Non-Euclidean Spring Embedders

Stephen G. Kobourov and Kevin Wampler

Abstract—We present a conceptually simple approach to generalizing force-directed methods for graph layout from Euclidean geometry to Riemannian geometries. Unlike previous work on non-Euclidean force-directed methods, ours is not limited to special classes of graphs, but can be applied to arbitrary graphs. 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. We also study the theoretical and practical considerations that arise when working with non-Euclidean geometries.

Index Terms—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. In particular, 3D mesh data can be parameterized on the sphere for texture mapping or graphs of genus one can be embedded on a torus without crossings. 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 [16], [17].

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, information visualization, and graphics, 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. The extended abstract [13] contains a summary of the results, whereas here we elaborate on the mathematical details and discuss some of the practical considerations of working with non-Euclidean geometries.

From a practical point of view, the hyperbolic and spherical cases are fairly straightforward and we have implemented spring embedder algorithms for both geometries. 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 Force-Directed Methods

Graph drawing has applications in many areas where relational data needs to be visualized, such as VLSI, software engineering, and databases; see the survey paper by Herman et al. [10]. It is the subject of an annual symposium and several books on the subject are available [2], [12]. While many algorithms have been developed for special classes of graphs, such as trees, tree-like graphs, and planar graphs, general graphs are most often visualized using force-directed methods.

Going back to 1963, the graph drawing algorithm of Tutte [24] is one of the first force-directed graph drawing methods, based on barycentric representations. More traditionally, the spring layout method of Eades [3] and the algorithm of Fruchterman and Reingold [6] both rely on Hooke's law for spring forces. In these methods, there are repulsive forces between all nodes, but also attractive forces between nodes which are adjacent.

Alternatively, forces between the nodes can be computed based on their graph theoretic distances, determined by the lengths of shortest paths between them. The algorithm of Kamada and Kawai [11] uses spring forces proportional to the graph theoretic distances. Other force-directed methods

- *S.G. Kobourov is with the Department of Computer Science, University of Arizona, Tucson, AZ 85721. E-mail: kobourov@cs.arizona.edu.*
- *K. Wampler is with the Department of Computer Science and Engineering, University of Washington, CSE 101, Allen Center, Box 352350, Seattle, WA 98195-2350. E-mail: wampler@cs.washington.edu.*

Manuscript received 15 Dec. 2004; revised 19 Apr. 2005; accepted 3 May 2005; published online 8 Sept. 2005.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-0256-1204.

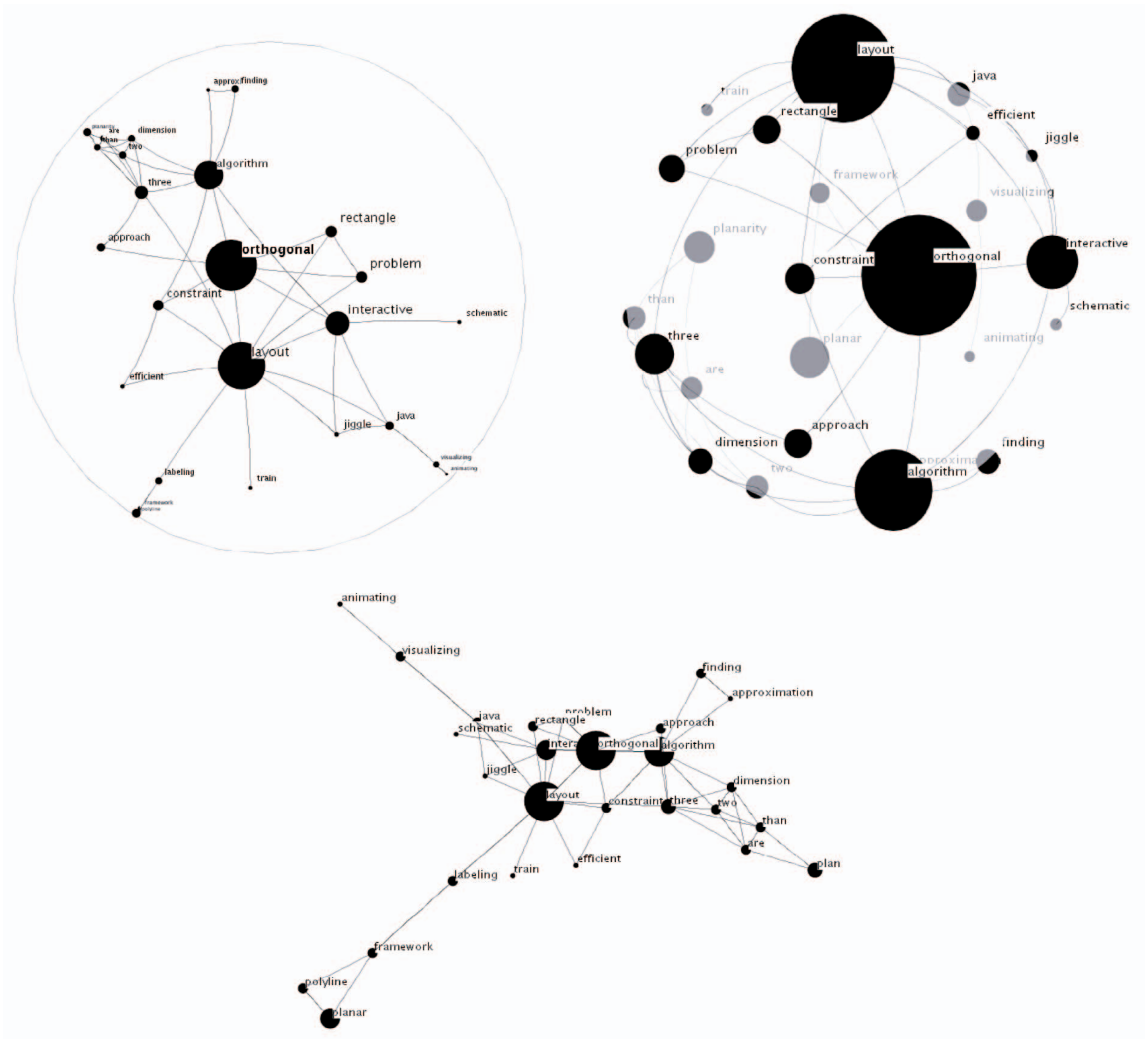

Fig. 1. Layouts of a title-word graph, obtained in $\mathbb{H}^2$, $\mathbb{S}^2$, and $\mathbb{R}^2$. The graph has 27 nodes and 50 edges.

include that of Sugiyama and Misue [23], based on magnetic fields.

In general, force-directed methods define an objective function which maps each graph layout into a number in $\mathbb{R}^+$ representing the energy of the layout. This energy function is defined in such a way that low energies correspond to layouts in which adjacent nodes are near some prespecified distance from each other, but in which nonadjacent nodes are well-spaced. A layout for a graph is then calculated by finding a (often local) minimum of this objective function; see Fig. 2.

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. 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. Harel and Koren [9] use a multiscale technique and Gajer et al. [7] combine this idea with intelligent placement of nodes to avoid local minima. Koren et al. [14] use techniques from spectral graph theory to quickly obtain layouts for even larger graphs.

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.

Ostry [21] considers constraining force-directed algorithms to the surface of three-dimensional objects. 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 method is inapplicable to certain geometries (e.g., hyperbolic geometry).

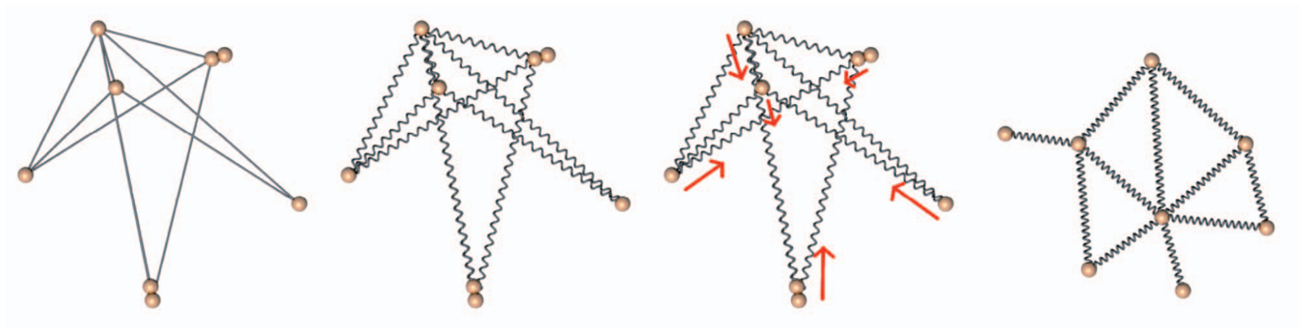

Fig. 2. An overview of a spring embedder in 2D Euclidean space.

Another example of graph embedding within a non-Euclidean geometry is described in the context of generating spherical parameterizations of 3D meshes. Gotsman et al. [8] describe a method for producing such an embedding using a generalization to spherical space of planar methods for expressing convex combinations of points. The implementation of the procedure is similar to the method described in this paper, but it may not lend itself to geometries other than spherical.

3 Hyperbolic Graph Drawing

Much of the work on non-Euclidean graph drawing has been done in hyperbolic space, which offers certain advantages over Euclidean space; see Munzner [17] and Munzer and Burchard [19]. 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, as shown by Lamping et al. [16]. 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 [18]. 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. Recently, van Wijk and Nuij [25] proposed a Poincaré's half-plane projection to define a model for 2D viewing and navigation. Eppstein [4] shows 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 our 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. Ontrup and Ritter [20] and Ritter [22] 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.

4 Non-Euclidean Spring Embedding

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 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.

We begin with a brief description of Riemannian geometry and manifolds. A *manifold* is a topological space that is locally Euclidean, that is, around every point, there is a neighborhood that is topologically the same as the open unit ball in $\mathbb{R}^n$. We then show how the properties of manifolds can be used to extend force-directed calculations

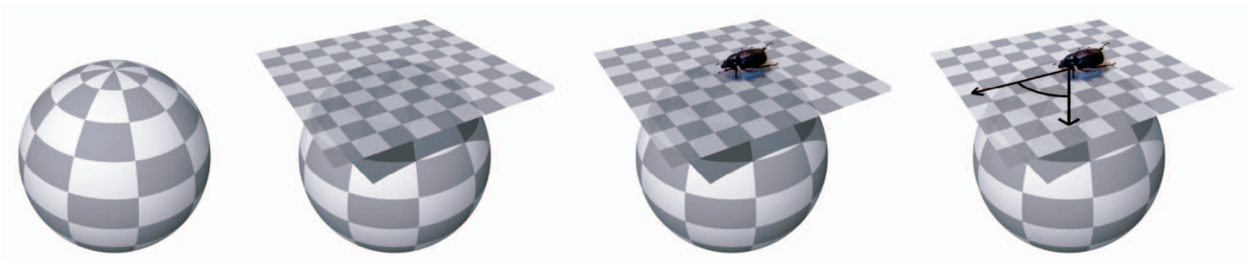
Fig. 3. The tangent plane: a bug's eye view of a Riemannian manifold.

to Riemannian geometries, such as spherical geometry, $\mathbb{S}^2$, and hyperbolic geometry, $\mathbb{H}^2$.

4.1 Basics of Riemannian Geometry

In 1894, Riemann described a generalization of the geometry of surfaces, which had been studied earlier by Gauss, Bolyai, and Lobachevsky. Two well-known special cases of Riemannian geometries are the two standard non-Euclidean types, spherical geometry and hyperbolic geometry. This generalization led to the modern concept of a Riemannian manifold.

Riemannian geometries have less convenient structure than Euclidean geometry, but they do 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. This means that, for every point on the manifold, it is possible to define local notions of length and angle. In effect, the tangent plane provides a *bug's-eye view* of the manifold; see Fig. 3.

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; see Fig. 4. We can similarly define the distance between two points, $d(x, y)$, as the length of a geodesic between them.

4.2 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. 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; see Fig. 5. Determining the direction in which x should 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). Pseudocode for a traditional Euclidean

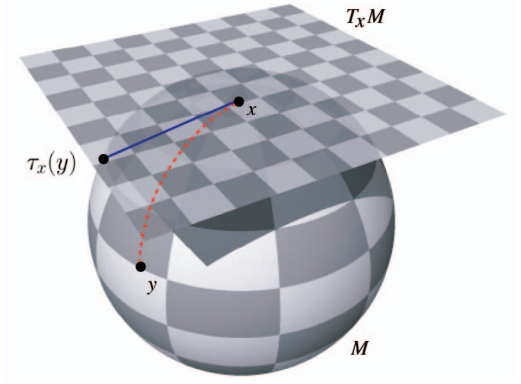

Fig. 4. A curve and its derivative in the tangent space.

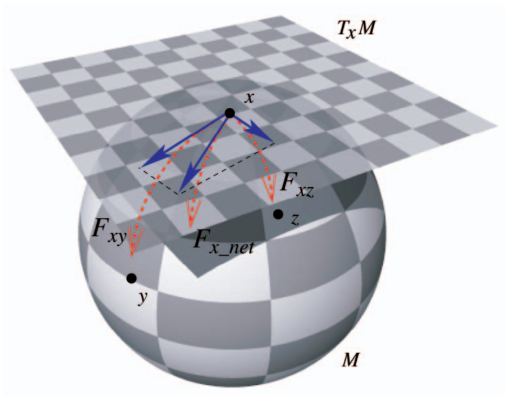

Fig. 5. Computing the net force on a node x as a result of its interaction with two other nodes, y and z.

spring embedder and its non-Euclidean counterpart are in Fig. 6.

Given a node $v_i \in V(G)$ 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 v_i as a vector in T_xM. Given the vector in T_xM, the new position, x' of v_i in T_xM is calculated using standard techniques, typically by multiplying f by a scalar. The desired position of v_i in M is then given by $\tau_x^{-1}(x)$. An overview of the process is in Fig. 7.

5 Hyperbolic Geometry

5.1 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.

The study of hyperbolic geometry began in the 18th century [1]. 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 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 lay out the tree with a uniform distribution of the nodes and with uniform edge lengths.

```
generic_algorithm($G$)
while not done do
    foreach $n \in G$ do
        position[$n$] := force_directed_placement($n$, $G$)
    end
end

non_Euclidean_algorithm($G$)
while not done do
    foreach $n \in G$ do
        $x$ := position[$n$]
        $G'$ := $\tau_x(G)$
        $x'$ := force_directed_placement($n$, $G'$)
        position[$n$] := $\tau_x^{-1}(x')$
    end
end
```

Fig. 6. A generic Euclidean spring embedder and its non-Euclidean counterpart.

5.2 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. 10.

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 nonintersecting if their associated chords are nonintersecting. 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 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.

5.3 Tangent Space Mapping

There are many possible ways to compute the mapping to and from the tangent space. Here, we present the details

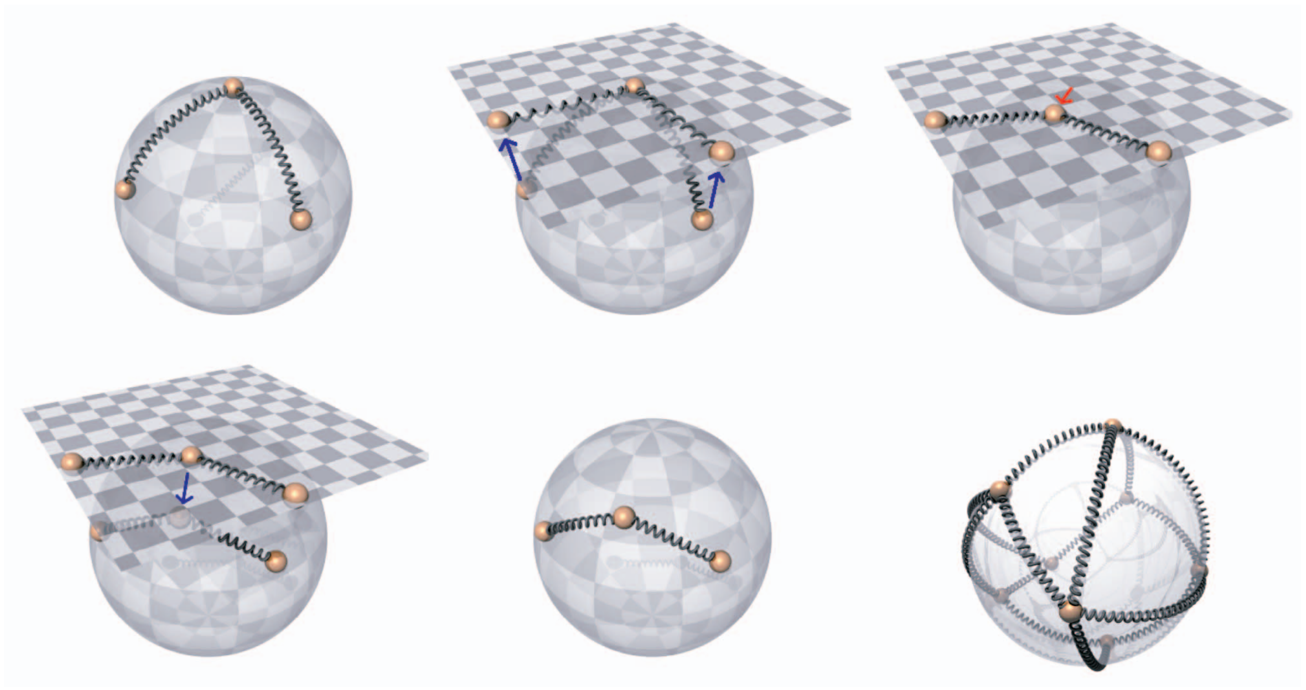
Fig. 7. An overview of a non-Euclidean spring embedder.

about one such mapping, which we also implemented. As illustrated in Fig. 6, the problem reduces to defining the mappings τ_x and τ_x^{-1} so that they meet the three criteria from Section 4.1.

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 are 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.

6 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 x 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 parameterization

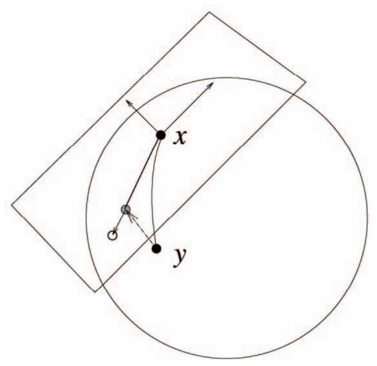

Fig. 8. Mapping to the tangent space via mapping to the tangent plane, followed by a renormalization for the length of the vector in the tangent space.

$emb(x) = (\cos\theta\sin\phi, \cos\phi, \sin\theta\sin\phi)$. We can calculate the tangent plane at any point on the sphere by taking the space spanned by the two partial derivative vectors:

$$u_x = (-\sin\theta\sin\phi, 0, \cos\theta\sin\phi),$$
$$v_x = (\cos\theta\cos\phi, -\sin\phi, \sin\theta\cos\phi).$$

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 $(emb(y)\cdot u_x, emb(y)\cdot v_x)$. 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. 8.

The inverse of this mapping $\tau_x^{-1}(y)$, illustrated in Fig. 9, 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 (for example, by $p_x = \tau_x(y)_y,\ p_y = -\tau_x(y)_x$). 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 $\theta = arctan\frac{z}{x}$, $\phi = \arccos y$.

7 Practical Considerations

We have implemented both the hyperbolic and the spherical layouts as a part of the `graphael` system [5]. In addition to specifying the tangent space mappings which define how the nodes of a graph respond to the forces of other nodes, there are further practical considerations to be addressed, particularly in the layout of larger graphs. Some of these considerations are specific to the space in which the graph is to be embedded, as, for example, hyperbolic space

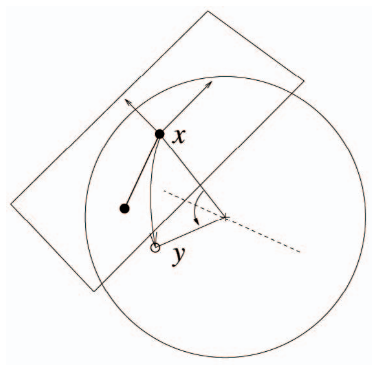

Fig. 9. Mapping from the tangent space via a rotation. The dotted line represents the axis of rotation, $up_x + vp_y$.

is susceptible to different layout problems than spherical. We present some of the images we obtained as well as describe some of the practical challenges posed by the non-Euclidean geometries.

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

Fig. 1 shows a title-word graph obtained from the graph drawing literature. 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* [15]. The size of a node is determined by the frequency of the corresponding word and an edge is placed between two nodes if they cooccur in at least one paper.

The images in Fig. 1 show layouts of the same graph obtained in $\mathbb{H}^2$, $\mathbb{S}^2$, and $\mathbb{R}^2$, obtained using our implementation of the algorithms described in this paper. Fig. 10 shows four views of the graph using different centers of attention and nicely illustrates the focus+context properties of hyperbolic space. Fig. 11 shows four views of the same graph in spherical space, again using different centers of attention.

7.2 Curvature Considerations

We have found that, for spaces of high curvature (both spherical and hyperbolic), there is a tendency to reach a local minimum that does not correspond to a visually pleasing layout. In spherical space, this typically occurs when the space is too small to accommodate an optimal layout of the graph. In such situations, the repulsive forces between nodes dominate and the layout "locks" into an unsatisfactory state.

In hyperbolic space, it is possible for a similar problem to occur. In this case, rather than due to lack of enough space to achieve a satisfactory layout, an overabundance of space allows the nodes to space themselves out too easily. For some intuitive explanation, consider that, since the circumference of a circle grows exponentially with its radius, in a space of high enough curvature (or, equivalently, large enough edge length), it is possible to lay out any fixed number of nodes near the boundary of the circle so that the distance between neighboring nodes on the circle is equal to the diameter of the circle. Thus, adjacent nodes are as likely to be neighboring on the circle as they are to be at opposite ends. This often results in a layout with many undesirable edge crossings.

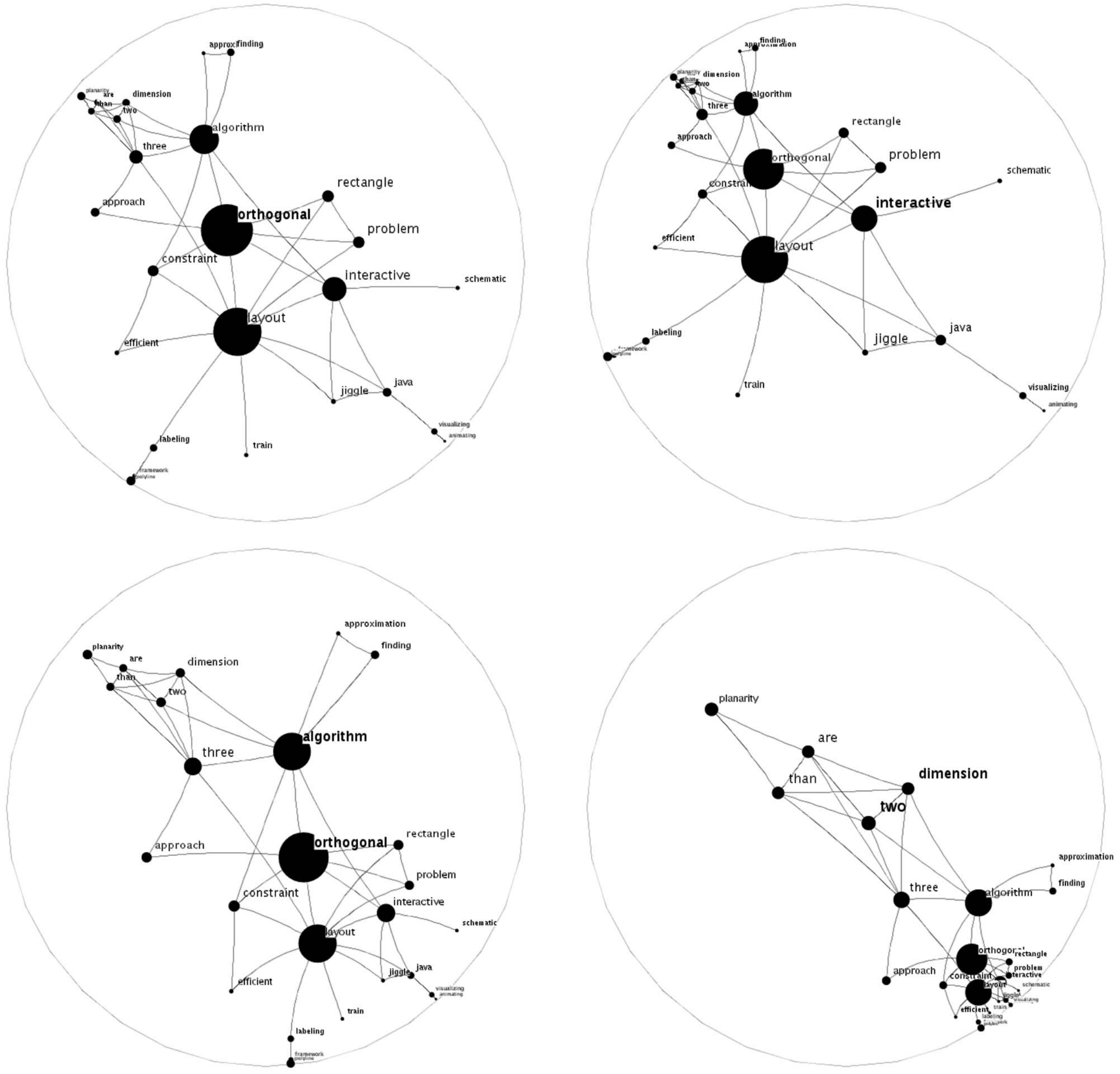

Fig. 10. Layouts of the title-word graph with different centers of attention in hyperbolic space.

We have considered ways to address the curvature problem. It is likely that multiscale layout methods along the lines of [7] and [9] will be of some help in reducing these effects. It is also likely that force equations designed specifically to work well within a particular geometry could avoid such unsatisfactory layouts. Finally, we have also found that better layouts can be achieved by beginning the iterative layout process in a space which is nearly Euclidean, then gradually increasing the magnitude of the curvature of the space during the layout process until it reaches the desired value.

It is worth noting that some graphs are better suited to non-Euclidean spaces than other. A regular grid, for example, is well suited to Euclidean space. In a hyperbolic space of high curvature, however, the layout of the grid is quite distorted. This can be thought of as the inverse case of the distortion that is necessary to embed a tree in Euclidean space. How well hyperbolic space suits a given graph can be observed by looking at a plot of the sum of all the magnitudes of the forces acting in a graph versus the curvature of the space the graph is embedded in; see Fig. 12. In graphs that can be embedded well in hyperbolic space, such as trees, after a certain curvature, we see monotonically decreasing stress. In graphs better suited to Euclidean space, such as grids, we instead see an early decrease followed by monotonically increasing stress.

7.3 Multiscale Considerations

Multiscale layout methods are needed even in the Euclidean case, in order to apply spring embedder algorithms. These methods typically greatly improve the speed and quality with which large graphs can be laid out. In

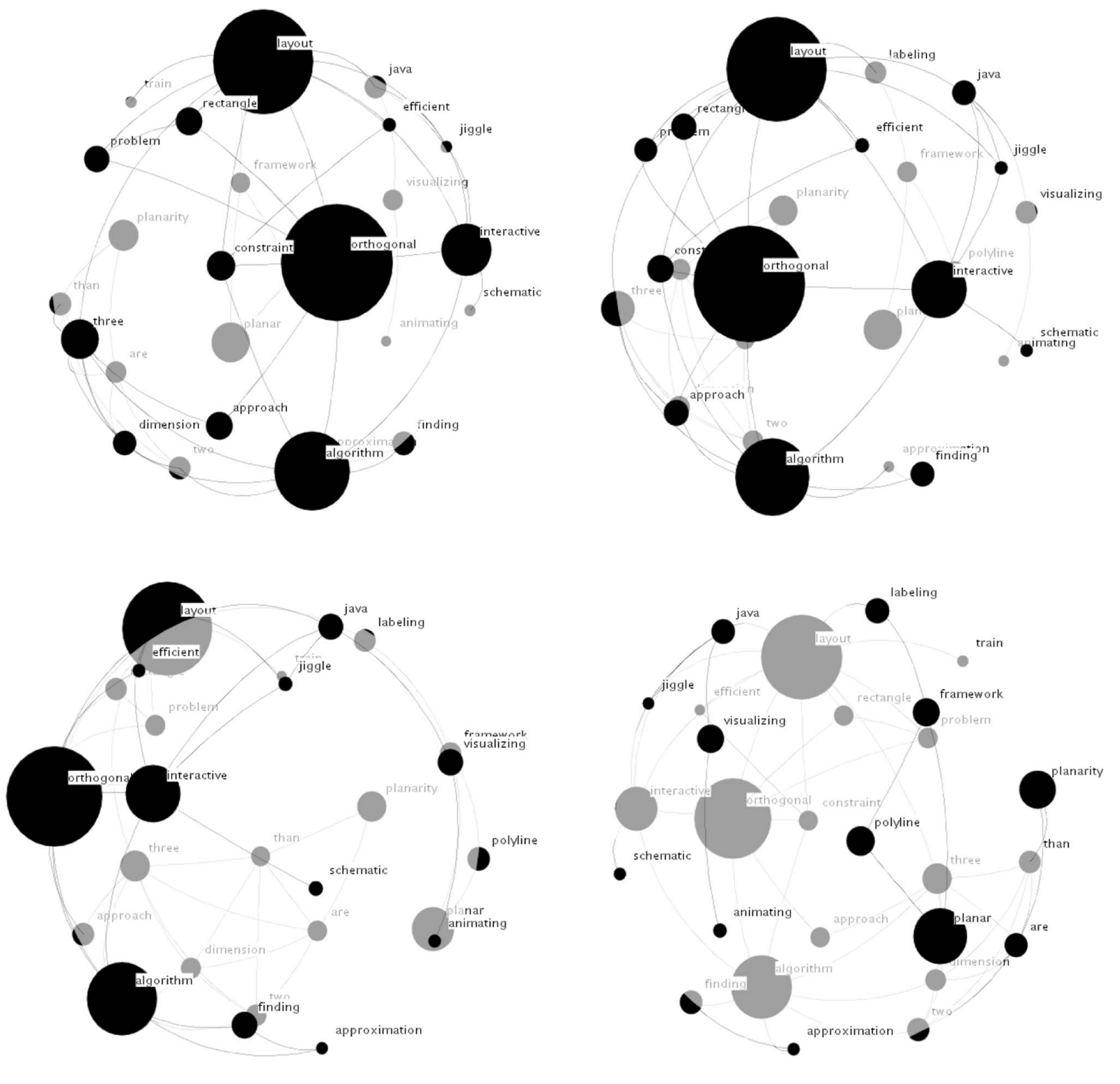

Fig. 11. Layouts of the title-word graph with different centers of attention in spherical space.

addition, multiscale layouts may be useful in avoiding highly suboptimal layouts. Unfortunately, such methods often employ techniques which cannot be directly applied using the framework for non-Euclidean graph layouts that we have set up. Consider, for example, estimating the position of a new node by interpolating the positions of already laid-out nodes [7]. For three or more nodes, this interpolation cannot be directly expressed in a general Riemannian geometry using the tools we have described.

Although it is possible to analytically calculate such interpolations for certain geometries, it is not always necessary to do so. Since such techniques often function as rough heuristics, in many cases, it is possible to perform the calculations in the tangent space of one of the nodes involved. For example, to interpolate the positions of some set of nodes, one node could be selected at random, and the calculations done within the tangent space of this node.

7.4 Oscillations in $\mathbb{S}^2$

Many spring embedders use a heuristic intended to expedite the graph's convergence to a steady layout—nodes which are changing their direction are slowed down while nodes moving in a straight line are sped up. This dampens oscillations in the layout process and normally improves layout quality. In spherical space, however, it is possible for a node to oscillate *and* move in a straight line, as occurs when a node repeatedly travels around the entire space. In such cases, these heuristics can actually degrade the layout by reinforcing this sort of motion. While such heuristics seem to pose no fundamental problems, they should be used with care.

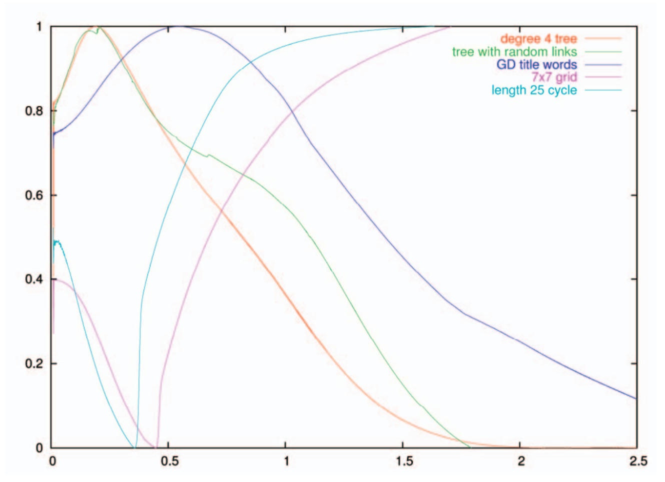

Fig. 12. Plot of the stress in the final layout of a graph as a function of the (negative of the) curvature of the space it is embedded in. This is only intended to compare the relative shapes of the stress curves and the y-axis has been normalized for each graph so that its values range from zero to one.

7.5 Precision in $\mathbb{H}^2$

Hyperbolic space also has its own set of problems which must be surmounted in calculating the layout of certain graphs. This problem arises from one of the most useful features of hyperbolic space—that the circumference of a circle grows exponentially with its radius. This means that, to achieve a given spatial resolution, floating-point representations of positions in hyperbolic space must use a number of bits proportional to the distance between nodes in the layout. If points are not represented in this manner, the layout of a large graph can be subject to serious floating-point errors. While the precision problem can be addressed with the traditional arbitrary precision floating-point representation, this would impose a speed penalty, prohibitively large for large graphs.

Note, however, that the further away two nodes are, the less accurately we need to calculate their absolute positions in tangent space. This makes it desirable to use a point representation better suited to calculating the relative position of nodes than their absolute positions. One such representation would be a discrete lattice over the hyperbolic plane, such as a hyperbolic tessellation. The position of each node can then be stored as a combination of a designation for the nearest lattice point and an offset from that point. This offset can be stored as a point in the Poincaré disk, with the origin being the nearest lattice point. The relative positions of two points can then be compared by taking the offset into account and walking the lattice from one to the other.

8 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. Multiscale methods and high dimensional embedding have been successfully used to extend Euclidean spring embedders. Generalizing the non-Euclidean spring embedders along the lines of [7] and [9] 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.

Acknowledgments

The authors would like to thank the anonymous referees of the journal and conference versions of the paper for their comments and suggestions. This work is partially supported by the US National Science Foundation under grant ACR-0222920.

References

[1] J.W. Anderson, *Hyperbolic Geometry.* Springer-Verlag, 1999.
[2] G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs.* Englewood Cliffs, N.J.: Prentice Hall, 1999.
[3] P. Eades, “A Heuristic for Graph Drawing,” *Congressus Numerantium,* vol. 42, pp. 149-160, 1984.
[4] D. Eppstein, “Hyperbolic Geometry, Möbius Transformations, and Geometric Optimization,” *Proc. MSRI Introductory Workshop Discrete and Computational Geometry,* 2003.
[5] C. Erten, P.J. Harding, S.G. Kobourov, K. Wampler, and G. Yee, “GraphAEL: Graph Animations with Evolving Layouts,” *Proc. 11th Symp. Graph Drawing,* pp. 98-110, 2003.
[6] T.M.J. Fruchterman and E.M. Reingold, “Graph Drawing by Force-Directed Placement,” *Software—Practice and Experience,* vol. 21, no. 11, pp. 1129-1164, 1991.
[7] 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,* vol. 29, no. 1, pp. 3-18, 2004.
[8] C. Gotsman, X. Gu, and A. Sheffer, “Fundamentals of Spherical Parameterization for 3D Meshes,” *ACM Trans. Graphics,* vol. 22, pp. 358-363, 2003.
[9] D. Harel and Y. Koren, “A Fast Multi-Scale Method for Drawing Large Graphs,” *J. Graph Algorithms and Applications,* vol. 6, pp. 179-202, 2002.
[10] I. Herman, G. Melançon, and M.S. Marshall, “Graph Visualization and Navigation in Information Visualization: A Survey,” *IEEE Trans. Visualization and Computer Graphics,* vol. 6, no. 1, pp. 24-43, Jan.-Mar. 2000.
[11] T. Kamada and S. Kawai, “An Algorithm for Drawing General Undirected Graphs,” *Information Processing Letters,* vol. 31, no. 1, pp. 7-15, Apr. 1989.
[12] M. Kaufmann and D. Wagner, *Drawing Graphs: Methods and Models.* Springer-Verlag, 2001.

[13] S.G. Kobourov and K. Wampler, "Non-Euclidean Spring Embedders," *Proc. 10th Ann. IEEE Symp. Information Visualization (InfoVis),* pp. 207-214, 2004.

[14] Y. Koren, L. Carmel, and D. Harel, "ACE: A Fast Multiscale Eigenvector Computation for Drawing Huge Graphs," *Proc. IEEE Symp. Information Visualization,* pp. 123-144, 2002.

[15] *Proc. Seventh Int'l Symp. Graph Drawing,* J. Kratochvil, ed., 1999.

[16] J. Lamping, R. Rao, and P. Pirolli, "A focus+context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies," *Proc. Computer Human Interaction,* pp. 401-408, 1995.

[17] T. Munzner, "H3: Laying Out Large Directed Graphs in 3D Hyperbolic Space," *Proc. IEEE Symp. Information Visualization,* L. Lavagno and W. Reisig, eds., pp. 2-10, 1997.

[18] T. Munzner, "Drawing Large Graphs with h3viewer and Site Manager," *Proc. Sixth Symp. Graph Drawing,* pp. 384-393, 1998.

[19] T. Munzner and P. Burchard, "Visualizing the Structure of the World Wide Web in 3D Hyperbolic Space," *Proc. Symp. Virtual Reality Modeling Language,* pp. 33-38, 1996.

[20] J. Ontrup and H. Ritter, "Hyperbolic Self-Organizing Maps for Semantic Navigation," *Proc. Advances in Neural Information Processing Systems 14,* pp. 1417-1424, 2001.

[21] D.I. Ostry, "Some Three-Dimensional Graph Drawing Algorithms," master's thesis, Univ. of Newcastle, Australia, 1996.

[22] H. Ritter, "Self-Organizing Maps on Non-Euclidean Spaces," *Kohonen Maps,* S. Oja and E. Kaski, eds., pp. 97-110, 1999.

[23] K. Sugiyama and K. Misue, "Graph Drawing by the Magnetic Spring Model," *J. Visual Languages and Computing,* vol. 6, no. 3, pp. 217-231, 1995.

[24] W.T. Tutte, "How to Draw a Graph," *Proc. London Math. Soc.,* vol. 13, no. 52, pp. 743-768, 1963.

[25] J.J. van Wijk and W.A.A. Nuij, "A Model for Smooth Viewing and Navigation of Large 2D Information Spaces," *IEEE Trans. Visualization and Computer Graphics,* vol. 10, no. 4, pp. 447-458, July/Aug. 2004.

Stephen G. Kobourov received BS degrees in computer science and mathematics from Dartmouth College (1995), the MS degree in computer science from Johns Hopkins University (1997), and the PhD degree in computer science from Johns Hopkins University (2000). He is currently an assistant professor in the Computer Science Department at the University of Arizona. His primary research interests are in geometric algorithms, graph drawing, and information visualization. He was program cochair of the International Symposium on Graph Drawing in 2002 and was a coorganizer of a Dagstuhl seminar on Graph Drawing in 2005.

Kevin Wampler received BS degrees in mathematics and computer science from the University of Arizona (2003) and is currently pursuing the PhD degree in computer science at the University of Washington. His primary research interests are in computer graphics, computer vision, artificial intelligence, and information visualization.