Bubble Tree Drawing Algorithm

Sébastien Grivet, David Auber, Jean-Philippe Domenger, Guy Melançon

Bubble Tree Drawing Algorithm

S. Grivet[1], D. Auber[1], J.P. Domenger[1] and G. Melançon[2]

[1]LaBRI-Université Bordeaux 1, 351 Cours de la Libération, 33405 Talence, France
[2]LIRMM Montepellier, France
email:auber/grivet/domenger@labri.fr/guy.melancon@lirmm.fr
fax:+33 5 56 84 66 69

Abstract. In this paper, we present an algorithm, called Bubble Tree, for the drawing of general rooted trees. A large variety of algorithms already exists in this field. However, the goal of this algorithm is to obtain a better drawing which make a trade off between the angular resolution and the length of the edges. We show that the Bubble Tree drawing algorithm provides a planar drawing with at most one bend per edge in linear running time. We compare the aesthetic criteria of this algorithm with two well-known algorithms on a data-set coming from a real application.

1 Introduction

Hierarchical representations of information still remain central in Information Visualization. Their success mainly resides in the wide spectrum of applications for which they are relevant. Some applications focus on hierarchical data, making the use of tree layouts an obvious choice. For instance, tree representations for the visual analysis of file systems [12] or phylogenies [1] are mandatory to reflect the structure of the data under study. Tree representations can still be relevant when the data is not hierarchical but consist in a general network or graph. In most cases, a tree is extracted from the network following an adequate search of the nodes and links. This approach makes sense when the task conducted by the user requires to select an entry point in the network, for instance, although the user is offered a hierarchical view of non hierarchical data. The display of a tree then makes it possible for the user to change its focus of interest, assuming this focus is consistently positioned at the center of the screen. The visual information supported by the layout can also be complemented by visual cues such as node size and labels.

Tree layout algorithms more or less belong to two distinct categories. The first category corresponds to the so-called hierarchical drawing of trees originally proposed by Reingold and Tilford [13] (extended by Walker [17]) and reconsidered recently by Buchheim et al [6]. Even if these algorithms do not perform well in terms of aspect ratios, their interest mainly resides in their ability to deal with varying node size. Another interesting feature is the possibility to map the layout into a radial representation, through a simple transformation more or less sending the bottom line of a top-down drawing to a circle [8]. A clear advantage

of such a representation is to allow direct comparison of the ancestor-descendant distance between nodes. However, this type of information is not sufficient when performing visual information retrieval or data mining.

The second category of algorithms differs from the first one in that the focus is not on the hierarchical structure of the information, but rather on scalability and on the possibility of displaying large amounts of data. Tree maps, for instance, do not intuitively reflect the hierarchical structure of the information [5, 15]. They are mainly used in contexts where the user needs to directly access attribute values and changes. Cone trees introduced in the pioneer work of Robertson [14] suggested the use of 3D for displaying and navigating large hierarchies. Several authors have later improved this technique [7, 10, 16].

In this paper we present some basic terminology about tree and aesthetic criteria. Then, we describe precisely the two principle stages of our algorithm. Subsequently, after a short discussion about the time complexity of this algorithm, we present a comparative study of the distribution of two well-known aesthetic criteria obtained by our and two other algorithms. Finally, we conclude the paper with several drawings of an entire Linux file system containing about 270.000 files.

2 Preliminaries

We define a (rooted) tree as a directed acyclic graph with a single source, called the *root* of the tree, such that there is a unique directed path from the *root* to any other node. Each node a on the path from the root to a node n is called an ancestor of n. For ease of read in the following we note $ancestor(n)$ the unique nearest ancestor of n. This function is not defined when n is the *root*. Each node d, such that it exists a directed path from n to d, is called a descendant of n. We note $outadj(n)$ (or children of n) the set of the nearest descendant of n. The degree of a node n, denoted as $deg(n)$, is the number of neighbors of a node n. We note n_i the i-th child of the node n.

Results from the graph drawing community show that if one wants to obtain a efficient drawing he/she needs to make a trade off between several aesthetic criteria [4]. Here is the definition of some aesthetic criteria that our drawing algorithm takes into account.

- Crossing number: The edges should not cross each others.
- Number of bends : the polyline used to draw an edge should have the least possible bends.
- Angular resolution : the minimal angle between two adjacent edges of a node n should be nearest to $\frac{2 \cdot \pi}{deg(n)}$.
- The drawing of a subtree does not depend on its position in the tree, isomorphic subtrees should be drawn identically up to a translation and a rotation.
- The order of children of a node should be respected in the final drawing.

3 Algorithm

The algorithm that we have set-up is recursive as the so-called Reingold and Tilford's algorithm [13]. It uses a depth first search traversal in order to draw the tree. As for the Reingold and Tilford's algorithm, the linear running time of the algorithm is achieved by using relative position for each nodes and by delaying the computation of node position in a second phase of the algorithm. The idea of the algorithm is to use enclosing circles (instead of contour in the R&T's algorithm) to represent space needed to draw a sub-tree.

In the following we will describe precisely the two stages of our algorithm. The first one is the computation of the position of the enclosing circle of each children's sub-tree of a node relatively to itself. The second one is the coordinates assignment that prevents to have crossings in the final drawing.

3.1 Computation of the relative positions

We use a suffix depth first search procedure to assigned the relative position γ_n of the center of the enclosing circle relative to $ancestor(n)$. If one considers that each sub-tree induced by a children of a node n has already been drawn and that one has got an enclosing circle for each sub-tree drawing, the first part of our algorithm consists in placing each enclosing circle on the plan and to prevent overlapping between them.

Enclosing circle location In order to enhance the angular resolution in the final drawing, the idea of this algorithm is to place each enclosing circle around the node n. This operation can be done by determining an angular sector θ_i for each enclosing circle C_i. If the enclosing circles are placed inside their respective angular sector it is straightforward that there is no overlapping.

Let $r_1, \ldots, r_{|outadj(n)|}$ be the radius of each enclosing circle. The first approach is to assign an angular sector θ_i proportionally to r_i such that

$$\sum_{i=1}^{|outadj(n)|} \theta_i = 2\pi.$$

However, one can attribute an angular sector larger than π which is too big to place a circle in. If such case arise, we give an angular sector equal to π to the bigger, and then we assign an angular sector to the others proportionally to their radius such that their sum is π. The total sum remains 2π.

Then we place each enclosing circle such that it is tangent internally to its respective angular sector. If the angular sector is too big, the enclosing circle and the node n could overlap. This could be avoided by computing correctly the distance δ_i from n to the center of an enclosing circle, using the following formula.

$$\delta_i = max(size(n) + r_i, \frac{r_i}{sin(\theta_i/2)})$$

Using δ_i we place the center of the enclosing circle on the interior angle bisector of θ_i. Each coordinate γ_{n_i} is computed relatively to the position of n.

$$\gamma_{n_i} = \begin{cases} x_i = \delta_i \; cos((\sum_{j=1}^{i} \theta_j) - \theta_i/2) \\ y_i = \delta_i \; sin((\sum_{j=1}^{i} \theta_j) - \theta_i/2) \end{cases}$$

This coordinates will be used in the second stage of the algorithm in order to compute the final drawing. The figure 1 summarizes all this part.

One can see that to fixing the angular sector to π when it is too big let a lot of space unused. We implement a $O(n \log(n))$ algorithm to solve this problem. We start with a angular resolution Θ equal to 2π, and a global radius R equal to $\sum_{i}^{|outadj(n)|} r_i$. We treat each circle in the decreasing order of their radius. If the sector calculated proportionally to the radius ($\theta_i = \Theta \cdot r_i/R$) is bigger than the maximum we can allocate for this circle ($\theta_i^{max} = 2 \arcsin(r_i/(r_i + size(n)))$), then we fix the sector to its maximum and we decrease the angular resolution Θ according to the angle of the sector θ_i^{max}, and the global radius R to the radius of the circle r_i. If all the sectors we set are maximum, we use the remaining angular resolution Θ to space the sectors themselves. According to our tests the difference between the two solutions is not perceptible for normal trees.

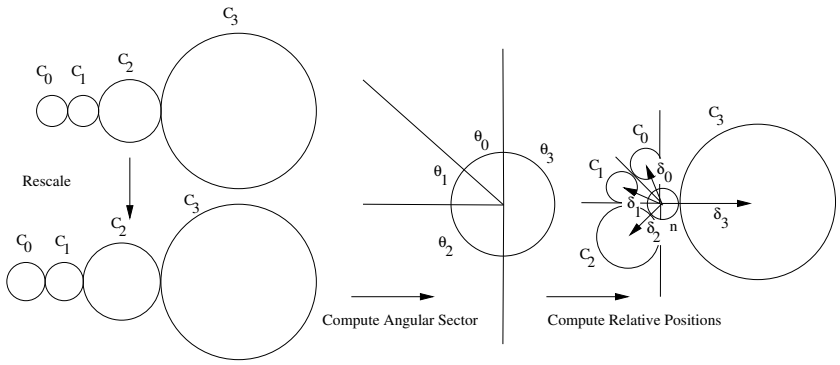

Fig. 1. Computation of locations.

Enclosing circle calculation Now, we need to find the smallest enclosing circle of the set of the enclosing circles we have placed. A detailed caparison between different methodologies is discussed in [9]. We use the incremental randomized algorithm proposed by Welzl [18] that gives the optimal solution in a linear average running time.

In [18] one needs to calculate the smallest enclosing circle of two circles (noted $EC_2(C_1, C_2)$), and of three circles (noted $EC_3(C_1, C_2, C_3)$). The computation of $EC2$ is obvious.

Let c_n (resp. r_n) be the center (resp. radius) of the circle C_n. Let c_e (resp. r_e) be the center (resp. radius) of the smallest enclosing circle C_e. The computation of $EC3$ can be done by solving the following system of equations:

$$\begin{cases} |c_1 c_e| = (r_e - r_1)^2 \\ |c_2 c_e| = (r_e - r_2)^2 \\ |c_3 c_e| = (r_e - r_3)^2 \end{cases}$$

The algorithm $EC(S, B)$ computes incrementally the enclosing circle of a set of circles S according to a set, noted B, of boundary circles of S. The definition of $EC(S, B)$ is the following:

S \ B	$\emptyset$	$\{b_1\}$	$\{b_1, b_2\}$
$\emptyset$	$\emptyset$	$\{b_1\}$	$\{EC_2(b_1, b_2)\}$
$S' \cup \{c\}$	$E = EC(S', \emptyset)$ if $c \in E$ then E else $EC(S', \{c\})$	$E = EC(S', \{b_1\})$ if $c \in E$ then E else $EC(S', \{b_1, c\})$	$E = EC(S', \{b_1, b_2\})$ if $c \in E$ then E else $\{EC_3(b_1, b_2, c)\}$

The computation of the smallest enclosing circle of a set S corresponds to calculate $EC(S, \emptyset)$. The selection of the circle c in $S' \cup \{c\}$ needs to be done randomly to ensure to have an average polynomial time [18]. To apply the optimizations proposed in [18], we use a double-ended queue [11] to store S. This queue is initialized randomly at the beginning of the algorithm. We always dequeue the selected circle c at the end of the queue. When the recursive procedure is finished, if $c \in E$, we enqueue c at the end, else at the beginning. Thus the element of S have only been reordered. Placing the circle which is not in E at the beginning of the queue optimize the order for the next calls.

If one wants a linear algorithm, one can use some heuristics in order to approximate the smallest enclosing circle. For instance, one can start with the circumcircle of the bounding box of the set of circles and then merge incrementally all the circles, using EC_2 as described in [3]. Another heuristic determines the center of the enclosing circle by calculating the barycenter of the center of the circles, weighted by the square of their radius. The radius of the enclosing circle is the maximum distance between its center and the center of the other circles augmented by their respective radius.

Our experimentations show that in average computing the smallest enclosing circle is 7% better than those heuristics. Nevertheless, the experimentation have shown that the drawings are almost the same when we use the smallest enclosing circle or one of the heuristics presented above.

Bend location To reserve an angular sector needed for the connection of a node n with its ancestor, we add a dummy enclosing circle C' during the enclosing circle placement process. The position of the bend β_n is the intersection between the enclosing circle C_n and the line containing the center of C' and n.

3.2 Coordinate assignment

After applying the previous algorithm we obtain for each node n an enclosing circle denoted C_n. Each center of C_n have a position γ_n relative to $ancestor(n)$. Thus we can compute the position δ_n of the center of C_n relative to the center of $C_{ancestor(n)}$. Each node n have a position, ζ_n, and a bend β_n position, relative to the center of C_n.

The final position of a node is obtained by using a prefix order traversal. After placing the root of the tree in the center of the view (coordinates (0,0)), we call a recursive function, detailed in the algorithm 1, to obtain the final absolute coordinates. In the following, we note P_n the final position of a node n and P_n^β the final position of the bend associated to n for the reconnection. The figure 2 summarizes the rotation scheme of the algorithm.

Algorithm 1: Coordinates Assignment.

input : n, the node to draw.
C_n^{abs}, the absolute coordinate of the center of C_n.
function *coordAssign*(node n,C_n^{abs})
1. begin
2. **Let** *rot* be the rotation operation of center C_n^{abs} such that :
$P_{ancestor(n)}$, $rot(\beta_n) + C_n^{abs}$ and C_n^{abs} are aligned.
3. **set** P_n to $rot(\zeta_n) + C_n^{abs}$
4. **set** P_n^β to $rot(\beta_n) + C_n^{abs}$
5. **for all** n_i in $outadj(n)$
6. **begin**
7. **call** *coordAssign*(n_i,$rot(\delta_{n_i}) + C_n^{abs}$) with $\delta_{n_i} = \zeta_n + \gamma_{n_i}$)
8. **end**
9. end coordAssign

Property 1. The bubble Tree drawing is planar.

Proof. The proof of the planarity of the Bubble Tree drawing consists in proving that the rotation operation that we make on the sub-tree drawing doesn't induce overlapping with the other sub-trees and that drawing of a polyline between $P_{ancestor(n)}$, P_n^β and P_n doesn't induce crossing in the final drawing.

The first part is obvious because the rotation is done around the center of an enclosing circle of the drawing and that the algorithm describe in section 3.1 ensures that there is no overlapping between enclosing circles.

For the second part, due to the angular sector induced by the inclusion of a dummy node, it is always possible to draw a line between P_n and β_n. The rotation operation used in the algorithm consists in aligning the bend, the center of the enclosing circle of n and the ancestor of n. Because the enclosing circle of n has been placed in an angular sector θ_n during the drawing of $ancestor(n)$ it is straightforward that β_n is on the bisector of θ_n and thus we can draw a line between $P_{ancestor(n)}$ and β_n. □

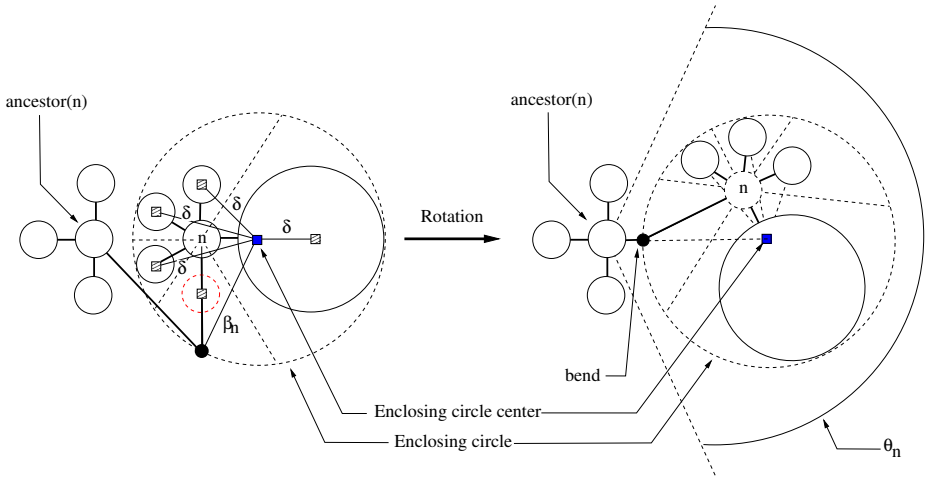

Fig. 2. Coordinates Assignment.

3.3 Space and Time Complexity

The time complexity of the Bubble Tree algorithm is the sum of the complexity of the two stages described above. For each of these steps we have a linear algorithm thus the complexity of the entire algorithm is linear. Furthermore, if one wants to obtain a layout using the optimal solution for the first stage we obtain a complexity in an expected $n \cdot \log(n)$ time. This complexity comes from the using of the Welzl's algorithm [18] and from the relative position computation (cf. 3.1) that requires a sort of the children of each node according to their enclosing circle radius. For the space complexity, the algorithm requires to store five values for each node and thus it is straightforward that it is linear in space.

4 Comparisons

In this section, we compare the results obtained with our algorithm and two well-known tree drawing algorithms. The first one is the Walker's algorithm [17] and the second one is the radial tree drawing proposed by Eades [8]. The data-set we use is an entire Linux file system, including users' directories, that contains about 270.000 nodes. In order to make a statistical analysis of the results obtained by each algorithm, we have extracted from this data-set all the sub-trees having at least 1.000 nodes. For each sub-tree, we have computed the layout obtained by each tree drawing algorithm and for each layout we have computed the standard deviation of two parameters that are the angle and the edge length (some results are presented in figure 3). In the following we describe briefly the both parameters and we discuss the results by showing the distribution of these two parameters on the entire tree. Note that the interpretation of the results on the sub-trees are similar to the one given for the entire tree. Then, we focus on the distribution

of parameters only on the entire tree. In both cases the distribution has been built by using a discrete normalized histogram (see figure 4 and 5). The mean is centered to zero and the size of the interval is one. In the following, we denote by $\sigma(Angles)$ (resp. $\sigma(EdgeLength)$) the standard deviation of distribution of the angles (resp. distribution of the edge lengths).

Algorithm	$\sigma(Angels)$	$\sigma(EdgeLength)$
BubbleTree (270 000)	0.0292999	0.0668552
BubbleTree (100 000)	0.0414186	0.0840859
BubbleTree (47 000)	0.0338936	0.0517702
Walker (270 000)	0.235554	0.0147631
Walker (100 000)	0.149192	0.0264255
Walker (47 000)	0.117964	0.0379802
TreeRadial (270 000)	0.12626	0.0197858
TreeRadial (100 000)	0.09409	0.0502502
TreeRadial (47 000)	0.130956	0.0319285

Fig. 3. Standard Deviation on several sub-trees.

4.1 Distribution of angles

The angular resolution is an aesthetic parameter for measuring graph drawing algorithm quality [4]. The original measure consists in computing the minimal angle between two edges in the final drawing. It is well known that the best value for this parameter should be for each node n equal to $\frac{2 \cdot \pi}{deg(n)}$. In our experimentation, in order to study the behavior of the angular resolution, we have measured for each node n, the difference between each angle, formed by incident edges to a node, and the optimal value. For each node n, this measure give us $deg(n)$ values that we have studied using the statistical method described above. For each algorithm, we presents the distribution of angles in figure 4.

Our algorithm and the Tree Radial algorithm are well balanced around the mean and the highest value of the histogram are very closed to the mean. Contrary, the Walker's algorithm presents two peaks, one located near the mean and one at the beginning of the interval. This property induces a strong standard deviation for the angle distribution associated with it. The interpretation will be done below.

When a distribution presents a peak closed to the mean, the signification is that the attribution of a small angular sector to an edge (an angle value inferior to the optimal value) is offset by the attribution of a high angular sector to another edge. If the number of these edges is relatively low, the result is better.

Our algorithm presents a small standard deviation and a very high peak very closed to the mean, these two properties induce that almost all edges have an optimal angle and the compensation is approximately one to one.

The tree radial algorithm has also a peak near the mean, but its standard deviation is higher ($\sigma(TreeRadial) > 4 \times \sigma(BubbleTree)$ see figure 4). We can see that density of angles is high near the right side of the mean, conversely, at the opposite left side the density is low near the mean. This fact is due to the strategy of the Tree Radial algorithm, because the angular sectors of the children depend on the angular sector attributed to their ancestor. Since the sum of the angular sectors attributed to the children of a given node n is equal to the angular sector of n, the distance between the optimal value and the angular sector of the edges linking n to one of its children is superior to the distance between the angular sector of n and the optimal value. Thus, the negative values of the distribution of the tree radial algorithm are fewer than the positive values but they are more distant.

The distribution of the Walker's algorithm has two peaks, this property induces that the density of angle near the mean is low. Now, we propose an explanation to this fact. Let n be a node and n' one of its children, we denote by e the edge linking n to n', $e1$ (resp. $e2$) the edge linking n' to n'_1 (resp $n'_{|outadj(n)|}$). The Walker's algorithm has the following property, the sum of the angles between e, e_1 and e, e_2 is greater than π. Thus, the angular value attributed to the edges e, e_1 and e_2 is superior to the optimal value and the angular values of the other edges (that link n' to its other children) are inferior to the optimal value. For each of these edges, the angular loss is proportional to the number of these edges. The second peak of the Walker's algorithm located at the extremity of the interval contains the majority of the leaves of the file system.

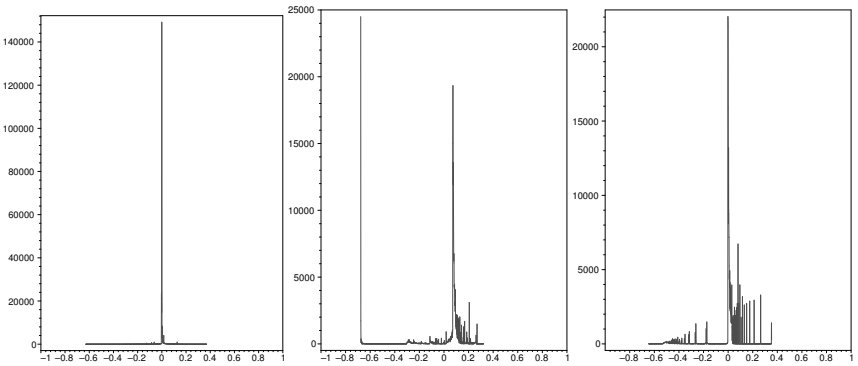

Fig. 4. Distribution of angles.

4.2 Distribution of edge length

In our experimentation, in order to study the behavior of the edge length, we have calculated the mean of the edge length and we have translated the mean to zero.

Considering the distribution of edge length (see figure 5) the peak is very closed to the mean and the standard deviation is low. We can consider that the edge length is quasi constant for these algorithms and that the radial algorithm provides a better result than the Walker's algorithm.

Our algorithm presents one high peak and five small peaks, each of those characterizes a type of node, the high peak located to the left side of the interval corresponds to the nodes that are leaves and where siblings are also leaves moreover the number of siblings is low. The next right peak corresponds to the leaves either with a higher number of siblings or with sibling that are not leaves. For the next peak either the number of sibling increases or the siblings are deeper and so on.

Each peak of the distribution characterizes a node configuration, this property allows us to visually detect the kind of node where we focus on.

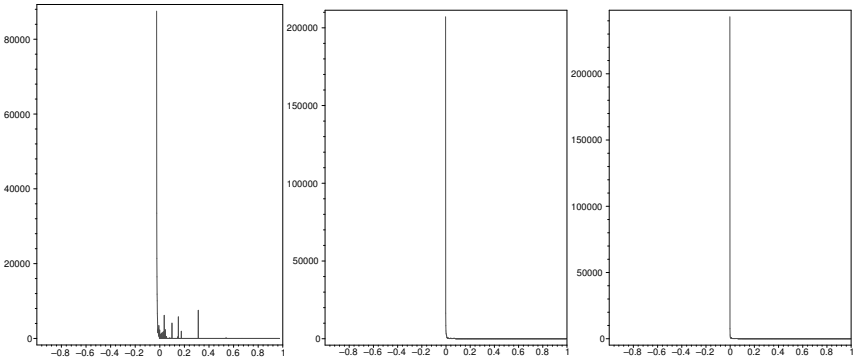

Fig. 5. Distribution of edge length.

4.3 Number of bends

Clearly the weak point of this algorithm is that we do not obtain a straight line drawing. A straightforward bound of the number of bends is the number of internal nodes in the tree. This bound can be equal to the number of nodes less one. However, in the final algorithm we automatically remove a bend β_n if it is straight lined with n and $ancestor(n)$. This operation do not change the theoretical bound that can be reached when one have a completely unbalanced tree. The figure 6 shows the spiral effect induced by the presence of several completely unbalanced sub-tree that induced a large number of

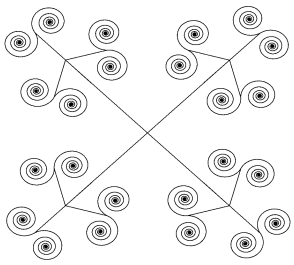

Fig. 6. Spiral Effect.

bends. At the opposite, the number of bends is equal to zero for a well balanced tree, and produces a fractal effect (see figure 8). On the set of sub-trees of our experimentation, we have measured that: the average number of bends is about 1% of the number of nodes, the best value is 0% and the worst value is 7.3%.

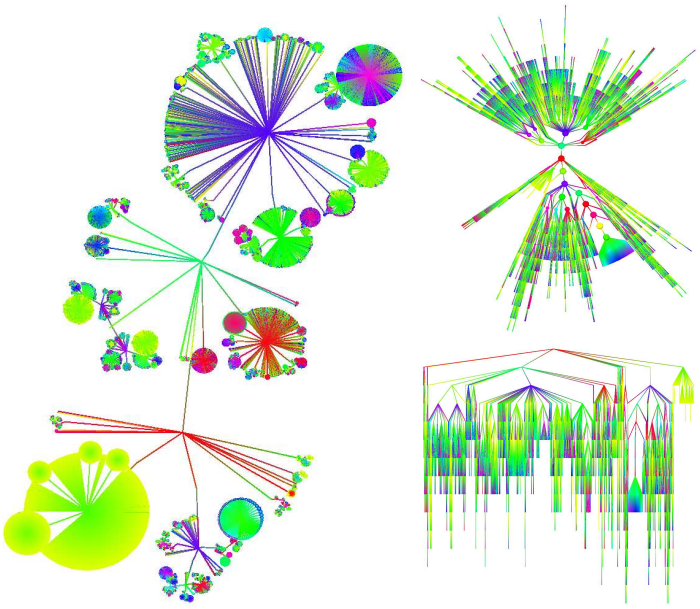

Fig. 7. Bubble Tree (left), Tree Radial (top/right), Tree Walker (bottom/right).

5 Conclusion

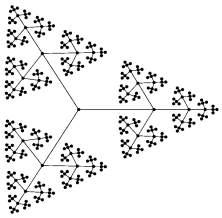

Fig. 8. Fractal Effect.

In this paper we have presented an algorithm for the drawing of general rooted trees. The strong point of the Bubble tree drawing is to privilege the angular resolution aesthetic criteria. Such a characteristic are very important for the purpose of Visual Information Retrieval. This algorithm has been implemented and compare with others by using the Tulip Software [2]. During interactive visualization of huge file-system it has clearly demonstrated its efficiency. Indeed, the Bubble Tree algorithm enables to easily detect isomorphic sub-trees even on graph having more than 270.000 nodes. Furthermore, small modifications of the tree structure imply small modifications of the

final drawing. This property is essential for the visual detection of the similarities.

References

1. N. Amenta and J. Klingner. Case study: Visualizing sets of evolutionary trees. In *IEEE Infovis'02*, pages 71–76, 2002.
2. D. Auber. Tulip. In P. Mutzel, M. Jünger, and S. Leipert, editors, *9th Symp. Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, pages 335–337. Springer-Verlag, 2001.
3. D. Auber. *Graph Drawing Softwares*, chapter Tulip- A Huge Graphs Visualization Framework, pages 80–102. Mathematics and Visualization series. Springer-Verlag, to appear 2003.
4. G. Battista, P. Eades, R. Tamassia, I. Tollis, and G. Tollis. *Graph Drawing : Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.
5. D.M. Bruls, C. Huizing, and J.J. VanWijk. Squarified treemaps. In *Data Visualization 2000*, proceedings of the joint Eurographics and IEEE TCVG Symposium on Visualization, pages 33–42. Springer, 2000.
6. C. Buchheim, M. Jünger, and S. Leipert. Improving walker's algorithm to run in linear time. Technical report, Zentrum für Angewandte Informatik Köln, Lehrstuhl Jünger, April 2002.
7. J. Carriere and R. Kazman. Interacting with huge hierarchies: Beyond cone trees. In G. and S. Eick, editors, *IEEE Symposium on Information Visualization*, pages 74–78. Atlanta, Georgia Institute for Electrical and Electronics Engineers, 1995.
8. P. Eades. Drawing free trees. *Bulletin of the Institute for Combinatorics and its Applications*, 5:10–36, 1992.
9. R.M. Freund, J. Sun, and S. Xu. Solution methodologies for the smallest enclosing circle problem. *Computational Optimization and Applications*, 24-26, 2003.
10. C.S. Jeon and A. Pang. Reconfigurable disc trees for visualizing large hierarchical information space. In *IEEE InfoVis'98*, pages 19–25, 1998.
11. D. E. Knuth. *The Art of Computer Programming*, volume 1. Addison-Wesley, 1973.
12. T. Munzner. H3: laying out large directed graphs in 3d hyperbolic space. In *IEEE Infovis'97*, pages 2–10, 1997.
13. E.M. Reingold and J.S. Tilford. Tidier drawings of trees. *IEEE Transactions on Software Engineering*, 7(2):223–228, 1981.
14. G.G. Robertson, J.D. Mackinlay, and S.K. Card. Cone trees: Animated 3d visualizations of hierarchical information. In *SIGCHI, Conference on Human Factors in Computing Systems*, pages 189–194. ACM, 1991.
15. B. Shneiderman. Tree visualization with tree-maps : A 2-d space filling approach. In *ACM Transaction on graphics*, pages 92–99, 1991.
16. Soon Tee Teoh and Kwan Liu Ma. Rings: A technique for visualizing large hierarchies. In S.G. Kobourov and M.T. Goodrich, editors, *Graph Drawing*, volume 2528 of *Lecture Notes in Computer Science*, pages 268–275. Springer, 2002.
17. J.Q. Walker. A node positioning algorithm for general trees. *Software Practice and Experience*, 20:685–705, 1990.
18. E. Welzl. Smallest enclosing disks (balls and ellipsoids). In Hermann A. Maurer, editor, *New Results and New Trends in Computer Science*, Lecture Notes in Computer Science, 555. Springer-Verlag, 1991.