Drawing Graphs on the Sphere

Scott Perry and Stephen Kobourov

Departament of Computer Science, University of Arizona, Tucson, AZ, USA

Abstract. We describe two approaches for drawing graphs on the sphere. 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. We also describe a different approach based on spherical multidimensional scaling, which performs dimensionality reduction directly on the sphere.

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 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. While this approach is the most general it is also still inherently a 2D Euclidean visualization wrapped around the sphere. Our second approach, spherical multidimensional scaling (MDS), truly takes advantage of the sphere. This approach is not as easy to generalize and currently is implemented only for the standard node-link diagram. 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 projection-reprojection spherical graph visualization is implemented and available as a visualization option in the GMap system at http://gmap.cs.arizona.edu. Rather than perform force calculations on the sphere, the layouts were produced in two dimensional Euclidean space, projected onto a sphere, and reprojected back onto the plane to display the layout in a web browser.

![Illustration of the spherical reprojection approach](image)

Fig. 2: 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.”

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 considers different projections from the sphere to the plane and different
reprojections back to the plane, to justify our choices for projection and reprojection functions. This leads to a detailed discussion in Section 4 about rendering graphs with labeled nodes, showing clusters on the sphere and providing interactions such as zooming and panning in the browser. Section 5 elaborates the direct approach to spherical graph visualization via MDS. 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], as well as in exhibitions such as the well-known “Places and Spaces” [3] and “Worldprocessor” [20] exhibitions; see Fig. 3. Sphere-based visualizations have been used for other types of data besides graphs [7].

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 [24]. 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 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 [29, 28, 30], especially in the context of trees. In fact, a survey on tree visualizations has more than ten different sphere-based approaches [32]. Concentric spheres have been used to visualize graphs by Sprenger et al. [36]. Self-organizing maps on the sphere have been also been considered by Ritter [31].

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 [33] and Kruskal [26]. 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) \( D = (d_{ij})_{i,j=1}^{n,n} \), between \( n \) objects (or \( n \) nodes), the objective function function for MDS is

\[
S(v_1, \ldots, v_n) = \sum_{i>j} (d_{ij} - \|x_i - x_j\|)^2.
\]

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 [19].

Cox [10] 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:

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

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

Wu and Takatsuka [38] 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 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 plain projections have been explored extensively in the field of cartography, and are the basis for two dimensional maps; see [23] 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.

3.1 Projection Basics

Two points on the surface of a sphere together with the center of the sphere define a plane, whose intersection with the surface of the sphere is a great circle. The shortest distance between two points on the surface of the sphere is defined by the the shorter of the two arcs of the great circle. The meridians of a sphere are great circles through the two poles and the equator is a great circle perpendicular to the meridians. Points on the surface of a unit sphere can be specified by longitude (parallels) and latitude (meridians). Alternatively, longitude and latitude can be interpreted as polar coordinates specified by the azimuthal angle and zenith angle, with respect to $xy$-plane defined by the equator and plane defined

\(^{1}\) https://xkcd.com/977/
by the prime meridian (Greenwich meridian). To specify spheres of different size (or points inside a sphere of unit size) a third parameter, radial distance, is needed.

Because projections of the sphere to the Euclidean plane cannot simultaneously preserve direction, area, shape and distance, each projection formula is created for a specific purpose. Projections that preserve area are equiareal, while those that preserve distance are classified as equidistant. If the direction along some of the great circles on the sphere remain unchanged, then the projection is a true-direction one, and if local shapes are preserved, then it is conformal.

Some of the commonly used projections first translate the sphere to another three dimensional object that is easier to flatten. Objects such as cylinders, cones, and planes can all form tangential contact points with a sphere. When translated, area, shape, distance, and direction at these contact points are not distorted. As points on the sphere become more distant from the other object, the amount of distortion in at least one of these traits increases.

Each of the three objects (cylinders, cones, and planes) used to translate the sphere to the plane produce uniquely shaped resulting images in the plane. Conic projections often capture a single hemisphere, and in the plane appear as an unraveled cone where the most accuracy occurs on a single parallel of the sphere. In planar projections, the plane that is projected to lies tangential to a single point on the sphere. Thus, the projection captures the relationship between other points on the sphere and the focus point. These projections can most accurately capture a single hemisphere. Cylindrical projections capture the entire sphere, by surrounding the sphere with the side of the cylinder along a single parallel. Often, as in the most common Mercator projection, the equator is used as the line of tangency and the meridians of the sphere are equally spaced, with distances between the parallels increasing towards the poles.

3.2 Plane to Sphere Inverse Equirectangular Projection

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 Carrée 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 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 Carrée projection, \((x = \lambda, y = \phi)\), allows the two dimensional graph to be quickly translated onto a sphere even for large numbers of nodes.
Fig. 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 [35].

3.3 Sphere to Plane Browser Orthographic Projection

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:

\[ x = R \cos \phi \sin(\lambda - \lambda_0) \]
\[ y = R[\cos \phi_1 \sin \phi - \sin \phi_1 \cos \phi \cos(\lambda - \lambda_0)] \]
\[ h' = \sin \phi_1 \sin \phi + \cos \phi_1 \cos \phi \cos(\lambda - \lambda_0) \]
\[ k' = 1.0 \]

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 [34].
4 Rendering Graphs in the Browser

The projection-reprojection approach for visualization of graphs on the sphere described above is available at http://gmap.cs.arizona.edu. The implementation relies on the JavaScript library D3.js \cite{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 \cite{16,25} 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 \cite{15}. The first step of GMap is to embed the graphs in the two-dimensional Euclidean plane \cite{18}. Current options include sfdp \cite{21}, a multi-level force directed algorithm which relies on the Barnes and Hut approximation algorithm to optimize long-distance force calculations, and neato \cite{22}, which uses the MDS approach.

4.1 XDOT Parsing

When visualizing just 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 \cite{2}, MapSets \cite{14}, BubbleSets \cite{9} (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 as it includes the positions for regions and nodes along with other relevant information such as colors \cite{17}. 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 format when performing projection calculations \cite{8}, 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”.

4.2 Making it All Work

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 90° 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.

![Image](image1.png)

(a) Minimal Coverage  
(b) Increased Coverage

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

### 4.3 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

Fig. 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.

4.4 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; see Fig. 7 for some example visualizations. 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. Larger graphs also result in more cluttered visualization, which is partially due to the restricted domain of a sphere with a set radius [37] and to the space required for labels for each node. The former issue is partially remedied through the zoom functionality. The latter problem requires semantic zoom functionality, which is still work in progress.
5 Spherical Multidimensional Scaling

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 the 2D Euclidean distance between the projected points). The difference between the input dissimilarities and the visualization dissimilarities is modeled by the stress function. While we will focus on metric MDS as a solution to our problem, the other forms are described by Cox [11].

5.1 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 $j$. For edge-weighted graphs this can be computed with Dijkstra’s algorithm and for unweighted graphs via a modified breadth first search. 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 [12]:

\[
d_{ij} = \lambda \arccos\left(\frac{x_i \cdot x_j}{\lambda^2}\right)
\]

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 [27]. 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 [12].

### 5.2 Implementation and Examples

For our prototype spherical MDS implementation we use the R package \texttt{smacof} (Scaling by MAjorizing a COmplicated Function) [13]. To apply this in the context of graphs, we must first create a dissimilarity matrix for the \( n \) nodes in the graph. For edge-weighted graphs this can be computed with Dijkstra’s algorithm and for unweighted graphs via a modified breadth first search. After processing this matrix through the ’smacof’ package, we retain the optimal configuration in coordinates in \( \mathbb{R}^3 \). Because of the CMDA penalty adjustment, we assume all points in this configuration lie on the surface of the sphere. Knowing the configuration is centered at \( \vec{0} \), we can calculate the radius, \( r \), of the sphere with the formula:

\[
r = \sqrt{x^2 + y^2 + z^2} = \left\| \begin{bmatrix} x \\ y \\ z \end{bmatrix} \right\|^2
\]

Using the OpenGL-based R rendering package \texttt{rgl} [1] we render a sphere of radius \( r \), and plot each node according to its point in the optimal configuration along with its corresponding
label. We then draw each edge along the shorter arc along the great circle. Note that the MDS approach calculates the positions of the nodes directly on the sphere, and so the resulting visualization naturally occupies most of the sphere; see Fig. 8. This eliminates the need for a coverage parameter which was needed in the projection-reprojection based approach.

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. In particular, this approach is available and functioning in GMap. It allows us to put on the sphere any graph visualization available in GMap (node-link diagram, GMap, LineSets, BubbleSets, MapSets), any layout algorithm available in GMap (e.g., sfdp, neato), and any clustering algorithm available in GMap (k-means, modularity, infomap, hierarchical).

The second approach of performing MDS directly on the sphere and takes better advantage of the underlying geometry but is difficult to extend and is not easy to deploy in a browser. Generalizing spherical MDS to different visualization styles and using different clustering techniques is still work-in-progress.
References