Layout and Display of Network Graphs on a Sphere

Anshul Guha[1] and Eliot Feibush[2(✉)]

[1] Yale University, New Haven, USA
[2] Princeton Plasma Physics Laboratory, Princeton, NJ, USA
efeibush@pppl.gov

Abstract. There are many advantages to displaying a network graph on a sphere instead of on a plane. For instance, 3D models can be rotated so that any node is in the center of the user's field of view. Moreover, an opaque sphere provides a natural filtering mechanism for selecting a small subset of data to display. However, the spherical geometry presents some challenges to modeling and displaying a graph, and the literature for 3D layout algorithms is not as rich as that of 2D layout algorithms. We developed a Visualization Pipeline to parse input data and visualize it on a 3D sphere. Within this pipeline, we have developed a 3D version of the Fruchterman-Reingold Layout Algorithm, and also present a method for creating 3D arcs that connect the nodes on a sphere. We created four modified force-directed algorithms, and determined which of their objective functions produced graphs with more evenly-distributed nodes. Our implementation functions in readily available 3D visualization programs and our browser-based display functions on all common operating systems and devices.

Keyword: 3D force directed graph algorithm

1 Introduction

Graphs are formally defined as an ordered pair (V, E), where V is a set of vertices and E is a set of edges that connect pairs of vertices. Node-link diagrams, which display vertices as points and edges as straight lines connecting vertices, are the most common method for visualizing graphs. Currently, most graph visualization research assumes that:

1. All vertices lie on a plane.
2. All edges are straight or curved lines in the same plane as the vertices.

These assumptions are quite reasonable since they ensure that the graphs can be printed on a flat 2D surface (paper) without distortion. Furthermore, research into 3D graphs has often focused on graphs where vertices are arbitrarily allowed

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-3-031-23473-6_6.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
N. Magnenat-Thalmann et al. (Eds.): CGI 2022, LNCS 13443, pp. 67–78, 2022.
https://doi.org/10.1007/978-3-031-23473-6_6

to lie anywhere in the 3D space. Although some graphs are better visualized in higher dimensions, or in non-euclidean geometries. This research is motivated by the need to show the influence of one research paper on subsequent papers. Even a modest 2D graph with 50 nodes and 200 edges can be cluttered so it is difficult to follow the connections. Modeling the graph in three dimensions significantly expands the space for locating nodes. Positioning the nodes on a sphere provides a strong visual context that is very familiar to people exploring a graph.

We developed a Visualization Pipeline to parse raw data and display the graphs on a sphere. Within this visualization pipeline we explore a variety of algorithms to display graphs on a sphere.

We implement four different force-directed algorithms for displaying graphs, all of which are inspired by the original Fruchterman-Reingold algorithm [6]. These algorithms have been adapted to address several problems which arise when force-directed diagrams are carried over to convex surfaces (like the sphere). For example,

1. Unlike planes, vectors which lie in a tangent plane to a sphere and have a head on the sphere do not have a tail on the sphere.
2. If two vertices repel each other with too much force, they can actually end up closer to each other on the other side of the sphere.

In this paper we analyze citation maps - graphs where each node corresponds to a separate scientific paper and each edge represents the event of one paper citing another - as our input data. However, the algorithms and software written apply generally to all graphs. Modeling graphs on a sphere requires connecting the nodes in each edge pair using curves that lie on the 3D sphere. As a simplifying assumption, we chose to connect these nodes with a 3D arc along the unique great circle path that connects them, since the great circle path is the shortest distance between any two points on the sphere. However connecting nodes with non-great-circle curves is possible, particularly if these paths are uniquely suited to avoid node and edge crossings.

The rest of the paper proceeds as follows: In Sect. 2 we give an overview of related work on force-directed algorithms and spherical graph visualization. Section 3 provides a high-level overview of the pipeline that transforms raw data into a spherical visualization in either Paraview or a browser. In Sect. 4 we describe the force-directed algorithms that calculate the final locations of the nodes on the sphere and provide the mathematical basis for drawing arcs on a sphere. Section 5 has an overview of the implementation of the browser-based visualization, followed by conclusions in Sect. 6.

2 Related Work

Force-directed algorithms were first explored by Tutte [14], whose algorithm relied on the barycentric representations of vertices on a 2D plane. His idea was revisited in 1984 by Eades [5] and in 1991 by Fruchterman and Reingold [6]. Both of these papers define attractive and repulsive forces, modeling the interactions between every pair of vertices by Hooke's Law. These papers paved the way for

more force-directed layout algorithms to be proposed. For instance, ForceAtlas and ForceAtlas2 were published by the creators of the 2D graph visualization software Gephi [2].

Spherical graph visualization has been most recently studied by Perry, Yin, Gray, and Kobourov [10]. They propose two algorithms for spherical visualization; one which computes a 2D visualization and another which uses spherical multi-dimensional scaling. In addition, they have also constructed a pipeline to parse data and create an in-browser spherical visualization.

Some other authors have also experimented with spherical visualization. For example, Munzner [9] has constructed many spherical and hyperbolic tree visualizations, and Sprenger [7] has used concentric spheres in graph visualization in the past.

Additionally, Brath and MacMurchy [3] have conducted a user study on how individuals (stock brokers) react to graph visualizations on a sphere. Moreover, in [12], Schulz presents a spherical graph layout algorithm for graphs with large numbers of nodes. He applies his layout algorithm to a citation map of Web of Science sociology papers. Similar papers have largely concluded that while spherical visualization has some drawbacks (such as being harder to implement), it also have many advantages, such as being able to rotate any vertex to the center of the screen.

Finally, some research has been conducted into 3D force-directed algorithms as well. For instance, Lu and Si [8] propose four clustering-based graph layout algorithms which are successful in reducing edge crossings in 3D graphs.

3 Visualization Pipeline

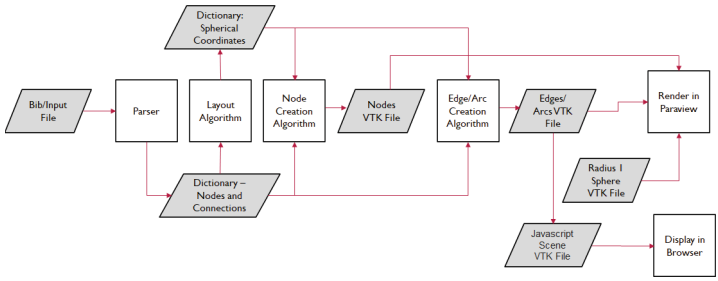


Fig. 1. A flowchart of the steps to convert input data into three separate VTK files representing a spherical graph layout: one to model the unit sphere, another to describe a small sphere for each node, and the third file that models each great circle arc that connects two nodes.

The first step of the pipeline is to run the data through a parser that stores the vertex and edge data as lists. (In our specific implementation, we wrote a

Parser that reads and interprets Bibtex files, so that the nodes of the graph are individual papers and the edges represent instances of papers referencing each other.) Call these lists V and E. Then V and E are passed into the Layout Algorithm, which returns a dictionary S of spherical coordinates - one for each node of the graph (Fig. 1).

S is then passed into a function called CreateNodes. CreateNodes generates a Nodes file containing instructions for the 3D visualization software to create a polygonal mesh approximating a sphere with radius 0.02 around each element of S, thereby creating a set of $3D$ spheres that represent each node in the final graph G. The points and polygonal mesh associated with these $3D$ spheres are written out to Nodes.

Next, S and E are passed into the CreateEdges algorithm. Each edge is actually an arc of a great circle of G. Because VTK has no function to automatically create arcs of spheres, each edge between two nodes at points A and B is approximated by equally spaced points along the arc connecting A and B. In typical views, connecting these points with short line segments renders curved arcs along the surface of the unit sphere.

Finally, a polygonal mesh is generated to model the unit sphere. Separating the components of the graph enables controlling the display and applying selection filters to the data.

Our implementation produces data sets that can be displayed in scientific visualization tools such as Paraview [1] and VisIt [4] which are both based on the Visualization Toolkit software [11]. These programs provide 3D viewing techniques and operators for filtering the nodes and edges in the graph. Both programs are freely available for Windows, Mac OS, and Linux. This approach to interoperability handles 3D viewing across platforms.

For added portability, our implementation generates Javascript files that can be displayed interactively in a browser.

4 Graph Visualization Algorithms

Our overarching goal with the graph layouts was to develop an algorithm that created evenly-distributed nodes with no node overlaps and minimal "dense" edge crossing. Usually, this meant some combination of:

1. Nodes are equally distributed across the sphere.
2. High-degree nodes are far away from each other.

4.1 Layout - Adaptation of Fruchterman-Reingold Algorithm

The Fruchterman-Reingold algorithm (FR) is a well-known layout algorithm which produces high-quality $2D$ graph layouts. To study the effectiveness of an FR-like algorithm on spheres, we developed four different algorithms for spherical graph visualization, each of which is modeled after the original 2-dimensional FR-algorithm. These spherical algorithms have not been previously studied.

The Spherical-FR-Layout Algorithm can be described as follows:

1. Define a static constant $k = \frac{4\pi}{\text{number of nodes}}$
2. Assign a random Cartesian coordinate to every node in V.
3. Run the Update Algorithm until the coordinates converge. (We ran the update algorithm for 50 iterations.)

(Note: Within the code, all rotations are performed by converting Cartesian coordinates to Spherical coordinates, carrying out the necessary rotation, and then converting back to Cartesian coordinates. Cartesian coordinates are used primarily to simplify the vector addition operations, and also because Paraview and THREE.js require Cartesian input.)

The Update-Algorithm can be described as follows:

1. For each node N in V, let $N.pos$ be the the original Cartesian coordinates of the point N. Additionally, for each N, create a vector called $N.update$ with an initial value of $[0, 0, 0]$.
2. Iterate through every pair of nodes (N_1, N_2), where both N_1 and N_2 are in V. Both N_1 and N_2 are modeled as positive electric charges which repel each other with a force proportional to the distance between them. $N_1.update$ and $N_2.update$ are modified so that the coordinates $N_1.pos + N_1.update$ and $N_2.pos + N_2.update$ move apart from each other.
3. Iterate through every pair of nodes (N_3, N_4), where both N_3 and N_4 are in V. N_3 and N_4 can be imagined to be connected by a spring with an equilibrium length of 0, so that the attractive force is directly proportional to the distance between them. Again, $N_1.update$ and $N_2.update$ are modified to move the $N_1.pos + N_1.update$ and $N_2.pos + N_2.update$ coordinates closer to each other.

The difference between the standard FR algorithm and our modified FR algorithm is encapsulated in two details. First, how is the distance between any two nodes calculated? There are two options:

1. The distance between two nodes can be defined to be equal to the Euclidean distance between them.
2. The distance between two nodes can be defined to be equal to the length of the shortest (great-circle) arc on the sphere that connects them.

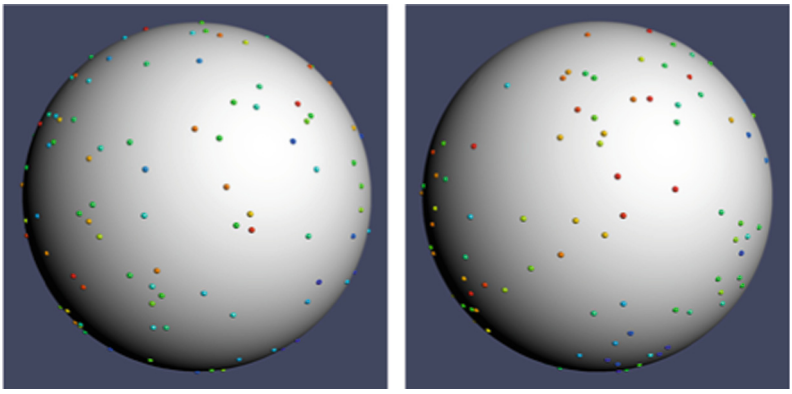
Another important detail to consider is that if $N.pos$ is on a sphere of radius 1, then $N.pos + N.update$ may no longer be on the sphere of radius 1. There are two ways to correct this:

1. Move $N.pos + N.update$ along the 'true' update vector and then to scale N up or down so that its coordinates once again have a radius of 1.
2. Move the point $N.pos$ along the unit sphere in the same direction as $N.update$ and for the same distance as the length of $N.update$.

After implementing all four variations on the FR algorithm, we judged that the graphs that were produced when we defined distance as the minimal arc length and strictly moved the nodes along the unit sphere itself were the most

aesthetically pleasing. A quantitative evaluation of this graph algorithm is provided in Sect. 4.2. These graphs tended to be the most evenly distributed and also contained the least number of node overlaps.

We have created an animation of the movement of nodes showing 50 steps of the Spherical-FR Layout Algorithm. It is available in the supplemental file. In the animation the nodes are initially arranged along lines of latitude and longitude, compared to the random starting locations in Fig. 2a. Both starting arrangements converge to the same final layout shown in Fig. 2b. After varying the relative strength of the attractive and repulsive forces, we found that the vertices always converge to some final location, unlike the $2D$ Fruchterman-Reingold algorithm, where the locations of the nodes sometimes oscillate or enter a chaotic state after many iterations.



(a) A Random Starting configuration for the Spherical-FR-Layout Algorithm.

(b) Final positions of the nodes after 50 iterations. At this point, the node positions have converged to their final locations.

Fig. 2. The movement of nodes in Spherical-FR-Layout Algorithm. This particular dataset contains 186 nodes and 791 edges. The edges are omitted from this example for the sake of clarity.

The 2D Fruchterman-Reingold Algorithm described in [6] differs in one key respect from our 3D adaptation. In particular, the 2D algorithm relies on a "cooling" function that limits the maximum displacement of every node after each iteration of the Update algorithm. This cooling function approaches zero as the number of iterations increases, which effectively forces the convergence of the 2D Fruchterman-Reingold algorithm. Without the cooling function the 2D Fruchterman-Reingold algorithm can lead to oscillating or chaotic behavior, depending on the relative strength of the attractive and repulsive forces. Our algorithm does not depend on a cooling function to ensure that all the nodes converge to a final location. This is because repeated testing with different parameters for the strength of the attractive and repulsive forces in our

algorithms all resulted in the vertices of the graph converging to a stable position. A theoretical explanation of this phenomenon (which is not true in the 2-dimensional case) is a potential avenue for future research.

4.2 Quantitative Evaluation

A quantitative consideration for graph quality is the length of the edges between the nodes. The goal is to eliminate long edges and increase the uniformity of edge length. We calculated the arc length of each edge in the citation graph and plotted histograms of the lengths in Fig. 4. The histogram at an early iteration of the algorithm shows some very long arcs greater than 5 rad. In the final iteration the very long arcs have been eliminated and a larger number of arcs are less than 1 rad. This indicates the progress of the Update Algorithm as it iterates through each pair of nodes.

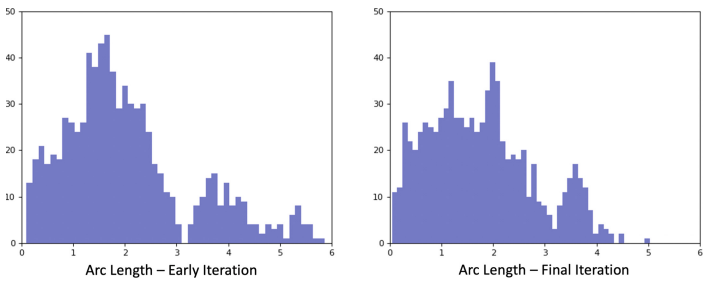


Fig. 3. Histogram of arc lengths (edges between nodes) at early and late iterations of the Update Algorithm.

4.3 Visualizing Spheres in Paraview

VTK files do not have a built-in specification for spheres but a polygonal mesh can be created to model our unit sphere. The points of the polygonal mesh are located at all spherical coordinates of the following forms:

1. $(\varphi, \theta) = \left(\frac{\pi m}{16}, \frac{\pi n}{8}\right)$ with $1 \leq m \leq 15$, $0 \leq n \leq 15$.
2. $(\varphi, \theta) = (0, 0)$ or $(0, \pi)$.

After generating this list of 242 vertices, the UnitSphere.vtk file also defines which coordinates to connect in order to create the 256 polygons of the mesh. For typical visualizations at 1920×1080 resolution, the resulting discrete polygonal mesh is visually indistinguishable from a continuous sphere representation.

Each node in the Nodes file is also created using a modified version of this method, where the spherical coordinates are converted to Cartesian coordinates, scaled down by a factor of 50, and then translated to their correct position. Ideally the spheres in the nodes file would be instances of a single master definition. Unfortunately VTK files do not have this capability. This approach can be implemented through Paraview's Python programming interface in a future version of our visualization pipeline.

4.4 Mathematical Basis for Visualizing Arcs in Paraview

Our method relies on spherical coordinates and rotation matrices to simplify the problem of drawing great-circle arcs on a sphere. To our knowledge, this method has not been described in the previous literature. However, we note that an alternate method for spherical interpolation can be derived by using quaternions to express rotations, as mentioned by Ken Shoemake in [13].

In order to model an arc between any two points A and B on a unit sphere, the key step is to calculate the coordinates of N equidistant points along the arc from A to B. We achieved good results with 64 points per arc.

Let the spherical coordinates of A and B be (φ_1, θ_1) and (φ_2, θ_2). The Cartesian coordinates (a_1, a_2, a_3) and (b_1, b_2, b_3) for A and B respectively can be calculated by the well-known equations, where $i \in \{1, 2\}$:

$$\begin{aligned} x_i &= \cos\varphi_i \sin\theta_i \\ y_i &= \sin\varphi_i \sin\theta_i \\ z_i &= \cos\theta_i \end{aligned} \tag{1}$$

Given the points $A = (a_1, a_2, a_3)$ and $B = (b_1, b_2, b_3)$, the shortest arc A on a unit sphere that connects A and B will be contained entirely in the great circle S that passes through both A and B. S, in turn, will be contained entirely within the plane P passing through A, B, and $O = (0, 0, 0)$.

Notice that the cross product

$$\begin{aligned} \vec{C} = \vec{OA} \times \vec{OB} = \begin{bmatrix} \hat{i} & \hat{j} & \hat{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix} \\ = (a_2b_3 - a_3b_2)\hat{i} + (a_3b_1 - b_3a_1)\hat{j} \\ + (a_1b_2 - a_2b_1)\hat{k} \end{aligned} \tag{2}$$

is perpendicular to P. So, P can be characterized as the set of all points (x, y, z) such that

$$\begin{aligned} 0 = \langle a_2b_3 - a_3b_2, a_3b_1 \\ - b_3a_1, a_1b_2 - a_2b_1 \rangle \cdot \langle x, y, z \rangle \\ \implies 0 = (a_2b_3 - a_3b_2)x + (a_3b_1 - a_1b_3)y \\ + (a_1b_2 - a_2b_1)z \end{aligned} \tag{3}$$

We now attempt to rotate the plane OAB so that it coincides with the xy-plane, in order to eliminate the z-coordinate and simplify the problem. This is equivalent to rotating $\vec{C}$ until it becomes parallel to the z-axis. Therefore we can let $C = (c_1, c_2, c_3) = (a_2b_3 - a_3b_2, a_3b_1 - b_3a_1, a_1b_2 - a_2b_1)$.

Now, applying the equations

$$\begin{aligned} \theta_C &= \arccos\left(\frac{c_3}{\sqrt{{c_1}^2 + {c_2}^2 + {c_3}^2}}\right) \\ \varphi_C &= \operatorname{atan2}(c_2, c_1) \end{aligned} \tag{4}$$

yields the spherical coordinates (φ_C, θ_C) of C. Let $R_{y,-\theta_C}$ and $R_{z,-\varphi_C}$ be the rotation matrices that rotate vectors by $-\theta_C$ degrees clockwise around the y-axis and by $-\varphi_C$ clockwise around the z-axis respectively. Then $C' = R_{y,-\theta_C} R_{z,-\varphi_C} C^T$ will have x and y-coordinates of 0. Similarly, $A' = R_{y,-\theta_C} R_{z,-\varphi_C} A^T$ and $B' = R_{y,-\theta_C} R_{z,-\varphi_C} B^T$ have z-coordinates of 0. Let

$$A' = \begin{bmatrix} a'_1 \\ a'_2 \\ 0 \end{bmatrix} \text{ and } B' = \begin{bmatrix} b'_1 \\ b'_2 \\ 0 \end{bmatrix} \tag{5}$$

Then one can define $\theta_A = \text{atan2}(a'_2, a'_1)$ and $\theta_B = \text{atan2}(b'_2, b'_1)$. The coordinates of N evenly spaced points from A' to B' are now much easier to parameterize:

1. In the case where $\theta_1 < \theta_2$ and $\theta_1 + \pi > \theta_2$, the points are of the form

$$S_i = \begin{bmatrix} \cos\left(\theta_1 + \frac{(\theta_2 - \theta_1)i}{N}\right) \\ \sin\left(\theta_1 + \frac{(\theta_2 - \theta_1)i}{N}\right) \\ 0 \end{bmatrix} \text{ with } 0 \leq i < N \tag{6}$$

2. In the case where $\theta_1 < \theta_2$ and $\theta_1 + \pi < \theta_2$, let $\theta_3 = \theta_1 + 2\pi$. Then the points are of the form

$$S_i = \begin{bmatrix} \cos\left(\theta_2 + \frac{(\theta_3 - \theta_2)i}{N}\right) \\ \sin\left(\theta_2 + \frac{(\theta_3 - \theta_2)i}{N}\right) \\ 0 \end{bmatrix} \text{ with } 0 \leq i < N \tag{7}$$

3. In the case where $\theta_1 > \theta_2$ and $\theta_2 + \pi < \theta_2$, the points are of the form

$$S_i = \begin{bmatrix} \cos\left(\theta_2 + \frac{(\theta_1 - \theta_2)i}{N}\right) \\ \sin\left(\theta_2 + \frac{(\theta_1 - \theta_2)i}{N}\right) \\ 0 \end{bmatrix} \text{ with } 0 \leq i < N \tag{8}$$

4. In the case where $\theta_1 > \theta_2$ and $\theta_2 + \pi > \theta_1$, let $\theta_3 = \theta_2 * 2\pi$. Then the points are of the form

$$S_i = \begin{bmatrix} \cos\left(\theta_1 + \frac{(\theta_3 - \theta_1)i}{N}\right) \\ \sin\left(\theta_1 + \frac{(\theta_3 - \theta_1)i}{N}\right) \\ 0 \end{bmatrix} \text{ with } 0 \leq i < N \tag{9}$$

The final step is to rotate A' and B' back to A and B. Notice that $A = R_{y,\varphi_C} R_{x,\theta_C} A'$, and $B = R_{y,\varphi_C} R_{x,\theta_C} B'$, as well as all points of the form S_i for $0 \leq i < 64$. Let $S'_i = R_{y,\varphi_C} R_{x,\theta_C} S_i$. Then the set of S'_i are the coordinates of N equidistant points on the shortest arc from A to B. Adding S'_i to the list of points in ArcEdges.txt allows these points to be rendered by Paraview, giving the appearance of a solid spherical arc connecting A and B, as in Fig. 4.

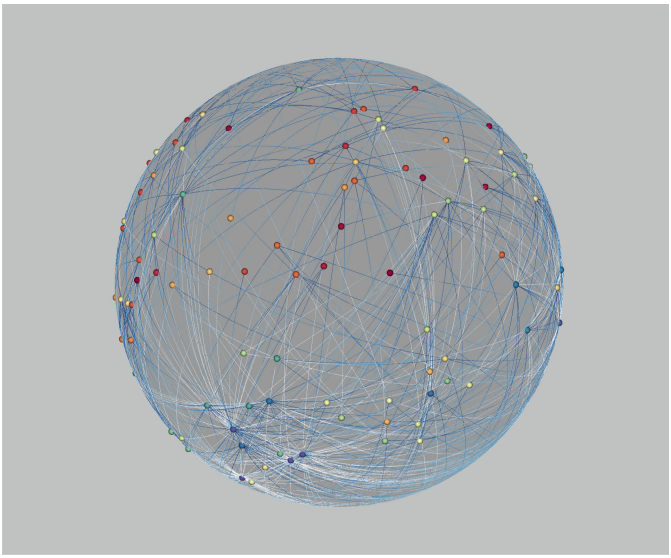


Fig. 4. A network graph created using the variant of the Fruchterman-Reingold algorithm described in Sect. 4.1. This particular graph is a directed citation graph of a set of research papers in gyrokinetics. The arc lines are shaded from blue to white. The node at the white end of the arc is the paper being referenced. (Color figure online)

5 Browser-Based Visualization

Displaying the visualization in a browser increases access and portability for researchers, as mentioned in [6]. We provide some implementation details. We used THREE.js, a JavaScript 3D library that is freely available online. (All imports linked to websites at www.unpkg.com, so our programs are not

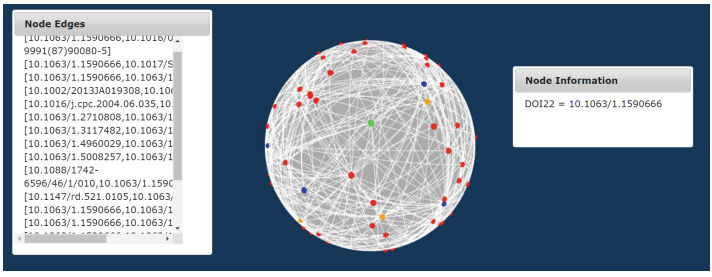


Fig. 5. To increase understanding we programmed two dialog boxes that were not present in the original Paraview visualization. When a node on the graph is clicked, it changes from red to green, and its name is displayed in the dialog box titled "Node Information". Furthermore, all edges containing the green node are also displayed in the dialog box titled "Node Edges". (Color figure online)

dependent on the host computer's local version of THREE.js.) Furthermore, using THREE.js (a wrapper for WEBGL) as well as Javascript allows displaying our visualizations on any device with a Javascript-enabled browser. The source code and a sample dataset are available at https://github.com/Bhombal/NGV.

6 Conclusion

In our research we created a visualization pipeline to parse raw data and display graphs on a 3D sphere. We built on the work of [10] and [6] by developing new versions of the Fruchterman-Reingold Algorithm for layout on a 3D sphere. Defining node distance as the minimal arc length and moving nodes along the unit sphere succeeded in producing graphs with reduced edge length. Our implementation enables the graphs to be viewed and manipulated within the Paraview application or within a web browser.

In the future, the possibility of connecting nodes with non-great-circle edges is vital to explore reduced edge crossings. Additionally, a theoretical explanation for why the modified Fruchterman-Reingold algorithm seems to always converge regardless of the relative strengths of the attractive and repulsive forces (instead of settling into an oscillating or chaotic pattern) will be informative.

References

1. Ahrens, J., Geveci, B., Law, C.: Paraview: an end-user tool for large-data visualization. Vis. Handb. **717**(8) (2005)
2. Bastian, M., Heymann, S., Jacomy, M.: Gephi: an open source software for exploring and manipulating networks. In: Proceedings of the International AAAI Conference on Web and Social Media, vol. 3, no. 1, pp. 361–362 (2009)
3. Brath, R., Macmurchy, P.: Sphere-based information visualization: challenges and benefits. In: 2012 16th International Conference on Information Visualisation, pp. 1–6 (2012)
4. Childs, H.: Visit: an end-user tool for visualizing and analyzing very large data. In: High Performance Visualization-Enabling Extreme-Scale Scientific Insight, pp. 357–372 (2012)
5. Eades, P.: A heuristic for graph drawing (1984)
6. Fruchterman, T.M., Reingold, E.M.: Graph drawing by force-directed placement. Softw. Pract. Exper. **21**(11), 1129–1164 (1991)
7. Gross, M.H., Sprenger, T.C., Finger, J.: Visualizing information on a sphere. In: Proceedings of VIZ 1997: Visualization Conference, Information Visualization Symposium and Parallel Rendering Symposium, pp. 11–16 (1997)
8. Jiawei Lu and Yain Whar Si: Clustering-based force-directed algorithms for 3D graph visualization. J. Supercomput. **76**, 12 (2020)
9. Munzner, T.: Exploring large graphs in 3D hyperbolic space. IEEE Comput. Graph. Appl. **18**(4), 18–23 (1998)
10. Perry, S., Yin, M.S., Gray, K., Kobourov, S.: Drawing Graphs on the Sphere. Association for Computing Machinery, New York, NY, USA (2020)
11. Schroeder, W., Martin, K., Lorensen, B.: The Visualization Toolkit. Kitware (2006)
12. Schulz, C.: Visualizing Spreading Phenomena on Complex Networks (2018)

13. Shoemake, K.: Animating rotation with quaternion curves. In: SIGGRAPH 1885, (1985)
14. Tutte, W.T.: How to draw a graph. Proc. London Math. Soc. s3–13(1), 743–767 (1963)