

# A Spring Modeling Algorithm to Position Nodes of an Undirected Graph in Three Dimensions

Aruna Kumar<sup>?</sup> and Richard H. Fowler<sup>?</sup>

**Technical Report**  
**Department of Computer Science**  
**University of Texas - Pan American**  
**Edinburg, Texas 78539**

<sup>?</sup>Architecture Labs: New Media Prototypes Intel Corporation  
Hillsboro, OR 97124

<sup>?</sup>Department of Computer Science  
University of Texas - Pan American  
Edinburg, TX 78539

## Abstract

A number of data presentation problems require the drawing or display of graphs. Methodologies for creating graph displays have typically focused on drawing on a two-dimensional surface. Recently, interest in computer-based visualization has increased attention on methodologies for the display of graphs in three dimensions. The spring embedder algorithm (Eades, 1984) is a heuristic approach to graph drawing based on a mechanical system in which a graph's edges are replaced by springs and vertices are replaced by rings. From the initial configuration of ring positions, the system oscillates until it stabilizes at a minimum-energy configuration. To date examples of the spring embedder algorithm have constrained the system of springs to move in a plane in order to draw the underlying graph on a two-dimensional surface. Allowing the spring system to reach its equilibrium in three dimensions would enable the use of interactive computer visualization as a tool in revealing the graph's structure. This paper reports that extension of the spring embedder algorithm from two to three dimensions.

---

## Table of Contents

1. Introduction
  2. A Spring Model in Three Dimensions
    - 2.1. Background
    - 2.2. A Dynamically Balanced Spring System
    - 2.3. Positioning Nodes in Three Dimensions
    - 2.4. Computing a Local Minimum
    - 2.5. Algorithm for Positioning Vertices in Three Dimensions
  3. Example Displays
  4. Conclusion
  5. Acknowledgments
  6. References
-

## 1. Introduction

A number of data presentation problems require the drawing or display of graphs. Methodologies for creating graph displays have typically focused on drawing on a two-dimensional surface. Recently, interest in computer-based visualization has increased attention on methodologies for the display of graphs in three dimensions. For visual presentations layout criteria center on how quickly and clearly the meaning of the diagram is conveyed to the viewer, i. e., the readability of the graph. Graph drawing algorithms have as their goal the layout and presentation of an inherently mathematical entity in a manner which meets various criteria for human observation. The aesthetics of a layout determine its readability and can be formulated as optimization goals for the drawing algorithm (Eades & Tamassia, 1989). For example, the display of symmetry and minimization of the number of edge crossings in two dimensional drawings are fundamental aesthetics for visual presentations. VLSI design is another area in which graph layout is an issue, but different criteria are applied, such as minimization of the area covered by the graph.

Given the wide application of graphs structures in display, there are relatively few algorithms for drawing general undirected graphs (Tamassia, Di Battista & Battini, 1988). This is due in part to the inability to specify the aesthetic criteria individuals use in understanding graphs (Eades & Xuemin, 1990). Nonetheless, for certain restricted classes of graphs in which graph-theoretic expressions of aesthetic criteria can be specified, satisfactory algorithms have been developed (Batini, Nardelli, & Tamassia, 1986; Carpano, 1980; Rowe et al., 1987). Comprehensive reviews can be found in Eades and Tamassia (1989) and Tamassia, Di Battista, and Batini (1988). The table below (after Messinger, Rowe, & Henry, 1991) summarizes representative graph drawing algorithms.

TABLE 1  
Representative Graph Drawing Algorithms

Type of Graph	Special Attributes	Author
Undirected Planar	Convex Layouts	Chiba, Yamanouchi, & Nishizeki, 1984
Undirected Planar	Manhattan Grid Embedd. Min. Edge Bends	Tamassia, 1987
General Undirected	Symmetric Drawings	Eades, 1984
General Undirected	Symmetric Drawings	Lipton, North, & Sandberg, 1985
General Undirected	Circular Drawings	Makinen, 1988
Directed Trees	Hierarchical Tree	Reingold & Tilford, 1981
General Directed	Hierarchical, Reduced Edge-crossings	Sigiyama, Tagawa, & Toda, 1981
General Directed	Hierarchical, Cycles, Graph Browser	Rowe et al., 1987
General Directed	Hierarchical, Edge-crossing/speed tradeoff	Robins, 1987

Some of the aesthetics for drawings of general undirected graphs are symmetry, minimization of edge crossings and bends in edges, uniform edge lengths, and uniform vertex distribution. (Eades & Tamassia, 1989). These aesthetics are such that optimality of one may prevent optimality in others. Additionally, graph layout algorithms in general can be viewed as optimization problems and are typically NP-complete or NP-hard. These two observations suggest a heuristic approach to general graph drawing for many applications.

The *spring embedder* algorithm (Eades, 1984) is a heuristic approach to graph drawing based on a physical system. This algorithm simulates a mechanical system in which a graph's edges are replaced by springs and vertices are replaced by rings connecting edges (springs) incident on a vertex. From the initial configuration of ring positions, the system oscillates until it stabilizes at a minimum-energy configuration. Among the parameters that control the forces acting on the rings and causing their movement are spring length, spring stiffness, spring type, and initial configuration. This a very general heuristic which can be combined with other algorithms (Esposito, 1988) to provide approximate solutions for competing aesthetics.

To date examples of the spring embedder algorithm have constrained the system of springs to move in a plane in order to draw the underlying graph on a 2-dimensional surface. Allowing the spring system to reach its equilibrium in three dimensions would enable the use of interactive computer visualization as a tool in revealing the graph's structure. This paper reports that extension of the spring embedder algorithm from two to three dimensions. The work and paper closely following the approach and explication of Kamada and Kawai (1989) in the extension.

## 2. A Spring Model in Three Dimensions

### 2.1. Background

An elastic body is one that experiences a change in volume or shape when the deforming forces act upon it but resumes its original size or shape when the deforming forces cease to act. The deformation of an elastic body is directly proportional to the magnitude of the applied force, provided the elastic limit is not exceeded. The elastic potential energy of a deformed body, which is our main concern, is the negative of the work done by the elastic forces when the body changes from the configuration defined as that of zero potential energy to the deformed configuration.

Consider a spring of negligible mass with one end attached to a wall. We choose the origin of the x axis to be the location of the free end of the spring when it is not stressed. If we apply an external force  $F$  to the free end, we find experimentally (to a good approximation) that the force required to stretch or compress the spring is proportional to the distance the free end moves from its unstressed location. The force  $F$  can be defined by the following equation:

$$F = KX.$$

where  $X$  is the distance the free end moves due to the stretching. The work that is done to stretch the free end of a spring from to a position is  $1/2 KX^2$ . The work done in stretching the spring is stored as potential energy. Thus the potential energy stored in the system is given by the following equation:

$$P. E. = \frac{1}{2} K X^2.$$

$K$  is the force constant of the spring.  $X$  is the elongation that the spring underwent as a consequence of the deforming forces acting on it.

The energy of the system is dependent on the length of elongation that the spring underwent. Hence there will be a gradual variation in the spatial positions of the extremities of the spring as it tends towards it's equilibrium position. The coordinate variables  $x$ ,  $y$ , and  $z$  are required to define the position

of the springs in three dimensions. Thus the variation of the energy of each spring with respect to the coordinate variables  $x$ ,  $y$  and  $z$  can be represented in the form of a partial differential equation. To arrive at the equilibrium position of the spring it will be necessary to solve the partial differential equations for each spring. The result will be the distances  $x$ ,  $y$ , and  $z$  respectively. Potential energy is always associated with a change in the configuration of a system of particles, in this case the rings or edges of the springs. The potential energy stored in the spring can be completely recovered by allowing the spring to move back to its unstressed position. Thus the final energy of the spring is at a minimum, possibly zero.

The above describes the method to be used in solving for the minimum potential energy of a single spring defined by the coordinate variables in three dimensions. In dealing with a system of springs it will be necessary to solve a system of partial differential equations to arrive at the equilibrium position of the entire system. If we are concerned with an arbitrary system of  $n$  vertices ( $n$  springs) then we have to solve  $3n$  simultaneous non-linear equations. The drawings to be represented here are considered to have straight edges and the position of the vertices of the structures are not restricted. We are primarily concerned with the balance of the layout. In this model the total balance of the layout is formulated as the square summation of the differences between desirable distances and real ones for all pairs of vertices.

## 2.2. A Dynamically Balanced Spring System

The description below closely follows Kamada and Kawai's (1989) terminology and methodology, extending their algorithm from positioning nodes in a plane to positioning nodes in three dimensions. The algorithm considers a dynamic three dimensional structure in which  $n$  particles are connected by springs. Vertices,  $v$ , in the graph correspond to particles,  $p$ , in the system. The total energy of the system of springs is given by

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{1}{2} k_{ij} (|p_i - p_j| - l_{ij})^2 \tag{1}$$

Balanced layouts are obtained by decreasing the energy ( $E$ ) of the system of springs. The all pairs shortest path algorithm is used to find the shortest distance  $d_{ij}$  between vertices.  $l_{ij}$  is the length of the spring between two particles,  $p_i$  and  $p_j$ . The distance  $d_{ij}$  between two vertices  $v_i$  and  $v_j$  in a graph is the length of the shortest path between  $v_i$  and  $v_j$ . With  $L$  the final length of an edge (spring), length  $l_{ij}$  is:

$$l_{ij} = L * d_{ij} \tag{2}$$

Absolute display space can be restricted based upon the distance between the maximum distance between pairs in a graph:

$$L = L_o / \max d_{ij} \tag{3}$$

where  $i < j$  and is the length of a side in the square display area.

$k_{ij}$  is the strength of the spring between  $p_i$  and  $p_j$ . Considering 1 as the square summation of the differences between final, minimum energy distances for vertices and current distances for all pairs of particles, is

$$k_{ij} = K / d_{ij}^2$$

4

where K is a constant expressing spring force.

The energy of the spring depends only on the force constant and the length to which it is stretched from its equilibrium position. The constant L<sub>side</sub> (refer appendix source code), is an arbitrary constant that scales the distances between the nodes, the scaled value is considered to be the length of the extended spring. The iterative process (Newton-Raphson method) then works to minimize the energy contained in the extended spring system.

### 2.3. Positioning Nodes in Three Dimensions

The position of the particles in a three dimensional space are given by their coordinate variables respectively, where x , y and z are the coordinate values. The energy E defined by 1 can be rewritten using these 3n variables as follows:

$$E = \sum \sum \frac{1}{2} k_{ij} \left\{ (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 + l_{ij}^2 - 2l_{ij} \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \right\} \quad .5$$

The algorithm computes the values of the 3n variables which minimize E(x<sub>1</sub>, x<sub>2</sub>, ..., x<sub>n</sub>, y<sub>1</sub>, y<sub>2</sub>, ..., y<sub>n</sub>, z<sub>1</sub>, z<sub>2</sub>, ..., z<sub>n</sub>). The components of the gradient of energy are not independent, arbitrary functions. It is therefore relatively hard to find a minimum of a multi-dimensional gradient. Since it is difficult to compute the value of the minimum energy for the entire system, we compute a local minimum and iterate the steps involved to compute the global minimum. There are no good, general methods for solving systems of more than one non-linear equation. Minimization in multi-dimensions is not equivalent to finding a zero of an N-dimensional gradient vector. The components of the gradient vector are not independent, arbitrary functions. The test of downhillness is one-dimensional. There is no analogous conceptual procedure for finding a multi-dimensional root, where downhill must mean simultaneously downhill in N separate function spaces, thus allowing a multitude of trade-offs, as to how much progress in one dimension is worth compared with progress in another. The efficient algorithms for finding minima rest on global and local minima indiscriminately.

The algorithm that computes the local minimum is based on the Newton - Raphson method. Perhaps the most celebrated of all one-dimensional root-finding routines is Newton's method, also called Newton-Raphson method. It readily generalizes to multiple dimensions and hence we chose to use this method. This method is distinguished from other root finding methods by the fact that it requires the evaluation of both f(x), and the derivative f'(x), at arbitrary points x. This method covers quadratically when the search interval envelops the root. However, it could prove disastrous if the search interval is far from the root. However in this spring system, it is a good choice because we control indirectly the root since we pre-define a minimum energy and stop the iteration when the minimum energy required is attained. The following condition must hold for the local minimum:

$$\frac{\partial E}{\partial x_m} = \frac{\partial E}{\partial y_m} = \frac{\partial E}{\partial z_m} = 0 \text{ for } 1 \leq m \leq n.$$

6

Satisfying 6 is the dynamic state that balances the forces of individual springs. The following are the partial derivatives of 5 with respect to x<sub>m</sub>, y<sub>m</sub>, and z<sub>m</sub>.

$$\frac{\partial E}{\partial x_m} = \sum k_{mi} \left\{ (x_m - x_i) - \frac{l_{mi} (x_m - x_i)}{\left( (x_m - x_i)^2 + (y_m - y_i)^2 + (z_m - z_i)^2 \right)^{1/2}} \right\} \quad 7$$

$$\frac{\partial E}{\partial y_m} = \sum k_{mi} \left\{ (y_m - y_i) - \frac{l_{mi} (y_m - y_i)}{\left( (x_m - x_i)^2 + (y_m - y_i)^2 + (z_m - z_i)^2 \right)^{1/2}} \right\} \quad 8$$

$$\frac{\partial E}{\partial z_m} = \sum k_{mi} \left\{ (z_m - z_i) - \frac{l_{mi} (z_m - z_i)}{\left( (x_m - x_i)^2 + (y_m - y_i)^2 + (z_m - z_i)^2 \right)^{1/2}} \right\}. \quad 9$$

These 3n simultaneous non-linear equations must be solved. However, the Newton - Raphson method can not be directly applied, because the equations are not independent.

#### 2.4. Computing a Local Minimum

Hence we compute only a local minimum of E by considering E to be a function of only  $x_m$ ,  $y_m$ , and  $z_m$ . This local minimum can be computed by a 3n-dimensional Newton - Raphson method. The above step is iterated until the required minimum energy of the system is obtained. During each iteration the system can be described as follows. Each iteration compares the particle's energy which has the maximum value of  $\nabla_m$ , which can be defined as follows

$$\nabla_m = \sqrt{\left\{ \frac{\partial E}{\partial x} \right\}^2 + \left\{ \frac{\partial E}{\partial y} \right\}^2 + \left\{ \frac{\partial E}{\partial z} \right\}^2}. \quad 10$$

to the next particle 's energy. Thus one particle at a time is fixed and compared with the maximum energy and moved to it's stable position. Mathematically the process can defined by the following equations:

$$x_m^{t+1} = x_m^t + \delta x, \quad y_m^{t+1} = y_m^t + \delta y, \quad z_m^{t+1} = z_m^t + \delta z \quad \text{for } t = 0, 1, 2, \dots \quad 11$$

$\delta x$ ,  $\delta y$ , and  $\delta z$  satisfy the linear equations 12, 13, and 14.

$$-\frac{\delta E}{\delta x_m}(x_m^t, y_m^t, z_m^t) = \frac{\delta^2 E}{\delta x_m^2}(x_m^t, y_m^t, z_m^t)\delta x + \frac{\delta^2 E}{\delta x_m \delta y_m}(x_m^t, y_m^t, z_m^t)\delta y + \frac{\delta^2 E}{\delta x_m \delta z_m}(x_m^t, y_m^t, z_m^t)\delta z \quad 12$$

$$-\frac{\delta E}{\delta y_m}(x_m^t, y_m^t, z_m^t) = \frac{\delta^2 E}{\delta y_m^2}(x_m^t, y_m^t, z_m^t)\delta y + \frac{\delta^2 E}{\delta y_m \delta x_m}(x_m^t, y_m^t, z_m^t)\delta x + \frac{\delta^2 E}{\delta y_m \delta z_m}(x_m^t, y_m^t, z_m^t)\delta z \quad 13$$

$$-\frac{\delta E}{\delta z_m}(x_m^t, y_m^t, z_m^t) = \frac{\delta^2 E}{\delta z_m^2}(x_m^t, y_m^t, z_m^t)\delta z + \frac{\delta^2 E}{\delta z_m \delta x_m}(x_m^t, y_m^t, z_m^t)\delta x + \frac{\delta^2 E}{\delta z_m \delta y_m}(x_m^t, y_m^t, z_m^t)\delta y \quad 14$$

The coefficients of the equations 12, 13, and 14 are the elements of a Jacobian matrix and are computed

from the partial derivatives of 7 , 8, and 9 with respect to only  $x_m$ ,  $y_m$ , and  $z_m$  in equation 15 - 23:

$$\frac{\partial^2 E}{\partial x_m^2} = \sum k_{mi} \left\{ 1 - \frac{l_{mi} \{(y_m - y_i)^2 + (z_m - z_i)^2\}}{\{(x_m - x_i)^2 + (y_m - y_i)^2 + (z_m - z_i)^2\}^{\frac{3}{2}}} \right\} \quad 15$$

$$\frac{\partial^2 E}{\partial x_m^2} = \sum k_{mi} \left\{ 1 - \frac{l_{mi} \{(y_m - y_i)^2 + (z_m - z_i)^2\}}{\{(x_m - x_i)^2 + (y_m - y_i)^2 + (z_m - z_i)^2\}^{\frac{3}{2}}} \right\} \quad 15$$

$$\frac{\partial^2 E}{\partial y_m \partial x_m} = \sum k_{mi} \frac{l_{mi} \{(x_m - x_i)^2 + (y_m - y_i)^2\}}{\{(x_m - x_i)^2 + (y_m - y_i)^2 + (z_m - z_i)^2\}^{\frac{3}{2}}} \quad 17$$

$$\frac{\partial^2 E}{\partial x_m \partial z_m} = \sum k_{mi} \frac{l_{mi} \{(x_m - x_i)^2 + (z_m - z_i)^2\}}{\{(x_m - x_i)^2 + (y_m - y_i)^2 + (z_m - z_i)^2\}^{\frac{3}{2}}} \quad 18$$

$$\frac{\partial^2 E}{\partial z_m \partial x_m} = \sum k_{mi} \frac{l_{mi} \{(x_m - x_i)^2 + (z_m - z_i)^2\}}{\{(x_m - x_i)^2 + (y_m - y_i)^2 + (z_m - z_i)^2\}^{\frac{3}{2}}} \quad 19$$

$$\frac{\partial^2 E}{\partial y_m \partial z_m} = \sum k_{mi} \frac{l_{mi} \{(y_m - y_i)^2 + (z_m - z_i)^2\}}{\{(x_m - x_i)^2 + (y_m - y_i)^2 + (z_m - z_i)^2\}^{\frac{3}{2}}} \quad 20$$

$$\frac{\partial^2 E}{\partial z_m \partial y_m} = \sum k_{mi} \frac{l_{mi} \{(y_m - y_i)^2 + (z_m - z_i)^2\}}{\{(x_m - x_i)^2 + (y_m - y_i)^2 + (z_m - z_i)^2\}^{\frac{3}{2}}} \quad 21$$

$$\frac{\partial^2 E}{\partial z_m^2} = \sum k_{mi} \left\{ 1 - \frac{l_{mi} \{(z_m - z_i)^2 + (x_m - x_i)^2\}}{\{(x_m - x_i)^2 + (y_m - y_i)^2 + (z_m - z_i)^2\}^{\frac{3}{2}}} \right\} \quad 22$$

$$\frac{\partial^2 E}{\partial y_m^2} = \sum k_{mi} \left\{ 1 - \frac{l_{mi} \{(x_m - x_i)^2 + (z_m - z_i)^2\}}{\{(x_m - x_i)^2 + (y_m - y_i)^2 + (z_m - z_i)^2\}^{\frac{3}{2}}} \right\} \quad 23$$

The unknowns  $\delta x$ ,  $\delta y$ , and  $\delta z$  found using equations 12 - 23. Iteration stops when some criterion is reached for  $\nabla_m$  at  $(x_m^t, y_m^t, z_m^t)$

By solving 12, 13, and 14 simultaneously and substituting for the unknown terms from equations 15 through 23 the unknowns  $\delta x$ ,  $\delta y$ , and  $\delta z$  can be determined as follows.

$$\delta y = \left\{ \left( \left( \frac{\partial E}{\partial z} * \frac{\partial^2 E}{\partial x \partial y} \right) - \left( \frac{\partial E}{\partial y} * \frac{\partial^2 E}{\partial x \partial z} \right) \right) * \left( \left( \frac{\partial^2 E}{\partial x \partial z} * \frac{\partial^2 E}{\partial x \partial y} \right) - \left( \frac{\partial^2 E}{\partial x} * \frac{\partial^2 E}{\partial y \partial z} \right) \right) \right. \\ \left. - \left( \left( \frac{\partial E}{\partial y} * \frac{\partial^2 E}{\partial x^2} \right) - \left( \frac{\partial E}{\partial x} * \frac{\partial^2 E}{\partial x \partial y} \right) \right) * \left( \left( \frac{\partial^2 E}{\partial z \partial y} * \frac{\partial^2 E}{\partial x \partial z} \right) - \left( \frac{\partial^2 E}{\partial x \partial y} * \frac{\partial^2 E}{\partial z} \right) \right) \right\} \\ \left\{ \left( \left( \frac{\partial^2 E}{\partial y} * \frac{\partial^2 E}{\partial x \partial z} \right) - \left( \frac{\partial^2 E}{\partial x \partial y} * \frac{\partial^2 E}{\partial z \partial y} \right) \right) * \left( \left( \frac{\partial^2 E}{\partial x \partial z} * \frac{\partial^2 E}{\partial x \partial y} \right) - \left( \frac{\partial^2 E}{\partial x} * \frac{\partial^2 E}{\partial y \partial z} \right) \right) \right. \\ \left. - \left( \left( \frac{\partial^2 E}{\partial x \partial y} * \frac{\partial^2 E}{\partial x \partial y} \right) - \left( \frac{\partial^2 E}{\partial y} * \frac{\partial^2 E}{\partial x} \right) \right) * \left( \left( \frac{\partial^2 E}{\partial z \partial y} * \frac{\partial^2 E}{\partial x \partial z} \right) - \left( \frac{\partial E}{\partial x \partial y} * \frac{\partial^2 E}{\partial z} \right) \right) \right\}$$

24

$$\delta z = \left( \left( \frac{\partial E}{\partial z} * \frac{\partial^2 E}{\partial y \partial x} \right) - \left( \frac{\partial E}{\partial y} * \frac{\partial^2 E}{\partial z \partial x} \right) - \left( \frac{\partial^2 E}{\partial y} * \frac{\partial^2 E}{\partial z \partial x} \right) \delta y - \left( \frac{\partial^2 E}{\partial y \partial x} + \frac{\partial^2 E}{\partial z \partial y} \right) \delta y \right) \\ \left( \left( \frac{\partial^2 E}{\partial y \partial z} * \frac{\partial^2 E}{\partial z \partial x} \right) \delta z - \left( \frac{\partial^2 E}{\partial z} * \frac{\partial^2 E}{\partial y \partial x} \right) \delta z \right)$$

25

$$\delta x = \left( - \frac{\partial E}{\partial z} - \left( \frac{\partial^2 E}{\partial z} * \delta z \right) - \left( \frac{\partial^2 E}{\partial z \partial y} * \delta y \right) \right) \\ \frac{\partial^2 E}{\partial z \partial x}$$

26

## 2.5. Algorithm for Positioning Vertices in Three Dimensions

Figure 1 below summarizes the algorithm for minimizing energy in a system in which vertices move in three dimensions. The input to the algorithm is the distance  $d_{ij}^d$  which is the shortest path between all pairs of vertices. The shortest pair algorithm used is also outlined below.

*All Pairs Shortest Path:*

```
input adjacency matrix of graph with n vertices, cost[1..n][1..n];
initialize shortest_path matrix, a[i,j]
initialize integers i , j and k
begin
  for i:= 1 to n do
    for j:= 1 to n do
      a[i,j] := cost[i,j]; {copy cost into a}
    for k:= 1 to n do { for a path with highest vertex index k}
      for i:=1 to n do {for all possible pairs of vertices}
        for j:=1 to n do
          if (a[i,k] + a[k,j]) < a[i,j]
            then a[i,j] := a[i,k] + a[k,j];
        end; (all pairs shortest path)
```

### Vertex Positioning Algorithm:

```

compute  $d_{ij}$  for  $1 \leq i \neq j \leq n$ ;
compute  $l_{ij}$  for  $1 \leq i \neq j \leq n$ ;
compute  $k_{ij}$  for  $1 \leq i \neq j \leq n$ ;
initialize  $P_1, P_2, \dots, P_n$ ;
while ( $P_m$  {
  let  $P_m$  be the particle satisfying  $\nabla_m = \max_i \nabla_i$ 
  while ( $\nabla_m > \epsilon$ ) {
    compute  $\delta x$  and  $\delta y$  by solving 12, 13, and 14
     $x_m := x_m + \delta x$ ;
     $y_m := y_m + \delta y$ ;
     $z_m := z_m + \delta z$ ;
  }
}

```

Figure 1: A spring modeling algorithm for positioning vertices in three dimensions

As discussed in Kamada and Kawai (1989) the running time for this algorithm is  $O(n^3)$  for the all pairs shortest path algorithm. For the main algorithm  $O(n)$  is the time required for the inner while loop (using the Newton - Raphson method) to compute the  $\nabla_m$  and to compute the  $\delta x$ ,  $\delta y$  and  $\delta z$ . The time required for the outer while loop is  $O(n)$ . Thus the total time required to terminate the while loops is  $O(T \cdot n)$  where T is the total number of inner loops. The total time required remains the same for particles represented in a two dimensional plane also. T depends both on the size and connectivity of a specific graph and on the initial positions of vertices. The algorithm is designed in such a manner that the number of iterations to be performed can be controlled effectively by lowering the convergence precision ( $\epsilon$ ) of the vertices.

### 3. Example Displays

The spring embedder algorithm extended to three dimensions exhibits the same ability to clearly display symmetries and make isomorphisms apparent as spring embedder algorithms for graph drawing in a plane. In addition a three dimensional display provides a means for overcoming some of the difficulties which remain for edge crossings by allowing the viewer to manipulate the graph in an interactive system so that projections can be found which "uncross" some edges.

Much of the utility of three dimensional graph viewing lies in the ability to convey, or allow the viewer to discover, more information by interactive viewing than is possible in a two dimensional display. As an example, Figure 2 is the layout of a weighted graph presented in Kamada and Kawai's paper. Using the original data the three dimensional solution is presented from three different views with lighting and perspective. The first, presented in Figure 3, is a view aligning the vertices of the three dimensional solution with the layout in a plane. Figure 4 is a view of vertex layout in three dimensions from the left hand side of the graph. The graph is viewed from "beneath" in Figure 5.



Figure 2. A weighted graph from Kamada and Kawaii (1989) drawn with vertices positioned using a spring algorithm in two dimensions.

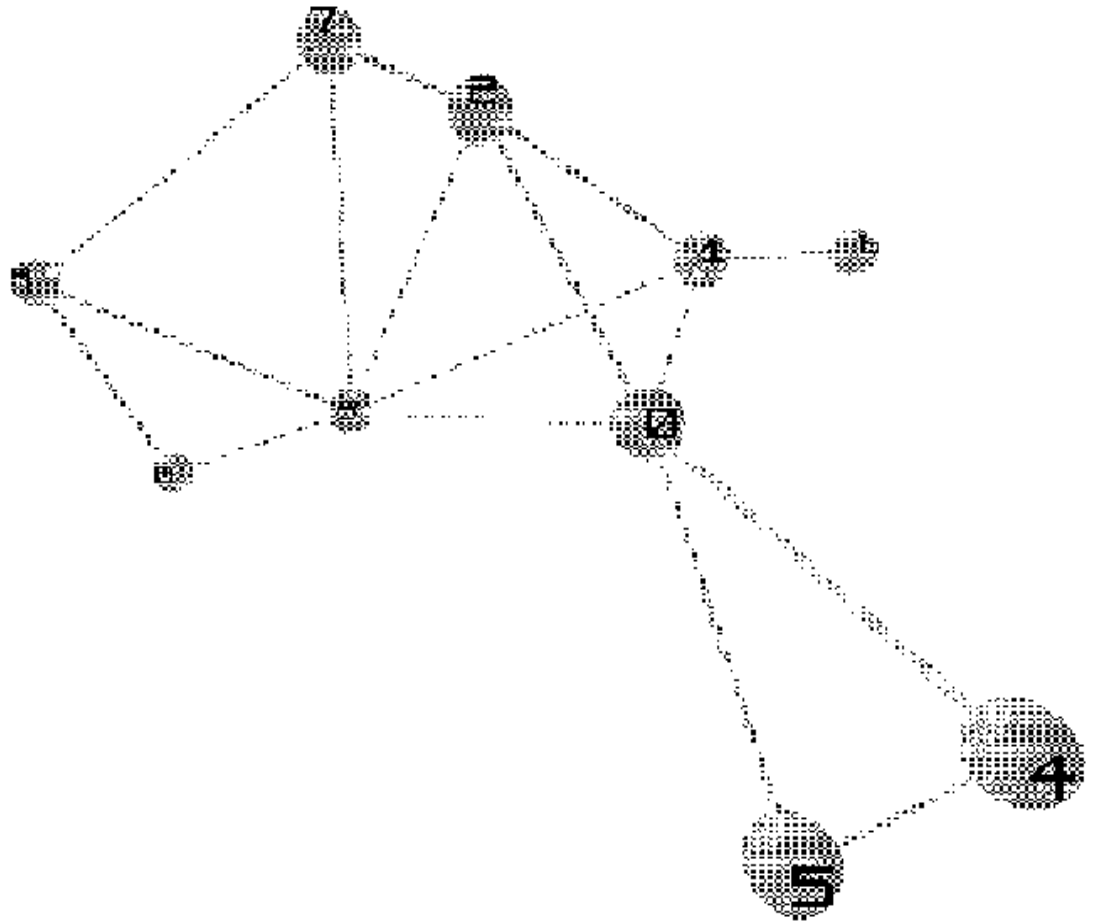


Figure 3. Three dimension vertex positioning aligned to correspond to Figure 2's vertex positioning in a plane.

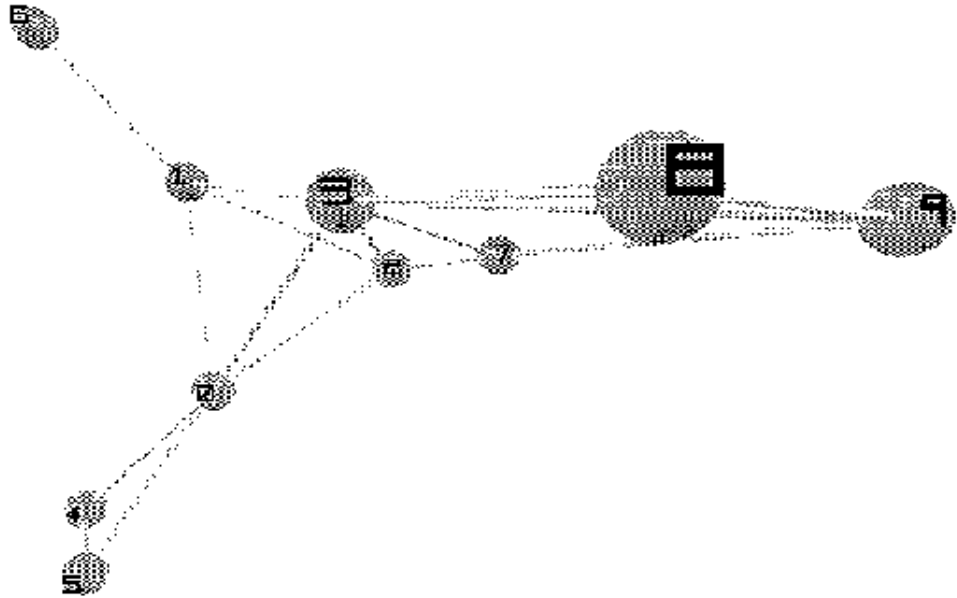


Figure 4. Three dimension solution viewed from left end of Figures 2 and 3. The relationships among vertices 4, 5, and 6 at the right are clearly shown.

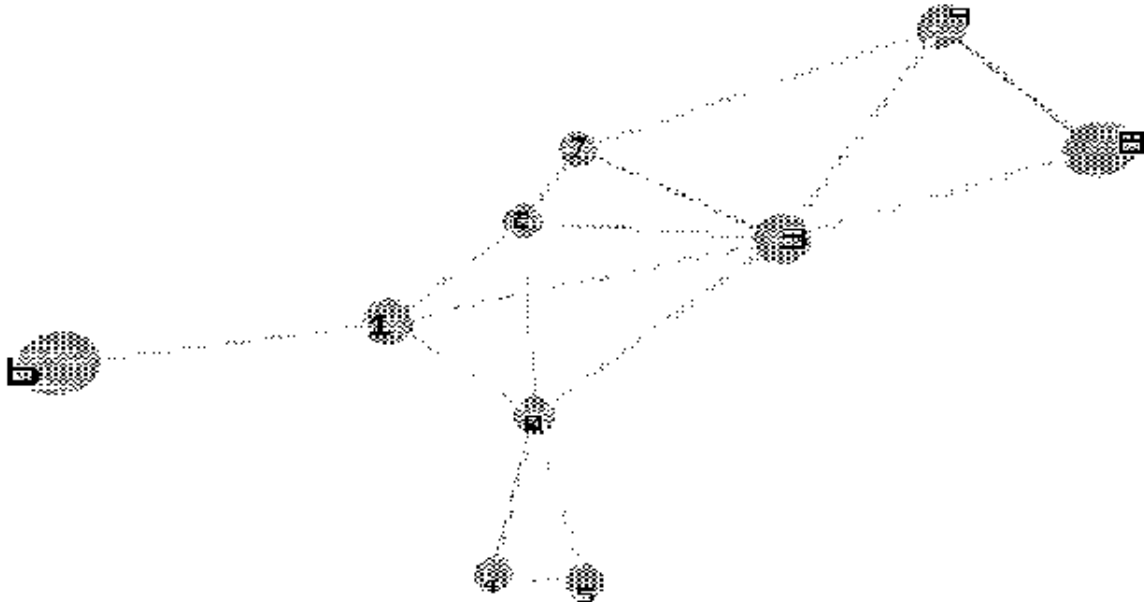


Figure 5. Three dimension solution viewed from "beneath" the views of Figures 2 and 3.

#### 4. Conclusion

The extension of a spring embedding algorithm to three dimensions affords one means to enhance viewing in an interactive graph visualization system. Such a system would allow the user to focus on various aspects which are of interest and provides an enriched display mechanism compared to the typical two dimensional static display.

#### 5. Acknowledgments

Work described in this paper was supported by NASA NAG9-551 to the second author.

Development of this algorithm was initiated by Brad Wilson using a program written by Roger Schvaneveldt and Jim McDonald developed from Kamada and Kawai (1989).

#### 6. References

- Chiba, N., Yamanouchi, T., & Nishizeki, T. (1984). Linear algorithms for convex drawings of planar graphs. In Bondy, J. A. & Murty, U. S. R. (Eds.) *Progress in Graph Theory*. Academic Press, 153-173.
- Eades, P. (1984). A heuristic for graph drawing. *Congressus Numerantium*, May, 149-160.

Eades, P. & Tamassia, R. (1989). Algorithms for drawing graphs: An annotated bibliography. Technical Report, CS-89-09, Brown University.

Eades, P. (1984). A heuristic for graph drawing. *Congressus Numerantium*, 42, 149-160.

Esposito, C. (1988). Graph graphics: Theory and practice. *Computer Mathematics Applications*, 15(4), 247-253.

Jablonowsky, D. & Guarna, V. A. (1983). GMB: A tool for manipulating and animating graph data structures. *Software Practice and Experience*, 19(3), 283-301.

Kamada, T. & Kawai, S. (1989). An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31, 7-15.

Lipton, R. J. (1985). A method for drawing graphs. *Proceedings of the 1st ACM Symposium on Computational Geometry*, 153-160.

Makinen, E. (1988). On circular layouts. *International Journal of Computing Mathematics*, 24, 29-37.

Reingold, E. E. & Tilford, J. S. (1981). Tidier drawings of trees. *IEEE Transactions on Software Engineering*, SE-7, 223-228.

Robins, G. (1987). *The ISI Grapher*. Information Sciences Institute. Marina Del Ray, CA.

Rowe, L. A., Davis, M., Messinger, E., Meyer, C., Spirakis, C., & Tuan, A. (1987). A browser for directed graphs. *Software Practice and Experience*, 17(1), 61-76.

Sugiyama, K., Tagawa, S., & Toda, M. (1981). Methods for visual understanding of hierarchical system structures, *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11, 109-125.

Tamassia, R. (1987). On embedding a graph in the grid with the minimum number of bends. *SIAM Journal of Computing*, 16, 421-444.

Tamassia, R., Di Battista, G & Battini, C. (1988). Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1), 61-79.

---

*Comments and questions to: [fowler@panam.edu](mailto:fowler@panam.edu)  
Last update October 7, 1996*