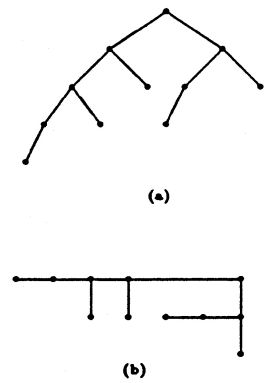


Figure 1:

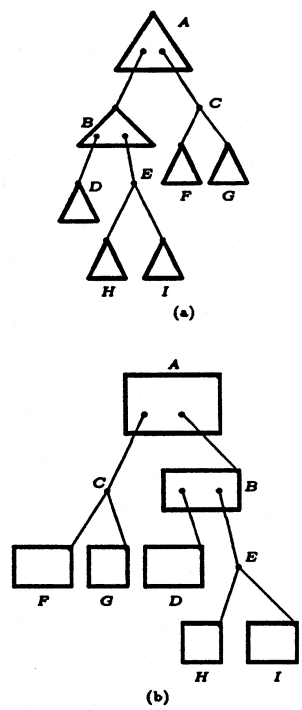


Figure 2:

## References


[1] P. Crescenzi, G. Di Battista and A. Piperno, A note on optimal area algorithms for upward drawings of binary trees, *Computational Geometry: Theory and Applications* **2**(1992) 187–200.

[2] A. Fujiwara, W. Chen, T. Masuzawa and N. Tokura, A cost optimal parallel algorithm for computing a balanced decomposition tree of a binary tree, *Technical Report of IEICE*, **COMP93-19** (1993) 93–100 (in Japanese).

[3] A. Garg, M.T. Goodrich, and R. Tamassia, Planar upward tree drawings with optimal area, Manuscript, 1995. An extended abstract "Area-efficient upward tree drawings" appeared in *Proc. 9th Symposium on Computational Geometry*, pp. 359–368, 1993.

[4] H. Gazit, G.L. Miller, and S-H Teng, Optimal tree contraction in an EREW model, in *Concurrent Computations: Algorithms, Architecture and Technology*, eds. S.K. Tewksbury, B.W. Dickinson, and S.C. Schwartz (Plenum Press, New York, 1988), pp. 139–156.

[5] M. Reid-Miller, G.L. Miller, and F. Modugno, List ranking and parallel tree contraction, in *Synthesis of Parallel Algorithms*, ed. J.H. Reif (Morgan Kaufmann, San Mateo, California, 1993) Chapter 3.

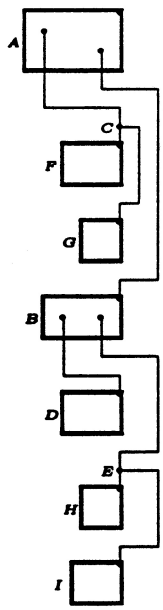


Figure 3:

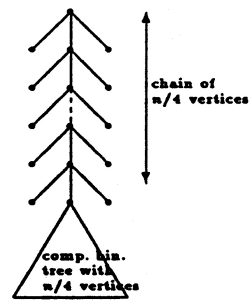


Figure 5:

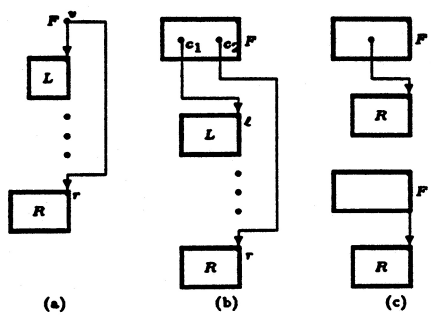


Figure 4:

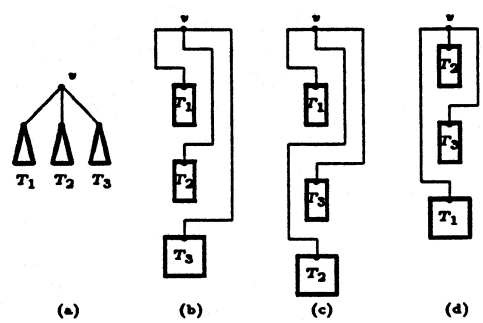


Figure 6: