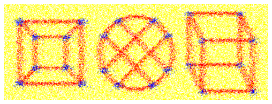

Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 15, no. 2, pp. 205–287 (2011)

Morphing Planar Graph Drawings with Bent Edges

Anna Lubiw[1] *Mark Petrick*[1]

[1]David R. Cheriton School of Computer Science,
University of Waterloo, Canada

Abstract

We give an algorithm to morph between two planar drawings of a graph, preserving planarity, but allowing edges to bend during the course of the morph. The morph is polynomial size and *discrete*: it uses a polynomial number of elementary steps, where each elementary step is a linear morph that moves each vertex along a straight line at uniform speed. Although there are previously-known planarity-preserving morphs that do not require edge bends, it is an open problem to find polynomial-size discrete morphs. We achieve polynomial size at the expense of edge bends.

Submitted:	Reviewed:	Revised:	Accepted:
May 2010	October 2010	October 2010	December 2010
	Final:	Published:	
	February 2011	February 2011	
Article type:		Communicated by:	
Regular paper		S. Kobourov	

Research supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC). A preliminary version of this work appeared in The International Conference on Topological and Geometric Graph Theory [21].

E-mail addresses: alubiw@uwaterloo.ca (Anna Lubiw) mdtpetrick@uwaterloo.ca (Mark Petrick)

1 Introduction

A *morph* from one drawing of a planar graph to another is a continuous transformation from the first drawing to the second. This paper is about morphs that preserve planarity—these exist only if the two drawings have the same faces and the same outer face. See Figure 1 for some examples. A special kind of morph is the *linear morph* of a planar straight-line drawing where every vertex moves at uniform speed along a straight line. The linear morph does not preserve planarity in general. A morph is *discrete* if it is composed of a sequence of linear morphs, i.e. it is piece-wise linear.

Developments in the theory of planar morphing run parallel to the developments in planar graph drawing, though they lag behind. In particular, the milestones in the history of planar graph drawing are: the existence results for straight-line planar drawings due to Wagner, Fáry [11] and Stein; Tutte's algorithm to construct such drawings [28]; and, in 1990, the polynomial time algorithms of de Fraysseix, Pach, Pollack [7] and independently Schnyder [23] to construct a straight-line drawing of an n-vertex planar graph on an $O(n) \times O(n)$ grid.

Mirroring these, the first result on morphing planar graph drawings was an existence result: between any two planar straight-line drawings there exists a morph in which every intermediate drawing is straight-line planar. This was proved for triangulations, by Cairns [6] in 1944, and extended to planar graphs by Thomassen [27] in 1983. Both proofs are constructive—they work by repeatedly contracting one vertex to another. The morphs are discrete. Unfortunately, they use an exponential number of linear morphs. In addition, they are horrible for visualization purposes, since the graph contracts to a triangle and then re-emerges.

The next development was an algorithm to morph between any two planar straight-line drawings, preserving planarity and straight-line edges, due to Floater and Gotsman [12] in 1999 for triangulations, and extended to planar graphs by Gotsman and Surazhsky [25, 26] beginning in 2001. The morphs are not given by means of explicit vertex trajectories, but rather by means of "snapshots" of the graph at any intermediate time t. The authors give a polynomial time algorithm to compute the snapshot for any specified t. By choosing sufficiently many values of t, the morph gives good visual results, but there are no guarantees. More specifically one would hope that between two consecutive values of t, a vertex does not change position too much, and follows a relatively straight path, but there is no guarantee of how fine-grained the set of t's must be to ensure this. In particular, the morph is not discrete. Furthermore, the morph suffers from the same drawbacks as Tutte's original planar graph drawing algorithm in that there is no nice bound on the size of the grid needed for the drawings.

The history of morphing planar graph drawings has not progressed to the analogue of the small grid results of de Fraysseix et al.: *It is an open problem to find a polynomial size discrete morph between two given drawings of a planar graph.*

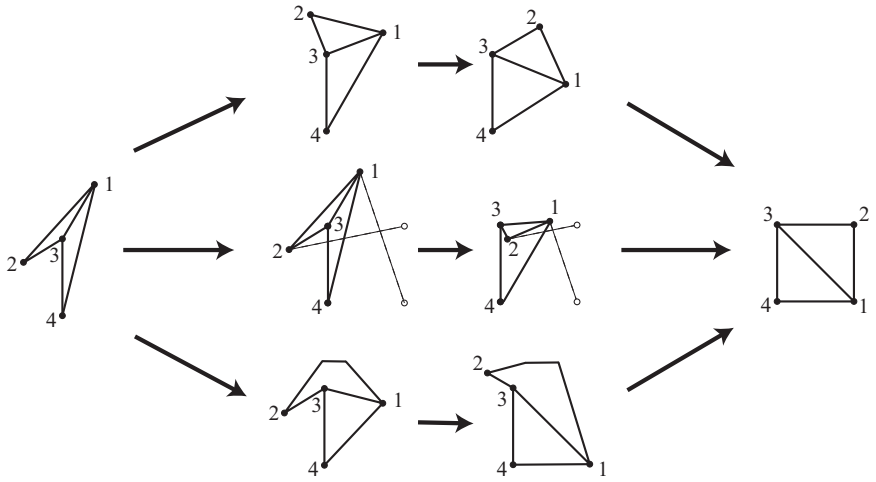


Figure 1: Example morphs: (*top*) a morph preserving planarity and straight-line edges; (*middle*) a linear morph (vertex trajectories shown as dashed lines) that does not preserve planarity; (*bottom*) a planarity preserving morph allowing edges to bend

In this paper we solve this problem provided that edges are allowed to bend during the course of the morph. In other words, we subdivide edges by adding new vertices that we call *bends*. We give a polynomial-time algorithm to find a planarity preserving morph between two drawings of a planar graph on n vertices, where the morph is composed of a sequence of $O(n^6)$ linear morphs. During the course of the morph, an edge becomes a polygonal path with $O(n^5)$ bends. Between successive linear morphs the drawings lie on an $O(n^3) \times O(n^3)$ grid.

1.1 Related Work

Morphing is well-studied in graphics (see [16]), though the focus there is often on *image space* morphs that transform pixels, as opposed to *object space* morphs that operate on geometric objects. For morphing of geometric objects, a main issue is establishing the right correspondence between the source object and the target object. In our work we assume that such a correspondence is given. A more mathematical treatment of morphing can be found in the survey by Alt and Guibas [2].

In graphics, morphs are primarily sought for animation and visualization purposes. However, structure-preserving transformations between geometric objects are useful in many other contexts. In motion planning [20], the goal is often physical transformation between two configurations, and the prohibition against self-intersection is inherent. Linkage reconfiguration problems—and unfolding problems more generally [8]—require in addition that metric properties such as

edge lengths be preserved.

In medical imaging, the problem of finding 3D interpolations between successive 2D slices is a morphing problem in which time becomes the third dimension [3]. Morphing is also relevant to similarity measures, with the connection made explicit in work on morphing of polylines [9, 5, 1]. Note that in this work, the correspondence between source and target is not pre-specified.

We now return to the more specific topic of morphing graph drawings. This has been addressed in the graph drawing literature by several practical approaches. Friedrich and Eades [13] formulate criteria for animation between two graph drawings and give a procedure that partially satisfies these criteria. A subsequent paper [14] augments this with clustering techniques, but neither method preserves planarity. The Floater-Gotsman-Surazhsky algorithm discussed above has been enhanced with a preliminary rigid motion stage in [10].

Restricting to special cases permits more theoretical guarantees. In previous work we gave an algorithm to morph between two planar orthogonal drawings of a graph, preserving planarity and orthogonality, and using a quadratic number of linear morphs [22]. For other work on preserving directions while morphing graph drawings, see [4, 24].

There is a substantial body of research on the special case of polygon morphing. Gotsman and Surazhsky tailor their general algorithm to the case of polygons [17]. A much stronger result, based on linkage reconfiguration, is given by Iben et al. [19] who give an algorithm to find intersection-free polygon morphs in which edge lengths change monotonically. There are other approaches to polygon morphing using a variety of different techniques (see references in [19]), but none of them preserve planarity. It would be especially interesting to have discrete versions of the Gage-Hamilton-Grayson result on shrinking planar curves [15, 18].

1.2 Terminology

A *planar drawing* of a graph $G = (V, E)$ assigns to each vertex $v \in V$ a distinct point $p(v)$ in the plane, and to each edge $e = (u, v)$ a path (or curve) from $p(u)$ to $p(v)$ in such a way that paths intersect only at a common endpoint. A planar *straight-line* drawing is a planar drawing in which every edge is drawn as a straight line segment. A *plane graph* is one that has a planar drawing. We assume that graphs are connected. Two planar drawings of a graph are *combinatorially identical* if they have the same cyclic order of edges around vertices and the same outer face.

We will consider drawings in which edges are drawn as polygonal paths. In other words, we work with straight-line drawings of a *subdivided* graph in which every edge has extra vertices added along it, called *bends*. We use the following convention: the input graph has "original vertices" and "edges"; the subdivided graph has "vertices" (original plus bends) and "segments". Segments are drawn as straight line segments, and an edge is a sequence of segments.

A *morph* from a drawing P of a graph G to a drawing Q of G is a continuous family of drawings $P(t)$, indexed by time $t \in [0, 1]$ where each $P(t)$ is a drawing

of G, and $P(0) = P$ and $P(1) = Q$. A morph *preserves planarity* if each $P(t)$ is planar.

A *linear morph* has the property that if the initial position of vertex v is point $p(v)$ and its final position is $q(v)$ then at time $t \in [0, 1]$ during the morph, v is at point $tq(v) + (1 - t)p(v)$. A morph is *discrete* if it is composed of a sequence of linear morphs.

2 The Morphing Algorithm

We give an algorithm that takes two combinatorially identical planar straight-line drawings P and Q of a connected graph G, and finds a planarity-preserving morph from P to Q using a polynomial number of elementary steps that are linear morphs. Edges are allowed to bend during the course of the morph.

Conceptually, the morph is simple. Let $v_1, v_2, \ldots, v_n$ be the vertices of Q ordered by increasing x-coordinate, with ties broken by increasing y-coordinate. We first locate v_1 in P—it must be on the outer face—and "pull it out" of the drawing P until it is at the far left, allowing the edges incident to v_1 to bend in compensation. We then repeat with v_2, v_3 and so on until the vertices of P appear in the same x-ordering as those of Q. If the "pulling out" is done with care, the edges of P will now be polygonal paths monotone in the x direction. We then perform a linear morph on P to adjust all vertices to the correct y coordinate, thus straightening all the polygonal paths.

We now introduce some terminology and discuss the morph in more detail. Throughout the morph, the drawing P changes but we will continue to refer to it as P. The algorithm begins with a set-up phase in which we discretize the x coordinates by placing a vertical line through each vertex of P. We also add n vertical lines $L_i, 1 \leq i \leq n$ to the left of the drawing of P. See Figure 2. Line L_i will be the eventual home of vertex v_i, and when all vertices v_i reach their line L_i then the vertices of P will be in the same x-order as those of Q. Throughout the course of the morph we will have a finite set of vertical lines, and we will enforce the invariant that every vertex lies on one of the vertical lines. Furthermore, we expand the definition of a *bend*: any intersection point between an edge's polygonal path and one of the vertical lines is a *bend*. We will not consider original vertices to be bends.

The other part of the set-up phase involves planning the route each vertex of P will follow as we pull it to its eventual position at the left. We augment Q with an extra vertex v_0 at the far left. We also augment Q with extra straight-line edges so that every vertex v_i, $i = 1, \ldots, n$ is adjacent to at least one vertex with smaller x-coordinate. (A triangulation of Q contains such edges.) We choose one *entering* edge e_i for each vertex v_i, $i = 1, \ldots, n$ so that e_i joins v_i to a vertex of smaller x-coordinate. Observe that if $e_i = (v_j, v_i)$ then $j < i$, and observe that the set of entering edges forms a tree rooted at v_0. See Figure 2.

We augment P to match the augmented Q and its embedding by routing each new edge as a polygonal path, preserving planarity (details below). We add vertical lines through any new bend vertices to maintain the invariant that

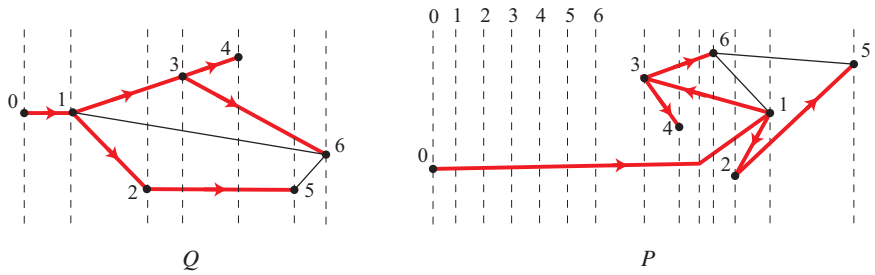


Figure 2: Set up phase. Edges with arrows are the entering edges that form a tree rooted at vertex 0. The new edge from vertex 0 to vertex 1 is drawn with a bend in P.

every vertex lies on a vertical line.

In the main body of the algorithm, vertices are pulled to the left one by one. Figure 3 shows an example. In iteration $i = 1, \ldots, n$, vertex v_i is pulled along the path of its entering edge e_i until it reaches line L_i. This is accomplished by repeatedly moving v_i along the last segment of e_i from the vertical line on which v_i currently lies to an adjacent vertical line—i.e. to the next bend along e_i. We call this the *main step* of the algorithm, and give details below. Each main step is preceded by a *straightening step* that modifies the paths of the edges incident to v_i. This is done in order to enforce certain properties of the drawing in the vicinity of v_i, which we describe in more detail below.

The final phase of the algorithm is a linear morph from P to Q. We prove that it preserves planarity by proving that, after the main body of the algorithm, the trapezoidizations of P and Q are combinatorially the same.

We summarize the algorithm as follows:

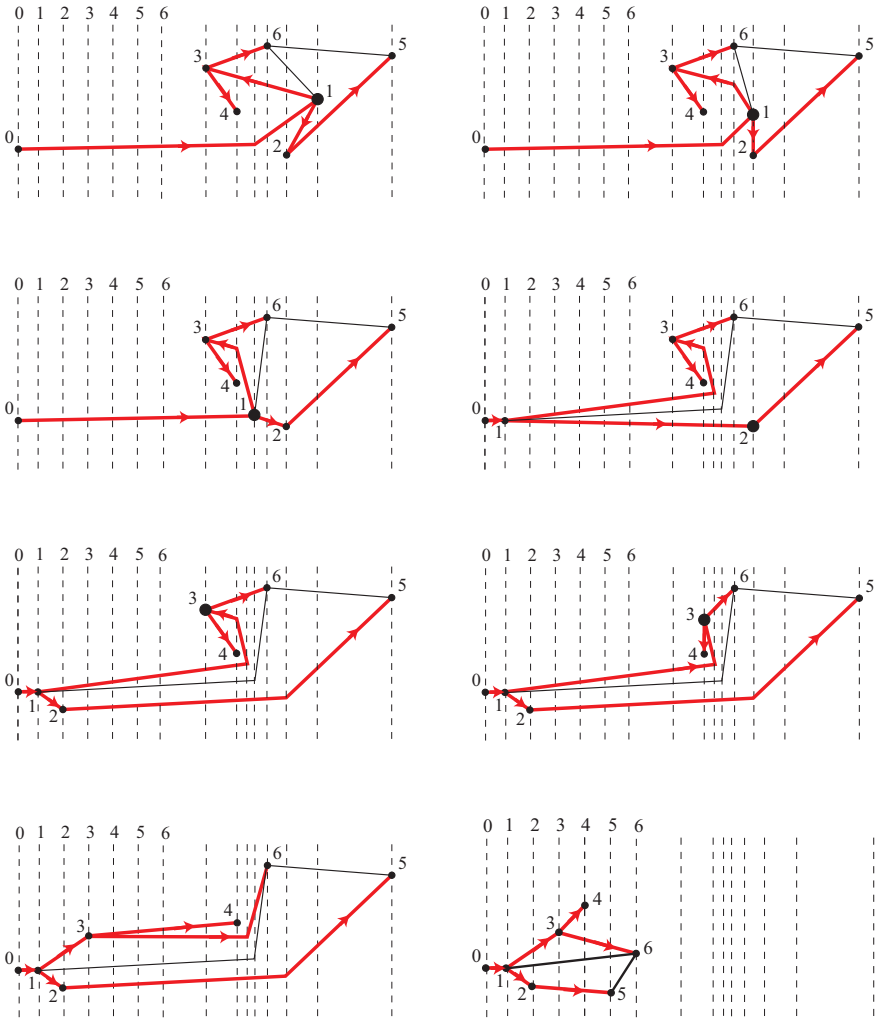
Algorithm 1: Morph

```
1 perform set-up phase on P and Q
2 morph P as follows
3 for i = 1...n do
4 |   repeat
5 |   |   perform the straightening step on v_i to enforce Property 1 (stated
  |   |   below)
6 |   |   morph to move v_i along the last segment of e_i
7 |   until v_i is on L_i
8 end
9 perform a linear morph on the vertices of P to match Q
```

We finish this section with a description of the property enforced by the straightening step of the algorithm, and a discussion of some other invariants maintained by the algorithm.

We define an edge to be an *incoming* edge to v_i if its other endpoint is

Figure 3: Example morph (read across the rows). The vertex that is about to move is drawn with a heavy dot. Several steps are omitted near the end.



a lower indexed vertex, and an *outgoing* edge otherwise. Thus the incoming edges to v_i are exactly those incident to the left of (or vertically below) v_i in drawing Q. Note also that the entering edge of v_i is one of its incoming edges. The *linear order* of incoming edges to v_i in Q is the order of incoming edges counterclockwise around v_i from topmost to bottommost. Note that in the drawing P the incoming edges appear contiguously in the cyclic ordering of edges around v_i.

The straightening step enforces the following property of the drawing P (see Figure 5).

Property 1

1. Either all incoming edges enter v_i strictly from the left of the line through v_i, or they all enter strictly from the right. Furthermore, their ordering—top-to-bottom if on the left, and bottom-to-top if on the right—matches their linear order in Q.

2. If l' is the adjacent vertical line on the side of the incoming edges, there are no other vertices along l' between the incoming edges.

We note that the second clause of Property 1.1 is redundant if v_i has an outgoing edge, but in case v_i has only incoming edges, it prohibits the possibility that the cyclic order of incoming edges is broken into different linear orders in Q and [the straightened] P.

Throughout the algorithm we maintain the following property of drawing P:

Property 2

1. the part of each edge that appears in the interval $L_1 \dots L_n$ is a monotone path

2. a vertical edge segment is never incident to a bend

Properties 2.1 and 2.2 both hold initially and after the set-up phase. Property 2.2 is easy to maintain: if morphing ever produces a vertical edge segment incident to a bend, we can morph the bend away.

The following subsections contain detailed descriptions of the four steps of the algorithm: set-up; straightening; the main step; and the final linear morph. The ordering of sections reflects the order of the steps in the algorithm, but the reader might wish to begin with the main step in Section 2.3. Analysis of the number of elementary steps and the run time is in Section 3.

2.1 Set-Up

The one set-up issue that warrants further explanation is adding the new entering edges to the drawing P. If entering edge e_i is added to Q, then it must be added to the same face in P. Furthermore, in case e_i is added to the outer face, there are still two ways to do this, and we must choose the one that gives the same resulting outer face as in Q.

Because planarity must be maintained, a new edge cannot always be drawn as a straight line segment, and must in general be drawn as a polygonal path

in P. We give some details about routing these paths. Note that there are at most n new edges, and each one joins two vertices in a common face, and no two cross (in the combinatorial sense) because they form a planar augmentation of drawing Q.

Construct a trapezoidization of P. Note that this is a conventional trapezoidization where a vertical segment extended through a vertex stops when it hits an edge. (In most of the paper, vertical lines extend all the way.) We obtain $O(n)$ trapezoids. Note that we allow vertically collinear vertices, so a trapezoid may have more than one vertex on the left or right. For each face of P the trapezoids are joined in a tree-like fashion. Each new edge e_i goes through a sequence of trapezoids that corresponds to a path. It remains to route the new edges through the trapezoids. At most $O(n)$ edges cross the vertical line segment between two adjacent trapezoids—space them nicely along the line segment. Each trapezoid τ is traversed by at most $O(n)$ [portions of] edges—draw these as disjoint paths in τ using one or two line segments each. See Figure 4. Note that each new edge e_i is routed as a polygonal path with $O(n)$ bends. In fact, there is great freedom in routing the new edges, and this bound is the only thing we need. Remember that we add new vertical lines through any bends.

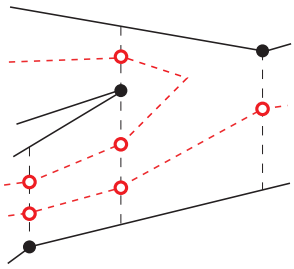


Figure 4: Routing new edges (dashed) through the trapezoids.

2.2 Straightening Step

In this section we show how to morph P to achieve Property 1 for vertex v_i just prior to each invocation of the main step of the algorithm. Figure 5 (left) shows an example where both conditions of Property 1 are violated: vertex v_i has an incoming edge from the left and from the right, and vertex p lies between two incoming edges on line l'. Observe that in both cases there is an edge that crosses a vertical line twice, forming an empty polygonal area (darkly shaded in the figure). More formally, a *redundant path* in a planar graph drawing is a subpath of an edge that forms a polygonal curve C with initial vertex s and final vertex t both on vertical line l, and with the property that C plus line segment st forms a simple polygon; in particular this means that no other part of the drawing crosses or touches line segment st. Our first goal is to collapse a redundant subpath using a planarity preserving morph. Following that, we

prove that eliminating redundant subpaths enforces Property 1, and we show how to find redundant subpaths.

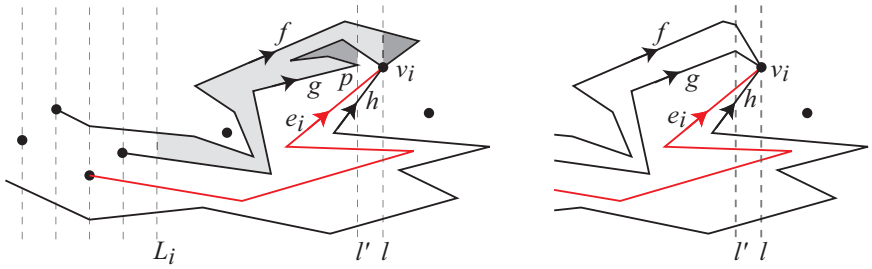


Figure 5: (*left*) Property 1 fails because incoming edge f enters v_i at the opposite side from e_i, and intervening point p lies on line l' between the incoming edges; (*right*) Property 1 holds after redundant paths (darkly shaded at left) are morphed away.

Lemma 1 *Suppose C is a k-vertex redundant path with initial vertex s and final vertex t on vertical line l. Let D be the simple polygon formed by C plus segment st. Then C can be morphed inside polygon D to segment st via a morph composed of $O(k)$ linear morphs each of which moves one vertex.*

Proof: The vertical lines partition D into trapezoids (including triangles as degenerate trapezoids). The trapezoids are connected in a tree, which we can root at the trapezoid that has st on its boundary. Starting from the leaves of this tree, we morph trapezoids away. See Figure 6. In the case of a triangle, with vertices x and y on one vertical line and z on an adjacent line, we morph by moving z to x. This is shown in Figure 6(a)–(b), (b)–(c) and (e)–(f). In the case of a trapezoid with more vertices, we morph by moving one vertex at a time along its vertical line until we reduce to a triangle. Figure 6(c)–(f) shows a 5-vertex trapezoid morphed away in three steps. This process uses one linear morph per vertex, with each linear morph moving one vertex. □

When a redundant path is morphed away, it leaves a vertical segment. If either endpoint of the vertical segment is a bend, we morph one bend away as shown in Figure 6 (g)–(h). This is done to maintain Property 2.2.

We now prove that if Property 1 fails, then there are redundant subpaths. Recall that drawing Q determines a linear order of incoming edges to v_i. Let $f = (v_j, v_i)$ and $g = (v_k, v_i)$ be consecutive edges in this linear order. The entering edges form a tree rooted at v_0. Let v_r be the least common ancestor of v_j and v_k in this tree. Take paths from v_r to v_j and from v_r to v_k in the tree. Adding the edges f and g yields a cycle B_{fg} in the graph. In drawing Q, this cycle is drawn as two x-monotone paths, the upper one from v_r to v_i through f, and the lower one from v_r to v_i through g. Note that the two monotone paths are in fact strictly monotone except that the last edge g may be a vertical

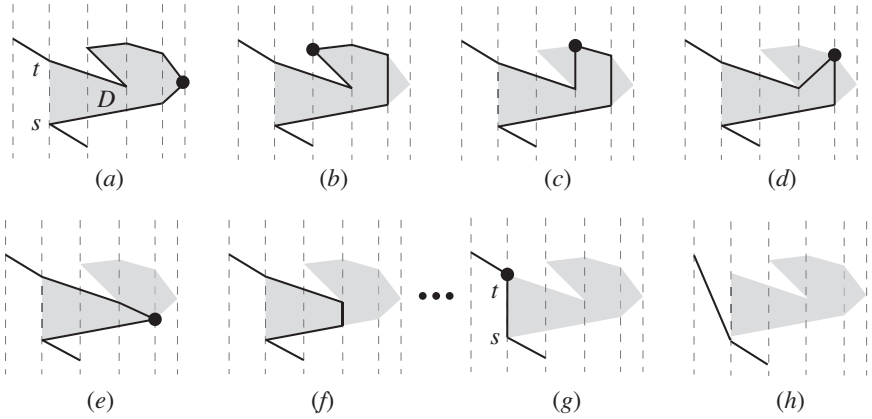


Figure 6: Morphing away the trapezoids of a redundant path and the final segment st. The vertex that is about to move is drawn with a heavy dot. Four steps are omitted between (f) and (g).

edge. Every vertex on or inside the cycle B_{fg} has index less than i. We now consider the appearance of this cycle in drawing P during the i^{th} iteration of the algorithm. With the exception of v_i, all vertices on or inside B_{fg} have already been pulled to their lines at the far left, to the left of L_i. By Property 2.1 all edges are monotone curves in the interval $L_1 \dots L_i$ and therefore all edges on or inside B_{fg} lie to the left of L_i except for f and g, which cross L_i once with f above g. Consider the polygon D_{fg} formed by the path of f from L_i to v_i, the path of g from v_i to L_i and the line segment on L_i between f and g. An example polygon is lightly shaded in Figure 5. Polygon D_{fg} is contained in cycle B_{fg} and the above discussion implies:

Claim 2 *Polygon D_{fg} contains no other part of the drawing.*

We now use this claim to show (in the following two lemmas) that if either condition of Property 1 fails, then there is a redundant path. We begin with Property 1.1. Note that although Figure 5 happens to show a case where elimination of redundant paths results in all incoming edges entering from the left, there is nothing in the following lemma or its proof that refers to left/right.

Lemma 3 *Suppose that during iteration i of the algorithm, with v_i on line l in drawing P, Property 1.1 fails. Then there is a redundant path that is a subpath of one of v_i's incoming edges with one endpoint at v_i and the other endpoint on l.*

Proof: We first show that no incoming edge reaches v_i via a vertical segment. The other end of such a segment cannot be a bend because of Property 2.2, and cannot be an initial vertex because the incoming edges come from vertices that have already been moved to the L lines in previous iterations of the algorithm.

The hypothesis implies that the counterclockwise wedge between the first and last incoming edge to v_i does not remain strictly to the left or to the right of l, and therefore that this wedge must include either the vertical ray above v_i or the vertical ray below v_i—suppose the former (the other case is symmetric). We obtain edges f and g, consecutive in the linear order of incoming edges, such that the vertical ray above v_i starts out strictly inside the polygon D_{fg} (as described above). By Claim 2, D_{fg} is empty. Traversing up line l from v_i, we are at first inside D_{fg}. The first point where we exit D_{fg} is a bend vertex of f or of g, and gives a redundant path in f or g. □

Lemma 4 *Suppose that during iteration i of the algorithm, with v_i on line l in drawing P, Property 1.1 holds, with incoming edges arriving from adjacent line l', but Property 1.2 fails. Then there is a redundant path that is a subpath of one of v_i's incoming edges with endpoints on l', and with one of the endpoints being a bend joined by a single incoming segment to v_i.*

Proof: Let B be the set of bends on l' that are sources of incoming segments to v_i. Let s be the line segment on l' between the topmost and bottommost bend of B. By hypothesis, there is an "intervening" vertex b on s that is not in B. Vertex b lies between two bends of B—suppose they are bends b_f and b_g of incoming edges f and g respectively, with f before g in the linear order of incoming edges, i.e f above g if l' is to the left of l, and f below g if l' is to the right of l. (In Figure 5 intervening bend p lies between consecutive incoming edges g and e_i. Note that in the figure, l' is to the left of l, but the proof does not make any such assumption.) Let D_{fg} be the polygon between f and g as described above. By Claim 2, D_{fg} is empty. The empty open triangle b_f, v_i, b_g lies inside D_{fg}, and therefore any intervening vertex in the interval $b_f b_g$ is a bend of D_{fg}. Let x_f be the intervening vertex closest to f in this interval, and let x_g be the intervening vertex closest to g in this interval (they may be the same vertex). If x_f is part of edge f then endpoints b_f and x_f form a redundant subpath of f. Otherwise x_f is part of edge g, and, since f and g cannot cross, therefore x_g must also be part of edge g. In this case endpoints b_g and x_g form a redundant subpath of g. □

In order to find redundant paths of the types guaranteed in Lemmas 3 and 4 it suffices to be able to find, from any vertex, the next vertex above/below on the same vertical line. To implement the algorithm, we will store all these sorted orders. Further discussion of run time can be found in Section 3.

2.3 Main Step of the Algorithm

In the main step of the algorithm, vertex v_i is moved from its current vertical line to an adjacent line along the path of entering edge e_i. The main step is invoked repeatedly during iteration i of the algorithm to move vertex i one bend at a time along the entering edge e_i until v_i reaches L_i. Iterations $1, \ldots, i-1$ have already moved vertices $v_1, \ldots, v_{i-1}$ to lines $L_1, \ldots, L_{i-1}$ respectively. When the main step is called, the straightening step has just enforced Property 1 for v_i.

We also assume that Property 2 holds so far, and prove below (Lemma 6) that it holds at the end of iteration i.

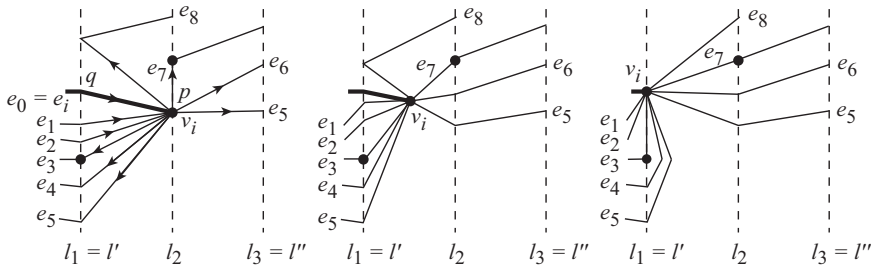


Figure 7: The main step of the algorithm moves v_i along the last segment of e_i. Edges outgoing to l_3 acquire new bends on l_2. Edges outgoing to l_1 may acquire new bends in-between l_1 and l_2.

The operation of the main step is local, altering only the position of v_i and the vertices (original or bend vertices) joined to v_i by a single segment. These vertices all lie on line l_2, the line of v_i, and its predecessor l_1 and its successor l_3. Since vertices only occur on vertical lines, portions of edges between adjacent vertical lines are straight, which makes it possible to use linear morphs. By Property 1 all incoming edges to v_i are contiguous and arrive from the same side—denote the line on that side (l_1 or l_3) by l', and denote the line on the other side by l''. By Property 1 there are no vertices along the line l' between the incoming edges. Note that all incoming edges come from lower numbered vertices that have already been pulled to the L lines at the far left. This does not imply that incoming edges enter from the left—see for example vertex 3 in Figure 3 which is pulled to the right along its entering edge before eventually being pulled left. However, it does imply that incoming edges cross l' at bend vertices.

Figure 7 shows an example of the main step of the algorithm in the case where $l' = l_1$. Notation: Let p be the position of v_i on l_2 and let q be point where the last segment of e_i from l' to v_i crosses l'. We morph as follows:

Vertex v_i moves from p to q.

Incoming edges arrive from bends on l'. Move these bends along l' to q. (See edges e_1 and e_2 in Figure 7.)

Outgoing edges have several cases:
(A) Any edge to l'' acquires a new bend on l_2, initially at p, and with final positions nicely spaced on l_2. (See e_5 and e_6.)
(B) There may be an edge lying on l_2, in which case it goes to an original vertex (not a bend). Leave the vertex fixed. (See e_7.)
(C) Finally, consider an edge to a vertex t on l'. Suppose first that the interval along l' between q and t contains only bends connected to v_i by a single segment. If t is a bend then move t to q. (See e_8.) If t is an original vertex then t stays

fixed and the edge (v_i, t) morphs to lie along l'. (See e_3.) For the last case, suppose that there is an *intervening* vertex along l' between q and t. In this case t stays fixed and the segment (v_i, t) morphs to a two-segment path that bends around the intervening vertex. (See e_4 and e_5.)

It remains to specify exactly where to place new bends. We will do that next, and we will show that each main step can be accomplished via two linear morphs, the dividing point being when the new bends appear in-between l' and l_2.

Follow the example in Figure 8 (but note that the example shows $l' = l_1$ and the text is more general). This figure does not show v_i's incoming edges (other than e_i) because they play no role in the construction. Let b_u be the lowest intervening vertex above q on line l', and let b_l be the highest intervening vertex below q on line l'. See Figure 8(a). Pick a range C on l_2 in which to place the new bends on the edges that go from v_i to l''. Expand C if necessary to include any vertices on l_2 joined via a single segment to v_i. Let W be the wedge from q to interval C. See Figure 8(b). Choose vertical line l between l' and l_2 such that the intersection of l and W lies strictly above the horizontal line through b_l and strictly below the horizontal through b_u. As described above, we perform a morph that moves v_i from p along the edge e_i to q. We divide the morph into two linear morphs: before and after v_i reaches l. Until v_i reaches l, the edge segments from l' to v_i remain straight. When v_i reaches l we place a new bend on every edge segment that crosses the horizontal lines through b_u and b_l. See Figure 8(d). These new bends stay fixed as v_i moves from l to q.

Observe that this construction maintains planarity. In particular, the edges that acquire new bends in the strip between l' and l will not intersect with each other or with the old outgoing edges of v_i (which live in the wedge W). We summarize with:

Claim 5 *The main step of the algorithm uses 2 linear morphs, maintains planarity, maintains Property 2.2, and shortens e_i by one segment.*

Recall that the main step of the algorithm is repeated in iteration i of the algorithm until v_i is on L_i. The argument that v_i eventually reaches L_i is not obvious, since the number of repetitions depends on the number of bends on e_i, and the main step may increase this number. We will bound the number of repetitions in Section 3. For now, we wrap up assuming that the algorithm halts:

Lemma 6 *At the end of the main loop of the algorithm, drawing P has been morphed so that vertices $v_0, \ldots, v_n$ lie on lines $L_0, \ldots, L_n$ respectively, all edges appear as monotone paths, and the incoming edges at each vertex enter from the left and in the same linear order (top to bottom) as in Q.*

Proof: We prove by induction on i that at the end of iteration i, vertices $v_0, \ldots, v_i$ lie on lines $L_0, \ldots, L_i$ respectively, Property 2 holds (i.e. all edges appear as monotone paths in the interval $L_0, \ldots, L_n$), and the incoming edges

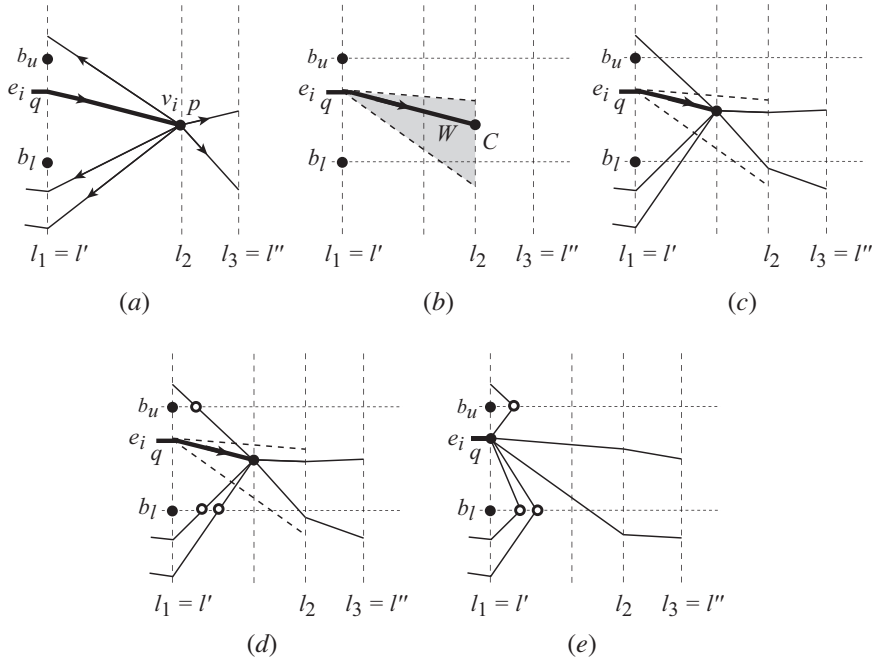


Figure 8: How to add new bends: (a) initial situation; (b) placing wedge W (shaded) and line l; (c) moving v_i to l; (d) placing new bends (hollow); (e) final situation.

at each vertex $v_1, \ldots, v_i$ enter from the left and in the same linear order (top to bottom) as in Q. The basis case is $i = 0$.

Consider iteration $i \geq 1$. By induction vertices $v_0, \ldots v_{i-1}$ lie on lines $L_0, \ldots, L_{i-1}$. By induction and Property 2, the incoming edges to v_i form monotone paths in the interval $L_1, \ldots, L_n$.

The main step affects only the vertical strips to the left and right of v_i, so paths in the range $L_1 \ldots L_n$ are affected only when v_i reaches those lines. Consider the invocation of the main step that moves v_i to the line L_n. At this point, the entering edge e_i consists of a monotone path in the interval $L_1, \ldots, L_n$ plus one segment from L_n to v_i. By Property 1 all the other incoming edges to v_i also arrive at v_i from L_n and their top-to-bottom ordering matches their linear order in Q. Each incoming edge consists of a monotone path in the interval $L_1, \ldots, L_n$ plus one segment from L_n to v_i.

The invocations of the main step that move v_i to $L_n, L_{n-1}, \ldots, L_i$ behave in a particularly simple way. Incoming edges always arrive from the left and outgoing edges leave to the right. The main step shortens the incoming edges by one segment and lengthens the outgoing edges by one segment. All of these edges form monotone paths in the interval $L_1, \ldots, L_n$, and the order of incoming edges to v_i does not change. Therefore, when v_i reaches L_i, Property 2 holds,

and the incoming edges to v_i enter from the left and in the same top to bottom order as in Q. By induction, the lemma is proved.

□

2.4 Final Linear Morph

The last step of the algorithm is a linear morph from P to Q. We justify correctness.

After the main loop of the algorithm, the drawing P has been morphed so that the x-coordinate ordering of the original vertices matches their index ordering, which is the x-coordinate ordering in Q with ties broken by y-coordinate. Furthermore, by Lemma 6, all the edges of P are monotone paths, and the incoming edges enter each vertex from the left, as in Q, and in the same linear order as in Q.

Consider the trapezoidization of P determined from the vertical lines and consider the trapezoidization of a perturbed version of Q where vertically collinear vertices are moved slightly apart to match the index ordering. By the above-mentioned properties, these trapezoidizations are combinatorially the same. A linear morph applied to a single trapezoid maintains planarity (allowing the trapezoid to collapse at the end of the morph). Thus the linear morph from P to Q maintains planarity, since it acts as a linear morph applied simultaneously to all the trapezoids.

3 Analysis

In this section we give a bound on the number of elementary moves required by our morph, and on the run time of the algorithm that computes the morph. In subsection 3.1 we discuss the grid size of intermediate drawings. All analyses are in terms of n, the number of vertices of the input graph.

We will prove that our morph is composed of $O(n^6)$ linear morphs, and that a complete description of the morph has size $O(n^6)$. The second result does not follow from the first since a single one of our linear morphs may move as many as n vertices.

We begin by discussing the main ideas and difficulties. The number of linear morphs used by the algorithm depends directly on the number of bends, but each step of the algorithm creates new bends. In particular, the final case of the main step of the algorithm (see the right-hand pane of Figure 7) may introduce new bends, which require new vertical lines. Each crossing of a new vertical line by an existing edge counts as a new bend. Thus, eliminating one bend of e_i may, in the worst case, cause a quadratic increase in the number of bends, which is potentially very dangerous. To circumvent the danger, we focus on *turns*, which are bends where the path changes x-direction, and we examine more closely the straightening step and the main step of the algorithm, and argue that turns are *propagated* rather than created. Looking at the big picture, the intuition is that, since iteration i of the algorithm causes each outgoing edge of v_i to follow the

path of the entering edge e_i, thus, at worst, we copy the turns of e_i onto all the outgoing edges of v_i.

We begin by defining a *turn* more precisely. Let e be an edge from v_j to v_k with $j < k$, drawn as some polygonal path. If b is a bend in e (i.e. a point distinct from v_j and v_k where e intersects one of the vertical lines) and the two segments of e incident to b lie on the same side of b (i.e. both to the left or both to the right) then b is a *turn*. Furthermore, there is one situation in which we count the initial endpoint v_j of e as a turn: if the entering edge e_j arrives at v_j on the same side as the outgoing edge e leaves, then v_j is a *turn* associated with e. In order to distinguish the two kinds of turns, we will call them *internal turns* and *endpoint turns* respectively.

We first bound the number of turns after the set-up phase.

Lemma 7 *After the set-up phase each edge has $O(n)$ turns; in the worst case there may be $\Omega(n^2)$ turns in total.*

Proof: In a straight-line drawing an edge can have at most one turn, the one at its initial endpoint. The set-up phase introduces edges with bends, but as noted in the detailed description in Section 2.1, each new edge is drawn as a polygonal path of $O(n)$ bends, hence $O(n)$ turns. See Figure 9 for an example of the lower bound construction. It is easy to see that this can be generalized. □

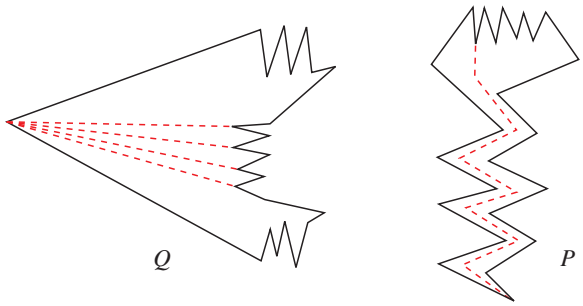


Figure 9: The set-up phase can create $\Theta(n^2)$ turns: (left) New entering edges added to Q (shown dashed); (right) Each new entering edge will be drawn with many turns in P. This can be generalized to $\Theta(n)$ entering edges with $\Theta(n)$ turns each.

We now argue that during the course of the main loop of the algorithm, turns are not created, but only propagated—specifically from e_i to the outgoing edges of v_i.

Lemma 8 *During iteration i, the only changes to turns are the following: (1) an internal turn of e_i may "propagate" to being an endpoint turn of [some of the] outgoing edges of v_i; (2) an endpoint turn of an outgoing edge of v_i may become an internal turn of that edge; (3) turns may vanish.*

Proof: First consider the straightening step, which eliminates a redundant subpath of an edge, say e, incoming to v_i. No internal turns are created along e, and all internal turns along the redundant subpath vanish. Note that there is at least one such turn that vanishes. Thus the changes to e are fine. The only remaining issue is that eliminating a redundant subpath of $e = e_i$ may change the direction that e_i enters v_i and thus cause new endpoint turns to appear on outgoing edges. This is covered by case (1).

Now consider the main step. See Figure 7. Let f be an edge that acquires a new internal turn (for example e_4 and e_5 in the figure). Then f originally leaves v_i on the same side as e_i and therefore has an endpoint turn at v_i. After the morph, f become incident to v_i on the opposite side, so we have case (2): an endpoint turn of f becomes an internal turn of f. The last thing to consider is the possibility of new endpoint turns. These only occur: (a) when an outgoing edge changes its side of incidence to v_i; or (b) when e_i changes its side of incidence to v_i. Case (a) is fine because no outgoing edge moves from the opposite side to the same side as the initial e_i. Case (b) only occurs if point q (the target point of v_i) was a bend of e_i, which is covered by case (1) in the Lemma statement. □

We now get a bound on the total number of turns, and thus on the total number of bends.

Lemma 9 *Over the course of the morph each edge has $O(n^2)$ turns.*

Proof: We count the total number of turns that propagate to a particular edge $e = (v_j, v_k)$. The entering edges e_i form a directed tree T rooted at v_0. The path in T from v_0 to v_j has at most n edges, and by Lemma 7 each edge has $O(n)$ original turns that may propagate onward. Thus there are at most $O(n^2)$ turns that can propagate to edge e. Since propagation is the only way that turns appear, this proves that there are $O(n^2)$ turns per edge. □

Lemma 10 *Over the course of the morph an edge will have at most $O(n^5)$ bends.*

Proof: By Lemma 9 above there are $O(n^2)$ turns per edge, thus $O(n^3)$ turns in the whole drawing. Each vertical line goes through a turn or a vertex so there are $O(n^3)$ vertical lines. Now any edge has $O(n^2)$ turns, and each section between consecutive turns can at worst cross all the $O(n^3)$ vertical lines. This gives a total of $O(n^5)$ bends per edge. □

To be more rigorous about turns and bends appearing and disappearing we should really label them. Label each turn with a 3-tuple consisting of: the edge on which the turn currently lies, the original edge that hosted the turn after the set-up phase, and the index of the turn $(1 \ldots n)$ on that original edge. We label a bend with the label of the preceding turn along its edge and the label of the turn whose vertical line created the bend. As the arguments above show, there are $O(n^2)$ turn labels per edge, and $O(n^5)$ bend labels per edge. Each turn/bend has a unique label, and a label never recurs.

Lemma 11 *Over the course of the algorithm the straightening steps use $O(n^6)$ linear morphs in total, and each of these linear morphs moves $O(1)$ vertices.*

Proof: By Lemma 1, eliminating a redundant polygon of k bends takes $O(k)$ linear morphs each moving $O(1)$ bends. Each morph eliminates a bend. By Lemma 10 there are $O(n^6)$ bends in total. This proves the lemma. □

Lemma 12 *Let d_i be the degree of vertex v_i. One main step of the algorithm in iteration i takes 2 linear morphs, each moving at most $d_i + 1$ vertices. It can be described in size $O(d_i)$.*

Proof: Claim 5 justified the use of 2 linear morphs per main step of the algorithm. The main step moves only v_i and the vertices joined to v_i by a single segment—hence at most $d_i + 1$ vertices. A linear morph that moves t points can be described in size $O(t)$. □

Lemma 13 *Over the course of iteration i the main steps of the algorithm use $O(n^5)$ linear morphs in total and can be described in size $O(d_i n^5)$. Over the course of the whole algorithm the main steps use $O(n^6)$ linear morphs and can be described in that size.*

Proof: Iteration i moves v_i along e_i until it reaches L_i. The number of main steps is equal to the number of bends on e_i. Combining Lemmas 10 and 12 gives the result for one iteration. To get the bound for all n iterations note that $\sum_i d_i$ is $O(n)$ for a planar graph. □

Theorem 1 *Between any two drawings of an n-vertex planar graph there is a planarity preserving morph that uses $O(n^6)$ linear morphs and has a complete description of size $O(n^6)$. Furthermore, there is an algorithm to compute a description of the morph that runs in time $O(n^6)$.*

Proof: The first statement follows from combining Lemma 11 for the straightening steps and Lemma 13 for the main steps.

We now discuss the run time of the algorithm. It suffices to implement the algorithm in such a way that each elementary step can be found in time proportional to the number of vertices that are moved, since we have an $O(n^6)$ bound on that. The cost of accessing v_i's neighbours is then covered. The one thing to worry about is the time spent on finding intervening vertices in the straightening step and in the main step, and the time spent on finding redundant paths in the straightening step.

To handle this we will maintain the sorted order of vertices along each vertical line. Then the cost of finding each redundant path in the straightening step is constant, and the cost of computing one of the main step morphs in iteration i is proportional to the degree of v_i.

After we insert a new vertical line we must explicitly find and store its sorted list of bends. The list can be found by following the list of bends on an adjacent vertical line, and the time to do this can be charged against the bends. Thus we can implement the algorithm to run in time $O(n^6)$. □

3.1 Analysis: Grid Size

Throughout the morph, the number of distinct x-coordinates is equal to the number of vertical lines, which is $O(n^3)$. This is because a vertical line goes through an original vertex or a turn and there are $O(n^3)$ turns by Lemma 9. A bound on the number of vertical lines is not quite a bound on grid size because the vertical lines are not equally spaced, but it is easy enough to enhance the algorithm to maintain the property that the vertical lines are $x = 1$, $x = 2$, etc. Whenever a new vertical line is about to be added, perform a linear morph to push all later vertical lines one unit to the right. Such a linear morph preserves planarity of the whole drawing because it preserves planarity of each trapezoid.

We can do a similar thing to force the y-coordinates to a grid. First note that the number of vertices along any vertical line is $O(n^3)$. This is because each edge crosses the line $O(n^2)$ times, once after each turn. We can maintain the property that vertices along each vertical line lie at consecutive coordinates $y = 1$, $y = 2$, etc. As before, this is done by adding an extra linear morph before each step of the algorithm to shift the vertices vertically and make room for new ones.

With these enhancements to the algorithm we can guarantee that the drawings between the linear morphs lie on the grid.

Theorem 2 *Between any two drawings of an n-vertex planar graph there is a planarity preserving morph that uses $O(n^6)$ linear morphs and has the property that every intermediate drawing lies on an $O(n^3) \times O(n^3)$ grid.*

Observe that we have lost the property that each linear morph in the main part of the algorithm moves only a local set of vertices (specifically, those joined to v_i by one segment). It may be possible to keep the description size of the morph down to $O(n^6)$ because the grid-maintaining morphs move sets of vertices that can be described succinctly, but we do not pursue this idea.

Note that for visualization purposes it may be much better to preserve long straight subpaths of edges than to do the overly rigid grid-preservation described above. A practical compromise might be to let the drawing grow naturally for a while, and then recoup a compact grid every once in a while.

Our morph does not guarantee nice vertex-edge distances: for example an edge from $(1, 1)$ to $(2, M)$ passes close to a vertex at $(1, 2)$. A consequence of this is that an edge can become very short during one of our linear morphs, even though the minimum edge length is 1 when the graph is on the grid.

4 Conclusion

We have given a planarity-preserving morph between any two combinatorially identical planar drawings of a graph. The main contribution is that our morph consists of a polynomial-size sequence of linear morphs. However, we had to add bends to the edges. We leave open the question of finding a morph that

consists of a polynomial-sized sequence of linear morphs and preserves straight-line edges.

Short of a new idea to solve this open question, it should still be possible to improve our results, because the $O(n^6)$ bound on the number of linear morphs seems excessive. One possibility would be to work with a trapezoidization (where each vertex extends a vertical segment only to the next edge) rather than extending vertical lines to form a complete grid. Another possibility would be to bundle v_i's incoming edges so that they can be handled as one edge—this might be possible by performing straightening steps in batches ahead of time rather than deferring them until required.

Another interesting problem would be to analyze the Floater-Gotsman-Surazhsky algorithm and prove that some efficiently computable set of time steps t yields a polynomially-bounded discrete morph.

Finally, we have not explored visualization at all. Is our strategy of pulling successive vertices out of the drawing at all useful for visualization purposes?

Acknowledgements

We thank two anonymous referees for suggestions that improved the presentation of the paper.

References

[1] H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. *Journal of Algorithms*, 49(2):262–283, 2003.

[2] H. Alt and L. Guibas. Discrete geometric shapes: Matching, interpolation, and approximation. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 121–153. Elsevier, 1999.

[3] G. Barequet and M. Sharir. Piecewise-linear interpolation between polygonal slices. In *SoCG '94: Proceedings of the Tenth Annual Symposium on Computational Geometry*, pages 93–102, 1994.

[4] T. C. Biedl, A. Lubiw, and M. J. Spriggs. Morphing planar graphs while preserving edge directions. In P. Healy and N. S. Nikolov, editors, *Graph Drawing*, volume 3843 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 2006.

[5] K. Buchin, M. Buchin, and Y. Wang. Exact algorithms for partial curve matching via the Fréchet distance. In *SODA '09: Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 645–654, 2009.

[6] S. S. Cairns. Deformation of plane rectilinear complexes. *American Mathematical Monthly*, 51:247–252, 1944.

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

[8] E. D. Demaine and J. O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, 2007.

[9] A. Efrat, J. Guibas, S. Har-Peled, J. Mitchell, and T. Murali. New similarity measures between polylines with applications to morphing and polygon sweeping. *Discrete and Computational Geometry*, 28(4):535–569, 2002.

[10] C. Erten, S. G. Kobourov, and C. Pitta. Intersection-free morphing of planar graphs. In G. Liotta, editor, *Graph Drawing*, volume 2912 of *Lecture Notes in Computer Science*, pages 320–331. Springer, 2004.

[11] I. Fáry. On straight-line representation of planar graphs. *Acta Sci. Math. (Szeged)*, 11:229–233, 1948.

[12] M. S. Floater and C. Gotsman. How to morph tilings injectively. *Journal of Computational and Applied Mathematics*, 101:117–129, 1999.

[13] C. Friedrich and P. Eades. Graph drawing in motion. *Journal of Graph Algorithms and Applications*, 6:353–370, 2002.

[14] C. Friedrich and M. Houle. Graph drawing in motion II. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, pages 122–125. Springer, 2002.

[15] M. Gage and R. Hamilton. The heat equation shrinking convex plane curves. *J. Differential Geometry*, 23(1):69–96, 1986.

[16] J. Gomes, L. Darsa, B. Costa, and L. Velho. *Warping and Morphing of Graphical Objects*. Morgan Kaufmann, 1999.

[17] C. Gotsman and V. Surazhsky. Guaranteed intersection-free polygon morphing. *Computers and Graphics*, 25:67–75, 2001.

[18] M. Grayson. The heat equation shrinks embedded plane curves to round points. *J. Differential Geometry*, 26(2):285–314, 1987.

[19] H. N. Iben, J. F. O'Brien, and E. D. Demaine. Refolding planar polygons. *Discrete and Computational Geometry*, 41(3):444–460, 2009.

[20] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[21] A. Lubiw and M. Petrick. Morphing planar graph drawings with bent edges. *Electronic Notes in Discrete Mathematics*, 31:45–48, 2008.

[22] A. Lubiw, M. Petrick, and M. Spriggs. Morphing orthogonal planar graph drawings. In *SODA '06: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 222–230, 2006.

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

[24] M. Spriggs. *Morphing parallel graph drawings*. PhD thesis, School of Computer Science, University of Waterloo, 2007.

[25] V. Surazhsky and C. Gotsman. Controllable morphing of compatible planar triangulations. *ACM Transactions on Graphics*, 20(4):203–231, 2001.

[26] V. Surazhsky and C. Gotsman. Intrinsic morphing of compatible triangulations. *International Journal of Shape Modeling*, 9(2):191–201, 2003.

[27] C. Thomassen. Deformation of plane graphs. *Journal of Combinatorial Theory Series B*, 34:244–257, 1983.

[28] W. T. Tutte. How to draw a graph. *Proc. London Math. Soc.*, 13(3):743–768, 1963.