Kinetic and Stationary Point-Set Embeddability for Plane Graphs*

Zahed Rahmati, Sue H. Whitesides, and Valerie King

Department of Computer Science, University of Victoria, Canada
{rahmati,sue,val}@uvic.ca

Abstract. We investigate a kinetic version of point-set embeddability. Given a plane graph $G(V, E)$ where $|V| = n$, and a set P of n moving points where the trajectory of each point is an algebraic function of constant maximum degree s, we maintain a point-set embedding of G on P with at most three bends per edge during the motion. This requires reassigning the mapping of vertices to points from time to time. Our kinetic algorithm uses linear size, $O(n \log n)$ preprocessing time, and processes $O(n^2 \beta_{2s+2}(n) \log n)$ events, each in $O(\log^2 n)$ time. Here, $\beta_s(n) = \lambda_s(n)/n$ is an extremely slow-growing function and $\lambda_s(n)$ is the maximum length of Davenport-Schinzel sequences of order s on n symbols.

Keywords: kinetic graph drawing, point-set embeddability, kinetic algorithm, plane graph.

1 Introduction

Problem Statement. Given a plane graph $G(V, E)$ on n vertices and a point-set $P = \{p_1, p_2, ..., p_n\}$, the problem of *point-set embeddability without mapping* is to draw G on P such that each vertex is mapped to a point of P and such that no two edges intersect except at common vertices. A *straight-line drawing* is a drawing in which each edge of G is mapped to a curve that is a line segment. A *k-bend drawing* is a drawing of G such that each edge is mapped to a chain (curve) of at most $k + 1$ line segments. The *kinetic point-set embedding problem without mapping*, with at most k bends per edge, is to construct a k-bend drawing without mapping of G on a set P of n moving points, where the trajectory of each point is an algebraic function of constant maximum degree s. In the kinetic setting, the trajectory of each point $p_i(t) = (x_i(t), y_i(t))$ is defined by two polynomial functions of constant maximum degree s, and the objective is to maintain the embedding during the motion. During the time period, the point-set embedding may change or develop edge crossings, so a kinetic algorithm is needed to repair the embedding by remapping the vertices.

Related Work. Cabello [9] proved that deciding whether a planar graph G can be embedded by straight-line edges without mapping onto a given set P of points is NP-complete, even when G is 2-connected. Recently, Durocher and Mondal [13] showed that this problem is NP-complete for 3-connected planar graphs. Biedl and Vatshelle [6]

* Partially supported by NSERC and a University of Victoria Graduate Fellowship.

W. Didimo and M. Patrignani (Eds.): GD 2012, LNCS 7704, pp. 279–290, 2013.

proved that the problem is NP-hard for 2-connected outer-planar graphs if the embedding is fixed, 3-connected planar graphs of constant treewidth, and triangulated planar graphs. Gritzmann *et al.* [15] showed that the class of planar graphs such that all vertices are on the outer face (*outerplanar* graphs) is the largest class of graphs that can be embedded with straight-line edges without mapping onto any point set in general position (no three or more points collinear). There are algorithms for special cases in which the graph G is a tree [8, 18] or an outerplanar graph [7, 15].

For k-bend drawing without mapping, Kaufmann and Wiese [19] gave a 1-bend drawing algorithm for 4-connected plane graphs and a 2-bend drawing algorithm for general plane graphs; for general graphs, their algorithm takes time $O(n \log n)$ (resp. $O(n^2)$) to draw a point-set embedding with at most three (resp. two) bends per edge. In particular, their algorithm draws a 3-bend drawing in $O(n \log n)$ time and then spends $O(n^2)$ time, using rotation, to transfer the 3-bend drawing to a 2-bend drawing. In addition, they proved that deciding whether there is a mapping such that each edge has at most one bend is NP-complete. Giacomo *et al.* [14] presented an $O(n \log n)$-time algorithm which improves the previous $O(n^2)$-time algorithm by Kaufmann and Wiese [19] and guarantees that no rotation is needed to obtain a 2-bend drawing.

Kinetic algorithms can model real-world phenomena where objects move with predictable trajectories in the short term, but may be subject to unpredictable changes in trajectory in the long term [2]. Basch, Guibas and Hershberger [5] introduced the *kinetic data structure* (*KDS*) framework to handle such situations. Using this framework, one can maintain a *target attribute* of a graph on a set of moving points, where the trajectory of each point $p_i(t) = (x_i(t), y_i(t))$ is defined by two algebraic functions of fixed degree. Kinetic versions of *proximity graph* problems have been studied extensively in the past decade. The vertices of proximity graphs represent points and the edges represent geometric relations between points. Kinetic studies of such graphs include *Delaunay triangulations*, *Pie Delaunay graphs*, and *Euclidean minimum spanning trees* [1, 16, 21].

To our knowledge there are no previous results for point-set embeddability of a plane graph on a set of points moving along predictable trajectories. While we currently have no particular application in mind, we believe that kinetic graph drawing is of inherent and compelling interest. We investigate the problem as follows. We find a Hamiltonian cycle of the graph G and then map the Hamiltonian cycle to the set of points P. For each edge v_iv_j of G, we draw a curve of line segments such that the slopes of the line segments are defined based on the difference between the subscripts i and j and the maximum slope of the edges of the Hamiltonian cycle. This assignment prevents intersections between edges during the motion except at times when two points change the ordering of their x-coordinates.

Kinetic Framework. Given a point-set embedding of a plane graph $G(V, E)$, in order to maintain the embedding as the points move, we define a set of *certificates* verifying the correctness of the embedding and a *priority queue* containing the failure times of these certificates. When the failure time of a certificate is equal to the current time, we invoke an update mechanism that replaces the invalid certificate(s) with a new, valid one(s).

The set of all algorithms and data structures used for maintaining the point-set embedding is called a *kinetic data structure* (*KDS*). For example, suppose we want to

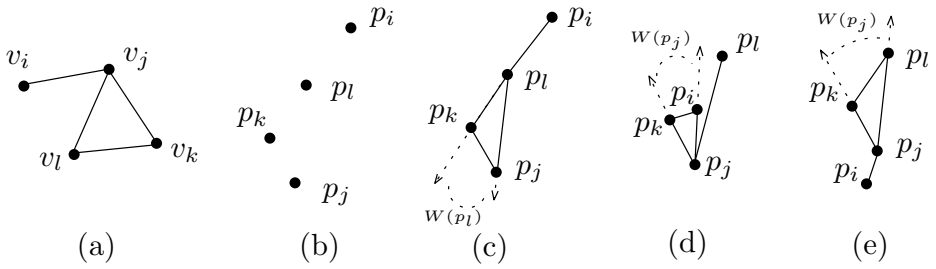

Fig. 1. (a) A plane graph $G(V, E)$ on vertices v_i, v_j, v_k, and v_l. (b) point set $P = \{p_i, p_j, p_k, p_l\}$. (c) A point-set embedding of G on P where v_i, v_j, v_k, and v_l map to p_i, p_l, p_j, and p_k, respectively. (d) New configuration after p_i moves inside the triangle $p_j p_k p_l$. (e) New configuration when $p_l p_i$ crosses the edge $p_j p_k$.

maintain the point-set embedding of the graph $G(V, E)$ in Figure 1.a on the set of points P in Figure 1.b, such that a point is outside the triangle created by the three other points; Figure 1.c depicts a drawing of G before the points move. During the motion, assume no three or more points are collinear in any positive interval of time. Let $p_j p_k p_l$ be a triangle and let $W(p_l)$ be a wedge whose sides are created by removing the edge $p_j p_k$ and extending the two line segments $p_l p_j$ and $p_l p_k$. As shown in Figure 1.c, the wedge $W(p_l)$ is the area between two half-lines $\overrightarrow{p_l p_j}$ and $\overrightarrow{p_l p_k}$. To maintain a valid embedding of the plane graph G as the points move in Figure 1.c, we create a certificate certifying that p_i is outside the wedge $W(p_l)$; the correctness of this certificate, over a time interval, certifies the correctness of the point-set embedding of G over that interval. Since we have the trajectory of the points, we can compute the (failure) time t when the point p_i moves inside the wedge $W(p_l)$. When the failure time of this certificate is equal to the current time t, then at time t^+, either point p_i moves inside the triangle $p_j p_k p_l$ or the edge $p_l p_i$ intersects $p_j p_k$; in both cases, the point-set embedding is no longer valid. Therefore, at the critical time t, we replace the previous certificate with a new one and update the mapping of G on P at time t^+. When point p_i crosses the half-line $\overrightarrow{p_l p_j}$ the new certificate should certify that the point p_l is outside the wedge $W(p_j)$ created by half-lines $\overrightarrow{p_j p_i}$ and $\overrightarrow{p_j p_k}$; the updated point-set embedding is the triangle $p_i p_j p_k$ and the edge $p_j p_l$ outside the triangle in the case that p_i moves inside the triangle $p_j p_k p_l$ (see Figure 1.d), or the triangle $p_j p_k p_l$ and the edge $p_i p_j$ outside the triangle otherwise (see Figure 1.e).

To measure the performance of a *KDS* there are four generally accepted criteria, called *efficiency*, *responsiveness*, *compactness*, and *locality*, which we now describe [5]. When a certificate fails, that doesn't necessarily imply that the attribute, *e.g.*, the validity of the point-set embedding, no longer holds. Events that don't change the target attribute are called *internal events*. Those events that produce changes in the target attribute are called *external events*; in Figure 1.c, the event is external. If the ratio between the number of the internal events and the number of the external events is polylogarithmic in the number of the points then the *KDS* is *efficient*. Typically, a *KDS* processes some internal events. Indeed our kinetic 3-bend drawing algorithm which we present in Section 3 processes internal events as well as external events.

If the processing time of an event is a polylogarithmic function of the number of points, the *KDS* is *responsive*. The *KDS* is *compact* if its size is equal to the number of points, within a polylogarithmic factor. Each point p_i has a trajectory, and the trajectory can change over time. When the trajectory changes, the certificates associated with the p_i must change. If the number of all certificates associated with p_i is a polylogarithmic function of the number of points, the *KDS* is *local*. The locality criterion anticipates that, while points move with predictable trajectories in the short term, they may be subject to unpredictable changes in the long term.

Our Results. We provide a kinetic algorithm maintaining a 3-bend drawing without mapping of a given plane graph $G(V, E)$ on a set of $n = |V|$ moving points, where the trajectory of each point is an algebraic function of constant maximum degree s. Our *KDS* uses $O(n)$ size, $O(n \log n)$ preprocessing time, and processes $O(n^2 \beta_{2s+2}(n) \log n)$ events, each in $O(\log^2 n)$ time. Here, $\beta_s(n) = \frac{\lambda_s(n)}{n}$ is an extremely slow-growing function of n and $\lambda_s(n)$ is the maximum length of Davenport-Schinzel sequences of order s on n symbols [20, 22]. In terms of the four standard *KDS* performance criteria, our *KDS* is *efficient*, *responsive*, *local*, and *compact*. In particular, in order to obtain a compact *KDS*, we provide a new $O(n \log n)$-time algorithm for point-set embedding of a given plane graph G on a stationary point set P. This algorithm first draws a 3-bend drawing in $O(n \log n)$ time and then, spends linear time to transfer the 3-bend drawing to a 2-bend drawing in a way that guarantees that no rotation is needed to obtain the 2-bend drawing. In Section 2, we introduce the main idea of constructing the 3-bend drawing and the 2-bend drawing without mapping and then, in Section 3, we maintain them kinetically.

2 k-Bend Drawing

In this section we first show an $O(n \log n)$ algorithm for point-set embedding of a given plane graph G on P with at most three bends per edge and then, given this drawing, we provide a 2-bend drawing in linear time.

A maximal planar subdivision with vertex set V is a triangulation, because no edge can be added without losing planarity. For any given plane graph $G(V, E)$, we add a set of edges E' to the graph G to make it maximally planar, and then we embed the graph $G(V, E \cup E')$ on the set of given points P and, finally, we remove the extra edges mapped from E'. The remaining drawing is the point-set embedding of the graph $G(V, E)$ on the set of points P. Therefore, throughout this paper we assume the given plane graph G is a triangulation.

2.1 3-Bend Drawing

We use a similar approach to that of Kaufmann and Wiese [19] to construct an initial drawing of the plane graph G on a set of stationary points P. Our algorithm draws a point-set embedding with at most three bends per edge that we later extend to the kinetic setting. The key insight from [19] is finding a *Hamiltonian cycle* that has at least one edge, called an *external edge*, on the outer face of G. A Hamiltonian cycle can be found in linear time [14, 19] by adding *dummy vertices*; below we explain the approach of [19].

Each 4-*connected* planar graph is Hamiltonian [23]. In fact, any maximal planar graph with at most two *separating triangles* is Hamiltonian [10, 17, 24]; a separating triangle is a triangle whose removal separates the graph into more components. We now review how to construct a 4-connected graph from any planar graph. Using the algorithm by Chiba and Nishizeki [12], the separating triangles can be found efficiently. Kaufmann and Wiese [19] destroy the separating triangles by adding dummy vertices and edges to create a 4-connected graph. Then, using the algorithm of Chiba and Nishizeki [11], a Hamiltonian cycle of the 4-connected graph can be found in linear time. Figure 2 depicts a graph with separating triangles. Adding two new vertices z_1 and z_2 creates a new graph (see Figure 3.a) having a Hamiltonian cycle (bold edges); z_1 is placed on the edge v_1v_6 and partitions it into two edges v_1z_1 and z_1v_6 and creates two new edges v_7z_1 and z_1v_8; the corresponding dummy point of the dummy vertex z_1, in Figure 3.b, is inserted in the middle of the segment p_7p_8. The new graph still has two separating triangles $v_1v_2v_4$ and $v_2v_4v_6$. For each dummy vertex z_k we add a dummy point to the given set of points $P = \{p_1, p_2, ..., p_n\}$. The dummy vertex is placed on an edge v_iv_j of the given graph G and partitions v_iv_j into two edges v_iz_k and z_kv_j. The corresponding dummy point is inserted in the middle of the segment p_ip_j. Let m be the number of dummy vertices, let $C = (u_1, u_2, ..., u_{n+m}, u_1)$ be the ordered vertices of the Hamiltonian cycle with external edge $e = u_1u_{n+m}$, and let chain $Q = \{q_1, q_2, ..., q_{n+m}\}$ be the list of P plus the dummy points sorted in increasing order by their x-coordinates. In particular, for two points $q_i = (x_i, y_i)$ and $q_j = (x_j, y_j)$, if $i < j$ then $x_i < x_j$; note that no two points have the same x-coordinate. Call the edges on the Hamiltonian cycle of the plane graph G *hull edges*, the edges inside the Hamiltonian cycle *interior edges*, and the edges outside of the Hamiltonian cycle *exterior edges*.

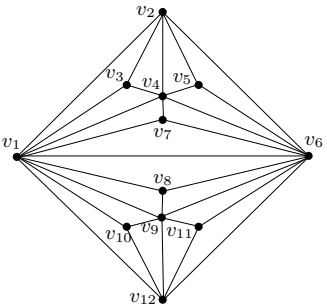

Fig. 2. A plane graph

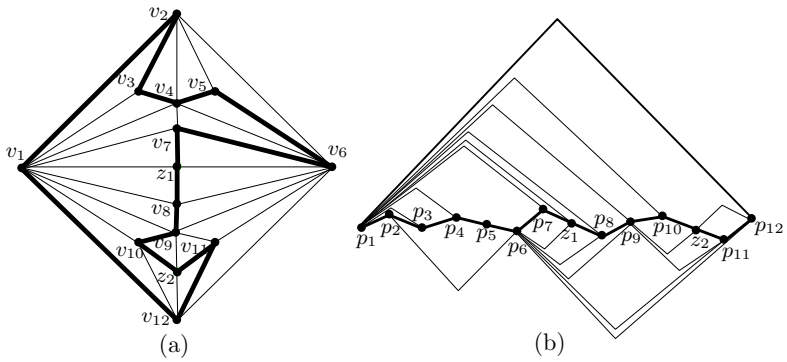

Fig. 3. (a) Adding two dummy vertices z_1 and z_2 and new edges creates a plane graph with a Hamiltonian cycle (bold edges). (b) A 3-bend drawing of G.

In order to support kinetic drawing, we assign different slopes to the edges than does the algorithm of Kaufmann and Wiese [19]; our slopes prevent intersections between interior edges during the motion of the points; see Section 3. Let δ be the maximum absolute slope of the edges of the chain Q. In particular, $\delta = \max_i |\frac{y_{i+1}-y_i}{x_{i+1}-x_i}|$ where $q_i = (x_i, y_i)$ and $q_{i+1} = (x_{i+1}, y_{i+1})$ are consecutive edges of the chain Q. To draw the point-set embedding we map the hull edge u_iu_{i+1} to the edge q_iq_{i+1}. For the external Hamiltonian edge u_1u_{n+m} and each interior edge u_iu_j, where $i < j$, we also draw an edge with one bend b_{ij} at the intersection of two lines, one through u_i with slope $(1 + \frac{j-i}{n+m})\delta$ and the other one through u_j with slope $-(1 + \frac{j-i}{n+m})\delta$; the mapping of the edge u_iu_j is $q_ib_{ij}q_j$ which has one bend at b_{ij}. The interior edges are drawn above the chain $q_1, q_2, ..., q_{n+m}$ and the exterior edges are drawn in a similar way below the chain.

Theorem 1. *The above point-set embedding of plane graph $G(V, E)$ onto the set of $n = |V|$ points P is crossing-free, has at most three bends per edge, and is constructed in $O(n \log n)$ time.*

Proof. After sorting the set of points P by x-coordinates in $O(n \log n)$ time, we map the hull edges to edges of the chain Q plus q_1q_{n+m}. This chain separates the interior edges and the exterior edges and it prevents intersections between these two types. In the following, we consider whether there are intersections among the interior edges; the proof that there is no intersection among the exterior edges is analogous. Let q_iq_j and q_kq_l be two interior edges; w.l.o.g., assume $i \leq k$. There are two possible situations: either $j \leq k$, in which case it is obvious that edge $q_ib_{ij}q_j$ doesn't cross edge $q_kb_{kl}q_l$ (Figure 3.b, see p_2p_6 and p_6p_8), or $j > k$, which implies that $j \geq l$ because the embedded plane graph G has no edge crossing (edge u_iu_j doesn't cross edge u_ku_l). In the second case ($i \leq k < l \leq j$), the slope of q_ib_{ij} (resp. $b_{ij}q_j$) is $(1+\frac{j-i}{n+m})\delta$ (resp. $-(1+\frac{j-i}{n+m})\delta$) which is sharper than the slope of q_kb_{kl} (resp. $b_{kl}q_l$) because $j - i > l - k$; therefore, b_{ij} is above b_{kl} which means that edge $q_ib_{ij}q_j$ doesn't cross edge $q_kb_{kl}q_l$ (Figure 3.b, see p_1p_3 and p_1p_4). By an argument similar to one in [19], it can be proved that each edge of the graph G is mapped to a chain of at most four line segments. □

2.2 2-Bend Drawing

Let q_k be a dummy point and let $q_ib_{ik}q_kb_{kj}q_j$ be a drawing of the edge u_iu_j of G that has three bends at b_{ij}, q_k, and b_{kj}. To reduce the number of bends to at most two bends per edge we replace the chain $b_{ik}q_kb_{kj}$ with a vertical line segment through q_k [19], saving a bend at q_k, see Figure 4. The new edge q_iq_k may cross other edges and destroy the planarity, see Figure 4.b, and so we need new assignments for the slopes of the edges in the point-set embedding. Let $r_{ij} = [i, j]$ denote the *range* of subscripts for the edge q_iq_j. The idea behind our 2-bend drawing algorithm is

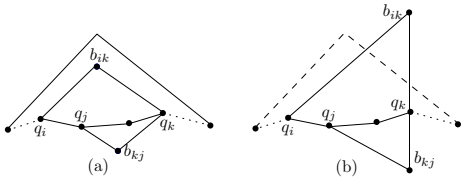

Fig. 4. Saving a bend at the dummy point q_k

to find the edges whose ranges contain the range r_{ij}, for all edges q_iq_j; if r_{kl} covers r_{ij} we assign slopes to segments of $q_kb_{kl}q_l$ so that they don't intersect the segments of $q_ib_{ij}q_j$. To store these nested layers of ranges we construct a *nested tree* $\mathcal{T}$ data structure. Each node n_{ij} of $\mathcal{T}$ corresponds to an edge q_iq_j and the subtree rooted at n_{ij} stores all ranges covered by r_{ij}. Figure 5.b shows the nested tree $\mathcal{T}$ for the graph in Figure 5.a.

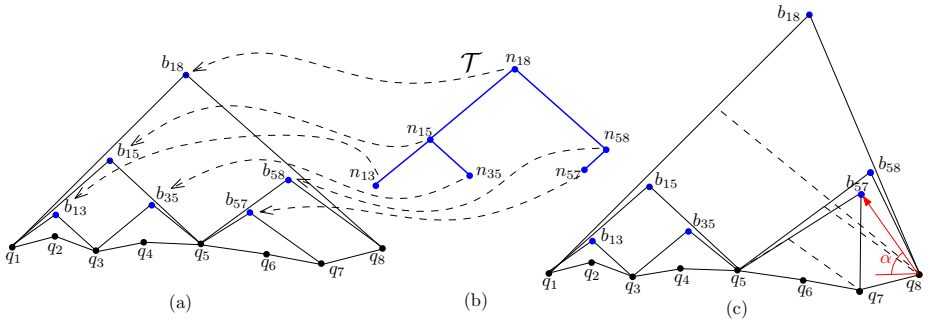

Fig. 5. (a) A 3-bend drawing. (b) The nested tree $\mathcal{T}$. (c) The 2-bend drawing.

Lemma 1. *Tree $\mathcal{T}$ can be built in $O(n)$ time from a 3-bend per edge drawing.*

Proof. Using a *stack* we construct the nested tree $\mathcal{T}$. For the external Hamiltonian edge q_1q_{n+m}, we push q_1 onto the stack, create a node n_{1n+m} as the root of $\mathcal{T}$ and a pointer pointing to this node. We process the endpoints of edges in order of increasing x-coordinate; if there are two or more edges incident to the same point, then we process these edges by decreasing order of their corresponding ranges. If we encounter the first endpoint of an edge q_iq_j, where $i < j$, we push the point q_i and insert into $\mathcal{T}$ a new rightmost child of the node to which the pointer points; after this the pointer must point to the newly created node. If we encounter the second endpoint q_j of q_iq_j, clearly, the top of the stack is the first endpoint q_i and we pop the point q_i and make the pointer point to the parent of the node n_{ij} (in Figure 5.a, after creating the root n_{18}, first we see the point q_1 of the edge q_1q_5, whose range r_{15} includes the range r_{13} and so we create the node n_{15} as the rightmost child of n_{18}; the pointer then points to n_{15}. After encountering the point q_1 of the edge q_1q_3 and creating the node n_{13}, the pointer points to the node n_{13}. Here, there are three q_1's in the stack corresponding to the three edges q_1q_8, q_1q_5, and q_1q_3. When we encounter q_3 we pop its corresponding q_1 and make the pointer point to n_{15}; continuing this process gives the nested tree $\mathcal{T}$ in Figure 5.b). The running time is clear. □

By traversing the nested tree $\mathcal{T}$ from the leaves to the root we show how to draw a 2-bend drawing. For each node n_{ij} of $\mathcal{T}$ corresponding to the edge q_iq_j we store two values δ_l and δ_r; the slopes of q_ib_{ij} and $b_{ij}q_j$ will be generated from δ_l and δ_r. Let n_{kl} be the parent of the node n_{ij} corresponding to the edge q_kq_l. If n_{ij} is a leaf and neither endpoint of q_iq_j is a dummy point we set $\delta_l = \delta_r = \delta$ and set the slopes of q_ib_{ij} and $b_{ij}q_j$ equal to $(1 + \frac{j-i}{n+m})\delta_l$ and $-(1 + \frac{j-i}{n+m})\delta_r$, respectively. If q_j (resp. q_i) is a dummy point we set $\delta_l = \delta$ (resp. $\delta_r = \delta$); then, b_{ij} is the intersection of the

vertical line through q_j (resp. q_i) and the line through q_i with slope $(1 + \frac{j-i}{n+m})\delta_l$ (resp. $-(1 + \frac{j-i}{n+m})\delta_r$). To assign slopes to the edges whose ranges cover the range r_{ij}, we find the slope α of the line through q_l (resp. q_k) and b_{ij} (see Figure 5.c) and set $\delta_r = |\alpha|$ (resp. $\delta_l = \alpha$). After assigning the slopes δ_l and δ_r to all leaves we can find the slopes of the edges corresponding to the internal nodes of $\mathcal{T}$. For each internal node n_{ij} we set δ_l (resp. δ_r) to be the maximum of the δ_l's (resp. δ_r's) of the children of the node n_{ij}; the slope of $q_i b_{ij}$ (resp. $b_{ij} q_j$) is $(1 + \frac{j-i}{n+m})\delta_l$ (resp. $-(1 + \frac{j-i}{n+m})\delta_r$). If one of the endpoints of the edge $q_i q_j$ corresponding to the internal node n_{ij} is a dummy point we handle n_{ij} as we did for a leaf.

The above process assigns slopes to the edges such that if the range r_{kl} covers the range r_{ij} then the slope of $q_k b_{kl}$ (resp. $b_{kl} q_l$) is sharper than the slope of $q_i b_{ij}$ (resp. $b_{ij} q_j$) except when one of the endpoints of $q_i q_j$, say p_j, is a dummy point; in this case as explained above, we compute the slope α and use it to assign a valid slope to $b_{kl} q_l$ so that it does not intersect $b_{ij} q_j$. Therefore, $q_i q_j$ doesn't cross $q_k q_l$. Lemma 1 together with the fact that traversing the nodes of the tree $\mathcal{T}$ takes linear time yields the following theorem.

Theorem 2. *Given a 3-bend drawing of the plane graph G on P, which can be constructed in $O(n \log n)$ time, a 2-bend drawing of G on P can be constructed in linear time.*

3 The Kinetic Drawing

Now, we kinetically maintain the drawing of Section 2. We give a *KDS* for maintaining the edges above the chain $q_1, q_2, ..., q_{n+m}$; the lower part can be maintained analogously.

3.1 Kinetic 3-Bend Drawing

In the 3-bend drawing of Section 2.1, each edge $q_i q_j$ above the chain is defined by two line segments $q_i b_{ij}$ and $b_{ij} q_j$ with positive slope $(1 + \frac{j-i}{m+n})\delta$ and negative slope $-(1 + \frac{j-i}{m+n})\delta$, respectively, where $\delta = \max_i |\frac{y_{i+1}-y_i}{x_{i+1}-x_i}|$. To maintain the point-set embedding over time, we maintain the maximum slope δ with a tool called the *kinetic tournament tree* [4]. We now describe it. Let $\widetilde{Q}$ be a set of moving objects such that each object has a time-varying value. The goal is to maintain over time the object with the maximum value. The kinetic tournament tree is a balanced binary tree that stores the objects at its leaves in an arbitrary order; each internal node stores the object with the maximum value of its two children. The root of the tree maintains the object in $\widetilde{Q}$ with the maximum value. For example, suppose we want to maintain the lowest point among a set of n moving points along the y-axis such that each point moves according to an algebraic function. We can use a kinetic tournament tree with the points at the leaves; the root of the tree maintains the lowest point during the time intervals between critical events where the ordering changes.

Here, we store the edges of the chain $q_1(t), q_2(t), ..., q_{n+m}(t)$ at the leaves of a kinetic tournament tree $\mathcal{KT}$. At each internal node of $\mathcal{KT}$, we store the *winner* edge of the two children, *i.e.*, the edge having the larger absolute slope. We also define a certificate certifying the winner is steeper than the other edge; the failure time of the certificate is the time when the other edge becomes the winner. This event is called a *tournament event*. When a tournament event at an internal node occurs, we apply the changes from the internal node to the root of the tournament tree, in $O(\log n)$ time, so that the edge at the root of the tree always has the steepest slope among all edges of the chain Q.

We maintain a list $L_{\mathcal{Q}}$ of the set of points Q sorted by increasing order of their x-coordinates. When the points move the order of the x-coordinates of two consecutive points may change. Let $q_{i'}$, q_i, q_j, and $q_{j'}$ be four consecutive points of $L_{\mathcal{Q}}$. For q_i and q_j, we maintain a certificate certifying that the x-coordinate of q_i is smaller than the x-coordinate of q_j; the failure time of the certificate is the time t when $x_i(t) = x_j(t)$ and the order is changed at time t^+. Call this an *order event*. Whenever an order event between q_i and q_j occurs, we delete two edges $q_{i'}q_i$ and $q_jq_{j'}$ from $\mathcal{KT}$ and add two new edges $q_{i'}q_j$ and $q_iq_{j'}$ into $\mathcal{KT}$ (see Figure 6). Thus, we need a dynamic version of the kinetic tournament tree $\mathcal{KT}$, called a *dynamic kinetic tournament tree* ($\mathcal{DKT}$) [3], which supports insertions and deletions. The following theorem gives the construction time for a $\mathcal{DKT}$ and bounds the total number of events in this tournament.

Theorem 3. *[3] A dynamic kinetic tournament tree $\mathcal{DKT}$, with a sequence of m insertions and deletions whose maximum size at any time is n (assuming $m \geq n$), generates at most $O(m\beta_{s+2}(n)\log n)$ events. Processing a tournament event takes $O(\log^2 n)$ time, and the $\mathcal{DKT}$ can be constructed in $O(n)$ time.*

Since the trajectory of each point is an algebraic function of constant degree at most s, the Euclidean length of each edge of the chain is an algebraic function of constant degree at most $2s$. Thus, the number of swaps between consecutive points in the sorted list $L_{\mathcal{Q}}$ is quadratic, and Theorem 3 implies the following.

Lemma 2. *$\mathcal{DKT}$ can be constructed in linear time; the number of tournament events is $O(n^2\beta_{2s+2}(n)\log n)$, and each event can be handled in $O(\log^2 n)$ time.*

In addition to the sorted list $L_{\mathcal{Q}}$ and $\mathcal{DKT}$, we need a priority queue, called an *event queue*, to maintain the failure times of the tournament events and the order events. Since the number of dummy vertices is $m = O(n)$, the sizes of the event queue and $\mathcal{DKT}$ are linear.

Lemma 3. *The proposed KDS (consisting of $L_{\mathcal{Q}}$, $\mathcal{DKT}$, and the event queue) uses linear space.*

Let $Inc(q_i)$ be the set of internal edges incident to q_i. Whenever the time of the next event in the queue is equal to the current time, we update the *KDS* and replace the failure certificates with new ones. When an order event between q_i and q_j occurs, edges in the set $Inc(q_i)$ (if non-empty) cross edges in the set $Inc(q_j)$ (if non-empty), see Figure 6.b. To remove the edge crossings and restore the embedding we allocate $Inc(q_i)$ to the point q_j and vice versa, see Figure 6.c. Then, we delete the order certificates corresponding to q_i and q_j from the event queue and replace them with new ones certifying

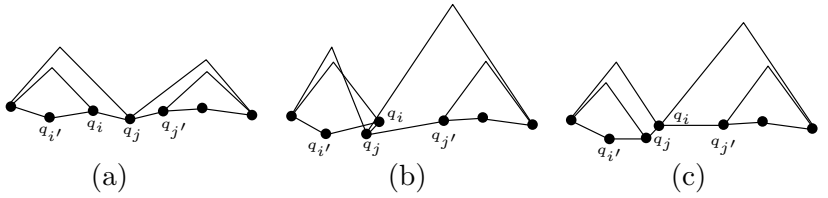

Fig. 6. (a) Before changing the ordering of q_i and q_j. (b) After points q_i and q_j change their order. (c) Allocating $Inc(q_i)$ to the point q_j and vice versa.

that the order of the x-coordinates of $q_{i'}$, q_j, q_i, and $q_{j'}$ is in increasing order; the failure times are the times when two consecutive points change their x-coordinate ordering.

Theorem 4. *Given an initial point-set embedding of a plane graph $G = (V, E)$ with at most three bends per edge on a set P of $n = |V|$ points, where the trajectory of each point is a known algebraic function of constant degree at most s, there is a kinetic data structure KDS that maintains the embedding and that satisfies the following properties. The KDS has linear size, and processes $O(n^2 \beta_{2s+2}(n) \log n)$ tournament events and $O(n^2)$ order events, each in $O(\log^2 n)$ time. The KDS is efficient, responsive, compact and local.*

Proof. The time required to apply a constant number of changes in the queue is $O(\log n)$. When an order event occurs a constant number of insertions and deletions is made in $\mathcal{DKT}$; each one takes $O(\log^2 n)$ time (see Lemma 2). The *compactness* and the *responsiveness*, which depend on the number of events and the time to process them, respectively, follow from Lemmas 2 and 3.

When a tournament event occurs, the winner stored at an internal node changes and the event may change the maximum slope δ. The only type of event that may cause edges of the drawing to cross is the order event. The ratio between the number of internal events (tournament events) and the number of external events (order events) is polylogarithmic in n, so the *KDS* is *efficient*. Each order event involves a constant number of other order events and $O(\log n)$ tournament events, so the number of all certificates associated with a particular point is polylogarithmic in n. Hence, the proposed *KDS* is *local*. □

3.2 Kinetic 2-Bended Drawing

Recall from Section 2.2 that the slopes of edge $q_i q_j$, for a 2-bend per edge drawing, arise from two values δ_l and δ_r stored at node n_{ij}. In order to maintain these values in the kinetic setting we define two dynamic kinetic tournament trees $\mathcal{DKT}_l$ and $\mathcal{DKT}_r$ whose nodes store δ_l's and δ_r's, respectively, of the children of node n_{ij}; the root of $\mathcal{DKT}_l$ (resp. $\mathcal{DKT}_r$) stores the larger value δ_l (resp. δ_r) of the children.

Let n_{kl} be the parent of node n_{ij} and let $\mathcal{P}_{kl}$ be the path from n_{kl} to the root of the nested tree $\mathcal{T}$. When the root of the tournament tree $\mathcal{DKT}_l$ or $\mathcal{DKT}_r$ changes, one of the children of n_{kl} is deleted and a new one is inserted; thus, an insertion and a deletion are done in the corresponding tournament tree of n_{kl}; this tournament tree may cause

an insertion and a deletion in the parent of n_{kl} and hence updates to all corresponding tournament trees on the path $\mathcal{P}_{kl}$. When two points q_i and q_j change their x-coordinate ordering, we update the values δ_l and δ_r of the nodes corresponding to these points, up to the root of $\mathcal{T}$. Using this process we obtain a compact *KDS* for 2-bend drawing.

4 Conclusion

We have introduced the investigation of graph drawing on moving points. In particular, we have introduced the kinetic data structure framework for maintaining a point-set embedding of a plane graph G on a set P of moving points, where each point moves according to an algebraic function of constant degree. We described an efficient kinetic algorithm for maintaining an embedding of G with at most three bends per edge. In terms of the standard evaluation criteria for a *KDS* framework, our *KDS* for maintaining a drawing for G with at most three bends per edge is efficient, responsive, local, and compact. We also gave a new $O(n \log n)$-time algorithm for point-set embedding with at most two bends per edge; we can kinetically maintain the 2-bend drawing but while this *KDS* is compact, it does not satisfy the other three performance criteria. Therefore, future directions for this research include finding a *KDS* for point-set embedding with at most two bends per edge that satisfies all four performance criteria and finding a *KDS* for straight-line drawings for some special graphs like outerplanar graphs and trees.

References

[1] Abam, M.A., Rahmati, Z., Zarei, A.: Kinetic Pie Delaunay Graph and Its Applications. In: Fomin, F.V., Kaski, P. (eds.) SWAT 2012. LNCS, vol. 7357, pp. 48–58. Springer, Heidelberg (2012)
[2] Agarwal, P.K., Eppstein, D., Guibas, L.J., Henzinger, M.R.: Parametric and kinetic minimum spanning trees. In: FOCS, pp. 596–605. IEEE Computer Society (1998)
[3] Agarwal, P.K., Kaplan, H., Sharir, M.: Kinetic and dynamic data structures for closest pair and all nearest neighbors. ACM Trans. Algorithms 5, 4:1–4:37 (2008)
[4] Basch, J.: Kinetic data structures. PhD Thesis, Stanford University (1999)
[5] Basch, J., Guibas, L.J., Hershberger, J.: Data structures for mobile data. In: Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1997, pp. 747–756. Society for Industrial and Applied Mathematics (1997)
[6] Biedl, T., Vatshelle, M.: The point-set embeddability problem for plane graphs. In: Proceedings of the 28th Symposuim on Computational Geometry, SoCG 2012, pp. 41–50. ACM, New York (2012)
[7] Bose, P.: On embedding an outer-planar graph in a point set. Comput. Geom. 23(3), 303–312 (2002)
[8] Bose, P., McAllister, M., Snoeyink, J.: Optimal algorithms to embed trees in a point set. J. Graph Algorithms Appl. 1 (1997)
[9] Cabello, S.: Planar embeddability of the vertices of a graph using a fixed point set is NP-hard. J. Graph Algorithms Appl. 10(2), 353–363 (2006)
[10] Chen, C.: Any maximal planar graph with only one separating triangle is hamiltonian. J. Comb. Optim. 7(1), 79–86 (2003)
[11] Chiba, N., Nishizeki, T.: The hamiltonian cycle problem is linear-time solvable for 4-connected planar graphs. J. Algorithms 10(2), 187–211 (1989)

[12] Chiba, N., Nishizeki, T.: Arboricity and subgraph listing algorithms. SIAM J. Comput. 14(1), 210–223 (1985)
[13] Durocher, S., Mondal, D.: On the Hardness of Point-Set Embeddability. In: Rahman, M. S., Nakano, S.-I. (eds.) WALCOM 2012. LNCS, vol. 7157, pp. 148–159. Springer, Heidelberg (2012)
[14] Giacomo, E.D., Didimo, W., Liotta, G., Wismath, S.K.: Curve-constrained drawings of planar graphs. Computational Geometry 30(1), 1–23 (2005)
[15] Gritzmann, P., Mohar, B., Pach, J., Pollack, R.: Embedding a planar triangulation with vertices at specified points. American Mathematical Monthly 98, 165–166 (1991)
[16] Guibas, L.J., Mitchell, J.S.B.: Voronoi Diagrams of Moving Points in the Plane. In: Schmidt, G., Berghammer, R. (eds.) WG 1991. LNCS, vol. 570, pp. 113–125. Springer, Heidelberg (1992)
[17] Helden, G.: Each maximal planar graph with exactly two separating triangles is hamiltonian. Discrete Applied Mathematics 155(14), 1833–1836 (2007)
[18] Ikebe, Y., Perles, M.A., Tamura, A., Tokunaga, S.: The rooted tree embedding problem into points in the plane. Discrete & Computational Geometry 11, 51–63 (1994)
[19] Kaufmann, M., Wiese, R.: Embedding vertices at points: Few bends suffice for planar graphs. J. Graph Algorithms Appl. 6(1), 115–129 (2002)
[20] Nivasch, G.: Improved bounds and new techniques for Davenport–Schinzel sequences and their generalizations. J. ACM 57, 17:1–17:44 (2010)
[21] Rahmati, Z., Zarei, A.: Kinetic Euclidean minimum spanning tree in the plane. Journal of Discrete Algorithms 16, 2–11 (2012)
[22] Sharir, M., Agarwal, P.K.: Davenport-Schinzel Sequences and their Geometric Applications. Cambridge University Press, New York (2010)
[23] Tutte, W.: A theorem on planar graphs. Trans. Amer. Math. Soc. 82, 99–116 (1956)
[24] Whitney, H.: A theorem on graphs. Ann. Math. 32, 378–390 (1931)