1 Algorithm

algorithm 1 Degree First

input: $d[n][n]$:distance between each pair of vertices
output: $p[n][3]$:position of vertices

function INIT($d[n][n]$)
    sort the vertices in descending order depending on the degree of each vertices(degree is the number of connected lines for each vertex) and generate $desc[n]$ array
    initialize $state[n]$ array which describes the current state of each vertex:0 represents unplaced vertices, 1 represents vertices that have been placed but not moved, and 2 represents vertices that have been moved
    take out the first element in $desc[n]$ array, and put it on the (0,0,0), in addition, change the state of this vertex into 1
end function

function MAIN($d[n][n], state[n], desc[n]$)
    for $i = 0 \rightarrow n - 1$ **do**
        $p \leftarrow$ find the first vertex in $state[n]$ whose state is 1
        $p_x \leftarrow position[p_(index)][0]$
        $p_y \leftarrow position[p_(index)][1]$
        $p_z \leftarrow position[p_(index)][2]$
        $degree_(current) \leftarrow$ find the number of vertices whose state is 1 or 2 which connected with p
        for $j = 0 \rightarrow 100$ **do**
            for $k = 0 \rightarrow degree_(current)$ **do**
                $c_x \leftarrow$ position$[degree_(current)][0]$
                $c_y \leftarrow$ position$[degree_(current)][1]$
                $c_z \leftarrow$ position$[degree_(current)][2]$
                $rate \leftarrow 0.1$
                $p_x \leftarrow p_x + rate \times \sqrt{(c_x - p_x)^2 + (c_y - p_y)^2 + (c_y - p_y)^2} - \frac{d[p_(index)][degree_(current)] \times (c_x - p_x)}{\sqrt{(c_x-p_x)^2+(c_y-p_y)^2+(c_y-p_y)^2}}$
                $p_y \leftarrow p_y + rate \times \sqrt{(c_x - p_x)^2 + (c_y - p_y)^2 + (c_y - p_y)^2} - \frac{d[p_(index)][degree_(current)] \times (c_y - p_y)}{\sqrt{(c_x-p_x)^2+(c_y-p_y)^2+(c_y-p_y)^2}}$
                $p_z \leftarrow p_z + rate \times \sqrt{(c_x - p_x)^2 + (c_y - p_y)^2 + (c_y - p_y)^2} - \frac{d[p_(index)][degree_(current)] \times (c_z - p_z)}{\sqrt{(c_x-p_x)^2+(c_y-p_y)^2+(c_y-p_y)^2}}$
            end for
        end for
        $position[p_(index)][0] \leftarrow p_x$
        $position[p_(index)][1] \leftarrow p_y$
        $position[p_(index)][2] \leftarrow p_z$
    end for
    find every vertices which are connected with the p and haven't been processed, put them randomly around the p, the distance between them to p depends on $d[n][n]$. In addition, change the state of these vertices to 2 in $state[n]$ array.
end function

The purpose of algorithm is to minimize the error between the total actual distance and total ideal distance. To satisfy it, we put vertices on the canvas in some sequence. The general idea is to put the vertex with the larger degree on the canvas. We first move the position of this vertex and expand it.

As for the movement, we find the correlation vertex of the current vertex (the correlation vertex is the vertex directly connected with the current vertex and has been on the canvas). If the vertex has no related vertices, skip this movement step; if the vertex has related vertices, the indices of the related vertices will be put into a temporary array, and one related vertex will be taken from the temporary array each time. The actual distance between the related vertex and the current vertex will be calculated and compared with the ideal distance (ideal distance is recorded in the distance relation array). If the actual distance is greater than the ideal distance, change the position of the current vertex, move the current vertex toward the relevant vertex along straight line of two vertices (the ratio of the moving distance can be set by the operator); if the actual distance is less than the ideal distance, change the position of the current vertex and move the current vertex away from the related vertex along straight line of two vertices; if the actual distance is equal to the ideal distance, current vertex doesn't move. After traversing the temporary array, movement step will repeat several times (the repetition number can be set by the operator). After moving the current vertex many times, the vertex position is determined and recorded in the position array.

As for the expansion, we find the vertices that are directly connected to the current vertex and haven't been on the canvas. If there are no vertices that meet the requirements, skip this expansion step; if there are vertices that meet the requirements, they will be randomly placed on the sphere with the current vertex as the center and the respective ideal distance as the radius according to the ideal distance from the current vertex.

When all vertices are moved and expended, the operation ends.