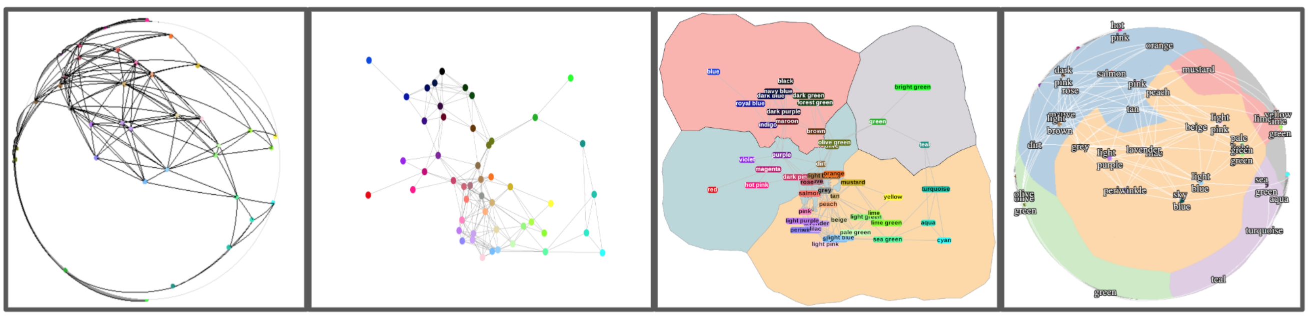
# Drawing Graphs on the Sphere

Scott Perry
scottperry@email.arizona.edu
University of Arizona

Mason Sun Yin
masonyin@email.arizona.edu
University of Arizona

Kathryn Gray
ryngray@email.arizona.edu
University of Arizona

Stephen Kobourov
kobourov@gmail.com
University of Arizona



(1) (2) (3) (4)

**Figure 1: An illustration of Multi-Dimensional Scaling (MDS) approach for drawing graphs on the sphere: (1) the output from spherical MDS which matches the geodesic distance between two points on the sphere to the corresponding graph distance, (2) the stereographic projection from the sphere to the Euclidean plane, (3) clustering similar nodes (e.g., based on graph distances); (4) the final output with the clusters projected back to the sphere.**

## ABSTRACT

Graphs are most often visualized in the two dimensional Euclidean plane, but spherical space offers several advantages when visualizing graphs. First, some graphs such as skeletons of three dimensional polytopes (tetrahedron, cube, icosahedron) have spherical realizations that capture their 3D structure, which cannot be visualized as well in the Euclidean plane. Second, the sphere makes possible a natural "focus + context" visualization with more detail in the center of the view and less details away from the center. Finally, whereas layouts in the Euclidean plane implicitly define notions of "central" and "peripheral" nodes, this issue is reduced on the sphere, where the layout can be centered at any node of interest.

We first consider a projection-reprojection method that relies on transformations often seen in cartography and describe the implementation of this method in the GMap visualization system. This approach allows many different types of 2D graph visualizations, such as node-link diagrams, LineSets, BubbleSets and MapSets, to be converted into spherical web browser visualizations. Next we consider an approach based on spherical multidimensional scaling, which performs graph layout directly on the sphere. This approach supports node-link diagrams and GMap-style visualizations, rendered in the web browser using WebGL.

**ACM Reference Format:**
Scott Perry, Mason Sun Yin, Kathryn Gray, and Stephen Kobourov. 2020. Drawing Graphs on the Sphere. In *International Conference on Advanced Visual Interfaces (AVI '20), September 28-October 2, 2020, Salerno, Italy*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3399715.3399915

## 1 INTRODUCTION

Graph visualizations, usually in the form of node-link diagrams in the two dimensional (2D) Euclidean plane, are a feature of many visualization tools and software packages. Force directed algorithms, otherwise known as spring embedders, model the graph layout problem using physical force analogies. The conceptual simplicity of such algorithms, and the generally aesthetically pleasing results, makes them useful for visualizing relational datasets. Some graphs, however, do not have an ideal representation in two dimensions due to their structure. For example, skeletons of three dimensional polytopes such as the tetrahedron and the cube are better visualized in three dimensions (3D) or even in non-Euclidean spaces


*AVI '20, September 28-October 2, 2020, Salerno, Italy*

ACM ISBN 978-1-4503-7535-1/20/09…$15.00
https://doi.org/10.1145/3399715.3399915

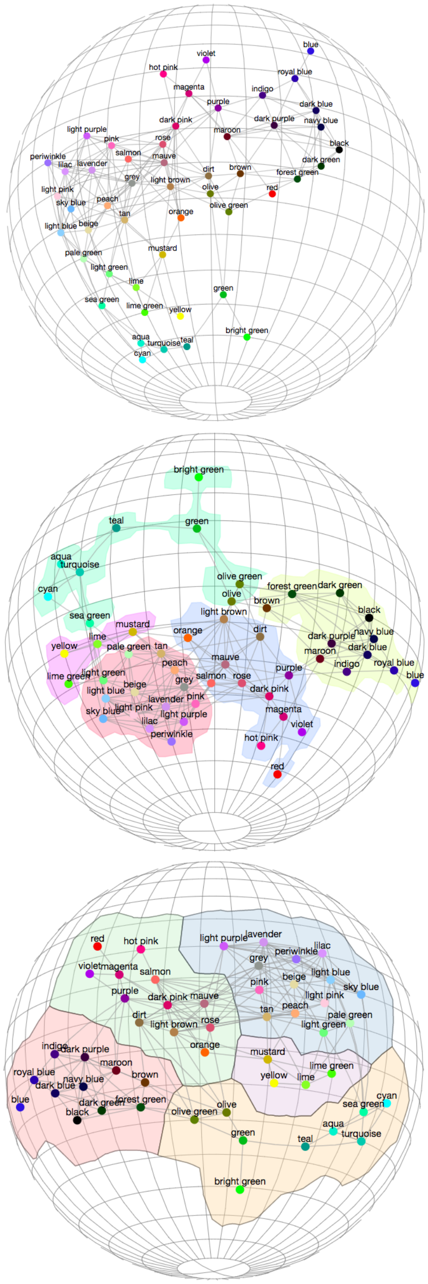



**Figure 2: Node link diagrams, BubbleSets, MapSets style graph visualizations on the sphere.**

such as the sphere. Trees and other hierarchical graphs are well-suited for visualization in hyperbolic space, where all edges can have uniform lengths with vertices uniformly distributed in the space. While there is some work on graph visualization in spherical and hyperbolic space there are no graph visualization tools or packages that provide this functionality. Further, existing work of this type requires 3D visualization environments and there are no implementations in the web browser.

In this paper we describe two different approaches to visualize graphs on the sphere. We first consider a projection-based method that relies on transformations often seen in cartography, and describe the implementation of these ideas in the GMap visualization system. This approach allows us to take the output of any 2D graph visualization method (e.g., a standard node-link diagram, a map-like visualization, etc.) and project this to the sphere and interact with the spherical visualization in the browser via panning, zooming and rotation; see Fig. 2.

While our first approach is still inherently a 2D Euclidean visualization wrapped around the sphere, our second approach, based on spherical multidimensional scaling (MDS), truly takes advantage of the space. It embeds a given graph on the sphere so that pairwise geodesic distances on the sphere correspond to the underlying pairwise graph distances. This method also allows for GMap-style visualization of clusters/countries, where the clusters can be computed based on the graph structure (modularity clustering) [2], or random-walk distances (InfoMap clustering) [37], or geometric distances ($k$-means) [31].

Due to the nature of spherical geometry relative to Euclidean geometry, applying a force-directed algorithm on the sphere or applying MDS on the sphere requires significant modifications. For example, on the sphere there is not necessarily a unique shortest path between two given points (e.g., there are infinitely many such paths between the north pole and the south pole), two points moving in parallel may intersect, moving a point away from another point might make the two closer if the movement it too large, etc.

The rest of the paper is organized as follows. In Section 2 we discuss work related to graph visualization in non-Euclidean spaces, cartographic projections, and multi-dimensional scaling. Section 3 discusses the first method of spherical graph visualization, based on the projection-reprojection method. This section also contains a detailed discussion about rendering graphs with labeled nodes, showing clusters on the sphere, and providing interactions such as zooming and panning in the browser. Section 4 describes the direct approach to spherical graph visualization via MDS. It provides details about MDS, spherical MDS, how to use the output to create clusters, and techniques to render the resulting sphere in WebGL. Discussion and limitations are in Section 5 , and conclusions and future work are in Section 6.

## 2 RELATED WORK

While there are not too many publications on visualizing graphs in spherical space, the idea has been used in the Map of Science project [4], in exhibitions such as “Places and Spaces" [3] and “Worldprocessor" [21]. Human subject studies show that map-like visualizations are as good or better than standard node-link representations of graphs, in terms of task performance, memorization and recall of the underlying data, as well as engagement [38, 39].

A force-directed algorithm for graph layouts has been generalized to arbitrary Riemannian geometries (which includes the sphere) by extending the Euclidean notions of distance, angle, and force-interactions to smooth non-Euclidean geometries via projections to and from appropriately chosen tangent spaces [26]. For example, on the sphere, a tangent plane exists for every point, and when the forces acting on a given node $v$ need to be computed, every other

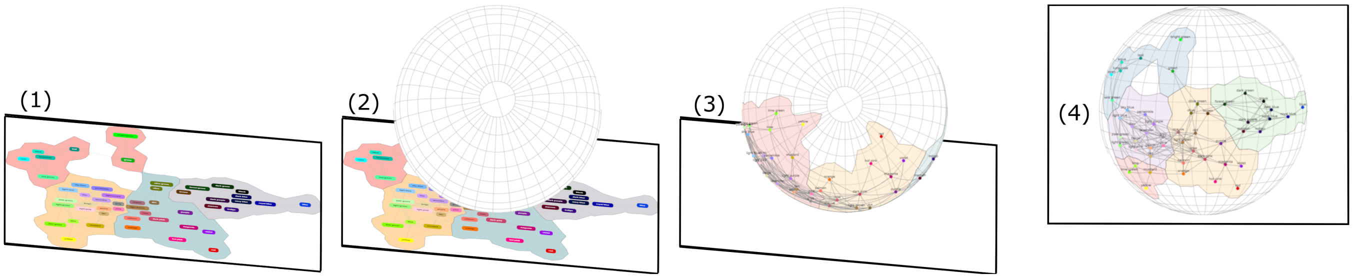



**Figure 3: Illustration of the spherical reprojection approach, where an input 2D visualization is inversely projected onto the sphere and then orthographically projected into the browser. We begin with the input 2D visualization (step 1), then (conceptually) place a tangent sphere (step 2) and inversely project the 2D input onto the sphere (step 3). The result is then orthographically projected onto the browser (step 4) which provides the spherical "look and feel."**

node is mapped onto the tangent plane at $v$ and the force calculation is performed in the plane and after the appropriate move, the node is projected back on the sphere.

Hyperbolic and spherical graph visualization have been studied by Munzner [33–35], especially in the context of trees. In fact, a survey on tree visualizations has more than ten different sphere-based approaches [40]. Concentric spheres have been used to visualize graphs by Sprenger et al. [45]. Self-organizing maps on the sphere have been also been considered by Ritter [36].

Recent work looks at spherical embeddings as a way to visualize graphs in an immersive setting. In this case, the user is placed inside the sphere and is able to interact and visualize the graph this way [29][48]. Our first method is similar to those used in [29], however, in our approach we can also visualize GMap [19], LineSets [1], MapSets [13], BubbleSets [8], and our visualizations work in the browser instead of an immersive environment.

Force-directed graph layout algorithms can be seen as a special case of multi-dimensional scaling (MDS). The main idea behind MDS is to find a placement of the nodes of a graph in such a way that pairwise distances between the nodes in the visualization match the graph distances between these nodes (e.g., computed via all pairs shortest path). MDS is a more general dimensionality reduction technique dating back to the 1960s. The problem was first studied in the non-metric setting by Shepard [42] and Kruskal [28]. Non-metric MDS recovers structure from measures of similarity, based on the assumption of a reproducible ordering between the distances rather than relying on the exact distances. The metric variant of MDS is more frequently used and it relies on the exact distances. The goal of metric MDS is to place objects in some low dimensional space so as to preserve the given pairwise distances between the objects. Given a distance matrix (pairwise dissimilarity matrix) $D = (d_{ij})^{n,n}_{i,j=1}$, between $n$ objects (or $n$ nodes), the objective function function for MDS is

$$S(v_1, \dots, v_n) = \sum_{i>j} \left(d_{ij} - \|x_i - x_j\|\right)^2 . \qquad (1)$$

The function defined in (1) is called the stress function. Some well known techniques for minimizing the stress function (1) are standard gradient descent, stochastic gradient descent [6], and stress majorization [20].

Cox [9] proposed a modification of MDS for the sphere. As the stress function above relies on preserving the rank order of dissimilarities in the distances, Cox argues that rather than using spherical arc distance in the formula, Euclidean distance between points on the surface of the sphere can be substituted, producing an equally suitable configuration. The equation for $d_{ij}$ then becomes:

$$d_{ij} = \{2 - 2\sin\theta_{i2}\sin\theta_{j2}\cos(\theta_{i1} - \theta_{j1}) - 2\cos\theta_{i2}\cos\theta_{j2}\}^{\frac{1}{2}},$$

where $\theta_{i1}$ is the azimuthal angle for point $i$ and $\theta_{i2}$ is the zenith angle for point $i$.

Wu and Takatsuka [47] consider the problem of visualizing high-dimensional data on the sphere and show how to generalize the notion of a self-organizing map to the sphere.

## 3 METHOD 1: PROJECTION-REPROJECTION

Our first approach uses projections of the graph and is implemented and functional at http://gmap.cs.arizona.edu. The implementation relies on the JavaScript library D3.js [5] to handle the computations of the projections and rendering of the graphics which are created with scalable vector graphics (SVG). D3.js allows for binding of data to the document object model (DOM), which allows for efficient display and manipulation. Conveniently, D3.js also contains implementations of common projections in its Geographies module.

In GMap, graphs are stored in the DOT format [17, 27] which specifies a list of nodes, each containing an identifying field along with a list of attributes. Following the nodes, a list of edges is defined using the identifying fields of the nodes. Further details of the specification are provided in the graphviz system [14]. The first step of GMap is to embed the graphs in the two dimensional Euclidean plane [19]. Current options include sfdp [22], a multi-level force directed algorithm which relies on the Barnes and Hut approximation algorithm to optimize long-distance force calculations, and neato [23], which uses the MDS approach.

### 3.1 XDOT Parsing

When visualizing only a node link diagram, having the coordinates of the vertices in the plane is enough to compute the spherical visualization. When dealing with more complex graph visualizations, such as GMap, LineSets [1], MapSets [13], BubbleSets [8] (all available within the GMap system), additional information is needed, such as the computed clusters, polygons in the plane, as well as labels, colors, etc. This information can be stored in different formats, such as PNG, SVG, or XDOT. We use XDOT [18] as it includes the positions for regions and nodes along with other relevant information such as colors. The XDOT file must first be parsed to extrapolate the vertices of each region in the map. D3.js needs geographic data to be presented in GeoJSON [7] format when performing projection calculations, with regions defined by vertices in counter-clockwise order. The following formula estimating the area under the region was applied to each set of vertices to determine the orientation of the region:

$$\text{orientation} = \sum_{i=0}^{n-1}(x_i - x_{i+1})(y_i + y_{i+1})$$

If the output of the summation for the region is negative, then the points are in counter-clockwise order. Otherwise, the order of the vertices must be flipped. Once the correctly ordered regions are defined, the vertices of each region are passed through the equirectangular inverse projection formula to map the regions to the sphere. Then, they are filtered through the orthographic projection formula to map them back onto the plane. Nodes and edges are handled in a similar manner, where a node is defined by a point, and edges are defined by two points. Nodes and edges do not require the orientation preprocessing step; instead, their geometric properties are captured by the GeoJSON tags, "Point" and "LineString".

## 3.2 Spherical Projections to the Plane

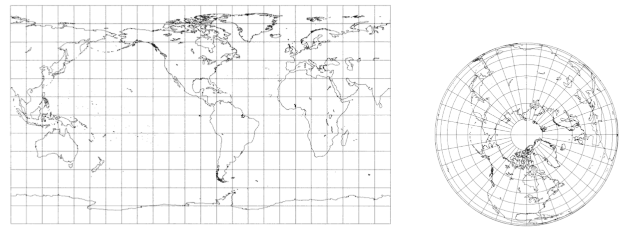
A sphere does not have a representation on the plane that perfectly preserves direction, area, shape, and distance. Thus, any representation of a sphere on the plane results in the loss of some information, and can be classified as a projection. Various sphere to plane projections have been explored extensively in the field of cartography, and are the basis for two dimensional maps; see [24] and the excellent xkcd comic[1].

Our approach for rendering a graph on the surface of a sphere can be divided into two separate tasks. The first is to model an existing graph in two dimensional Euclidean geometry with a sphere, and the second is to display that sphere in a web browser and provide a "spherical look and feel." We begin with a brief review of the essentials of spherical projection formulas.

*3.2.1 Plane to Sphere.* Before we can visualize the graph on the surface of a sphere we need to select a projection from the plane (where the input graph visualizations live) to the surface of the sphere. The *equirectangular* projection is a cylindrical projection which maps a sphere onto a Cartesian grid that contains equal-sized squares. Meridians are mapped to vertical lines and parallels to horizontal lines. The simplest form of this projection is the Plate Carrée, where the line of tangency between the cylinder with the sphere is the equator; see Fig. 4(a). Taking $\lambda$ as the longitude and $\phi$ as the latitude of a point on the sphere, the Plate Carée projection to a point in the plane is given by the simple formula: $(x = \lambda, y = \phi)$.

The Plate Carrée projection is neither conformal nor equiareal, but for visualizing graphs on the surfaces of the sphere this seems acceptable. In the context of graph visualization, the output can be

[1]https://xkcd.com/977/



(a) (b)

**Figure 4: Visualizing Earth with (a) an equirectangular projection where distortion of shapes and areas clearly increase near the poles; and (b) an orthographic projection based on shooting rays from infinity above the north pole [44].**

summarized succinctly by the position of all the nodes (as points). The shape distortion in spherical projections is less important in the context of the graph layout, than maintaining proportionally similar distances between the nodes (points). In graph visualization systems such as GMap the output is usually bounded by a rectangular region and since cylindrical projections produce rectangular results, they are a natural choice for mapping from the plane onto a sphere. Further, the simplicity of the Plate Carée projection, $(x = \lambda, y = \phi)$, allows the two dimensional graph to be quickly translated onto a sphere even for large numbers of nodes.

*3.2.2 Sphere to Browser Plane.* In the previous step we projected a graph visualization from the Euclidean plane onto the sphere. Now we need to show this sphere in a web browser. Since there are no good ways to handle 3D objects in the browser, we utilize yet another projection from the sphere back to the Euclidean plane, which provides the "look and feel" of a sphere.

While cylindrical projections produce rectangular shaped layouts in a plane, planar projections result in circular shaped layouts. As we aim to provide a spherical "look and feel" in the browser, a planar projection such as the *orthographic* one is a suitable choice. An orthographic projection maps the sphere to the plane by casting rays from infinity, through the sphere, and orthogonal to the projection plane. Then each point on the sphere has a ray through it and that point is mapped to the point on the plane that the ray intersects. If projecting the entire sphere, some pairs of spherical points will be mapped to the same point in the plane (when a ray passes through the sphere at two points). Thus the orthographic projection is usually applied to a single hemisphere, and the mapping is one-to-one and onto; see Fig. 4(b).

Orthographic projections can be computed as follows:

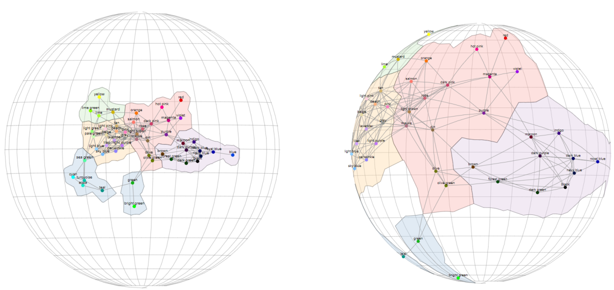
$$\begin{aligned} x &= R\cos\phi\sin(\lambda - \lambda_0) \\ y &= R[\cos\phi_1\sin\phi - \sin\phi_1\cos\phi\cos(\lambda - \lambda_0)] \\ h' &= \sin\phi_1\sin\phi + \cos\phi_1\cos\phi\cos(\lambda - \lambda_0) \\ k' &= 1.0 \end{aligned}$$

where $\phi_1$ and $\lambda_0$ are the latitude and longitude, respectively, of the center point and origin of the projection, $h'$ is the scale factor along a line radiating from the center, and $k'$ is the scale factor in a direction perpendicular to a line radiating from the center [43].

## 3.3 Other Considerations

The equirectangular inverse projection maps the input plane onto the entire sphere, and as a result, the orthographic projection produces an overlapping visualization. To deal with this issue we restrict orthographic projection so that it projects only a hemisphere rather than the entire sphere. This is accomplished in D3.js by clipping by $\frac{\pi}{2}$ the visualization boundary. This in effect hides the "back" of the sphere.

A different problem arises when drawing text labels, as text is dealt with in a different manner than nodes, edges, and regions. We build a custom clipping function to also hide the labels in the "back" of the sphere, so that labels are drawn only for the nodes/edges/regions that are visible. To do this we need to solve the following problem: given a set of points, does each point exist on a hemisphere with a specified center that lies on the surface of the sphere? We can identify the center of our desired hemisphere in a similar way that we inverted points from the plane onto the sphere. Instead, we take the center point of the projection from the browser, and pass it through the inverse equidistant projection formula used previously. Then, for each node's spherical coordinates, we calculate the distance from the node to the selected center. For simplicity, we use a unit sphere as the intermediate mapping step. Thus, if the distance from a node to the center point is greater than $\frac{\pi}{2}$, then neither the node, nor the label should be drawn.



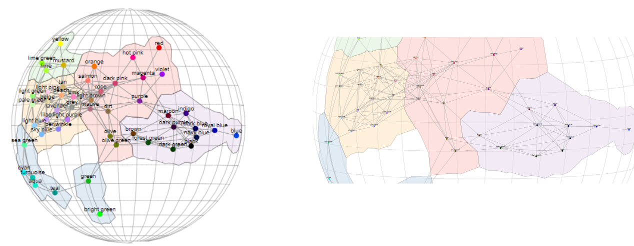
(a) Minimal Coverage (b) Increased Coverage

**Figure 5: As the coverage slider of GMap increases, the elements of the sphere are scaled, while the sphere stays at constant size.**

## 3.4 Zoom and Coverage

When visualizing graphs on the sphere there are two related but different notions of scale to deal with. Zooming a spherical visualization is similar to zooming a plane visualization: zooming in shows a smaller portion of the image at larger scale. Coverage is related to what portion of the sphere is occupied by the visualization. When the coverage is small, the visualization occupies only a tiny part of the surface of the sphere; when the coverage is large, the visualization occupies most of the sphere.

The coverage setting for spherical drawings in GMap corresponds to the inverse of the scale of the equirectangular projection. However, this scale represents the distance between points of the output that an equirectangular projection would produce given a sphere. In our pipeline, the two dimensional plane (rather than the sphere) is our input and that has a fixed size. Hence, increasing the scale of the projection makes the two dimensional graph cover less of the sphere when the inverse projection is applied. Increasing the coverage slider in GMap decreases the scale of the projection and the graph covers more of the sphere; see Fig. 5



(a) Minimal Zoom (b) Increased Zoom

**Figure 6: As the zoom slider of GMap increases, the sphere and contents are both scaled at a proportional rate.**

The zoom parameter corresponds directly to the scale of the orthographic projection. Increasing the scale of this projection increases the output size in the browser of every node, edge, and region linearly. By adjusting the zoom parameter one can decide the context in which to view the nodes/edges/regions in the center of the view: from just a few neighbors, to most of the graph; see Fig. 6.

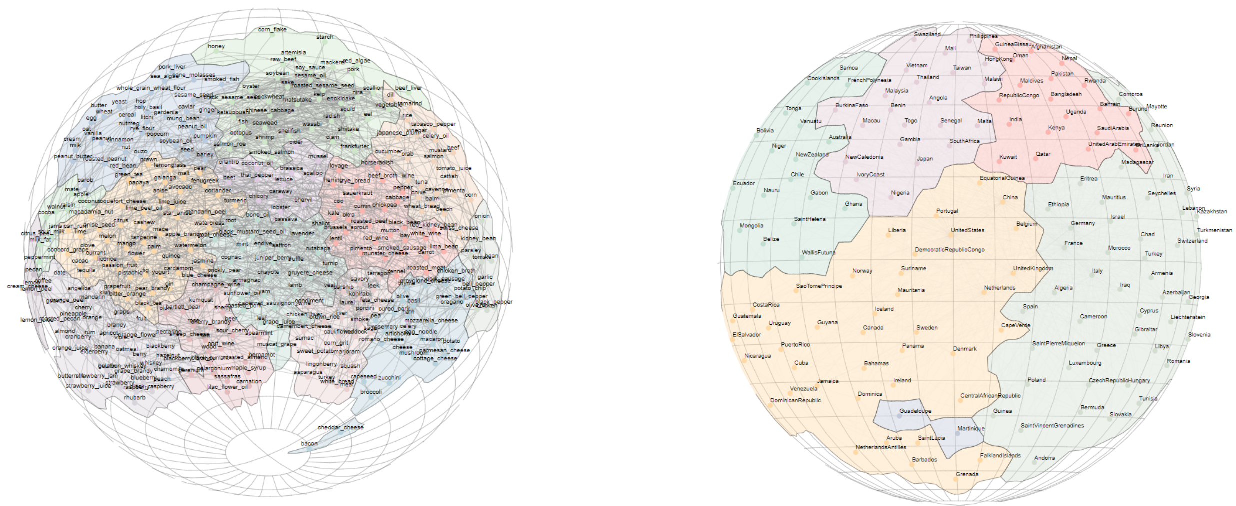
## 3.5 Spherical Layout Examples

The software is available for use at http://gmap.cs.arizona.edu by selecting the "Spherical" visualization type checkbox under the advanced options tab. While the layout and clustering algorithms are performed on the server, the performance of spherical visualizations is dependent on the local machine due to D3.js handling many of the calculations. Graphs with a few hundred nodes can be easily visualized, although larger ones slow down the system; see Fig. 7.

# 4 METHOD 2: SPHERICAL MDS

The projection-reprojection approach for visualizing graphs on the sphere provides a fairly straight-forward way to visualize and interact with a spherical graph visualization inside a standard web browser. However, this approach does not take full advantage of the sphere. For example, graphs that correspond to 3D polytopes (tetrahedron, cube, icosahedron, etc.), visualized with the projection-reprojection approach still look like 2D plane layouts. To take full advantage of the sphere we need an approach that embeds the graph directly on the sphere. One natural direction is to generalize multidimensional scaling from Euclidean space to spherical space.

Recall that MDS is a dimensionality reduction technique that relies on comparing the pairwise dissimilarities of the input data (typically the distance between high dimensional points), with the pairwise dissimilarities represented in the visualization (typically



**(a) Recipes** **(b) Trade**

**Figure 7: Examples of graphs on the sphere: (a) the recipes graph shows recipe ingredients with edges placed between pairs that frequently co-occur in recipes, and (b) the trade graph connects pairs of countries that have significant trade partnerships.**

the 2D Euclidean distance between the projected points). The difference between the input dissimilarities and the visualization dissimilarities is modeled by the *stress* function.

We implement an approach that uses MDS to create a layout on the sphere. This is implemented and available at http://masonsunyin.pythonanywhere.com/polls/

While we will focus on metric MDS as a solution to our problem, the other forms are described by Cox [10].

## 4.1 MDS for Graph Drawing

Computing a good graph layout can be naturally modeled as an MDS problem as follows: Given $G = (V, E)$ with $|V| = n$, we can compute the all-pairs-shortest-path $n \times n$ matrix with entries $\delta_{ij}$ corresponding to the length of the shortest path between nodes $i$ and $j$. We compute this with Dijkstra's algorithm. For any embedding of the graph on the sphere we have another $n \times n$ matrix where an entry $d_{ij}$ corresponds to the actual pairwise distances between nodes $i$ and $j$ on the sphere, using the length of the shorter arc of the great circle defined by $i$, $j$ and the center of the sphere. Then the MDS formulation of the spherical embedding problem is to position the nodes on the surface of the sphere so as to minimize the difference between all pairwise entries $|\delta_{ij} - d_{ij}|$. To compute the values $d_{ij}$ we use the following formula [11]:

$$d_{ij} = \lambda \arccos(\frac{x_i x_j}{\lambda^2})$$

where $\lambda$ is the radius of the sphere and $x_i, x_j$ are the vectors representing nodes $i, j$ in the visualization space. Therefore, we can define the optimal configuration representing the graph as one that minimizes the sum of squares of differences between $\delta_{ij}$ and $d_{ij}$ for every pair of nodes:

$$\text{stress} = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} (\delta_{ij} - d_{ij})^2$$

where $w_{ij}$ is the weight or importance of that pair. In our implementation we set $w_{ij} = 1$ for all pairs.

While the resulting configuration defines points in $\mathbb{R}^3$, Constrained Monotone Distance Analysis (CMDA) can be used to enforce the constraint that nodes are placed on the surface of a sphere in the visualization space [30]. In CMDA, a parameter is used to penalize nodes that are not on the sphere. This is modeled by the function:

$$\sigma_\kappa(X) = \min_{\Delta \epsilon D_L} \sigma(X, \Delta) + \kappa \min_{\Delta \epsilon D_C} \sigma(X, \Delta)$$

where $\Delta \epsilon D_L$ is the graph theoretic distance matrix and $\Delta \epsilon D_C$ contains the spherical constraints. The penalty parameter is $\kappa$ and the stress functions are $\sigma_L(X, \Delta)$ and $\sigma_C(X, \Delta)$ where $X$ is the configuration matrix for the point positions on the sphere in Euclidean coordinates and $\Delta$ is the dissimilarity matrix for the stress function. When $\kappa = \infty$, the second term of $\sigma_\kappa(X)$ is forced to zero, and we minimize the first term under the conditions that the second term is zero, meaning that all points lie on the sphere. This adjusted stress function can be optimized via stress majorization [11].

While we could parameterize the sphere using two coordinates, we leave it in three dimensions and use CMDA as this method will extend simply to other conics, such as ellipsoids and hyperbolas.

## 4.2 Sphere to Plane using Stereographic Projection

Stereographic projection is used in the MDS version to bring points already found on the sphere to the plane for clustering and then projecting the clustered points back to the sphere.

This projection takes a point on the sphere and rotates it until it becomes the north pole. A tangent plane is placed at the south pole. Then lines are drawn from the north pole through each of the nodes on the sphere. The projections of the nodes on the plane are the points of intersection of these lines and the plane. This gives each point on the sphere a unique projection point on the plane.

The stereographic projection is found using the following, where $x_s, y_s, z_s$ are the Cartesian points on the sphere and $x_p, y_p$ are the Cartesian points on the plane:

$$x_p = \frac{x_s}{1 - z_s}$$
$$y_p = \frac{y_s}{1 - z_s}$$

The inverse mapping follows a similar pattern, except the line is drawn through the north pole and the point on the plane. The intersection of this line with the sphere is the projected values. This is used to bring the 2D clusters onto the sphere.

The inverse projection can be written as follows:

$$x_s = \frac{2x_p}{1 + x_p^2 + y_p^2}$$
$$y_s = \frac{2y_p}{1 + x_p^2 + y_p^2}$$
$$z_s = \frac{x_p^2 + y_p^2 - 1}{1 + x_p^2 + y_p^2}$$

where $x_s, y_s, z_s$ are points on the sphere in Cartesian coordinates and $x_p, y_p$ indicate points on the plane.

## 4.3 Implementation of the MDS version

The implementation can be broken down in three steps: (1) compute the dissimilarity matrix and node positions on the sphere based on spherical MDS; (2) project the sphere stereographically to the plane and compute cluster boundaries; (3) projecting back to the sphere and render the final visualization.

*4.3.1 Using MDS to Find Positions.* MDS expects pairwise dissimilarity values for all pairs of nodes in the graph. If the graph is given with similarities instead of dissimilarities, the given similarities are normalized between [0, 1] and the dissimilarities in the matrix are given by 1− this value. To compute the final dissimilarity matrix, we use Disktra's all pairs shortest paths algorithm on the given graph. After computing the dissimilarity matrix for the graph, we apply spherical MDS. We implemented spherical MDS in python, based on the R package `smacof` (Scaling by MAjorizing a COmplicated Function) [12]. The output of this step are spherical coordinates for all nodes in the input graph.

*4.3.2 Clustering with MDS.* After computing spherical node positions we cluster the nodes. This step uses the stereographic projection to bring the points into the 2D plane, using existing graph clustering algorithms, and then reprojecting those clusters back onto the sphere using the inverse stereographic projection.

When projecting, we would like to choose a point as the north pole (and origin of the stereographic projection) such that most of the nodes are as far as possible from it. This corresponds to choosing the center of a large empty region on the sphere as the north pole and reduces the area of the plane occupied by the projected points (which could be arbitrarily large if there are nodes close to the north pole). Currently we apply a simple heuristic to rotate the sphere randomly $n$ times (in our implementation $n = 100$), projecting, and then selecting the projection with the smallest final area.

Once the nodes are projected to the plane they are clustered to create the desired map-like look, using one of several algorithms, including modularity clustering [2], InfoMap clustering [37], and $k$-means clustering [31]. Once the polygons for each cluster have been computed, the polygons are reprojected onto the sphere.

## 4.4 Rendering

The rendering is done using WebGL [25]. We use the points on the sphere output by the spherical MDS and use inverse stereographic projection to get the cluster boundaries to the sphere. Again, clipping is used to hide the back of the sphere.

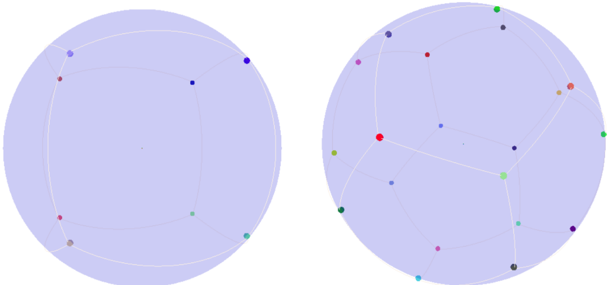
The rendering of the spherical graph consists of three main objects, points, lines, and triangles. For example, the sphere itself is generated using tiny triangles that interlock with each other, resulting in a curved surface that looks smooth to the naked eye. Nodes are drawn by converting the selected point on the sphere into a small sphere, which is drawn using the THREE.js spherical geometry method. Labels are attached to the nodes using HTML which allows us to hide them when they are behind the sphere. Edges are drawn using the shortest (geodesic) path along the sphere.

To render the cluster polygons on the sphere we convert each given cluster into a large collection of small triangles, filled in with the same color. As the sphere itself is made out of many tiny 2D triangles stitched together, our cluster drawing algorithm attempts to split each polygon to the same resolution of triangles on the sphere to create a spherical look.

Before the clusters are rendered on the sphere, several steps are needed to help the rendering perform well. First, the clustering is performed, which gives back the locations of each point of the cluster on the 2d plane. Each polygon is then cut into small triangles using the earcut algorithm [15, 41] as implemented by mapbox (https://github.com/mapbox/earcut). This algorithm splits the polygon into triangles, where each triangle is a set of three points from the polygon. No new points are created, so each point in a triangle corresponds to a point in the 2d polygon, and the triangles can be easily projected to the sphere without changing the original polygon. After the triangles are mapped onto the sphere, there are still some projected triangles that are too large to render properly on the sphere. To reduce the size of triangles, each triangle (now a set of three arcs) on the sphere is recursively bisected until every arc of the resulting triangles is no larger than a specified threshold. This ensures that the polygons appear on the surface of the sphere rather than inside the sphere.

Some of the clustering algorithms produce clusters that are not contiguous polygons: a cluster can be a set of disjoint polygons or a polygon with a hole that contains another cluster. Such cases require special treatment. Through the use of THREE.js, we can refer to the Shape and Edges objects to extract holes. A Shape object is constructed from the points of a cluster and then the Edges object extracts edges from the points that define the shape. The Edges object contains all edges, inside and outside the polygon. If the edges are on the inside they serve as the contour lines for the holes. The inside edges are then stitched together based on the start and end positions of each line and are reorganized as new "clusters," and can be filled in grey or left empty if the data identifies that region as a hole.

### 4.5 Examples



(a) Cube Graph (b) Dodecahedral Graph

**Figure 8: Examples of graphs embedded using spherical MDS. Note how the 3D polytopes "cover" the entire sphere.**

When applied to graphs that correspond to 3D polytopes, the spherical MDS approach produces the desired results; see Fig. 8. Similarly, when applied to graphs that do not have a latent 2D embedding, the spherical MDS approach produces layouts that take advantage of the ability to "wrap around" the sphere. The Last.fm graph is extracted from the larger graph described in [16]. Nodes are music bands, weighted by number of listeners. Edges connect pairs of bands with weights determined by number of shared co-listeners. For this examples we extract the top 100 most popular bends and consider the largest induced connected component. The graph has 59 nodes and 170 edges; see Fig. 9. The colors graph contains 38 nodes and 184 edges. The nodes are the most popular colors as found in a survey by xkcd [32] and the weight of the edges corresponds to their distances in 3D RGB space; see Fig. 10.

## 5 DISCUSSION AND LIMITATIONS

Larger graphs result in cluttered spherical visualization, which is even more of a problem than in the plane as the sphere has finite size for any given radius while the plane is infinite [46]. This issue is partially remedied through the zoom functionality. A semantic zoom functionality on the sphere remains an open problem.

Another inherent limitation of spherical visualization is that it can hide nodes, edges and clusters at the back of the sphere. This could be partially remedied by changing the opacity of the sphere. However, given that most users are familiar with physical globes, this may turn out to be a desirable feature rather than a bug.

In this paper we focused on two different methods for getting graph visualizations on the sphere and allowing for interactions in the browser. Naturally, a human-subjects study would be helpful to determine whether spherical visualizations in the browser are as intuitive and usable as we would like to believe.

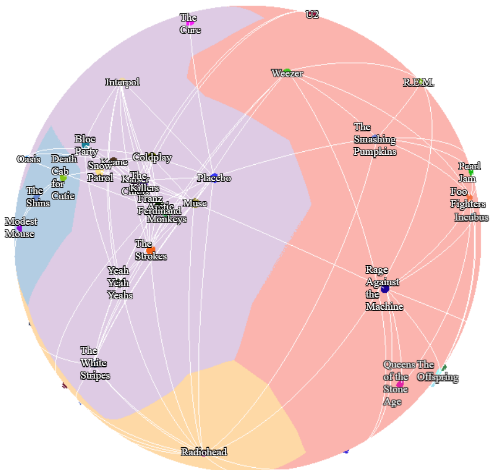



**Figure 9: The Last.fm graph obtained by spherical MDS.**

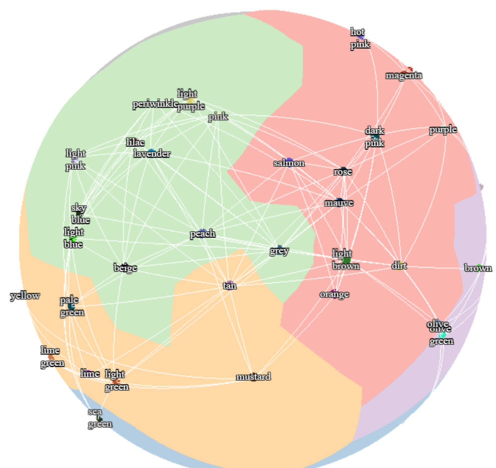



**Figure 10: The colors graph obtained by spherical MDS.**

## 6 CONCLUSIONS AND FUTURE WORK

We described two approaches for visualizing graphs on the sphere. The first projection-reprojection approach provides a simple way to interact with spherical graph visualizations in the browser and is easily extensible to different visualization styles and different underlying layout algorithms.

The second approach applies MDS directly on the sphere and takes better advantage of the underlying geometry, but cannot be directly applied to visuaizations other than and GMap-style and node-link visualizations. So while this approach lays out the graph in a manner more appropriate to the sphere, it lacks the options of the first approach.

In the future, we would like to make any type of 2D plane visualization generalizable to the sphere in the browser. We would like also like to experiment with other non-Euclidean geometries in the browser, such as the hyperbolic plane, which offers even more promising "focus+context" visualization and interactions.

# REFERENCES


[1] B. Alper, N. Riche, G. Ramos, and M. Czerwinski. Design Study of LineSets, a Novel Set Visualization Technique. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2259–2267, 2011.

[2] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast Unfolding of Communities in Large Networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[3] K. Börner. Places and Spaces. http://scimaps.org/home.html, 2012.

[4] K. Börner, R. Klavans, M. Patek, A. M. Zoss, J. R. Biberstine, R. P. Light, V. Larivière, and K. W. Boyack. Design and Update of a Classification System: The UCSD Map of Science. *PloS one*, 7(7):e39464, 2012.

[5] M. Bostock, V. Ogievetsky, and J. Heer. $D^3$ Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

[6] L. Bottou. Large-Scale Machine Learning with Stochastic Gradient Descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[7] H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, and T. Schaub. The GeoJSON format. Internet Engineering Task Force, RFC 7946, 2016.

[8] C. Collins, G. Penn, and S. Carpendale. Bubble Sets: Revealing Set Relations with Isocontours over Existing Visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1009–1016, 2009.

[9] T. F. Cox and M. A. Cox. Multidimensional Scaling on a Sphere. *Communications in Statistics-Theory and Methods*, 20(9):2943–2953, 1991.

[10] T. F. Cox and M. A. Cox. *Multidimensional Scaling*. Chapman and Hall/CRC, 2000.

[11] J. de Leeuw and P. Mair. Multidimensional Scaling Using Majorization: SMACOF in R. *Journal of Statistical Software, Articles*, 31(3):1–30, 2009.

[12] J. De Leeuw, P. Mair, and M. J. de Leeuw. Package SMACOF, 2013. https://cran.r-project.org/package=smacof.

[13] A. Efrat, Y. Hu, S. Kobourov, and S. Pupyrev. MapSets: Visualizing Embedded and Clustered Graphs. In *International Symposium on Graph Drawing*, pages 452–463. Springer, 2014.

[14] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull. Graphviz – Open Source Graph Drawing Tools. In *International Symposium on Graph Drawing*, pages 483–484. Springer, 2001.

[15] A. Fournier and D. Y. Montuno. Triangulating Simple Polygons and Equivalent Problems. *ACM Trans. Graph.*, 3(2):153–174, Apr. 1984.

[16] E. Gansner, Y. Hu, S. Kobourov, and C. Volinsky. Putting Recommendations on the Map: Visualizing Clusters and Relations. In *Proceedings of the third ACM Conference on Recommender Systems*, pages 345–348, 2009.

[17] E. R. Gansner and J. Ellson. The DOT language. https://www.graphviz.org/doc/info/lang.html, 2002.

[18] E. R. Gansner and J. Ellson. XDOT Output Formats. https://www.graphviz.org/doc/info/output.html#d:xdot, 2002.

[19] E. R. Gansner, Y. Hu, and S. Kobourov. GMap: Visualizing Graphs and Clusters as Maps. In *2010 IEEE Pacific Visualization Symposium (PacificVis)*, pages 201–208. IEEE, 2010.

[20] E. R. Gansner, Y. Koren, and S. North. Graph Drawing by Stress Majorization. In *International Symposium on Graph Drawing*, pages 239–250. Springer, 2004.

[21] I. Günther. Worldprocessor. com. In *ACM SIGGRAPH 2007 art gallery*, page 200. ACM, 2007.

[22] Y. Hu. Efficient, High-quality Force-directed Graph Drawing. *Mathematica Journal*, 10(1):37–71, 2005.

[23] T. Kamada and S. Kawai. An Algorithm for Drawing General Undirected Graphs. *Information processing letters*, 31(1):7–15, 1989.

[24] M. Kennedy, S. Kopp, et al. *Understanding Map Projections*. ESRI, 2000.

[25] Khronos WebGL Working Group. WebGL specification https://www.khronos.org/registry/webgl/specs/latest/1.0/.

[26] S. Kobourov and K. Wampler. Non-Eeuclidean Spring Embedders. *IEEE Transactions on Visualization and Computer Graphics*, 11(6):757–767, 2005.

[27] E. Koutsofios, S. North, et al. Drawing graphs with *dot*. Technical report, Technical Report 910904-59113-08TM, AT&T Bell Laboratories, Murray Hill, NJ, 1991.

[28] J. B. Kruskal. Multidimensional Scaling by Optimizing Goodness of Fit to a Nonmetric Hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[29] O. Kwon, C. Muelder, K. Lee, and K. Ma. A Study of Layout, Rendering, and Interaction Methods for Immersive Graph Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(7):1802–1815, July 2016.

[30] J. C. Lingoes and I. Borg. *Geometric Representations of Relational Data: Readings in Multidimensional Scaling*. Mathesis Press, 1979.

[31] S. Lloyd. Least Squares Quantization in PCM. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[32] R. Munroe. xkcd Color Name Survey. https://xkcd.com/color/rgb/.

[33] T. Munzner. H3: Laying out Large Directed Graphs in 3D Hyperbolic Space. In *Proceedings of VIZ'97: Visualization Conference, Information Visualization Symposium and Parallel Rendering Symposium*, pages 2–10. IEEE, 1997.

[34] T. Munzner. Exploring Large Graphs in 3D Hyperbolic Space. *IEEE Computer Graphics and Applications*, 18(4):18–23, 1998.

[35] T. Munzner. *Interactive Visualization of Large Graphs and Networks*. PhD thesis, Stanford University, 2000.

[36] H. Ritter. Self-Organizing Maps on non-Euclidean Spaces. In *Kohonen maps*, pages 97–109. Elsevier, 1999.

[37] M. Rosvall and C. T. Bergstrom. Maps of Random Walks on Complex Networks Reveal Community Structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008.

[38] B. Saket, C. Scheidegger, S. Kobourov, and K. Börner. Map-based Visualizations Increase Recall Accuracy of Data. *Computer Graphics Forum*, 34(3):441–450, 2015.

[39] B. Saket, P. Simonetto, S. Kobourov, and K. Börner. Node, Node-Link, and Node-Link-Group Diagrams: An Evaluation. *IEEE Transactions on Visualization & Computer Graphics*, 20(12):2231–2240, 2014.

[40] H.-J. Schulz. Treevis. Net: A Tree Visualization Reference. *IEEE Computer Graphics and Applications*, 31(6):11–15, 2011.

[41] R. Seidel. A Simple and Fast Incremental Randomized Algorithm for Computing Trapezoidal Decompositions and for Triangulating Polygons. *Computational Geometry*, 1(1):51 – 64, 1991.

[42] R. N. Shepard. The Analysis of Proximities: Multidimensional Scaling with an Unknown Distance Function. *Psychometrika*, 27(2):125–140, 1962.

[43] J. P. Snyder. *Map Projections – A Working Manual*, volume 1395. US Government Printing Office, 1987.

[44] J. P. Snyder and P. M. Voxland. *An Album of Map Projections*, volume 1453. US Government Printing Office, 1989.

[45] T. C. Sprenger, M. H. Gross, A. Eggenberger, and M. Kaufmann. A Framework for Physically-Based Information Visualization. In *Visualization in Scientific Computing*, pages 71–83. Springer, 1997.

[46] G. Van Brummelen. *Heavenly Mathematics: The Forgotten Art of Spherical Trigonometry*. Princeton University Press, 2012.

[47] Y. Wu and M. Takatsuka. Visualizing Multivariate Network on the Surface of a Sphere. In *Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation-Volume 60*, pages 77–83. Australian Computer Society, Inc., 2006.

[48] Y. Yang, B. Jenny, T. Dwyer, K. Marriott, H. Chen, and M. Cordeil. Maps and Globes in Virtual Reality. *Computer Graphics Forum*, 37(3):427–438, 2018.