Convexifying Polygons with Simple Projections

Jorge Alberto Calvo* Danny Krizanc† Pat Morin‡
Michael Soss§ Godfried Toussaint§

Newly revised on December 19, 2000

Abstract

It is known that not all polygons in 3D can be convexified when crossing edges are not permitted during any motion. In this paper we prove that if a 3D polygon admits a non-crossing orthogonal projection onto some plane, then the 3D polygon can be convexified. If an algorithm to convexify the planar projection exists and runs in time P, then our algorithm to convexify the 3D polygon runs in $O(n + P)$ time.

1 Introduction

A closed chain of n line segments with lengths $l_1, ..., l_n$ embedded in R^3 forms a space polygon. We are concerned with simple space polygons which are trivially knotted (also called unknots). The problem we address is that of convexifying a polygon in 3D, that is, reconfiguring it to a planar convex polygon while maintaining the edges rigid with fixed lengths and not allowing the edges to cross each other during any motion. Recently Cantarella and Johnston [5] and independently, Biedl, et al. [2] studied the embedding classes of such objects and discovered that there exist stuck (or locked) unknotted simple polygons. In the setting of linkage convexification, the existence of these locked polygons is key, since it implies that not every unknotted linkage in 3D can be convexified.

On the other hand, Biedl, et al. [2] showed that a planar polygon in 3D can always be convexified in $O(n^2)$ time with $O(n^2)$ simple motions in which no more than four joints are rotated at once. Recently Jeff Erickson pointed out that with a small modification the algorithm in [2] runs in linear time with linearly many simple motions. A *simple motion* is defined as a movement of the chain where only a constant number of angles at vertices and dihedral angles change, and all such changes are monotonic. Therefore, during a simple motion of k joints, there exists a partition of the polygon into k pieces (determined by the k joints), each of which remains rigid with respect to itself during the motion. If the number of angles changing is larger than a constant, the motion is said to be a *complex motion*.

In January 2000, Connelly, Demaine, and Rote presented a proof that any simple 2D polygon can be convexified in the plane [6]. Their algorithm for convexification involves integrating a vector field, and complexity is not measured in terms of simple or complex motions. Streinu [11] has recently published an algorithm which convexifies a planar

*Dept. of Mathematics, North Dakota State University, Fargo, ND. jorge_calvo@ndsu.nodak.edu
†Dept. of Mathematics, Wesleyan University, Middletown, CT. dkrizanc@wesleyan.edu
‡School of Computer Science, Carleton University, Ottawa, ON. morin@scs.carleton.ca
§School of Computer Science, McGill University, Montreal, QC. {soss, godfried}@cs.mcgill.ca

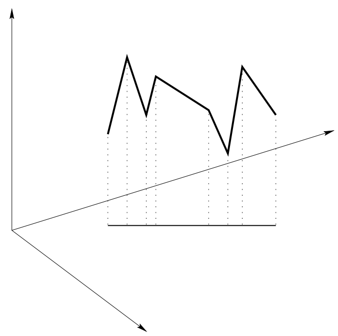

Figure 1: Coplanar edges whose projection is a single edge.

polygon in $O(n^2)$ motions, although the complexity of determining each individual motion is not measured.

Such results are relevant to understanding how small-scale rigidity influences the shape of DNA and other complex circular (ring) molecules [8, 9].

In this paper, we discuss a class of non-planar polygons in three dimensions that can be convexified, polygons with simple orthogonal projections.

2 Polygons with Simple Projections

Suppose a 3D polygon has a simple orthogonal projection to some plane, which we assume to be the xy-plane. By keeping the height (z-coordinate) of each vertex fixed, we can convexify the projection on the xy-plane while making the polygon in 3D track this motion. Thus we can reconfigure any polygon with a simple projection into a polygon with a convex projection.

We now give an algorithm to convexify a polygon with a convex projection, thereby implying that a polygon with a simple projection can be convexified.

3 Polygonal Chains and Springs

Our algorithm works by repeatedly reconfiguring the polygon while maintaining convexity in the projection. The overall motion is a simple combination of basic primitives, and as such, the motions are easy to compute and visualize. We now explain our first primitive motion, which straightens a monotonic polygonal chain which lies on a single vertical plane.

Consider a polygonal chain which is entirely contained in a vertical plane, such as the one illustrated in Figure 1. If we wish to straighten this series of edges, we need only to pull the rightmost vertex to the right, moving only the last two vertices (and changing the angle at the third). When the second vertex straightens, we freeze that joint permanently as straight, thereby effectively deleting a vertex, resulting in a polygonal chain of one fewer vertex. We then proceed by induction until the chain is a single line segment.

As this motion resembles the straightening of a spring (see Figure 2), we refer to this straightening as a *spring-unfolding* of a polygonal chain. The straightening motions are illustrated in Figure 3. The rightmost vertex is in constant motion directly to the right. At each step the two right edges move until straightened, at which point they are frozen to henceforth act as a single edge. The procedure is then repeated with the two rightmost edges (including the newly fused edge) until either the entire chain

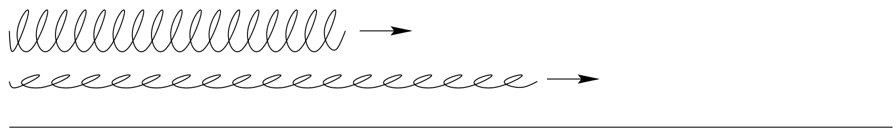

Figure 2: Pulling a spring until it straightens.

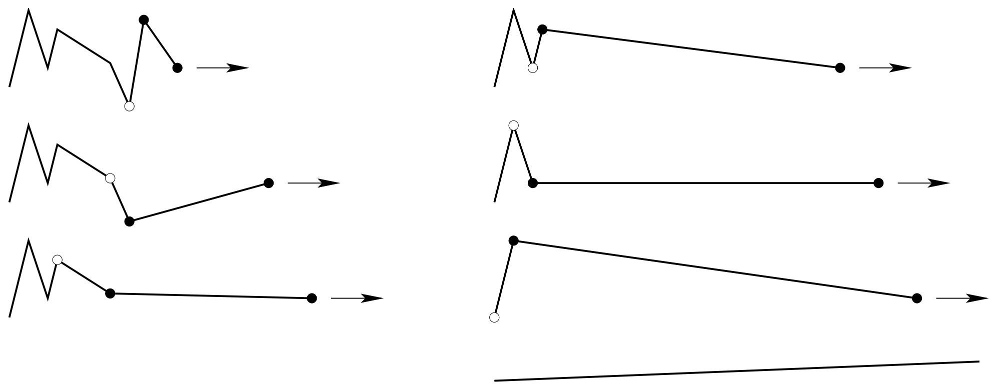

Figure 3: A spring-unfolding of a planar chain. The black vertices are in motion while the angle at the white vertex is changing. When the two rightmost edges are straightened, the new antepenultimate vertex becomes white.

is straightened or the right endpoint comes to a halt. It is vital to realize that the projection of the polygonal chain is, at all times, in the same vertical plane. Thus its projection is always a line segment, growing throughout the motion.

In keeping with this terminology, we will define a *spring* as a consecutive sequence of at least two (non-colinear) edges whose simple orthogonal projection is a single line segment. Thus the polygonal chain in Figure 1 is a spring.

It is a simple matter to compute when each vertex straightens given a RAM computer which can compute square roots (and therefore distances). Because only two edges move at once, each motion, which straightens a vertex, can be computed in constant time. Thus the entire spring can be straightened in time linear in the number of its vertices.

4 The Spring Algorithm

Our algorithm requires that the convex projection be a triangle. Changing a polygon from one convex position to another is not difficult and can be accomplished in linear time [1, 10]. As mentioned above, by keeping the height (z-coordinate) of each vertex fixed, we can reconfigure the projection on the xy-plane while making the polygon in three-space track this motion.

Once the projection is a triangle, note that all the edges of the polygon lie along three vertical planes. Therefore the polygon is composed of three or fewer springs. We adopt the terminology *k-spring* as a polygon which contains exactly k springs. For example, a polygon with a triangular projection $\triangle abc$ that has only one edge projecting onto ab, but several edges projecting onto each ac and bc is a 2-spring.

The Spring Algorithm works just as one would intuitively straighten a triangle made of springs. From a high-level standpoint, we repeatedly pull on vertices, straightening a spring each time. Since the straightening (spring-unfolding) of a spring implies a straightening of vertices, we are guaranteed to finish in linearly many such motions. For low-level details, it will become apparent that we can easily compute which vertices of a spring move and at which time, as their motions are uniquely linked to the high-level description of the springs.

The following is our algorithm in a high-level form.

The Spring Algorithm

1. While the polygon has more than four edges,
   (a) If the polygon is a 3-spring, reconfigure it into a 2-spring (as per Figure 4).
      - Fix the position of the edge $\overline{ab}$. Keeping the height of c fixed, pull c orthogonally away from $\overline{ab}$ until one of the springs (ac or bc) straightens. At any moment, each of the three springs remain in a single vertical plane, and thus the projection is always a triangle during this step.

   (b) Else if the polygon is a 2-spring, reconfigure it into a 1-spring (as per Figure 5).
      - Let $\overline{ab}$ be the non-spring edge; this edge remains fixed. Keeping the height of c fixed, pull c orthogonally away from $\overline{ab}$ until one of the springs (ac or bc) straightens. As in the previous step, each of the three springs remains in a single vertical plane, so that the projection is always a triangle throughout this step.

   (c) Else if the polygon is a 1-spring, reconfigure it into a 2- or 3-spring or a quadrilateral (as per Figure 6).
      - Let $\overline{ab}, \overline{bc}$ be the non-spring edges, labelled such that the angle at c is acute. Fix in space the median[1] vertex d of the springed edge ac. Keeping c at a fixed height, rotate $\overline{bc}$ to open the internal angle at b while unfolding the spring cd until one of the two following possibilities occurs.
        i. If the edges $\overline{ab}$ and $\overline{bc}$ achieve colinear projections, we attain a 3-spring, unless $\overline{ab}$ and $\overline{bc}$ happen to be colinear in three-space, in which case we attain a 2-spring.
        ii. If the spring cd straightens, rotate $\overline{cd}$ to open the internal angle at c while keeping d at a fixed height and unfolding the spring ad until one of the three following conditions is satisfied. The edges $\overline{bc}$ and $\overline{cd}$ could achieve colinear projections, resulting in a 2-spring (or 1-spring should b, c, and d be colinear). The edge $\overline{ab}$ and the spring ad could achieve colinear projections, resulting in a 1-spring. Lastly, the spring ad could straighten, resulting in a 3D quadrilateral.

2. If the polygon has four edges, label it $abcd$. Rotate c about the diagonal $\overline{bd}$ until the quadrilateral is planar. Convexify the planar quadrilateral.

3. Return the convexified polygon.

We first prove the correctness of the algorithm. Since each step involves the straightening of vertices until a triangle or quadrilateral is obtained, certainly the algorithm will finish. We now prove the hypotheses on which the algorithm is based, that the polygon never self-intersects and that the projection is a triangle or convex

[1] Suppose the spring ac contains k vertices. By the phrase, "median vertex d of the edge ac," we mean a vertex d such that each ad and dc contain roughly $k/2$ vertices. To be exact, the two chains ad and dc would contain $\lceil k/2 \rceil$ and $\lceil (k+1)/2 \rceil$ vertices, respectively.

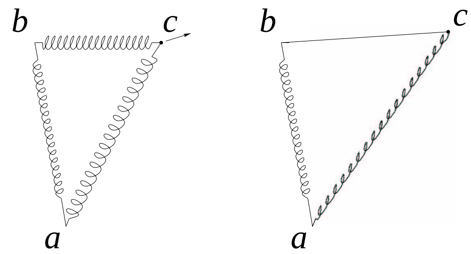

Figure 4: Step 1a. Reconfiguring a 3-spring into a 2-spring.

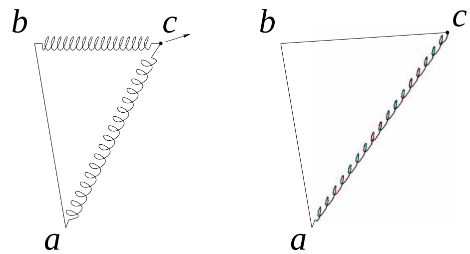

Figure 5: Step 1b. Reconfiguring a 2-spring into a 1-spring.

quadrilateral at all times. No spring can intersect itself, so intersections, if possible, could only be between two springs, resulting in a non-convex projection. Note that during the execution of Steps 1a and 1b the projection is a triangle, so there is no possibility for a non-convex projection. During Step 1c, the vertex c is rotated about b, causing the projection to be a quadrilateral. Consider the motion of c during this step. Because $\angle acb$ is acute, the quadrilateral will stay convex. (If c were obtuse, then the angle at d would initially become reflex.) From this point on, if any vertex of the projection straightens, it becomes a triangle. Therefore the projection is convex at all times.

Each high-level motion (Step 1a, 1b, or 1c) is easily computable. We simply add up the lengths of all edges in the springs to discover their length when straightened to see which spring straightens first. For the low-level description of the motions of individual vertices for each spring, we can determine which vertex opens and when during its spring-unfolding since we know the motion of the spring and therefore how its length increases over time. As noted in Section 3, a spring-unfolding of a chain can be computed in time linear in the number of vertices in the spring, and the spring-unfolding is determined solely by the motion of the endpoints of the spring. Therefore each high-level motion can be computed in time linear in the number of vertices of the polygon.

During every two or three iterations of the while loop (Step 1), the polygon will be in a 1-spring configuration. If the polygon has n (unstraightened) vertices at this point in the algorithm, $n-1$ of these will be contained in the spring. During the execution of Step 1c, the spring is separated into two nearly equal parts, each of which contains no less than $\frac{n-4}{2}$ vertices. After two or three more iterations of the algorithm, the polygon is again a 1-spring. Therefore at least one of these two springs of $\frac{n-4}{2}$ vertices each was straightened. Therefore after at most every three iterations of the algorithm, the polygon's complexity, and thus the time complexity of each iteration, is approximately halved. By this geometric progression, the entire algorithm finishes in linear time.

We further note that at no point in time are more than seven vertices opening or

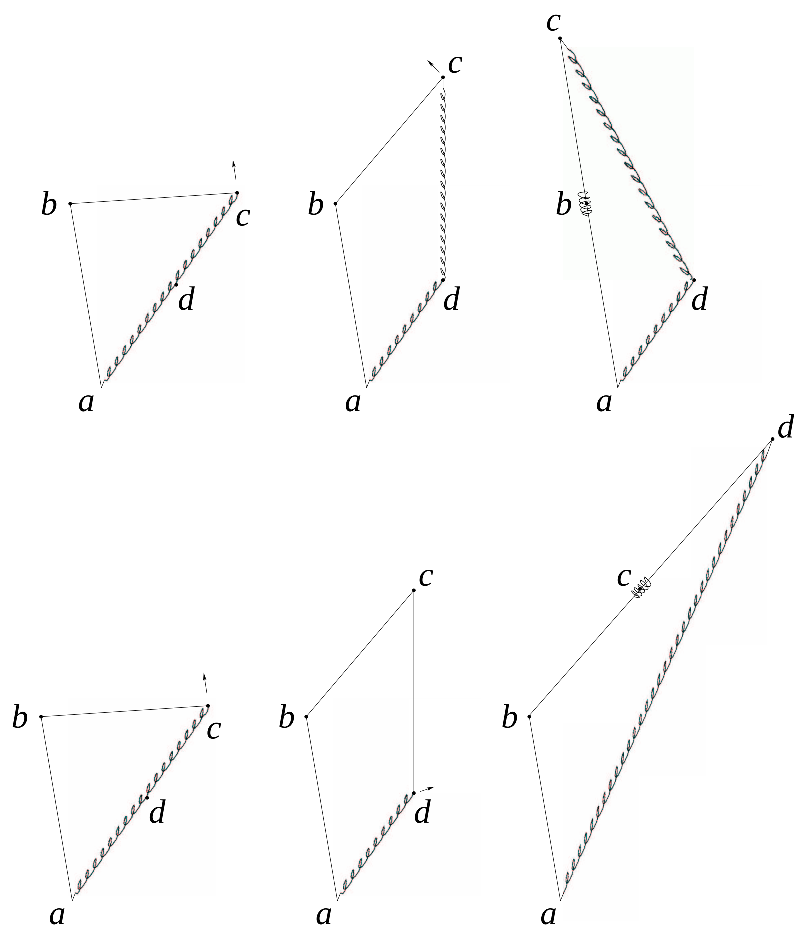

Figure 6: Step 1c. Reconfiguring a 1-spring. The top diagram illustrates Step 1(c)i; the bottom illustrates Step 1(c)ii.

closing. During Step 1 only two adjacent edges of the projection are in motion at any one time. The three endpoints adjacent to these two edges are in motion, and for each of two edges in the projection corresponding to springs, two additional vertices on each are involved in the spring-unfoldings. Therefore the total number of vertices changing at any one time is seven. Recalling the definition of a simple motion, a motion in which the angles at most a constant number of vertices change, we arrive at the following theorem.

Theorem 1 *A polygon which admits a convex orthogonal projection can be convexified in $O(n)$ time with $O(n)$ simple motions.*

We can now apply Theorem 1 to a variety of previous results on convexifying polygons in the plane, by first convexifying the projection and then applying the Spring Algorithm. Building on Streinu's algorithm to convexify a planar polygon, we obtain Theorem 2.

Theorem 2 *A polygon which admits a simple orthogonal projection can be convexified with $O(n^2)$ motions.*

There are currently two classes of planar polygons which have special convexifying algorithms. Algorithms have been designed to convexify monotone polygons [3] and star-shaped polygons [7]. Applying Theorem 1 to these results yields the following two theorems.

Theorem 3 *A polygon which admits a monotonic orthogonal projection can be convexified in $O(n^2)$ time with $O(n^2)$ simple motions.*

Theorem 4 *A polygon which admits a star-shaped orthogonal projection can be convexified in $O(n^2)$ time with $O(n)$ complex motions.*

In light of the above, it is important to be able to determine if a polygon has an orthogonal projection or one of the mentioned special cases. Bose, Gómez, Ramos, and Toussaint have an algorithm to determine whether or not a polygon has a simple orthogonal projection in $O(n^4)$ time, and another which determines whether or not a polygon has a monotonic projection in $O(n^2)$ time [4]. To our knowledge, determining whether or not a polygon has a star-shaped projection is an open problem.

5 Concluding Remarks

Building on the recent results by Connelly, Demaine, and Rote [6] and by Streinu [11], we have proven that any polygon which admits a simple orthogonal projection can be convexified. One can of course easily construct convexifiable polygons which do not possess such a projection; thus the condition is sufficient but not necessary. It is obvious that a polygon must be unknotted if it is to be convexifiable. What other conditions are sufficient, and what others are necessary? For example, can a polygon with a simple perspective projection be convexified?

References

[1] Oswin Aichholzer, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, Mark Overmars, Michael A. Soss, and Godfried T. Toussaint. Reconfiguring convex polygons. In *Proceedings of the Twelfth Canadian Conference on Computational Geometry*, Fredericton, Canada, August 2000.

[2] T. Biedl, E. Demaine, M. Demaine, S. Lazard, A. Lubiw, J. O'Rourke, M. Overmars, S. Robbins, I. Streinu, G. T. Toussaint, and S. Whitesides. Locked and unlocked polygonal chains in 3D. In *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 866–867, 1999.

[3] Therese C. Biedl, Erik D. Demaine, Sylvain Lazard, Steven M. Robbins, and Michael A. Soss. Convexifying monotone polygons. In *Proceedings of the Tenth International Symposium on Algorithms and Computation*, pages 415–424, Chennai, India, December 1999.

[4] Prosenjit Bose, Francisco Gomez, Pedro Ramos, and Godfried T. Toussaint. Drawing nice projections of objects in space. In *Graph Drawing (Proc. GD '95)*, volume 1027 of *Lecture Notes Comput. Sci.*, pages 52–63. Springer-Verlag, 1996.

[5] Jason Cantarella and Heather Johnston. Nontrivial embeddings of polygonal intervals and unknots in 3-space. *Journal of Knot Theory and its Ramifications*, 7(8):1027–1039, 1998.

[6] Robert Connelly, Erik D. Demaine, and Günter Rote. Every polygon can be untangled. In *Proceedings of the Sixteenth European Workshop on Computational Geometry*, Eliat, Israel, March 2000.

[7] H. Everett, S. Lazard, S. Robbins, H. Schröder, and S. Whitesides. Convexifying star-shaped polygons. In *Proc. 10th Canadian Conference on Computational Geometry*, pages 2–3, 1998.

[8] Maxim D. Frank-Kamenetskii. *Unravelling DNA*. Addison-Wesley, 1997.

[9] C. Holden. Random samples: Locked but not knotted. *Science*, 283:931, February 12, 1999.

[10] W. J. Lenhart and S. H. Whitesides. Reconfiguring closed polygonal chains in Euclidean d-space. *Discrete Comput. Geom.*, 13:123–140, 1995.

[11] Ileana Streinu. A combinatorial approach to planar non-colliding robot arm motion planning. In *Proceedings of the Forty-First Annual Symposium on Foundations of Computer Science*, Redondo Beach, California, November 2000.