DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SCHOOL OF ENGINEERING
UNIVERSITY OF NEW MEXICO

A Low Complexity Recursive Force-Directed Tree Layout Algorithm Based on the Lennard-Jones Potential

Lisong Sun [1]
Department of Computer Science
The University of New Mexico
Albuquerque, NM 87131
e-mail: sunls@unm.edu

Steve Smith
Los Alamos National Laboratory
Los Alamos, NM 87545
e-mail: sas@lanl.gov

Thomas Preston Caudell
Department of Electrical and Computer Engineering
The University of New Mexico
Albuquerque, NM 87131
e-mail: tpc@eece.unm.edu

UNM Technical Report: EECE-TR-03-001

Report Date: April 16, 2003

[1]Partially funded by Los Alamos National Laboratory and High Performance Computing, Education and Research Center at the University of New Mexico.

Abstract

In this paper, a low complexity force-directed tree layout algorithm based on the Lennard-Jones potential is described. The recursive method lays out sub-trees as small disks contained in their parent disk. Inside each disk, children disks are dynamically laid out using a new force directed simulation. Unlike most other force directed layout methods which run in quadratic time for each simulation step, this algorithm runs in $O(n^{\frac{m+1}{m}})$ time per each step for a tree with n nodes, depth m and all the nodes having uniform number of children. The layout uses space efficiently and reflects both global structure and local detail. The method supports runtime insertion and deletion. Both operations and the evolving process are rendered with smooth animation to preserve visual continuity. The method could be used to monitor in real time, visualize and analyze a wide variety of data which has a rooted tree structure, e.g. internet hosts could be laid out by domain name (DNS) hierarchies. This paper gives a description of the algorithm, a complexity analysis and an example of how the algorithm is implemented to visualize DNS tree.

Keywords

Graph Layout, Tree Layout, Lennard-Jones Potential, Force-Directed Simulation, DNS Hierarchy

1 Introduction

In the domain of computer science, a graph G is defined as a set of vertexes (or nodes) V connected by a set of edges (or links) E. Graph like structures are pervasive. For example, route maps of airline companies, infrastructure of computer networks, the relationship between people who work in a same company etc. One way to understand the information coded in these graphs is to draw graphical representations of them. Since drawing by hand is tedious and error-prone, it is natural to expect computers to draw graphs automatically, assigning spatial coordinates to vertexes and connecting them with edges.

Graphs, such as the flight route maps, are not hard to draw since the precise locations of the vertexes (cities) are already given. For other graphs, such information is not available and computer needs to determine where to plot the vertexes and how to draw the edges that connect the vertexes. Various graph layout algorithms have been developed to solve this problem.

Two decades ago, Peter Eades proposed a graph layout heuristic[6] which is called the "Spring Embedder" algorithm. As described later in a review[3], edges are replaced by springs and vertexes are replaced by rings that connect the springs. A layout can be found by simulating the dynamics of such a physical system. Figure 1 shows how the method works. This method and other methods, which involve similar simulations to compute the layout, are called "Force Directed" algorithms.

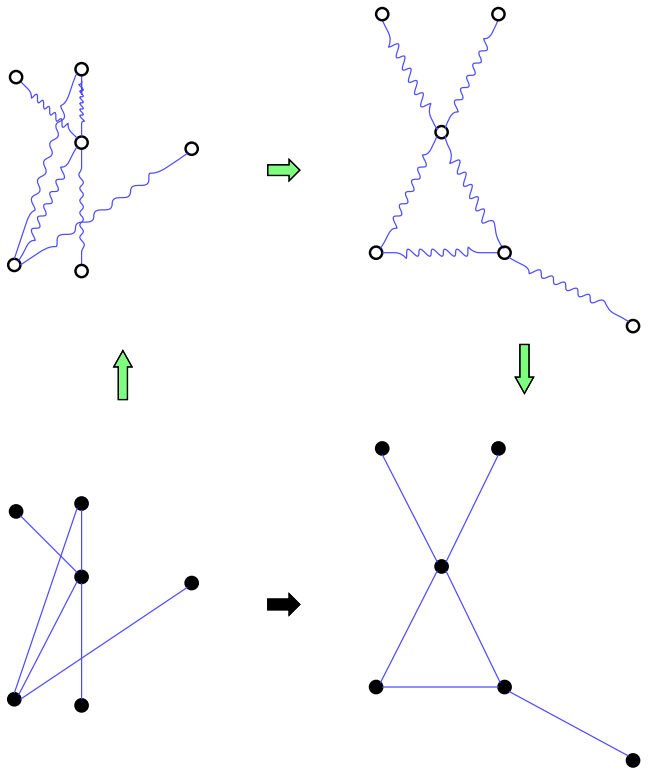


Figure 1. Spring Embedder Algorithm

Because of the underlying analogy to a physical system, the force directed graph layout methods tend to meet various aesthetic standards, such as efficient space filling, uniform edge length, symmetry and the capability of rendering the layout process with smooth animation (visual continuity, also described by Eades as "preserving the mental map"[7]). Having these nice features, the force directed graph layout has become the "work horse" of layout algorithms. It has been successfully adapted to many domains with variations of implementation[15][11][16][10]. Force directed layout methods commonly have computational scaling problems. When there are more than a few thousand vertexes in the graph, the running time of the layout computation

can become unacceptable. This is caused by the fact that in each step of the simulation, the repulsive force between each pair of unconnected vertexes needs to be computed, costing a running time of $O(\frac{V^2}{2} - E)$. Here V is the number of vertexes and E is the number of edges in the graph. This complexity is hard to escape for general graphs with no hierarchical structure. In this paper, we focus on a special but common type of graph, the tree.

It is interesting to note that the available tree visualization methods[19][17][8][9][13][14][18][12][4], do not uses force directed layout methods. One reason for this is the scaling problem mentioned above. The hierarchical structure of trees makes it possible to find a direct mapping from a node in the tree to a proper coordinate. Unfortunately the attractive properties of force directed methods, such as efficient space filling and visual continuity are lost.

There are several conventions for drawing trees[3][9][8], for example, the classic planar straight-line drawing convention (Figure 2), which represents nodes as dots and edges as straight lines connecting the two nodes, and the containment convention (Figure 3), which represents nodes as squares or disks with children nodes contained inside parent nodes. The first convention is the most commonly used in tree layout and drawing algorithms. On the other hand, the containment convention may be used to increase the efficiency of space filling and reducing the visual complexity. A few implementations of the containment convention can be found[18][12][4]. These algorithms are not efficient in using the rendering space and none of them renders dynamic tree structures.

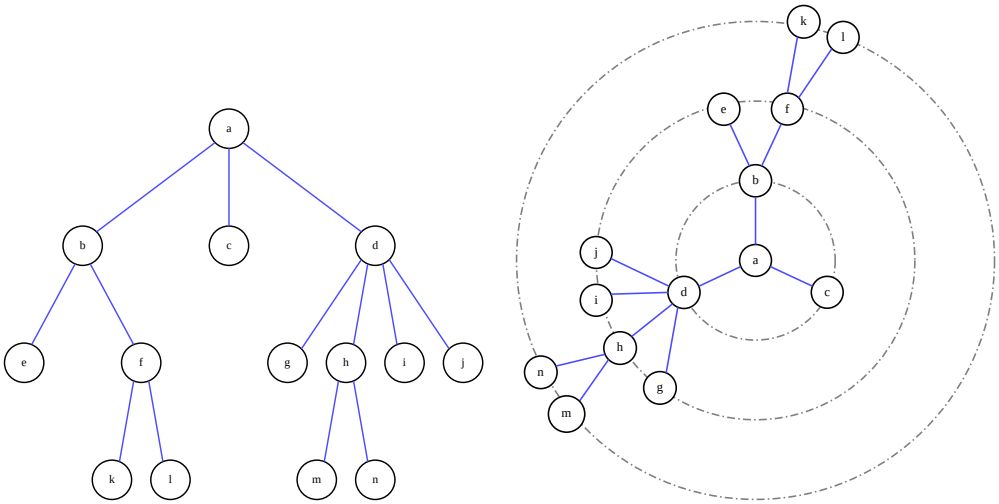


Figure 2. Straight-Line Drawing Convention

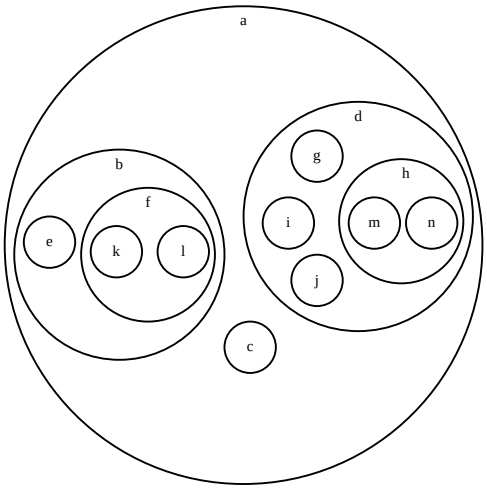


Figure 3. Containment Convention

The algorithm presented here follows the containment convention and makes use of force directed simulation. The quadric running time is avoided by applying the divide and conquer approach. By dividing the tree into sub-trees, only siblings of the same parent need to interact during the simulation, allowing the decomposition of the general tree layout problem into nested, independent sub-problems. In this paper, a recursive layout algorithm utilizing a novel force simulation is introduced which has a much decreased complexity.

The remainder of the paper is structured as follows. Section 2 provides a detailed description of the method. Section 3 gives the complexity analysis and shows the results from several tests with respect to running time, visual effect and different forms of representations. A discussion can be found in Section 4 and Section 5 gives conclusion and future work.

2 Method

The implementation of the method is explained in this section. A general outline of the algorithm is given in Section 2.1. Following that is the detailed description of the simulation and rendering process.

2.1 General Algorithm

This method works with data that has a rooted tree structure. Here rooted tree means the root node of the tree is fixed and does not change over time. A free tree could be transformed into a rooted tree by picking one node as the root, but it might not reflect the same hierarchical semantics as a rooted tree hence not preferable to be laid out using this algorithm.

Each node of the tree is represented as a disk. Each child node is contained inside its parent node which is represented as a larger disk. Each node recursively contains all of its proper descendants and all the nodes are contained inside the disk that represents the root node.

Inside each node, the positions of its children are determined by simulating a physical system. In the simulation, each node is treated as a particle. The mass of the particle is proportional to the size of the sub-tree rooted in the node. Each node has a position which is relative to the center of its parent disk. The position will be updated after each step of simulation. Each node has a relative-radius which equals the square root of the size of the sub-tree rooted in the node. The relative-radius will be used to plot the current disk inside its parent and it is updated whenever an insertion or deletion occurs in the sub-tree. Each node also has a children-radius which is the minimum radius to fully contain all its children disks. It is computed by finding the maximum sum of the distance from the center of current node to the center of each children disks and the children disk's relative-radius. The children-radius will be used to compute the scale factor when rendering the children of the current node and is updated after each step of the simulation.

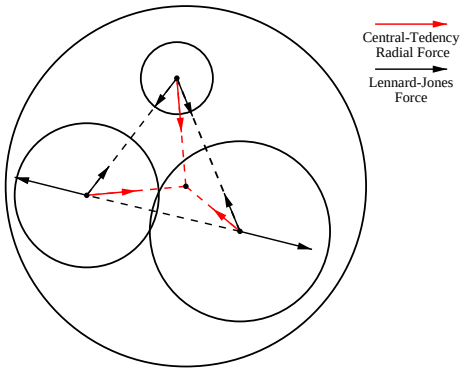


Figure 4. Two Types of Forces in the Simulation

Two types of forces are computed in this physical system. One of them is the force derived from the Lennard-Jones potential. This force exists between each pair of the contained siblings and ensures that they will not overlap upon each other. The second force is a central-tendency radial force which pulls all the children toward the center of the parent disk. A simple illustration of the system is given is Figure 4. Detail of this simulation could be found in Section 2.2.

Once the method for laying out one node's children is defined, the whole tree can then be laid out recursively starting with the root. The rendering is done in the same recursive manner. Detail of this implementation could be found in Section 2.3.

2.2 Simulation Process

In each simulation step, all of the forces exerted on each particle are vectorialy summed. The particle will move in the direction of the total force. The displacement is proportional to the total force divided by the mass of the particle. A maximum displacement is used to keep the particles from moving too far in each step. Since the potential field generated by such a particle system is very complicated, some particles will start to vibrate at various frequency. To reduce the vibrations, a simple filter is applied on the trajectory of each particle. There is no momentum term in the force equation which means kinetic energy is totally "damped".

Two optional boolean flags are used to indicate whether the layout inside the current node has converged (Reached a balanced layout) and whether the layout of the sub-tree has converged. These flags are used to turn off the layout of the converged pieces. If the flags indicate that a sub-tree has fully converged, the simulation process inside that sub-tree will stop. When a new node is inserted, the flags of the nodes along the insertion path will be set to false, which will initiate the layout along the path. The same will be done for deletion. This time saving mechanism makes the actual layout process run faster than the analyzed results given in Section 3.1. In order to get stable layout times, all the running time tests were carried out with this mechanism disabled.

Algorithm : LAYOUTFUNCTION($radial\_factor, LJ\_factor$)

for each $i \in child\_container$
  $i.force = -i.position * i.mass * radial\_factor$
$child\_radius = 0$
for each $i \in child\_container$
  for each $j \in child\_container$ **and** $j \neq i$
    $i.force = i.force + \text{LJFORCE}(i, j, LJ\_factor)$
  $i.position = i.position + \frac{i.force}{i.mass}$
  $child\_radius = \text{MAX}(\text{LEN}(i.position) + i.radius, child\_radius)$
for each $i \in child\_container$
  $i.\text{LAYOUTFUNCTION}(radial\_factor, LJ\_factor)$

Detailed description of the Lennard-Jones potential and how it is used to derive the forces is given in the next subsection. Following that is the description of the central-tendency radial force.

- **Lennard-Jones Force**
  The Lennard-Jones (LJ) potential [1] is a simple mathematical model for the simulation of potential energy between two particles. It has been widely used in computational chemistry to simulate the interaction between two atoms or molecules[20][1]. The normal form of LJ potential is shown below.

$$\phi_{LJ}(r) = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right]$$

[1] Named after Sir John Edward Lennard-Jones (1894-1954), English theoretical chemist, one of the early founders of quantum chemistry.

Here r is the distance between the centers of two particles, ϕ_{LJ} is the potential, ε is a constant related to the energy unit and σ is the constant related to the equilibrium distance. Figure 5 shows the general shape of the potential. When two particles get too close, the potential increases rapidly which will cause a strong repulsive force. When r is greater than the equilibrium distance, there will be a mildly attractive force between the two particles. When r equals $\sqrt[6]{2}\sigma$, the potential is at a minimum and there is no force.

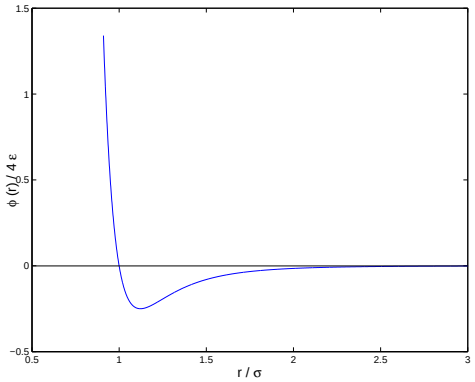


Figure 5. Lennard-Jones Potential

In the layout algorithm, such a potential exists between each pair of siblings contained within a parent node. The force can easily be calculated by taking the first derivative of the potential.

$$F_{LJ}(r) = -4\varepsilon\left[12\left(\frac{\sigma^{12}}{r^{13}}\right) - 6\left(\frac{\sigma^{6}}{r^{7}}\right)\right]$$

A unit vector which indicates the direction of the force in two or higher dimensions points along a line between the two interacting nodes. ε is an adjustable parameter defined at the beginning. σ is set to be the sum of the radius of the two sibling disks, which reflects the desired distance between the two disks. In practice, σ can be larger than the sum, for example, the sum multiplied by 1.1. This gives a marginal space between each pair of the sibling disks. The marginal space enhances the visual effect and reduce the chance of overlap.

- **Central-Tendency Radial Force**
  In order to keep all the children close to the center of the parent disk, a central-tendency radial force is applied on each child inside a parent disk. In this implementation, the radial force equals the mass of the child times the distance between the center of the child disk and the center of the parent disk. An optional factor can be used to make the simulation more flexible. The radial force is proportional to the mass which pulls larger disks closer to the center of their parent disk. This will increase the space filling. A discussion of this issue can be found in Munzner's Ph.D. dissertation[18]. The radial force is proportional to the distance to the center of the parent disk. It makes the force similar to the effect of springs which connect each child disk to the center of their parent disk. The disks further away from the center will be pulled back by a larger radial force, which will ultimately decrease the radius the parent disk needed to contain all the children and increase the space filling inside the parent disk.

2.3 Rendering Process

The rendering was done inside the UNM virtual reality development environment "Flatland"[5] using openGL. Similar to the simulation process, the rendering is done recursively. Following openGL convention, glPushMatrix is used to keep the current state and glPopMatrix is used to resume the previously stored state.

Algorithm : RENDERFUNCTION($depth$)

PLOTDISK($radius = 1.0$)
GLSCALE($\frac{1}{child\_radius}$)
for each $i \in child\_container$
  GLPUSHMATRIX()
  GLTRANSLATE($i.position$)
  GLSCALE($i.radius$)
  i.RENDERFUNCTION($depth + 1$)
  GLPOPMATRIX()

The rendered result using above method is shown in Figure 7 and Figure 8. Although the containment relationship was used throughout the simulation process, it is not necessary to represent the tree in this way. A simple variation is to plot the connections between parent and children instead of painting the disks. This representation will then become a straight-line convention type. An optional third dimension can be introduced to avoid edge crossings. In this case, some 3D geometric shapes are used to represent edges and nodes. The result of various representations can be found in Section 3.4.

3 Result

This section starts with the complexity analysis of the method described above. Following that is the result from a set of running time tests. The third part of this section gives an example of how the layout method was implemented to visualize internet structure and traffic. The result from various representations is given at the end of this section.

3.1 Complexity Analysis

For a node with k children, the computation of the LJ force takes $O(k^2)$ time and computation of the radial force takes $O(k)$ time. The rendering process takes constant time for each node hence linear time $O(n)$ for the tree. The total running time needed to layout and render a tree of n nodes is then

$$\sum_{i=1}^{n} degree(i)^2$$

Here $degree(i)$ refers to the number of children the node i has.

For a full k_{th}-tree of depth m (a tree with all the non-leaf nodes having k children and all the leaf nodes at depth m) the total running time becomes $\sum_{i=1}^{k^{m-1}} k^2 = k^{m+1}$. Since there are $n = k^m$ nodes in the tree, $O(k^{m+1}) = O(n^{\frac{m+1}{m}})$.

For a random tree, the result will depend on how the randomness behaves. In general the performance will be dominated by the node with maximum number of children. That is $\sum_{i=1}^{n} degree(i)^2 \approx max\_degree^2$.

In both cases, the total running time is slightly slower than linear. Compared with most of the other force directed methods which runs in quadratic time, the complexity of this algorithm is significantly reduced.

3.2 Running Time Test

Full k_{th}-trees of depth 3 and 4 are generated for k equals 6 to 12. Running time for each step of the layout process was measured. The result is shown in Table 1. Each of the time values listed is the average of $1,000$ samples. The test was done on a laptop computer with 1.7GHz mobile Intel P4 CPU.

Degree k	**Time/ms**	k^{m+1}
6	0.971	1296
7	1.611	2401
8	2.614	4096
9	3.937	6561
10	5.712	10000
11	8.251	14641
12	11.31	20736

Tree depth: $m = 3$

Degree k	**Time/ms**	k^{m+1}
6	6.093	7776
7	12.00	16807
8	21.39	32768
9	36.46	59049
10	59.40	100000
11	93.51	161051
12	151.0	248832

Tree depth: $m = 4$

Table 1. Layout Running Time for Full k_{th}-Tree

Running times were plotted versus k^{m+1} in Figure 6. The plots show a clear linear relationship between them. This conforms to the previous analysis that k_{th}-tree's running time is $O(k^{m+1})$.

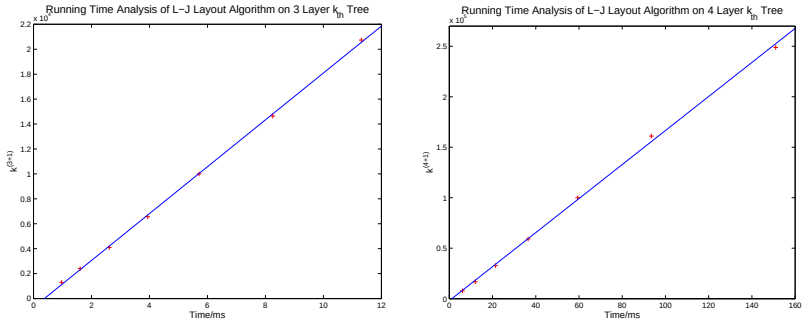


Figure 6. Linear Regression of Layout Running Time for Full k_{th}-Tree of Depth m *vs.* k^{m+1}

3.3 Internet Layout Example

This algorithm is being used as part of a research project to visualize internet structure and traffic to detect network intrusions. Large numbers of internet host names were collected from an internet firewall as test data. Since internet domain names have a natural hierarchical tree structure, this data is well suited for the proposed algorithm. For example, all of the domain names ended with ".edu" belongs to the same sub-tree. "cs.unm.edu" is a child of "unm.edu". All the highest level domain names such as ".com", ".net", ".edu" and ".gov" are children of the root of the internet.

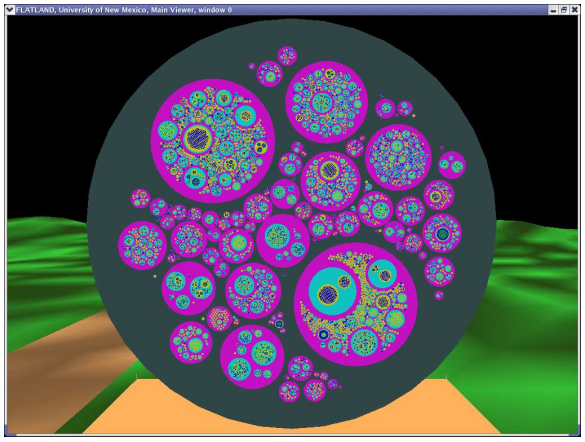


Figure 7. Layout Result of 8,000 Internet Host Names

Figure 7 shows the layout result of a tree with $8,000$ host names. The diameters of the disks reflect the number of descendants they have which gives the visual information about the structure of the tree. Detail inside sub-trees can be obtained by zooming into the figure. This is especially helpful as trees grow large. Figure 8 shows the detail of a sub-tree located in upper left corner of Figure 7.

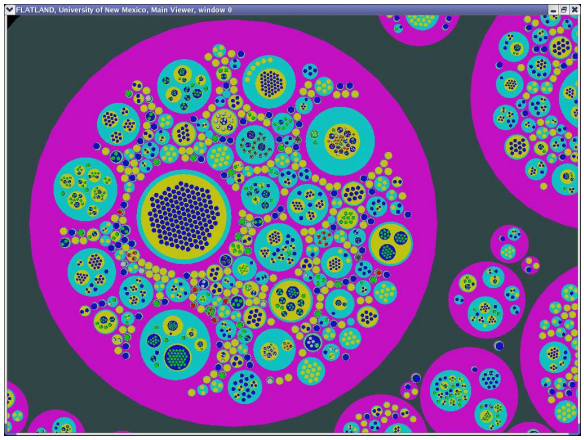


Figure 8. Zoomed In Detail of a Sub-Tree In Figure 7.

Layout times for each step are measured together with the frame rates. The maximum degrees and their squares are also listed in the table below.

Number of Nodes	Max Degree	Max Degree^2	Time/ms	Frame Rate/fps
1000	80	6400	18	13.5
2000	164	26896	48	9.90
3000	188	35344	76	6.65
4000	219	47961	106	4.69
5000	244	59536	138	3.39
6000	280	78400	175	2.81
7000	320	102400	222	2.68
8000	376	141376	281	2.50

Table 2. Layout Running Time, Max Degree and Frame Rate of DNS Tree

A plot of layout time versus the square of the maximum degree is given in Figure 9. A linear relationship is shown in the plot which met the previous complexity analysis about random trees.

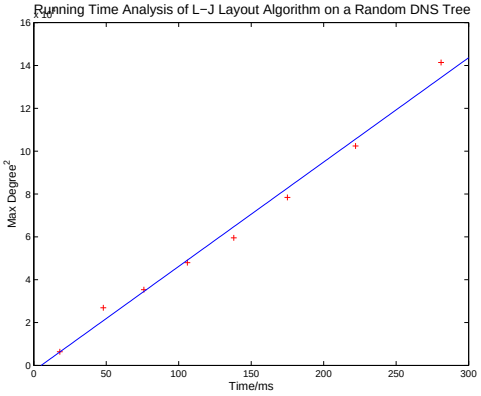
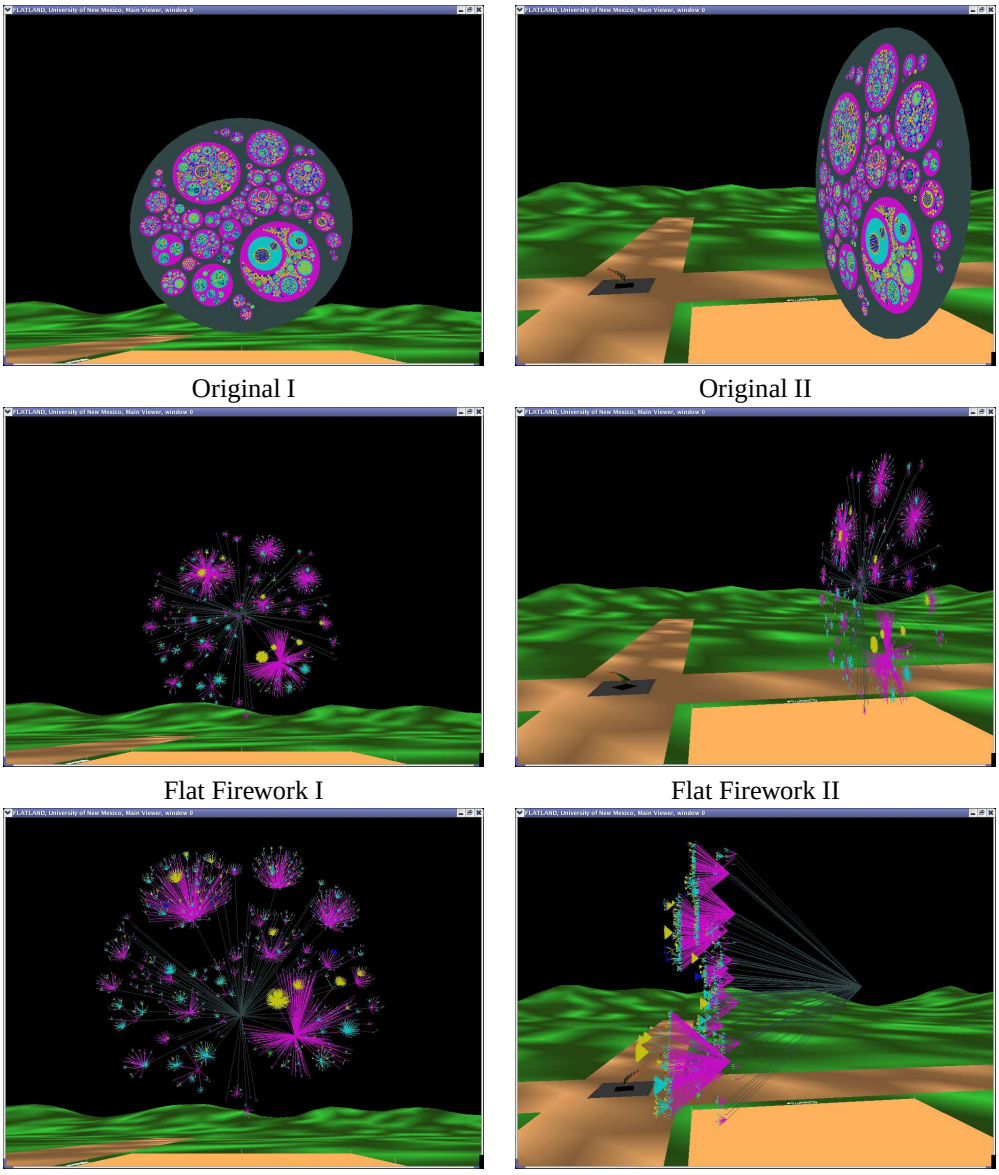


Figure 9. Linear Regression of Running Time *vs.* Max Degree^2

3.4 Various Representations

As described in Section 2.2 and Section 2.3, although the underlying simulation process uses the containment relationship between parents and children, in the representation it is not necessary to conform to this. Figure 10 shows five different representations of the same tree structure using the same simulation mechanism.

In Figure 10, "Original I and II" directly reflect the mechanism of the simulation; "Flat Firework I and II" use the coordinates of the nodes computed by the simulation and connect parents and children with straight lines; "Firework I and II" translate the children in the third dimension; "Crystal I and II" replace the straight lines with cylinders; the last representation, "Dome I and II", is different from the previous four since it does not use the coordinates computed by the simulation directly. A non-linear mapping method is designed to project each node in the root disk onto a corresponding point on the hemisphere. The "Dome" representation is implemented in the Network Intrusion Project at Los Alamos National Laboratory.

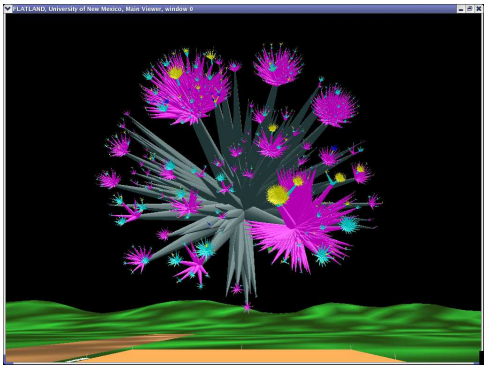


Original I | Original II

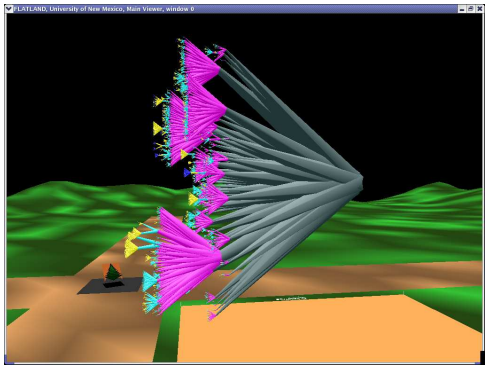
Flat Firework I | Flat Firework II

Firework I | Firework II

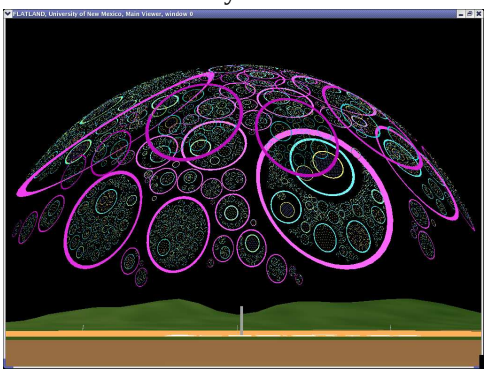
Figure 10. Five Different Representations of a Tree With 4,000 Nodes (Continued in next page)



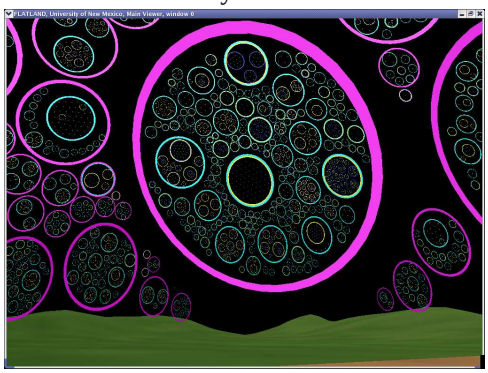
Crystal I



Crystal II



Dome I



Dome II

Figure 10 Continued. Five Different Representations of a Tree With 4,000 Nodes

4 Discussion

Compared with other tree drawing algorithms, the method described here is novel in the sense that parent-child relationships are now expressed as containment relationships. The edge crossings, which are hard to avoid in other force directed layout methods and can cause confusion between different edges, are no longer a problem since edges are not drawn. Since there is no need to maintain the up-down or central-radial relationship between parents and children (See Figure 2), the space filling is increased. The Lennard-Jones potential plays the key role in avoiding overlaps. It gives the disks both a "hard border", which will stop the neighbors from moving too close, and a mild attraction to neighbors further away, which will keep the disks packed tight. Without this feature, the recursive layout method would not be valid. It is possible to replace the Lennard-Jones potential with other type of potentials which have a similar shape shown in Figure 5. The Lennard-Jones potential is chosen in this algorithm since it is straight forward to understand and easy to implement. The central-tendency radial force is effective in keeping all the children disks packed tight inside their parent disk. Fine tuning of the movement distance limit stops the particles from jumping when they got too close to each other. The filter successfully removed the vibrations. Both of them are important to keep the main structure stable during the layout process.

Efficiency, the biggest concern for most of the force directed layout methods, is improved by breaking large trees into small sub-trees and laying out the pieces recursively. The running time tests show a significant improvement over other similar methods. Since insertion and deletion operations can only affect a small region of the tree, the whole structure will change little. In fact, only the nodes along the path and their siblings need to be adjusted. All the other nodes with a converged layout will not be recomputed. This time saving mechanism makes the layout process run even faster. Meanwhile, it also helps to preserve visual continuity. In the internet layout test, 8,000 host names were inserted one by one during the layout process. The rendered image changed smoothly during the insertion and an observer can easily track details inside the sub-domains. This feature is

essential for tasks such as monitoring changes of dynamic tree structures.

Although the efficiency of the layout algorithm is addressed well by this technique, there remain efficiency considerations in the rendering and display of these trees. When real-time rendering is required and there is a large number of nodes, for example $10,000$, the frame rate will drop below 3 frame per second (See Table 2). Naturally, as graphic cards get more powerful, the problem will be partially relieved. Meanwhile, techniques such as the Continuous Semantic Zooming[21] method proposed by Summers and other similar methods[2] could also be used to address the problem. Image-based rendering techniques are also promising to provide better than linear speedup as well.

5 Conclusion And Future Work

In this paper, a new tree layout method is introduced. The method is fast because of its "Divide and Conquer" nature. The design is simple because of its recursive layout and rendering mechanism. The visual effect is smooth because of its underlying simulation process.

More interactive functionalities will be developed in the future. Mechanisms will be implemented to increase the frame rate. Meanwhile it will be interesting and challenging to find out whether this method could be used on other types of graphs.

Beyond the implementation in the research area of graph layout, the simulation process proposed in this paper could also be used as a circle packing or sphere packing algorithm. A packing efficiency analysis will be given in the future.

Acknowledgment

Special thanks to Timothy B. Eyring who gave us many valuable suggestions during the development of the method. We also thank Satyam Babu Kothapally and Paul Weber for helping us to obtain internet host data.

References

[1] M. P. Allen and D. J. Tildesley. *Computer Simulation of Liquids*. Oxford University Press, 1987.

[2] Lyn Bartram, Albert Ho, John Dill, and Frank Henigman. The continuous zoom: A constrained fisheye technique for viewing and navigating large information space. *Proceedings of ACM UIST'95*, pages 207–215, November 1995.

[3] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Loannis G. Tollis. Algorithms for drawing graphs: An annotated bibliography. *Computational Geometry: Theory and Applications*, 4(5):235–282, 1994.

[4] Richard Boardman. Bubble trees: The visualization of hierarchical information trees. *Extended Abstracts of ACM CHI 2000 Conference*, 2000.

[5] Thomas Preston Caudell. http://www.hpcerc.unm.edu/homunculus, 2003.

[6] Peter Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[7] Peter Eades, W. Lai, Kazuo Misue, and Kozo Sugiyama. Preserving the mental map of a diagram. *Proceedings of Compugraphics*, 9:24–33, 1991.

[8] Peter Eades, Tao Lin, and Xuemin Lin. Minimum size h-v drawings. *Advanced Visual Interfaces - Proceedings of the Internations Workshop AVI '92*, pages 386–394, 1992.

[9] Peter Eades, Tao Lin, and Xuemin Lin. Two tree drawing conventions. *International Journal of Computational Geometry and Applications*, 3(2):133–153, 1993.

[10] Peter Eades and Xuemin Lin. Spring algorithms and symmetry. *Theoretical Computer Science*, 240(2):379–405, 2000.

[11] Thomas M.J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software–Practice and Experience*, 21(11):1129–1164, 11 1991.

[12] groxis. http://www.groxis.com, 2003.

[13] Ivan Herman, Guy Melançon, Maurice M. de Ruiter, and Maylis Delest. Latour — A tree visualisation system. *Lecture Notes in Computer Science*, 1731:392+, 2000.

[14] Ivan Herman, Guy Melançon, and M. Scott Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[15] T. Kamada and S. Kawai. Automatic display of network structures for human understanding. *Information Processing Letters*, 31:7–15, 1989.

[16] Xuemin Lin. *Analysis of Algorithms for Drawing Graphs*. PhD thesis, Department of Computer Science, University of Queensland, Australia, 1992.

[17] Sven Moen. Drawing dynamic trees. *IEEE Software*, 7(4):21–28, July 1990.

[18] Tamara Munzner. *Interactive Visualization of Large Graphs and Networks*. PhD thesis, Stanford University, June 2000.

[19] Edward M. Reingold and John S. Tilford. Tidier drawings of trees. *IEEE Transactions on Software Engineering*, 7(2):223–228, 1981.

[20] Robert J. Silbey and Robert A. Alberty. *Physical Chemistry*. J. Wiley and Sons, Inc., 3 edition, 2001.

[21] Kenneth L. Summers. *Visualization of Programs Using Proximity to Trigger Continuous Semantic Zooming: An experimental Study*. PhD thesis, University of New Mexico, 2002.