Non-Euclidean Spring Embedders*

Stephen G. Kobourov† Kevin Wampler‡

University of Arizona

Abstract

We present a method by which force-directed algorithms for graph layouts can be generalized to calculate the layout of a graph in an arbitrary Riemannian geometry. The method relies on extending the Euclidean notions of distance, angle, and force-interactions to smooth non-Euclidean geometries via projections to and from appropriately chosen tangent spaces. In particular, we formally describe the calculations needed to extend such algorithms to hyperbolic and spherical geometries.

CR Categories: G.2.2 [Discrete Mathematics]: Graph Theory—Graph Algorithms; H.5.0 [Information Systems]: Information Interfaces and Presentation—General I.3.6 [Computing Methodologies]: Computer Graphics—Methodology and Techniques

Keywords: force-directed algorithms, spring embedders, non-Euclidean geometry, hyperbolic space, spherical space, graph drawing, information visualization

1 Introduction

Some of the most flexible algorithms for calculating layouts of simple undirected graphs belong to a class known as force-directed algorithms. Also known as spring embedders, such algorithms calculate the layout of a graph using only information contained within the structure of the graph itself, rather than relying on domain-specific knowledge. Graphs drawn with these algorithms tend to be aesthetically pleasing, exhibit symmetries, and tend to produce crossing-free layouts for planar graphs.

However, existing force-directed algorithms are restricted to calculating a graph layout in Euclidean geometry, typically $\mathbb{R}^2$, $\mathbb{R}^3$, and more recently $\mathbb{R}^n$ for larger values of n. There are, however, cases where Euclidean geometry may not be the best option: certain graphs may be known to have a structure which would be best realized in a different geometry, such as on the surface of a sphere or on a torus. Furthermore, it has also been noted that certain non-Euclidean geometries, specifically hyperbolic geometry, have properties which are particularly well suited to the layout and visualization of large classes of graphs [10, 11].

We present a method by which a force-directed algorithm can be generalized so that it can compute a graph layout in any of a large class of geometries (known as Riemannian geometries), so long as the mathematics describing how the geometries behave are well described. Because of the particular usefulness of hyperbolic geometry and spherical geometry, with respect to graph drawing, we also present these mathematical properties for the case of $\mathbb{H}^2$, two dimensional hyperbolic space and $\mathbb{S}^2$, spherical space. Our method relies on extending the Euclidean notions of distances and angles to Riemannian geometries via projections to and from appropriately chosen tangent spaces.

From a practical point of view, the hyperbolic and spherical cases are fairly straightforward and we have implemented both of them. Thus, we are able to compare layouts obtained with the traditional Euclidean force-directed methods and those obtained with the generalized force-directed methods in hyperbolic space and in spherical space, such as those in Fig. 1.

2 Related Work

2.1 Force-Directed Layouts

Force-directed algorithms are a well-known and powerful tool for laying out arbitrary graphs [1, 4, 8]. Such methods define an objective function which maps each graph layout into a number in $\mathbb{R}^+$ representing the energy of the layout. Generally, this energy function is defined in such a way that low energies correspond to layouts in which adjacent nodes are near some pre-specified distance from each other, but in which non-adjacent nodes are well-spaced. A layout for a graph is then calculated by finding a (often local) minimum of this objective function.

One particularly useful way to find such a local minimum is through a gradient descent method. In this model we calculate forces (often via the negative gradient of the energy function) which result from the interaction between the nodes in the graph. Generally, there are repulsive forces between all nodes, but also attractive forces between nodes which are adjacent [4]. Alternatively, forces between the nodes can be computed based on their graph distance, as determined by the lengths of shortest paths between them [8]. The nodes are then moved according to the net force acting upon them, and the process is repeated until a steady state is reached or a maximum number of iterations is exceeded.

While early force-directed algorithms work well for small graphs, recently such algorithms have been extended to deal with graphs with hundreds of thousands of vertices using multi-scale and spectral techniques [5, 7, 9].

With few exceptions, spring embedders thus far have been restricted to n-dimensional Euclidean space. This restriction is due in part to the simplicity of the algorithms when formulated in Euclidean space, and in part to a reliance on the convenient structure of Euclidean space with well-defined notions of distances and angles. Some work, however, has been done on constraining force-directed algorithms to the surface of three-dimensional objects [15]. This work is based on a differential equation formulation of the motion of the nodes in the graph, and is flexible in that it allows a layout on almost any object, even multiple objects. Since the force calculations are made in Euclidean space, however, this

*This work is partially supported by the NSF under grant ACR-0222920.

†Email: kobourov@cs.arizona.edu

‡Email: wamplerk@cs.arizona.edu

IEEE Symposium on Information Visualization 2004
October 10-12, Austin, Texas, USA
0-7803-8779-1/04/$20.00 ©2004 IEEE

method is inapplicable to certain geometries (e.g., hyperbolic geometry).

Another example of graph embedding within a non-Euclidean geometry is described in the context of generating spherical parameterizations of 3D meshes [6]. This method produces such an embedding using a generalization to spherical space of planar methods for expressing convex combinations of points. This results in a non-linear system of equations in three dimensions which when solved yields an embedding on the unit sphere. Although no specific way of solving this system of equations is described, it is likely that certain ways of doing so would operate in a manner similar to the method described in this paper, though the former is not readily generalizable to non-spherical geometries.

2.2 Hyperbolic Graph Drawing

Much of the work on non-Euclidean graph drawing has been done in hyperbolic space [11, 13] which offers certain advantages over Euclidean space. For example, in hyperbolic space it is possible to compute a layout for a complete tree with both uniform edge lengths and uniform distribution of nodes. Furthermore, some of the embeddings of hyperbolic space into Euclidean space naturally provide a fish-eye view of the space, which is useful for "focus+context" visualization [10]. Previous algorithms for calculating the layouts of graphs in hyperbolic space, however, are either restricted by their nature to the layout of trees and tree-like graphs, or to layouts on a lattice.

The hyperbolic tree layout algorithms function on the principle of hyperbolic sphere packing, and operate by making each node of a tree, starting with the root, the center of a sphere in hyperbolic space. The children of this node are then given positions on the surface of this sphere and the process recurses on these children. By carefully computing the radii of these spheres it is possible to create aesthetically pleasing layouts for the given tree. Although some applications calculate the layout of a general graph using this method, the layout is calculated using a spanning tree of the graph and the extra edges are then added in without altering the layout [12]. This method works well for tree-like and quasi-hierarchical graphs, or for graphs where domain-specific knowledge provides a way to create a meaningful spanning tree. However, for general graphs (e.g., bipartite or densely connected graphs) and without relying on domain specific knowledge, the tree-based approach may result in poor layouts.

Methods for generalizing Euclidean geometric algorithms to hyperbolic space, although not directly related to graph drawing, have also been studied [2]. It is shown that many algorithms which operate in Euclidean space can be extended to hyperbolic space by exploiting the properties of a Euclidean model of the space (such as the Beltrami-Klein or Poincaré). Our work follows a similar vein in that we use the Poincaré model to implement the hyperbolic case of out technique, though it differs in that this mapping alone is not sufficient, as the notions of distance and linearity in the Poincaré model do not match their Euclidean counterparts.

Hyperbolic and spherical space have also been used to display self-organizing maps in the context of data visualization [14, 16]. These methods extend the traditional use of a regular (Euclidean) grid, on which the self-organizing map is created, with a tessellation in spherical or hyperbolic space. An iterative process is then used to adjust which elements in the data-set are represented by the intersections. Although the hyperbolic space method seems a promising way to display high-dimensional data-sets, the restriction to a lattice is often undesirable for graph visualization.

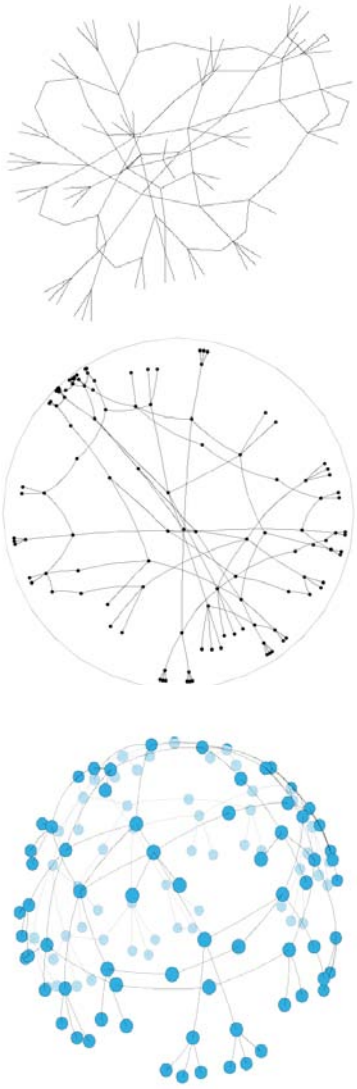

Figure 1: Drawings of the same simple, undirected graph obtained in Euclidean $\mathbb{R}^2$ space, hyperbolic $\mathbb{H}^2$ space and spherical $\mathbb{S}^2$ space. The graph has 121 nodes and 140 edges.

3 Non-Euclidean Spring Embedding

3.1 Basics of Riemannian Geometry

Current implementations of force-directed algorithms perform their calculations in $\mathbb{R}^n$, the standard Euclidean space. Euclidean geometry has properties which afford many conveniences for calculating a graph layout with a force-directed method. In particular, Euclidean space has a very convenient structure; it is easy to define distances and angles, and the relationship between the vector representing the net force on an object and the appropriate motion of that object is quite straightforward.

A non-Euclidean geometry does not afford all of the conveniences above, so it is more difficult to define how the

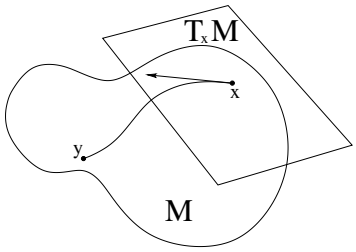

Figure 2: A curve and its derivative in the tangent space.

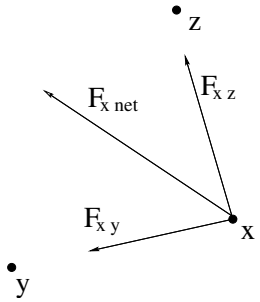

Figure 3: Net force on x by two other points, y and z.

forces acting upon a graph should be calculated, and how those forces should affect the layout of the graph. There is, however, a straightforward way to do this, provided we restrict ourselves to geometries which are smooth.

Such geometries are known as Riemannian geometries, and while they have less convenient structure than Euclidean geometry, they retain many of the characteristics which are useful for force-directed graph layouts. A Riemannian manifold M has the property that for every point $x \in M$, the tangent space T_xM is an inner product space; see Fig. 2. This means that for every point on the manifold, it is possible to define local notions of length and angle.

Using the local notions of length we can define the length of a continuous curve $\gamma : [a, b] \to M$ by

$$\text{length}(\gamma) = \int_a^b ||\gamma'(t)||dt.$$

This leads to a natural generalization of the concept of a straight line to that of a *geodesic*, where the geodesic between two points $u, v \in M$ is defined as a continuously differentiable curve of minimal length between them. These geodesics in Euclidean geometry are straight lines, and in spherical geometry they are arcs of great circles. We can similarly define the distance between two points, $d(x, y)$ as the length of a geodesic between them.

3.2 Application to Spring Embedders

As mentioned above, one of the convenient properties of Riemannian manifolds is that at every point there exists a well-structured tangent space. We utilize these tangent spaces to generalize spring embedders to arbitrary Riemannian geometries.

In Euclidean space the relationship between a pair of nodes is defined along lines: the distance between the two nodes is the length of the line segment between them and forces between the two nodes act along the line through them. These notions of distance and forces can be extended to a Riemannian geometry by having these same relationships be defined in terms of the geodesics of the geometry, rather than in terms of Euclidean lines.

The tangent space is also useful in dealing with the interaction between one point and several other points in non-Euclidean geometries. Consider three points x, y, and z in a Riemannian manifold M where there is an attractive force from x to y and z; see Fig. 3.

As can be easily seen in the Euclidean case (but also true in general) the net force on x is not necessarily in the direction of y or z, and thus the natural motion of x is along neither the geodesic toward y, nor that toward z. Determining the direction x will move requires the notion of angle.

Since the tangent space at x, being an inner product space, has enough structure to define lengths and angles, we do the computations for calculating the forces on x in T_xM. In order to do this, we define two functions for every point $x \in M$ as follows:

$$\tau_x : M \to T_xM$$
$$\tau_x^{-1} : T_xM \to M.$$

These two functions map points in M to and from the tangent space of M at x, respectively. We require that τ_x and τ_x^{-1} satisfy the following constraints:

1. $\tau_x^{-1}(\tau_x(y)) = y$ for all $y \in M$
2. $||\tau_x(y)|| = d(x, y)$
3. τ_x preserves angles about the origin

Using these functions it is now easy to define the way in which the nodes of a given graph $G = (V, E)$ interact with each other through forces. In the general framework for this algorithm we consider each node individually, and calculate its new position based on the relative locations of the other nodes in the graph (repulsive forces) and on its adjacent edges (attractive forces). Given a node $n \in V(G)$ with position x we use τ_x to map the positions of the relevant nodes of G into T_xM (nodes that are used in computing x's new location). A standard force equation can then be used to calculate the force, f upon n as a vector in T_xM. Given the vector in T_xM, the new position, x' of n in T_xM is calculated using standard techniques, typically by multiplying f by a scalar. The desired position of n in M is then given by $\tau_x^{-1}(x)$. Pseudo-code for this process is summarized in Fig. 4.

4 Hyperbolic Geometry

4.1 Motivation

One of the most useful applications for our non-Euclidean force-directed method is that it allows the layout of a general graph to be calculated in hyperbolic space (space of constant negative curvature). This provides a functionality beyond current hyperbolic graph layout techniques. Such functionality is desirable because of both the geometric properties of hyperbolic space and of the properties of some of the more common ways of mapping hyperbolic geometry into Euclidean space.

Hyperbolic geometry is particularly well suited to graph layout because it has "more space" than Euclidean geometry – in the same sense that spherical geometry has "less

```
generic_initial_layout(G)
while not done do
    foreach n ∈ G do
        position[n] := force_directed_placement(n, G)
    end
end

non_Euclidean_initial_layout(G)
while not done do
    foreach n ∈ G do
        x := position[n]
        G' := τ_x(G)
        x' := force_directed_placement(n, G')
        position[n] := τ_x^{-1}(x')
    end
end
```

Figure 4: A generic Euclidean spring embedder and its non-Euclidean counterpart.

space". To illustrate this, consider the relationship between the radius and circumference of a circle in a two-dimensional geometry. In Euclidean geometry the relationship is linear with a factor of 2π. In spherical geometry, however, the circumference is bounded above by a constant (the circumference of a great circle on the sphere). With hyperbolic geometry the opposite is the case; the circumference of a circle increases exponentially with its radius.

The applicability of this geometric property to graph layout is well illustrated with the example of a tree. The number of nodes at a certain depth in the tree typically increases exponentially with the depth. Thus, layouts in Euclidean space result in characteristic long edges near the root and short edges near the leaves. In hyperbolic space, however, it is possible to layout the tree with a uniform distribution of the nodes and with uniform edge lengths.

4.2 Hyperbolic Projections

In order to display a layout in hyperbolic geometry, it is necessary to map the figure into the (two-dimensional) Euclidean geometry of a computer monitor. There are numerous ways of doing this, two of the most common being the Poincaré disk and Beltrami-Klein projections. In both of these cases the hyperbolic space is mapped onto the open unit disk $\{z \in \mathbb{R}^2 : |z| < 1\}$. To obtain such projections it is necessary to distort the space, which in these cases takes the form of compressing the space near the boundary of the unit disk, giving the impression of a fish-eye view. This naturally provides a useful focus+context technique for visualizing the layouts of the graph; see Fig. 5 and Fig. 6.

In the Beltrami-Klein projection straight lines are mapped to straight lines, but angles are not necessarily preserved. Thus, each line in hyperbolic space is mapped to a chord of the unit disk, and two lines are non-intersecting if their associated chords are non-intersecting. Furthermore, the distance between two points, (x, y) and (u, v), in the Beltrami-Klein model is not given by their Euclidean distance, but rather by

$$\arccos\left[\frac{1 - xu - yv}{\sqrt{(1 - x^2 - y^2)(1 - u^2 - v^2)}}\right].$$

The Poincaré disk model preserves angles, but distorts lines. A line in hyperbolic space is mapped to a circular arc

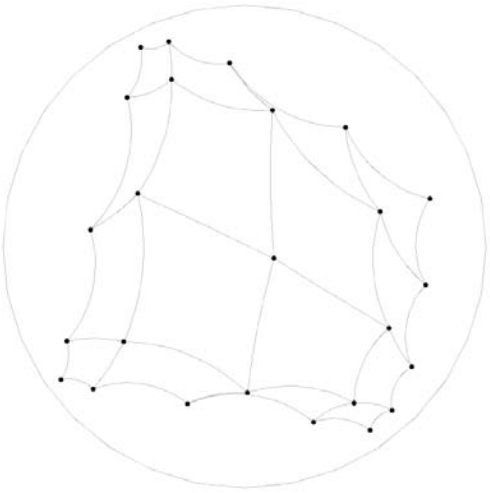

Figure 5: A 5×5 mesh laid out in a hyperbolic geometry and projected onto the Poincaré disk.

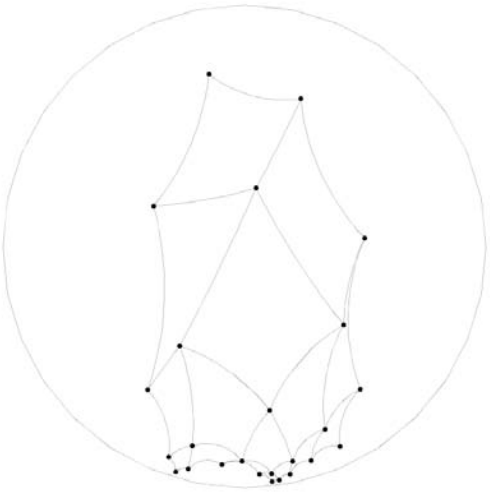

Figure 6: The same mesh as in Fig. 5, but projected with a different center of attention.

which intersects the unit circle at right angles (chords passing through the origin are considered to be such arcs). As with the Beltrami-Klein model, distances in the projection are not equal to the the hyperbolic distances between the points. The Poincaré disk model also compresses the space slightly less at the edges, which in some cases can have the advantage of allowing a better view of the context around the center of projection. In this paper we focus on an implementation which uses the Poincaré disk model.

4.3 Tangent Space Mapping

There are many possible ways to compute the mapping to and from the tangent space. Here we present the details about one such mapping, which we also implemented. As illustrated in Fig. 4, the problem reduces to defining the mappings τ_x and τ_x^{-1} so that they meet the three criteria from Section 3.1.

Internally, each node in the graph is assigned a position $z = (x, y)$ within the unit disk, representing the Poincaré coordinates of that node. Using the Poincaré coordinates for the positions of points allows us to take advantage of the property that in a Poincaré projection angles are preserved and circles and lines are mapped to circles and lines. Since hyperbolic space is uniform, we can 'recenter' the projection about any point, z_0, by applying a conformal (angle preserving) mapping which maps z_0 to the origin, the boundary of the unit circle to itself, and which maps circles and lines to circles and lines. By treating the position of the node as a complex number, we can define such a mapping as the linear fractional transformation:

$$f_{z_0}(z) = \frac{z - z_0}{1 - \bar{z_0} z}.$$

It is also easy to compute the inverse of this function:

$$f_{z_0}^{-1}(z) = \frac{-z - z_0}{-\bar{z_0} z - 1}.$$

By using f to recenter the projection about z_0 we force all geodesics passing through z_0 to be projected as line segments passing through the origin. Furthermore, the Euclidean angle formed between two such lines is equal to the angle by which the two corresponding geodesics intersect. This satisfies criteria 1, and 3 for the function τ_z, but the norm of the points (their distances from the origin) after the mapping f is not equal to their distances from z_0 in the hyperbolic space (as a consequence the range of the inverse function is also only the unit disk). To remedy this, we rescale the points such that their distances from the origin is indeed equal to their hyperbolic distance from z_0. Note that this does not alter angles at the origin. This is accomplished with another mapping, denoted by g as follows:

$$g_{z_0}(z) = \frac{z}{||z||} \log\left(\frac{1 + ||z||}{1 - ||z||}\right).$$

It is also possible to find the inverse of this mapping:

$$g_{z_0}^{-1}(z) = \frac{z}{||z||}\left|\frac{1 - e^{||z||}}{1 + e^{||z||}}\right|.$$

Now we can define τ_{z_0} by composing these two mappings:

$$\tau_{z_0} = g \circ f.$$

Similarly, we can define $\tau_{z_0}^{-1}$ as:

$$\tau_{z_0}^{-1} = f^{-1} \circ g^{-1}.$$

It can be verified that τ_{z_0} and $\tau_{z_0}^{-1}$ as defined above, indeed satisfy the three criteria for functions mapping to and from the tangent space, and thus these two functions are sufficient to implement a spring embedder in hyperbolic geometry.

5 Spherical Geometry

As a further example for generalizing spring embedders to non-Euclidean geometry we also consider spherical geometry. As with hyperbolic geometry, spherical geometry has a constant curvature and the equations for mapping to and from the tangent space can be calculated analytically.

Each point in a spherical geometry is defined by its coordinates, $\theta \in [0, 2\pi)$ and $\phi \in [0, \pi)$, representing the longitude and latitude of the point, respectively. This spherical geometry can then be embedded as a sphere in three-dimensional Euclidean space by the parametrization $(\cos\theta \sin\phi, \cos\phi, \sin\theta \sin\phi)$. We can calculate the tangent plane at any point on the sphere by taking the space spanned by the two partial derivative vectors:

$$u = (-\sin\theta \sin\phi, 0, \cos\theta \sin\phi)$$

$$v = (\cos\theta \cos\phi, -\sin\phi, \sin\theta \cos\phi).$$

Note that if applied at either of the poles, these equations fail to yield a valid space, so in these cases the u and v vectors can be hard-coded to, for example, $(1, 0, 0)$ and $(0, 0, 1)$.

We can now compute $\tau_x(y)$ for any points x and y in the spherical geometry by projecting the embedding of y, $emb(y)$ onto the tangent plane at x with $(emb(y) \cdot u_x, emb(y) \cdot u_y)$. To complete the mapping we have only to set the length of this vector equal to the length of the geodesic between x and y:

$$r \cdot \arccos\left[\sin\phi_x \sin\phi_y \cos\theta_y - \theta_x + \cos\phi_x \cos\phi_y\right],$$

where r is the radius of curvature of the geometry. An illustration of this mapping can be seen in Fig. 7.

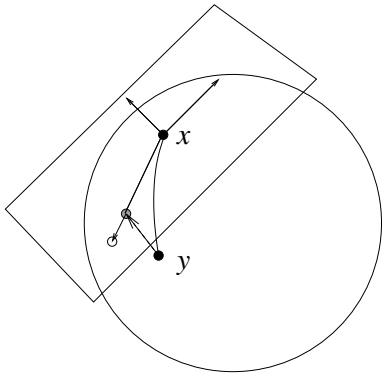

Figure 7: Mapping to the tangent space via mapping to the tangent plane, followed by a renormalization for the length of the vector in the tangent space.

The inverse of this mapping $\tau_x^{-1}(y)$, illustrated in Fig. 8, can also be computed in a similar geometric manner. First, we compute a vector, p, perpendicular to that from $\tau_x(y)$ in the tangent space. The vector p is then mapped to the corresponding vector in three dimensional space by $up_x + vp_y$. This vector is perpendicular to the plane containing the origin, $\tau_x^{-1}(x)$ and $\tau_x^{-1}(y)$. Thus the desired point $\tau_x^{-1}(y)$ can be obtained by rotating $\tau_x^{-1}(x)$ about this axis so that the arc length traveled by $\tau_x^{-1}(x)$ is equal to the norm of $\tau_x^{-1}(y)$. In radians, this angle is $\frac{|y|}{r}$. Since this rotated vector is in Euclidean space, the calculation can be completed by projecting it back onto the sphere by calculating $\theta = arctan\frac{z}{x}$, $\phi = \arccos y$.

6 Example Layouts in $\mathbb{H}^2$ and $\mathbb{S}^2$

In Figures 9-11 we consider a title-word graph obtained from the graph drawing literature [3]. This graph has 27 nodes and 50 edges. The graph nodes correspond to title-words from papers in the Proceedings of the 1999 Symposium on Graph Drawing. The size of a node is determined by the frequency of the corresponding word and an edge is placed between two nodes if they co-occur in at least one paper.

The images in the Figures 9-11 were obtained using our implementation of the algorithms described in this paper.

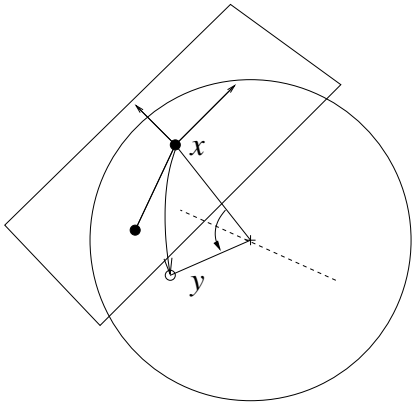

Figure 8: Mapping from the tangent space via a rotation.

Fig. 9 shows three views of the graph using different centers of attention, and nicely illustrates the focus+context properties of hyperbolic space. Fig. 10 shows three views of the same graph in spherical space, again using different centers of attention. Finally, Fig. 11 contains layouts of the same graph obtained in $\mathbb{R}^2$, $\mathbb{H}^2$ and $\mathbb{S}^2$.

7 Conclusion and Future Work

We presented a simple algorithm for generalizing a spring embedder to an arbitrary Riemannian geometry. This method relies on only very general features of spring embedders, and thus can be applied in principle to most force-directed layout methods. We also presented the details for the specific cases of hyperbolic and spherical geometries as well as some layouts obtained with our implementation.

Although the methods presented here are sufficient to generalize a spring embedder into any Riemannian geometry, there are still many practical concerns that need to addressed. While the mathematics needed to determine τ_x and τ_x^{-1} are relatively simple for the cases of hyperbolic and spherical geometries, this is not always the case. It is not even possible, in general, to analytically calculate the geodesic between two points in an arbitrary geometry. It is likely the case that for more complex geometries approximate methods will have to be used to determine τ_x and τ_x^{-1}.

Perhaps most importantly with regard to information visualization, we would like to make our method scalable. As with traditional force-directed algorithms, our method does not work well for very large graphs. Finding low energy states becomes increasingly difficult as the input graphs get larger. Multi-scale methods and high dimensional embedding have been successfully used to extend Euclidean spring embedders. Generalizing the non-Euclidean spring embedders along the lines of [5, 7] should be possible. This would allow us to experiment with the layouts of very large graphs in these geometries, and thus to fully exploit their properties to better visualize large data-sets.

8 Acknowledgments

We would like to thank the anonymous referees for pointing out two publications, relevant to our work [2, 6].

References

[1] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[2] D. Eppstein. Hyperbolic geometry, Möbius transformations, and geometric optimization. In *MSRI Introductory Workshop on Discrete and Computational Geometry*, 2003.

[3] C. Erten, P. J. Harding, S. G. Kobourov, K. Wampler, and G. Yee. GraphAEL: Graph animations with evolving layouts. In *11th Symposium on Graph Drawing*, pages 98–110, 2003.

[4] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software — Practice and Experience*, 21(11):1129–1164, 1991.

[5] P. Gajer, M. T. Goodrich, and S. G. Kobourov. A multi-dimensional approach to force-directed layouts of large graphs. *Computational Geometry: Theory and Applications*, 29(1):3–18, 2004.

[6] C. Gotsman, X. Gu, and A. Sheffer. Fundamentals of spherical parameterization for 3D meshes. In *ACM Transactions on Graphics, 22*, pages 358–363, 2003.

[7] D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. *Journal of graph algorithms and applications*, 6:179–202, 2002.

[8] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, Apr. 1989.

[9] Y. Koren, L. Carmel, and D. Harel. ACE: A fast multiscale eigenvector computation for drawing huge graphs. In *Proceedings of IEEE Symposium on Information Visualization*, pages 123–144, 2002.

[10] J. Lamping, R. Rao, and P. Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of Computer Human Interaction*, pages 401–408. ACM, 1995.

[11] T. Munzner. H3: Laying out large directed graphs in 3D hyperbolic space. In L. Lavagno and W. Reisig, editors, *Proceedings of IEEE Symposium on Information Visualization*, pages 2–10, 1997.

[12] T. Munzner. Drawing large graphs with h3viewer and site manager. In *6th Symposium on Graph Drawing*, pages 384–393, 1998.

[13] T. Munzner and P. Burchard. Visualizing the structure of the World Wide Web in 3D hyperbolic space. In *Symposium on the Virtual Reality Modeling Language*, pages 33–38, 1996.

[14] J. Ontrup and H. Ritter. Hyperbolic self-organizing maps for semantic navigation. In *Advances in Neural Information Processing Systems 14*, pages 1417–1424, 2001.

[15] D. I. Ostry. Some three-dimensional graph drawing algorithms. Master's thesis, University of Newcastle, Australia, 1996.

[16] H. Ritter. Self-organizing maps on non-euclidean spaces. In S. Oja, E. & Kaski, editor, *Kohonen Maps*, pages 97–110. Elsevier, Amsterdam, 1999.

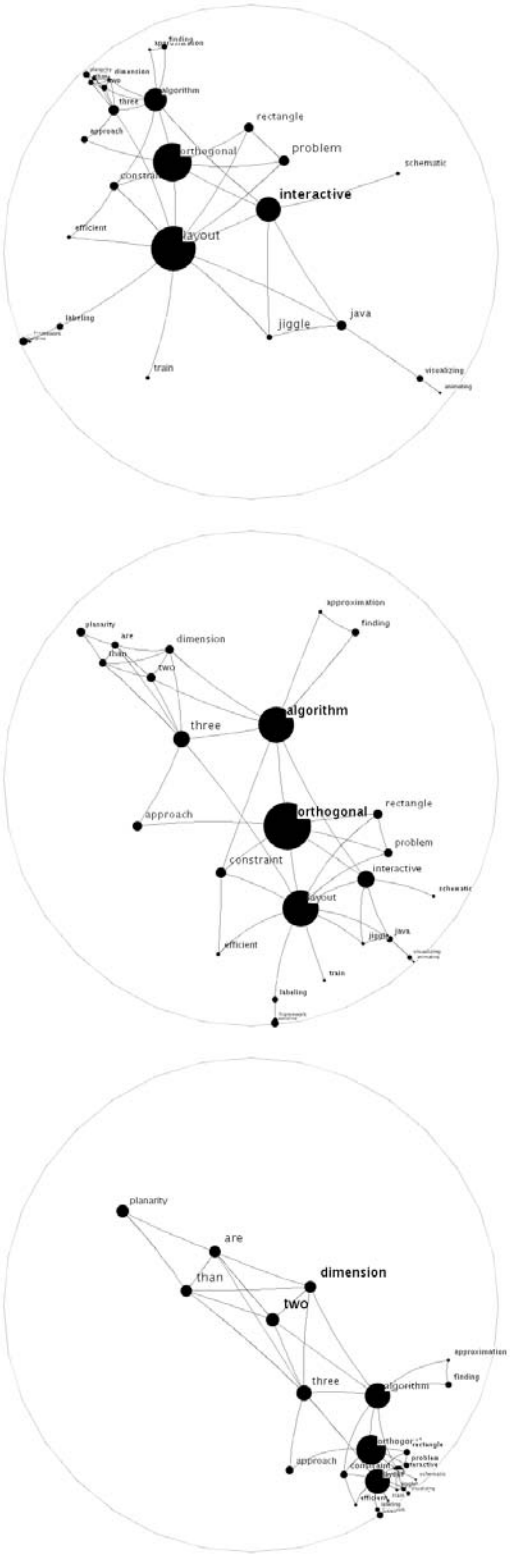

Figure 9: Layouts of the title-word graph with different centers of attention in hyperbolic space.

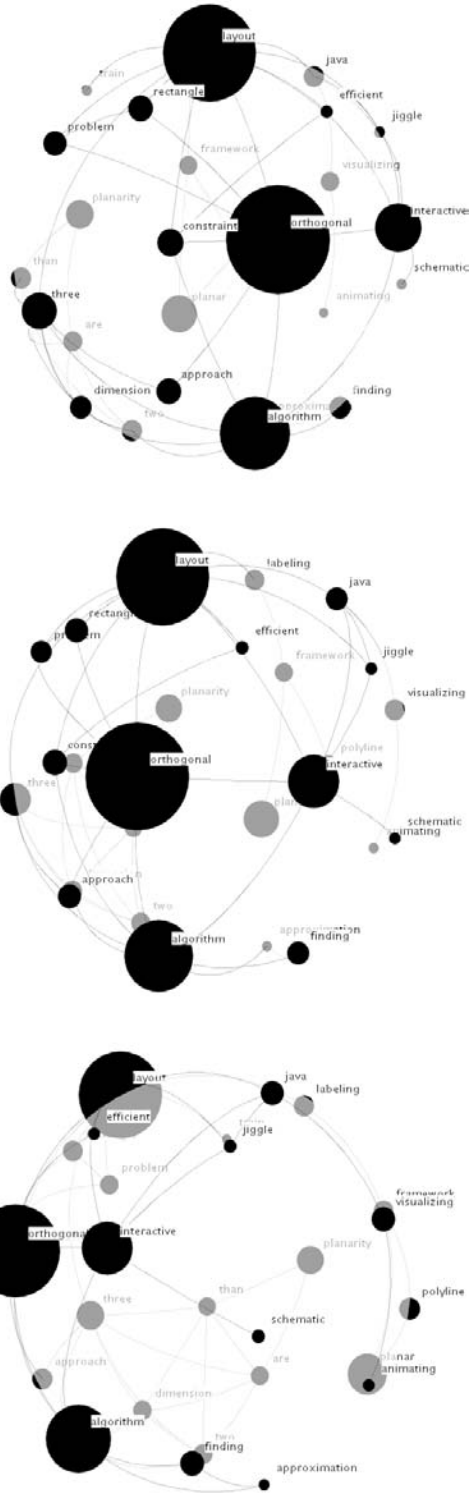

Figure 10: Layouts of the title-word graph with different centers of attention in spherical space.

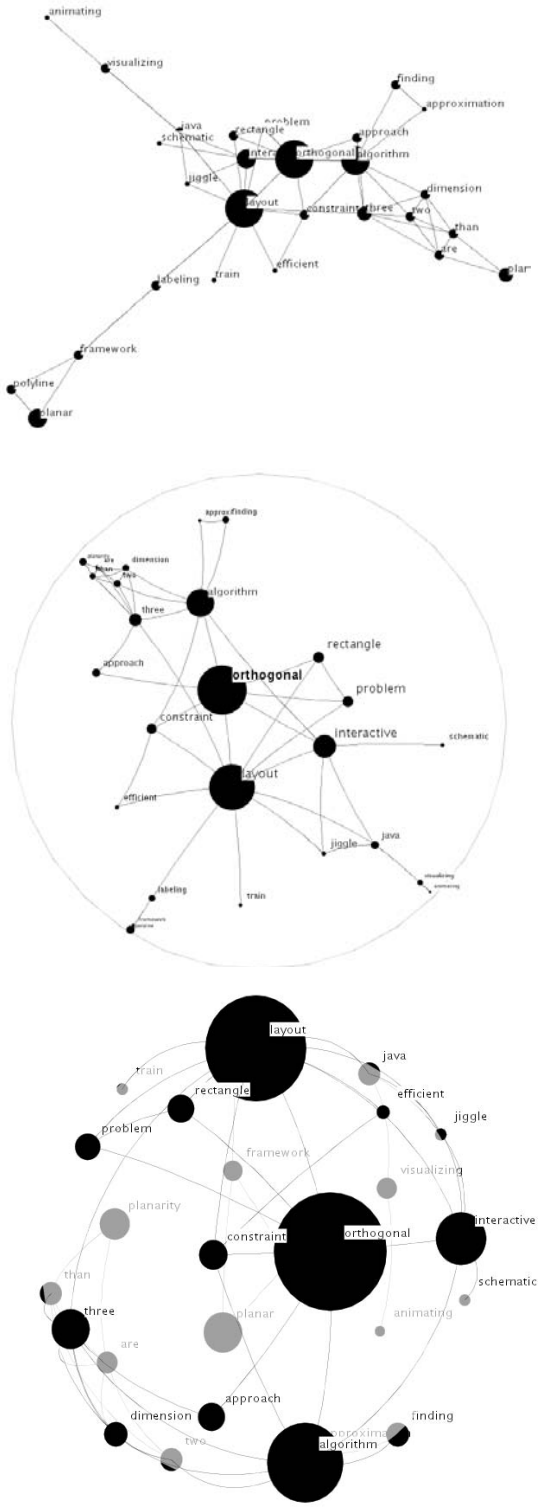

Figure 11: Layouts of the title-word graph, obtained in $\mathbb{R}^2$, $\mathbb{H}^2$ and $\mathbb{S}^2$. The graph has 27 nodes and 50 edges.