Fast spherical drawing of triangulations: an experimental study of graph drawing tools

Luca Castelli Aleardi, Gaspard Denis, Eric Fusy

Fast spherical drawing of triangulations: an experimental study of graph drawing tools *

Luca Castelli Aleardi[†] Gaspard Denis[‡] Éric Fusy[§]

Abstract

We consider the problem of computing a spherical crossing-free geodesic drawing of a planar graph: this problem, as well as the closely related spherical parameterization problem, has attracted a lot of attention in the last two decades both in theory and in practice, motivated by a number of applications ranging from texture mapping to mesh remeshing and morphing. Our main concern is to design and implement a linear time algorithm for the computation of spherical drawings provided with theoretical guarantees. While not being aesthetically pleasing, our method is extremely fast and can be used as initial placer for spherical iterative methods and spring embedders. We provide experimental comparison with initial placers based on planar Tutte parameterization. Finally we explore the use of spherical drawings as initial layouts for (Euclidean) spring embedders: experimental evidence shows that this greatly helps to untangle the layout and to reach better local minima.

1 Introduction

In this work we consider the problem of computing in a fast and robust way a spherical layout (crossing-free geodesic spherical drawing) of a genus 0 simple triangulation. While several solutions have been developed in the computer graphics and geometry processing communities [1, 2, 3, 15, 18, 23, 26, 29], very few works attempted to test the practical interest of standard tools [6, 8, 11, 20, 7] from graph drawing developed for the non-planar (or non-Euclidean) case. On one hand, force-directed methods and iterative solvers are successful to obtain very nice layouts achieving several desirable aesthetic criteria, such as uniform edge lengths, low angle distortion or even the preservation of symmetries. Their main drawbacks rely on the lack of rigorous theoretical guarantees and on their expensive runtime costs, since their implementation requires linear solvers (for large sparse matrices)

*This work has been accepted to appear as extended abstract in the Proc. of the *17th Int. Symposium on Experimental Algorithms (SEA 2018).*

[†]LIX, Ecole Polytechnique, France, amturing@lix.polytechnique.fr

[‡]LIX, Ecole Polytechnique, France, gaspard.denis@hotmail.fr

[§]LIX, Ecole Polytechnique, France,fusy@lix.polytechnique.fr

or sometimes non-linear optimization methods, making these approaches slower and less robust than combinatorial graph drawing tools. On the other hand, some well known tools such as linear-time grid embeddings [10, 24] are provided with worst-case theoretical guarantees allowing us to compute in a fast and robust way a crossing-free layout with bounded resolution: just observe that their practical performances allow processing several millions of vertices per second on a standard (single-core) CPU. Unfortunately, the resulting layouts are rather unpleasing and fail to achieve some basic aesthetic criteria that help readability (they often have long edges and large clusters of tiny triangles).

Motivation. It is commonly assumed that starting from a good initial layout (referred to as *initial guess* in [23]) is crucial for both iterative methods and spring embedders. A nice initial configuration, that is closer to the final result, should help to obtain nicer layouts (this was explored in [13] for the planar case). This could be even more relevant for the spherical case, where an initial layout having many edge-crossings can be difficult to unfold in order to obtain a valid spherical drawing. Moreover, the absence of natural constraints on the sphere prevents in some cases from eliminating all crossings before the layouts collapse to a degenerate configuration. One of the motivations of this work is to get benefit of a prior knowledge of the graph structure: if its combinatorics is known in advance, then one can make use of fast graph drawing tools and compute almost instantaneously a crossing-free layout to be used as starting point for running more expensive force-directed tools.

Related works. A first approach for computing a spherical drawing consists in projecting a (convex) polyhedral representation of the input graph on the unit sphere: one of the first works [25] provided a constructive version of Steinitz theorem (unfortunately its time complexity was quadratic). Another very simple approach consists in planarizing the graph and to apply well known tools from mesh parameterizations (see Section 2.1 for more details): the main drawback is that, after spherical projection, the layout does not always remain crossing-free. Along another line of research, several works proposed generalizations of the barycentric Tutte parameterization to the sphere. Unlike the planar case, where boundary constraints guarantees the existence of crossing-free layouts, in the spherical case both the theoretical analysis and the practical implementations are much more challenging. Several works in the geometry processing community [3, 15, 23, 29] expressed the layout problem as an energy minimization problem (with non-linear constraints) and proposed a variety of iterative or optimization methods to solve the spherical Tutte equations: while achieving nice results on the tested 3D meshes, these methods lack rigorous theoretical guarantees on the quality of the layout in the worst case (for a discussion on the existence of non degenerate solutions of the spherical Tutte equations we refer to [18]). A very recent work [1] proposed an adaptation of the approach based on the Euclidean orbifold Tutte parameterization [2] to the spherical case: the experimental results are very promising and come with some theoretical guarantees (a couple of weak assumptions are still necessary to guarantee the validity of the drawing). However the layout computation becomes much more expensive since it involves solving non-linear problems, as reported in [1]. A few papers in the

graph drawing domain also considered the spherical drawing problem. Fowler and Kobourov proposed a framework to adapt force-directed methods [16] to spherical geometry, and a few recent works [6, 8, 11, 7] extend some combinatorial tools to produce planar layouts of non-planar graphs: some of these tools can be combined to deal with the spherical case, as we will show in this work (as far as we know, there are not existing implementations of these algorithms).

1.1 Our contribution

- Our first main contribution is to design and implement a fast algorithm for the computation of spherical drawings. We make use of several ingredients [11, 6, 7] involving the well-known *canonical orderings* and can be viewed as an adaptation of the *shift* paradigm proposed by De Fraysseix, Pach and Pollack [10]. As illustrated by our experiments, our procedure is extremely fast, with theoretical guarantees on both the runtime complexity and the layout resolution.

- While not being aesthetically pleasing (as in the planar case), our layouts can be use as initial vertex placement for iterative parameterization methods [3, 23] or spherical spring embedders [20]. Following the approach suggested by Fowler and Kobourov [13], we compare our combinatorial algorithm with two standard initial placers used in previous existing works [23, 29] relying on Tutte planar parameterizations: our experimental evaluations involve runtime performances and statistics concerning edge lengths.

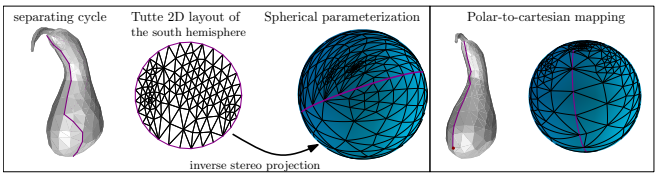
- As an application, we show in Section 5 how spherical drawings can be used as initial layouts for (Euclidean) spring embedders: as illustrated by our tests, starting from a spherical drawing greatly helps to entangle the layout and to escape from bad local minima.

All our results are provided with efficient implementations and experimental evaluations on a wide collection of real-world and synthetic datasets.

2 Preliminaries

Planar graphs and spherical drawings. In this work we deal with *planar maps* (graphs endowed with a combinatorial planar embedding), and we consider in particular *planar triangulations* which are simple genus 0 maps where all faces are triangles (they correspond to the combinatorics underlying genus 0 3D triangle meshes). Given a graph $G = (V, E)$ we denote by $n = |V|$ (resp. by $|F(G)|$) the number of its vertices (resp. faces) and by $N(v_i)$ the set of neighbors of vertex v_i; $\mathbf{x}(v_i)$ will denote the Euclidean coordinates of vertex v_i.

The notion of planar drawings can be naturally generalized to the spherical case: the main difference is that edges are mapped to *geodesic arcs* on the unit sphere $\mathbb{S}^2$, which are minor arcs of great circles (obtained as intersection of $\mathbb{S}^2$ with an hyperplane passing through



Projected Gauss-Seidel($\mathbf{x}^0$, λ, ε)

```
r = 0; // iteration counter
do {
  for(i = 1; i ≤ n; i++) {
    s = (1 − λ)x^r(v_i) + λ Σ_j w_ij x^r(v_j)
    x^{r+1}(v_i) = s/‖s‖
  }
  r++;
} while (‖x^r − x^{r−1}‖ > ε))
```

Figure 1: (left) Two spherical parameterizations of the `gourd` graph obtained obtained via Tutte's planar parameterization. (right) The pseudo-code of the *Projected Gauss-Seidel* method.

the origin). A *geodesic drawing* of a map should preserve the cyclic order of neighbors around each vertex (such an embedding is unique for triangulations, up to reflexions of the sphere). As in the planar case, we would aim to obtain *crossing-free* geodesic drawings, where geodesic arcs do not intersect (except at their extremities). In the rest of this work we will make use of the term *spherical drawings* when referring to drawings satisfying the requirements above. Sometimes, the weaker notion of *spherical parameterization* (an homeomorphism between an input mesh and $\mathbb{S}^2$) is considered for dealing with applications in the geometry processing domain (such as mesh morphing): while the bijectivity between the mesh and $\mathbb{S}^2$ is guaranteed, there are no guarantees that the triangle faces are mapped to spherical triangles with no overlaps (obviously a spherical drawing leads to a spherical parameterization).

2.1 Initial Layouts

Part of this work will be devoted to compare our drawing algorithm (Section 3) to two spherical parameterization methods involving Tutte planar parameterization: both methods have been used as *initial placers* for more sophisticated iterative spherical layout algorithms.

Inverse Stereo Projection layout (`ISP`). For the first initial placer, we follow the approach suggested in [23] (see Fig. 1). The faces of the input graph G are partitioned into two components homeomorphic to a disk: this is achieved by computing a vertex separator defining a simple cycle of small size (having $O(\sqrt{n})$ vertices) whose removal produces a balanced partition (G^S, G^N) of the faces of G. The two graphs G^S and G^N are then drawn in the plane using Tutte's barycentric method: boundary vertices lying on the separator are mapped on the unit disk. Combining a Moebius inversion with the inverse of a stereographic projection we obtain a spherical parameterization of the input graph: while preserving some of the aesthetic appeal of Tutte's planar drawings, this map is bijective but cannot produce in general a crossing-free spherical drawing (straight-line segments in the plane are not mapped to geodesics by inverse stereographic projection). In our experiments we adopt a growing-region heuristic to compute a simple separating cycle: while not having theoretical guarantees, our approach is simple to implement and very fast, achieving balanced partitions in practice (separators are of size roughly $\Theta(\sqrt{n})$ and the balance ratio $\varrho = \frac{\min(|F(G^S)|,|F(G^N)|)}{|F(G)|}$

is always between 0.39 and 0.49 for the tested data) [1] .

Polar-to-Cartesian layout (PC). The approach adopted in [29] consists in planarizing the graph by cutting the edges along a simple path from a south pole v^S to a north pole v^N. A planar rectangular layout can be computed by applying standard Tutte parameterization with respect to the azimuthal angle $\theta \in (0, 2\pi)$ and to the polar angle $\phi \in [0, \pi]$: the spherical layout, obtained by the polar-to-cartesian projection, is bijective but not guaranteed to be crossing-free.

2.2 Spherical drawings and parameterizations

The spherical layouts described above can used as initial guess for more sophisticated iterative schemes and force-directed methods for computing spherical drawings. For the sake of completeness we provide an overview of the algorithms that will be tested in Section 4.

Iterative relaxation: projected Gauss-Seidel. The first method can be viewed as an adaptation of the iterative scheme solving Tutte equations (see Fig. 1). This scheme consists in moving points on the sphere in tangential direction in order to minimize the spring energy

$$\mathcal{E} = \frac{1}{2} \sum_{i=1}^{n} \sum_{j \in N(i)} w_{ij} \|\mathbf{x}(v_i) - \mathbf{x}(v_j)\|^2 \tag{1}$$

with the only constraint $\|\mathbf{x}(v_i)\| = 1$ for $i = 1 \dots n$ (in this work we consider uniform weights w_{ij}, as in Tutte's work). As opposite to the planar case, there are no boundary constraints on the sphere, which makes the resulting layouts collapse in many cases to degenerate solutions. As observed in [18, 23] this method does not always converge to a valid spherical drawing, and its practical performance strongly depends on the geometry of the starting initial layout $\mathbf{x}^0$. While not having theoretical guarantees, this method is quite fast allowing to quickly decrease the residual error: it thus can be used in a first phase and combined with more stable iterative schemes leading in practice to better convergence results [23] (still lacking of rigorous theoretical guarantees).

Alexa's method In order to avoid the collapse of the layout, without introducing artificial constraints, Alexa [3] modified the iterative relaxation above by penalizing long edges (that tend to move all vertices in a same hemisphere). More precisely, the vertex v_i is moved according to a displacement $\triangle_i = c \frac{1}{deg(v_i)} \sum_j (\mathbf{x}(v_i) - \mathbf{x}(v_j)) \|\mathbf{x}(v_i) - \mathbf{x}(v_j)\|$ and then reprojected on the sphere. The parameter c regulates the step length, and can be chosen to be proportional to the inverse of the longest edge incident to a vertex, improving the convergence speed.

[1] The computation of small cycle separators for planar triangulations is a very challenging task. This work does not focus on this problem: we refer to recent results [14] providing the first practical implementations with theoretical guarantees.

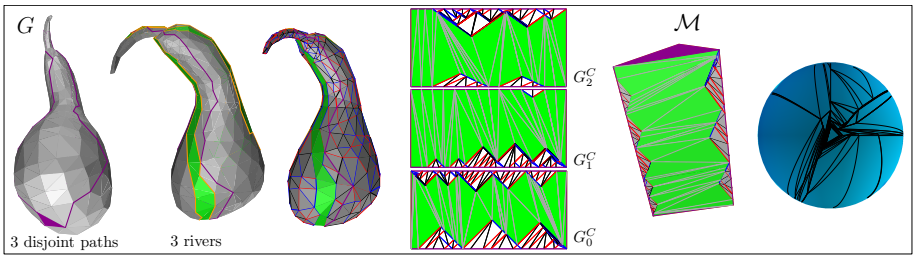


Figure 2: Computation of a spherical drawing based on a prism layout of the `gourd` graph (326 vertices). Three vertex-disjoint chord-free paths lead to the partition of the faces of G into three regions which are each separated by one river (green faces). Our variant of the `FPP` algorithm allows to produce three rectangular layouts, where boundary vertex locations do match on identified (horizontal) sides. One can thus glue the planar layouts to obtain a 3D prism: its central projection on the sphere produces a spherical drawing. Edge colors (blue, red and black) are assigned during the incremental computation of a *canonical labeling* [11], according to the Schnyder wood local rule.

(Spherical) Spring Embedders. While spring embedders are originally designed to produce 2D or 3D layouts, one can adapt them to non euclidean geometries. We have implemented the standard *spring-electrical model* introduced in [16] (referred to as `FR`), and the spherical version following the framework described by Kobourov and Wampler [20] (called `Spherical FR`). As in [16] we compute attractive forces (between adjacent vertices) and repulsive forces (for any pair of vertices) acting on vertex u, defined by:

$$F_a(u) = \sum_{(u,v)\in E} \frac{\|\mathbf{x}(u)-\mathbf{x}(v)\|}{K}(\mathbf{x}(u)-\mathbf{x}(v)),\ F_r(u) = \sum_{v\in V, v\neq u} \frac{-CK^2(\mathbf{x}(v)-\mathbf{x}(u))}{\|\mathbf{x}(u)-\mathbf{x}(v)\|^2}$$

where the values C (the strength of the forces) and K (the optimal distance) are scale parameters. In the spherical case, we shift the repulsive forces by a constant term, making the force acting on pairs of antipodal vertices zero.

3 Fast spherical embedding with theoretical guarantees: `SFPP` layout

We now provide an overview of our algorithm for computing a spherical drawing of a planar triangulation G in linear time, called `SFPP` layout (the main steps are illustrated in Fig 2). We make use of an adaptation of the *shift* method used in the incremental algorithm of de Fraysseix, Pach and Pollack [10] (referred to as `FPP` layout): our solution relies on the combination of several ideas developed in [11, 6, 7]. For the sake of completeness, a more detailed presentation in given the Appendix.

Mesh segmentation. Assuming that there are two non-adjacent faces f^N and f^S, one can find 3 disjoint and chord-free paths P_0, P_1 and P_2 from f^S to f^N (planar triangulations are 3-connected). Denote by u_0^N, u_1^N and u_2^N the three vertices of f^N on P_0, P_1 and P_2 (define similarly the three neighbors u_0^S, u_1^S, u_2^S of the face f^S). We first compute a partition of the faces of G into 3 regions, cutting G along the paths above and removing f^S and f^N. We thus have three quasi-triangulations G_0^C, G_1^C and G_2^C that are planar maps whose inner faces are triangles, and where the edges on the outer boundary are partitioned into four sides. The first pair of opposite sides only consist of an edge (drawn as vertical segment in Fig. 2), while the remaining pair of opposite sides contains vertices lying on P_i and P_{i+1} respectively (indices being modulo 3): according to these definitions, G_i^C and G_{i+1}^C share the vertices lying on P_{i+1} (drawn as a path of on horizontal segments in Fig. 2).

Grid drawing of rectangular frames. We apply the algorithm described in [11] to obtain three rectangular layouts of G_0^C, G_1^C and G_2^C: this algorithm first separates each G_i^C into two sub-graphs by removing a so-called *river*: an outer-planar graph consisting of a face-connected set of triangles which corresponds to a simple path in the dual graph, starting at f^S and going toward f^N. The two-subgraphs are then processed making use of the *canonical labeling* defined in [11]: the resulting layouts are stretched and then merged with the set of edges in the river, in order to fit into a rectangular frame. Just observe that in our case a pair of opposite sides only consists of two edges, which leads to an algorithm considerably simpler to implement in practice. Finally, we apply the two-phases adaptation of the shift algorithm described in [6] to obtain a planar grid drawing of each map G_i^C, such that the positions of vertices on the path P_i in G_i^C do match the positions of corresponding vertices on P_i in G_{i+1}^C. The grid size of drawing of G_i^C is $O(n) \times O(n)$ (using the fact that the two opposite sides $(u_i^N, \ldots, u_i^S)$ and $(u_{i+1}^N, \ldots, u_{i+1}^S)$ of G_i^C are at distance 1).

Spherical layout. To conclude, we glue together the drawings of G_0^C, G_1^C and G_2^C computed above in order to obtain a drawing of G on a triangular prism. By a translation within the 3D ambient space we can make the origin coincides with the center of mass of the prism (upon seeing it as a solid polyhedron). Then a central projection from the origin maps each vertex on $\mathcal{M}$ to a point on the sphere: each edge (u, v) is mapped to a geodesic arc, obtained by intersecting the sphere with the plane passing trough the origin and the segment relying u and v on the prism (crossings are forbidden since the map is bijective).

Theorem 1. *Let G be a planar triangulation of size n, having two non-adjacent faces f^S and f^N . Then one can compute in $O(n)$ time a spherical drawing of G, where edges are drawn as (non-crossing) geodesic arcs of length at least $\Omega(\frac{1}{n})$.*

Some heuristics. We use as last initial placer our combinatorial algorithm of Section 3. For the computation of the three disjoint paths P_0, P_1 and P_2, we adopt again an heuristic based on a growing-region approach: while not having theoretical guarantees on the quality of the partition and the length of the paths, our results suggest that well balanced partitions are achieved for most tested graphs. A crucial point to obtain a nice layout resides in the

choice of the *canonical labeling* (its computation is performed with an incremental approach based on vertex removal). A bad canonical labeling could lead to unpleasant configurations, where a large number of vertices on the boundaries of the bottom and top sub-regions of each graph G_i are drawn along the same direction: as side effects, a few triangles use a lot of area, and the set of interior chordal edges in the river can be highly stretched, especially those close to the south and north poles. To partially address this problem, we design a few heuristics during the computation of the canonical labeling, in order to obtain more balanced layouts. Firstly, we delay the conquest of the vertices which are close to the south and north poles: this way these extremal vertices are assigned low labels (in the canonical labeling), leading to smaller and thicker triangles close to the poles. Moreover the selection of the vertices is done so as to keep the height of the *triangle caps* more balanced in the final layout. Finally, we adjust the horizontal stretch of the edges, to get more equally spaced vertices on the paths P_0, P_1 and P_2.

4 Experimental results and comparison

Experimental settings and datasets. In order to obtain a fair comparison of runtime performances, we have written pure Java implementations of all algorithms and drawing methods presented in this work. [2] Our tests involves several graphs including the 1-skeleton of 3D models (made available by the *AIM@SHAPE* repository) as well as random planar triangulations obtained with an uniform random sampler [22].

In our tests we take as an input the combinatorial structure of a planar map encoded in OFF format: nevertheless we do not make any assumption on the geometric realization of the input triangulation in 2D or 3D space. Moreover, observe that the fact of knowing the combinatorial embedding of the input graph G (the set of its faces) is a rather weak assumption, since such an embedding is essentially unique for planar triangulations and it can be easily retrieved from the graph connectivity in linear time [21]. We run our experiments on a HP EliteBook, equipped with an Intel Core i7 2.60GHz (with Ubuntu 16.04, Java 1.8 64-bit, using a single core, and 4GB of RAM for the JVM).

4.1 Quantitative evaluation of aesthetic criteria

In order to obtain a quantitative evaluation of the layout quality we compute the spring energy $\mathcal{E}$ defined by Eq. 1 and two metrics measuring the edge lengths and the triangle areas. As suggested in [13] we compute the average percent deviation of edge lengths, according to

$$\mathfrak{el} := 1 - \left(\frac{1}{|E|} \sum_{e \in E} \frac{|l_g(e) - l_{avg}|}{\max(l_{avg}, l_{max} - l_{avg})} \right)$$

where $l_g(e)$ denotes the geodesic distance of the edge e, and l_{avg} (resp. l_{max}) is the average geodesic edge length (resp. maximal geodesic edge length) in the layout. In a similar manner

[2] Datasets, source codes and runnable Java applications are available `http://www.lix.polytechnique.fr/ amturing/software.html`

			preprocessing		Layout computation		PC	ISP
mesh	vertices	faces	rivers comput.	canonical labeling	shift algorithm	prism projection	linear solver	linear solver
Egea	8268	16K	0.015	0.017	0.005	0.017	0.24	0.16
Gargoyle	10002	20K	0.016	0.018	0.007	0.025	0.26	0.22
Bunny	26002	52K	0.017	0.031	0.019	0.036	1.14	0.75
Iphigenia	49922	99K	0.023	0.049	0.025	0.046	2.38	1.44
Camille's hand	195557	391K	0.076	0.121	0.073	0.125	17.02	7.92
Eros	476596	950K	0.162	0.260	0.132	0.255	50.54	29.99
Chinese dragon	655980	1.3M	0.174	0.314	0.157	0.433	89.64	53.12

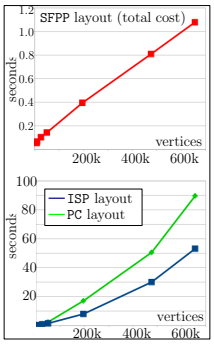


Table 1: This table reports the runtime performance of all steps involved in the computation of the SFPP layout obtained with the algorithm of Section 3. The overall cost (red chart) includes the preprocessing phase (computing the three rivers and the canonical labeling) and the layout computation (running the two-phases shift algorithm, constructing and projecting the prism). The last two columns report the timing cost for solving the linear systems for the ISP and PC layouts (see blue/green charts), using the MTJ conjugate gradient solver. All results are expressed in seconds.

we compute the average percent deviation of triangle areas, denoted by $\mathfrak{a}$. The metrics $\mathfrak{el}$ and $\mathfrak{a}$ take values in $[0 \dots 1]$, and higher values indicate more uniform edge lengths and triangle areas [3].

4.2 Timing performances: comparison

The runtime performances reported in Table 1 clearly show that our SFPP algorithm has an asymptotic linear-time behavior and in practice is much faster than other methods based on planar parameterization. For instance the ISP layout adopted in [23] requires to solve large linear systems: among the tested Java libraries (MTJ, Colt, PColt, Jama), we found that the linear solvers of the MTJ have the best runtime performances for the solution of large sparse linear systems (in our tests we run the *conjugate gradient* solver, setting a numeric tolerance of 10^{-6}). Observe that a slightly better performance can be achieved with more sophisticated schemes or tools (e.g. Matlab solvers) as done in [2, 23]. Nevertheless the timing cost still remains much larger than ours: as reported in [2] the orbifold parameterization of the dragon graph requires 19 seconds (for solving the linear systems, on a 3.5GHz Intel i7 CPU).

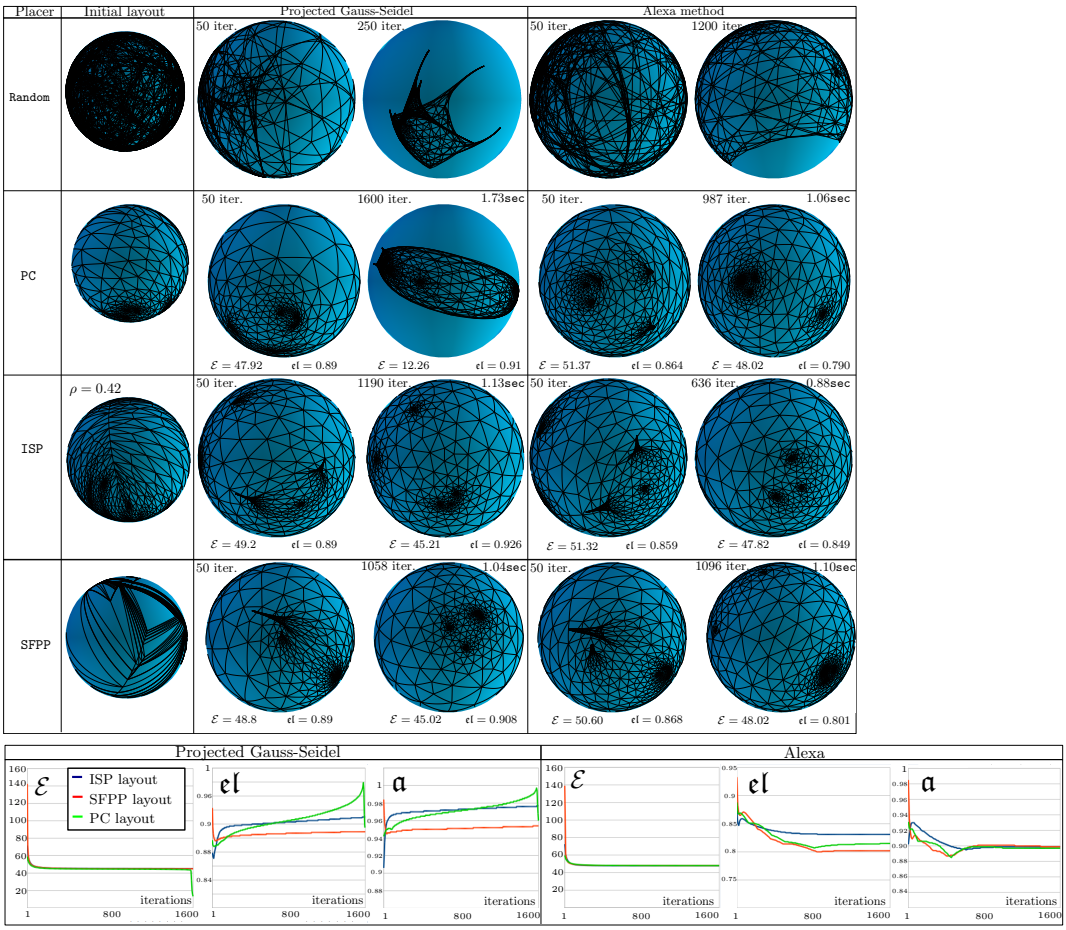


Figure 3: These pictures illustrate the use of different initial placers as starting layouts for two iterative schemes on the `dog` graph (1480 vertices). For each initial layout, we first run 50 iterations of the projected Gauss-Seidel and Alexa method, and then we run the two methods until a valid spherical drawing (crossing free) is reach. The charts below show the energy, area and edge length statistics obtained running 1600 iterations of the projected Gauss-Seidel and Alexa methods.

4.3 Evaluation of the layout quality: interpretation and comparisons

All our tests confirm that starting with random vertex locations is almost always a bad choice, since iterative methods lead in most cases to a collapse before reaching a valid spherical drawing (spherical spring embedders do not have this problem, but cannot always eliminate edge crossings, see Fig. 4). Our experiments (see Fig. 3 and 4) also confirm two well known facts: Alexa's method is always more robust compared to the projected Gauss-Seidel

[3]Observe that one common metric considered in the geometric processing community is the (angle) distortion: in our case this metric cannot be taken into account since our input is a combinatorial structure (without any geometric embedding).

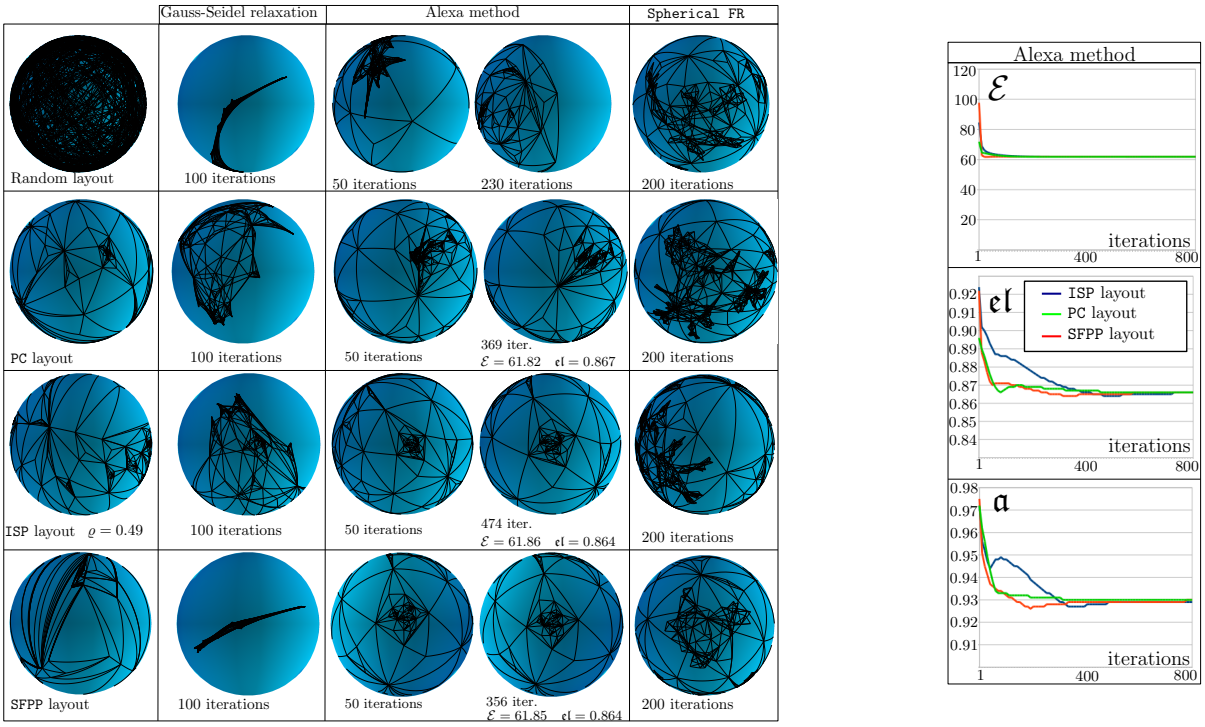


Figure 4: Spherical layouts of a random triangulation with $1K$ faces. While the projected Gauss-Seidel relaxation always collapse, Alexa method is more robust, but also fails when starting from a random initial layout. When using the `ISP`, `PC` or our `SFPP` layouts Alexa method converges toward a crossing-free layout: starting from the `SFPP` layout allows getting the same aesthetic criteria as the `ISP` or the `PC` layouts (with even less iterations). Spring embedders [13] (`Spherical FR`) prevent from reaching a degenerate configuration, but have some difficulties to unfold the layout. The charts on the right show the plot of the energy, edge lengths and areas statistics computed when running 800 iterations of Alexa method (we compute these statistics every 10 iterations).

relaxation, and the `ISP` provides a better starting point compared to the `PC` layout (one can more often converge towards a non-crossing configuration before collapsing, since vertices are distributed in a more balanced way on the sphere).

Layout of mesh-like graphs. When computing the spherical layout of mesh-like structures, the `ISP` layout seems to be a good choice as initial guess (Fig. 3 show the layout of the `dog` mesh). The drawing is rather pleasing, capturing the structure of the input graph and being not too far from the final spherical Tutte layout: we mention that the results obtained in our experiments strongly depend on the quality of the separator cycle. Our `SFPP` layout clearly fails to achieve similar aesthetic criteria: nevertheless, even not being pleasing in the first few iterations, it is possible to reach very often a valid final configuration (crossing-free) without collapsing, and whose quality is very close, in terms of energy and

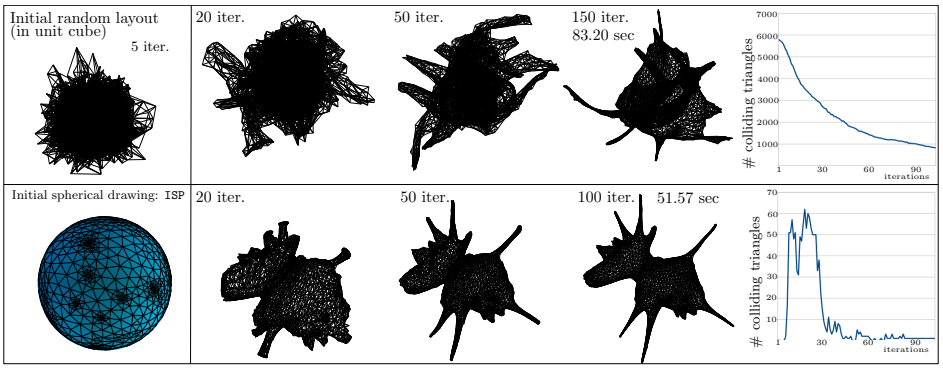


Figure 5: These pictures illustrate the use of spherical drawings as initial placers for force-directed methods: we compute the layouts of the `cow` graph (2904 vertices, 5804 faces) using our 3D implementation of the `FR` spring embedder [16]. In the charts on the right we plot the number of colliding 3D triangles, over 100 iterations of the algorithm.

edge lengths and area statistics, to the ones obtained starting from the `ISP` layout, as confirmed by the charts in Fig. 3. As we observed for most of the tested graphs, when starting from the `SFPP` layout the number of iterations required to reach a spherical drawing with good aesthetics is larger than starting from an `ISP` layout. But the convergence speed can be slightly better in a few cases: Fig. 3 shows a valid spherical layout computed after 1058 iterations of the Gauss-Seidel relaxation (1190 iterations are required when starting from the `ISP` layout).

The charts in Fig. 3 show that our `SFPP` has higher values of the edge lengths and area statistics in the first iterations: this reflects the fact that our layout has a polynomial resolution and thus triangles have a bounded aspect ratio and side lengths. In the case of the `ISP` parameterization there could be a large number of tiny triangles clustered in some small regions (the size of coordinates could be exponentially small as n grows).

Layout of random triangulations. When drawing random triangulations the behavior is rather different: the performances obtained starting from our `SFPP` layout are often better than the ones achieved using the `ISP` layout. As illustrated by the pictures in Fig.4) and 6, Alexa's method is able to reach a non-crossing configuration requiring less iterations when starting from our `SFPP` layout: this is observed in most our experiments, and clearly confirmed by the plots of the energy and statistics $\mathfrak{el}$ and $\mathfrak{a}$ that converge faster to the values of the final layout (see charts in Fig. 4).

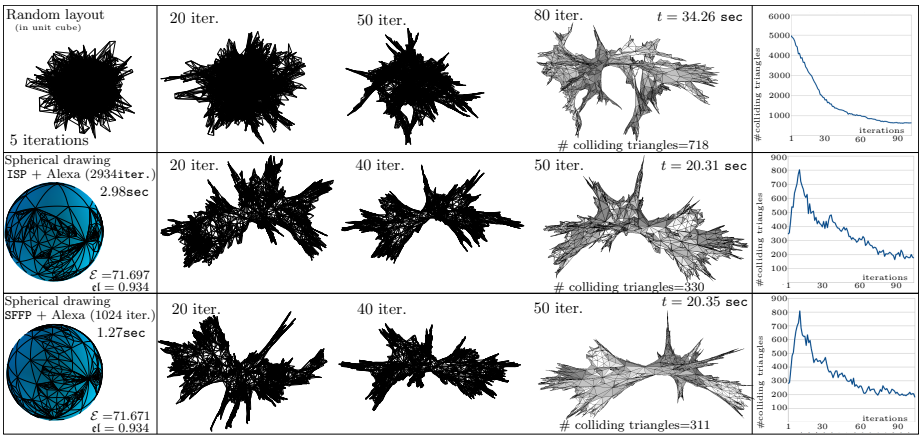


Figure 6: These pictures illustrate the use of spherical drawings as initial placers for the 3D version of the `FR` spring embedder [16], for a random planar triangulation with $5K$ faces.

5 Spherical preprocessing for Euclidean spring embedders

In this section we investigate the use of spherical drawings as initial placers for spring embedders in 3D space. The fact of recovering the original topological shape of the graph, at least in the case of graphs that have a clear underlying geometric structure, is an important and well known ability of spring embedders. This occurs for the case of regular graphs used in Geometry Processing (the pictures in Fig. 5 show a few force-directed layouts of the `cow` graph), and also for many mesh-like complex networks involved in physical and real-world applications (such as the networks made available by the `Sparse Matrix Collection` [9]). In the case of uniformly random embedded graphs (called maps) of a large size n on a fixed surface $\mathcal{S}$, the spring embedding algorithms (applied in the 3D ambient space) yield graph layouts that greatly capture the topological and structural features of the map (the genus of the surface is visible, the "central charge" of the model is reflected by the presence of spikes, etc.), a great variety of such representations can be seen at the very nice simulation gallery of Jérémie Bettinelli (`http://www.normalesup.org/~bettinel/simul.html`). While common software and libraries (e.g. `GraphViz` [12], `Gephi` [4], `GraphStream`) for graph visualization provide implementations of many force-directed models, as far as we know they never try to exploit the strong combinatorial structure of surface-like graphs.

Discussion of experimental results. Our main goal is to show empirically that starting from a nice initial shape that captures the topological structure of the input graph greatly improves the convergence speed and layout quality.

In our first experiments (see Figures 5 and 6) we run our 3D implementation of the spring electrical model `FR` [16], where we make use of exact force computation and we adopt the cooling system proposed in [28] (with repulsive strength $C = 0.1$). We also perform

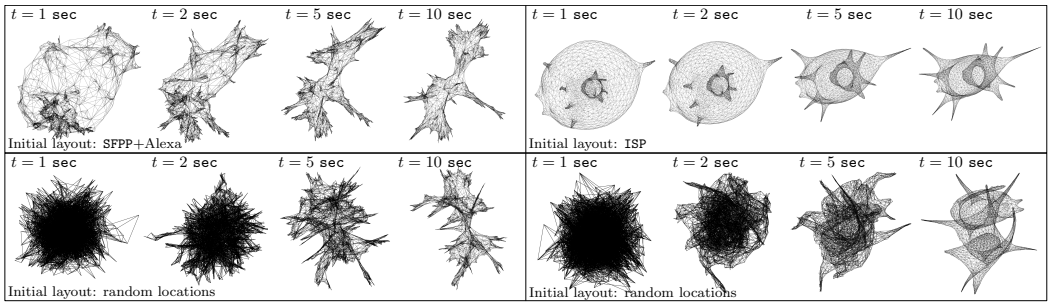


Figure 7: The spherical drawings of the graphs in Fig. 5 and 6 are used as initial placers for the `Yifan Hu` algorithm [19]: we test the implementation provided by `Gephi` (after rescaling the layout by a factor 1000, we set an optimal distance of 10.0 and a parameter $\vartheta = 2.0$).

some tests with the `Gephi` implementation of the `Yifan Hu` layout algorithm [19], which is a more sophisticated spring-embedder with fast approximate calculation of repulsive forces (see the layouts of Fig. 7). In order to quantify the layout quality, we evaluate the number of self-intersections of the resulting 3D shape during the iterative computation process [4]. To be more precise, we plot (over the first 100 iterations) the number of triangle faces that have a collision with a non adjacent triangle in 3D space. The charts of Fig. 5 and 6 clearly confirm the visual intuition suggested by pictures: when starting from a good initial shape the force-directed layouts seem to evolve according to an inflating process, which leads to better and faster untangle the graph layout. This phenomenon is observed in all our tests (on several mesh-like graphs and synthetic data): experimental evidence shows that an initial spherical drawing is a good starting point helping the spring embedder to reach nicer layout aesthetics and also to improve the runtime performances. Finally observe that from the computational point of view the computation of a spherical drawing has a negligible cost: iterative schemes (e.g. Alexa method) require $O(n)$ time per iteration, which must be compared to the complexity cost of force-directed methods, requiring between $O(n^2)$ or $O(n \log n)$ time per iteration (depending on the repulsive force calculation scheme). This is also confirmed in practice, according to the timing costs reported in Fig 5, 6 and 7.

6 Concluding remarks

One main feature of our `SFPP` method is that it always computes a crossing-free layout: unfortunately edge crossings can appear during the beautification process, when running iterative algorithms (projected Gauss-Seidel iteration, Alexa method or more sophisticated schemes). It could be interested to adapt to the spherical case existing methods [27] (which are designed for the Euclidean case) whose goal is to dissuade edge-crossings: their application could lead to produce a sequence of layouts that converge to the final spherical drawing

[4] We compute the intersections between all pairs of non adjacent triangles running a brute-force algorithm: the runtimes reported in Fig. 5 and 6 do not count the cost of computing the triangle collisions.

while always preserving the map. The promising results of Section 5 would suggest that starting from an initial nice layout could lead to faster algorithms and better results for the case of mesh-like structures. It could be interesting to investigate whether this phenomenon arises for other classes of graphs, such as quadrangulated or 3-connected planar graphs, or non planar (e.g. toroidal) graphs, for which fast drawing methods also exist [6, 17].

References

[1] N. Aigerman, S. Z. Kovalsky, and Y. Lipman. Spherical orbifold tutte embeddings. *ACM Trans. Graph.*, 36(4):90:1–90:13, 2017.

[2] N. Aigerman and Y. Lipman. Orbifold tutte embeddings. *ACM Trans. Graph.*, 34(6):190:1–190:12, 2015.

[3] M. Alexa. Merging polyhedral shapes with scattered features. *The Visual Computer*, 16(1):26–37, 2000.

[4] M. Bastian, S. Heymann, and M. Jacomy. Gephi: An open source software for exploring and manipulating networks. In *Proc. of the Third Int. Conf. on Weblogs and Social Media, ICWSM 2009, 2009*, 2009.

[5] E. Brehm. 3-orientations and Schnyder 3-tree-decompositions. *Master's Thesis, FB Mathematik und Informatik, Freie Universität Berlin*, 2000.

[6] L. Castelli-Aleardi, O. Devillers, and É. Fusy. Canonical ordering for triangulations on the cylinder, with applications to periodic straight-line drawings. In *Graph Drawing - 20th International Symposium*, pages 376–387, 2012.

[7] L. Castelli-Aleardi, É. Fusy, and A. Kostrygin. Periodic planar straight-frame drawings with polynomial resolution. In *LATIN 2014: Theoretical Informatics - 11th Latin American Symposium*, pages 168–179, 2014.

[8] E. W. Chambers, D. Eppstein, M. T. Goodrich, and M. Löffler. Drawing graphs in the plane with a prescribed outer face and polynomial area. *J. Graph Algorithms Appl.*, 16(2):243–259, 2012.

[9] T. A. Davis and Y. Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, 2011.

[10] H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.

[11] C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Planar drawings of higher-genus graphs. *J. Graph Algorithms Appl.*, 15(1):7–32, 2011.

[12] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull. Graphviz - open source graph drawing tools. In *Proc. of Graph Drawing*, pages 483–484, 2001.

[13] J. J. Fowler and S. G. Kobourov. Planar preprocessing for spring embedders. In *Graph Drawing - 20th International Symposium*, pages 388–399, 2012.

[14] E. Fox-Epstein, S. Mozes, P. M. Phothilimthana, and C. Sommer. Short and simple cycle separators in planar graphs. *ACM Journal of Experimental Algorithmics*, 21(1):2.2:1–2.2:24, 2016.

[15] I. Friedel, P. Schröder, and M. Desbrun. Unconstrained spherical parameterization. *J. Graphics Tools*, 12(1):17–26, 2007.

[16] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Softw., Pract. Exper.*, 21(11):1129–1164, 1991.

[17] D. Gonçalves and B. Lévêque. Toroidal maps: Schnyder woods, orthogonal surfaces and straight-line representations. *Discrete & Computational Geometry*, 51(1):67–131, 2014.

[18] C. Gotsman, X. Gu, and A. Sheffer. Fundamentals of spherical parameterization for 3d meshes. *ACM Trans. Graph.*, 22(3):358–363, 2003.

[19] Y. Hu. Efficient, high-quality force-directed graph drawing. *The Mathematica Journal*, 10(1), 2006.

[20] S. G. Kobourov and K. Wampler. Non-euclidean spring embedders. *IEEE Trans. Vis. Comput. Graph.*, 11(6):757–767, 2005.

[21] H. Nagamochi, T. Suzuki, and T. Ishii. A simple recognition of maximal planar graphs. *Inf. Process. Lett.*, 89(5):223–226, 2004.

[22] D. Poulalhon and G. Schaeffer. Optimal coding and sampling of triangulations. *Algorithmica*, 46(3-4):505–527, 2006.

[23] S. Saba, I. Yavneh, C. Gotsman, and A. Sheffer. Practical spherical embedding of manifold triangle meshes. In *(SMI2005)*, pages 258–267, 2005.

[24] W. Schnyder. Embedding planar graphs on the grid. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, volume 90, pages 138–148, 1990.

[25] A. Shapiro and A. Tal. Polyhedron realization for shape transformation. *The Visual Computer*, 14(8/9):429–444, 1998.

[26] A. Sheffer, C. Gotsman, and N. Dyn. Robust spherical parameterization of triangular meshes. *Computing*, 72(1-2):185–193, 2004.

[27] P. Simonetto, D. W. Archambault, D. Auber, and R. Bourqui. Impred: An improved force-directed algorithm that prevents nodes from crossing edges. *Comput. Graph. Forum*, 30(3):1071–1080, 2011.

[28] C. Walshaw. A multilevel algorithm for force-directed graph-drawing. *J. Graph Algorithms Appl.*, 7(3):253–285, 2003.

[29] R. Zayer, C. Rössl, and H. Seidel. Curvilinear spherical parameterization. In *Int. Conf. on Shape Modeling and Applications (SMI 2006)*, page 11, 2006.

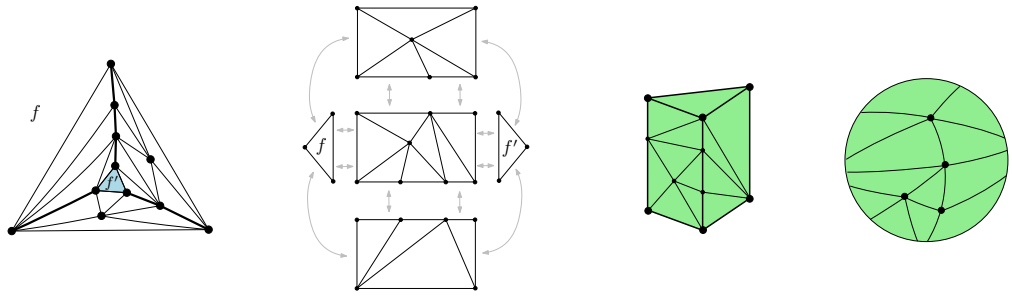


Figure 8: These pictures illustrate the computation of a spherical drawing using our SFPP algorithm.

A Appendix

A.1 Proof of Theorem 1

For the sake of completeness we provide a detailed description of all the steps of the linear-time algorithm computing a SFPP layout of a triangulation G of the sphere, as sketched in Section 3: we combine and adapt many ingredients developed in [11, 6, 7].

Cutting G along three disjoint paths. Let G be a triangulation on the sphere. We assume that there exist two non adjacent faces f and f' (with no common incident vertices). If not, one can force the existence of two such faces by adding a new triangle t within a face (and adding edges so as to triangulate the area between t and the face-contour).

The first step is to compute 3 vertex-disjoint chord-free paths that start at each of the 3 vertices of f_S and end at each of the 3 vertices of f_N.

Schnyder woods [24, 5] provide a nice way to achieve this. Taking f as the outer face, where v_0, v_1, v_2 are the outer vertices in clockwise (CW) order and inserting a vertex v of degree 3 inside f', we compute a Schnyder wood of the obtained triangulation, and let P_0, P_1, P_2 be the directed paths in respective colors $0, 1, 2$ starting from v: by well-known properties of Schnyder woods, these paths are chord-free and are disjoint except at v, and they end at the 3 vertices v_0, v_1, v_2 of f. Deleting v and its 3 incident edges, (and thus deleting the starting edge in each of P_0, P_1, P_2) we obtain a triple of disjoint chord-free paths from f' to f. Let u_0, u_1, u_2 be the vertices on f' incident to P_0, P_1, P_2.

As in [7] we call *4ST triangulation* a graph embedded in the plane with a polygonal outer contour and with triangular inner faces, with 4 distinguished vertices w_0, w_1, w_2, w_3 (in cw order) incident to the outer face, and such that each of the 4 outer paths delimited by the marked outer vertices is chord-free. The external paths between w_0 and w_1 and between w_2 and w_3 are called *vertical*, and the two other ones are called *horizontal*. A 4ST is called *narrow* if the two vertical paths have only one edge. For $i \in \{0, 1, 2\}$ let G_i be the narrow 4ST whose outer contour (indices are taken modulo 3) is made of the path P_i, the edge $\{u_i, u_{i+1}\}$, the path P_{i+1}, and the edge $\{v_i, v_{i+1}\}$.

Note that G can be seen as a *prism*, with f and f' as the two triangular faces and with G_0, G_1, G_2 occupying the 3 lateral quadrangular faces of the prism.

Computing compatible drawings of the 3 components G_0, G_1, G_2. A *straight-frame drawing* of a 4ST H is a straight-line drawing of H where the outer face contour is an axis-aligned rectangle, with the 4 sides of the rectangle corresponding to the 4 paths along the contour. The *interspace-vector* of each of the 4 paths is the vector giving the lengths (in the drawing) of the successive edges along the path, where the path is traversed from left to right for the two horizontal ones and is traversed from bottom to top for the two vertical ones. In order to obtain a drawing of G on the prism (which then yields a geodesic crossing-free drawing on the sphere, using a central projection), we would like to obtain *compatible* straight-frame drawings of G_0, G_1, G_2, i.e., such that for $i \in \{0, 1, 2\}$ the interspace-vectors of P_i in the drawing of G_i and in the drawing of G_{i-1} are the same.

Using an adaptation —given in [7]— of the algorithm by Duncan et al. [11], one gets the following result, where a vector of positive integers is called *even* if all its components are even, and the *total* of a vector is the sum of its components:

Lemma 1 (from [7]). *Let H be a narrow 4ST with m vertices. Then one can compute in linear time a straight-frame drawing of H such that the interspace-vectors $U = (u_1, \ldots, u_p)$ and $V = (v_1, \ldots, v_q)$ of the two horizontal external paths are even, and the grid-size of the drawing is bounded by $4m \times (4m + 1)$.*

Moreover, for any pair $U' = (u'_1, \ldots, u'_p)$ and $V = (v'_1, \ldots, v'_q)$ of even vectors such that $U' \geq U$, $V' \geq V$, and U' and V' have the same total s, one can recompute in linear time a straight-frame drawing of H such that the interspace vectors of the two horizontal external paths are respectively U' and V', and the grid-size is $4s \times (4s + 1)$.

For $i \in \{0, 1, 2\}$ let k_i be the number of vertices of G_i. By the first part of Lemma 1, G_i admits a straight-frame drawing —where P_i and P_{i+1} are the two horizontal external paths— such that the interspace-vector U_i along P_i and the interspace-vector V_{i+1} along P_{i+1} are even, and the grid size is bounded by $4k_i \times (4k_i + 1)$, with k_i the number of vertices in G_i.

We let W_i be the vector $\max(U_i, V_i)$, and let s_i be the total of W_i, and set $s := \max(s_0, s_1, s_2)$. It is easy to check that $s \leq 8n$. We then let W'_i be obtained from W_i by adding $s - s_i$ to the last component. Then we set U'_i and V'_i to W'_i. Note that we have $U'_i \geq U_i$ and $V'_i \geq V_i$ for $i \in \{0, 1, 2\}$, and moreover all the vectors $U'_0, V'_0, U'_1, V'_1, U'_2, V'_2$ now have the same total, which is s. We can thus use the second part of Lemma 1 and obtained straight-frame drawings of G_i (for $i \in \{0, 1, 2\}$) on the grid $4s \times (4s + 1)$ where the interspace-vector for the bottom (resp. upper) horizontal external path is U'_i (resp. is V'_{i+1}). Since $U'_i = V'_i$ for $i \in \{0, 1, 2\}$, the drawings of G_0, G_1, G_2 are compatible and can thus be assembled to get a drawing of G on the prism (see Figure 8 for an example), which then yields a drawing on the unit sphere using a central projection (with the origin at the center of mass of the prism seen as a solid polyhedron). Note that the prism has its 3 lateral faces of area $O(n \times n)$, hence is at distance $O(n)$ from the origin. Since every edge of G drawn on the prism clearly has length at least 1 (as in any straight-line drawing with integer coordinates) we conclude that after the central projection every edge of G has length $\Omega(1/n)$, as claimed in Theorem 1.

Remark. To improve the distribution of the points on the sphere, one aims at 3 paths P_0, P_1, P_2 such that the graphs G_0, G_1, G_2 are of similar sizes. A simple heuristic (using the approach based on Schnyder woods mentioned above), is to do the computation *for every face f_S* non-adjacent to f_N, and keep the one that maximizes the objective parameter $\sum_{0 \leq i < j \leq 2} ||G_i| - |G_j||$.