A Framework for Drawing Planar Graphs with Curves and Polylines

Michael T. Goodrich*
Department of Computer Science
The Johns Hopkins University
Baltimore, MD 21218
goodrich.cs.jhu.edu

Christopher G. Wagner†
Department of Mathematical Sciences
The Johns Hopkins University
Baltimore, MD 21218
wagner@brutus.mts.jhu.edu

Abstract

We describe a unified framework of aesthetic criteria and complexity measures for drawing planar graphs with polylines and curves. This framework includes several visual properties of such drawings, including aspect ratio, vertex resolution, edge length, edge separation, and edge curvature, as well as complexity measures such as vertex and edge representational complexity and the area of the drawing. In addition to this general framework, we present algorithms that operate within this framework. Specifically, we describe an algorithm for drawing any n-vertex planar graph in an $O(n) \times O(n)$ grid using polylines that have at most two bends per edge and asymptotically-optimal worst-case angular resolution. More significantly, we show how to adapt this algorithm to draw any n-vertex planar graph using cubic Bézier curves, with all vertices and control points placed within an $O(n) \times O(n)$ integer grid so that the curved edges achieve a curvilinear analogue of good angular resolution. All of our algorithms run in $O(n)$ time.

*Work by this author is supported in part by the U.S. Army Research Office under Grant DAAH04-96-1-0013 and by NSF under Grant CCR-9732300.

†Work by this author is supported in part by ONR grant N00014-96-1-0829.

1 Introduction

One of the main contributions of research in graph drawing has been the formal identification of aesthetic criteria that graph drawings can possess together with the trade-offs that exist between these criteria and various complexity measures of drawings. Such formal specifications allow us to quantify the qualities that make a drawing "nice" to look at and studies of various trade-offs further our understanding of which aesthetic goals can be realistically achieved. Examples of such studies are too numerous to enumerate, but some well-known examples include the importance of the drawing *area* identified by de Fraysseix, Pach, and Pollack [3], the illumination of the value of *bend minimization* in polyline drawings by Tamassia [13], the discussion of *aspect ratio* by Chan *et al.* [1], the identification of *angular resolution* as a signficant aesthetic criterion by Formann *et al.* [5], and its further study by Kant [10], Garg and Tamassia [7], and Malitz and Papakostas [11].

This paper is directed at continuing this tradition by articulating a unified framework for describing aesthetic criteria and complexity measures for drawings of graphs that place vertices at points in the plane and represent edges with smooth curves (that is, curves with at least $\mathcal{C}^1$ continuity) or polylines (that is, polygonal chains). We review or generalize several well-known graph-drawing design goals for this framework, including aspect ratio, vertex resolution, and bend minimization, as well as introduce design goals that are not as well-known, such as a curvilinear analogues to angular resolution and edge resolution. In addition to articulating this framework for curve and polyline drawings, we also describe new planar graph drawing algorithms that operate within this framework.

1.1 Related Prior Work

Before we describe our framework and our algorithms for this framework, however, let us review some related prior work. Specifically, since we reviewed above much of the prior work on the identification and study of general aesthetic criteria, let us review here some of the prior work relating to the drawing of planar graphs with curves and polylines.

One of the now-classic results for drawing planar graphs is an algorithm by de Fraysseix, Pach, and Pollack [3] for drawing an n-vertex planar graph in an $O(n) \times O(n)$ integer grid using straight line segments to represent edges. Originally presented as an $O(n \log n)$ time algorithm, Chrobak and Payne [2] show how to cleverly use a tree data structure to reduce the running time of this algorithm to $O(n)$. This algorithm does not achieve good angular resolution, however, because edges incident on the same vertex are not necessarily nicely "spread out." But this algorithm introduces an important concept known as the *canonical ordering* of the vertices of a plane graph, which is vital to several other graph drawing algorithms, including several algorithms by Kant [10]. In particular, Kant uses this ordering in an algorithm to draw a planar graph with vertices placed in an $O(n) \times O(n)$ integer grid and edges drawn as polygonal chains with at most three bends per edge in order to obtain good angular resolution. Forshadowing the topic of our paper, Kant observes that polyline drawings can be modified to give drawings with nicely curved edges, by, say, using splines.

Unfortunately, several issues related to curve drawings are not addressed by Kant. For example, if one applies standard methods for "smoothing" polylines, there may be no guarantee that edges that previously did not cross will not cross in their smoothed form. In

addition, there are several complexity issues related to polyline smoothing that should be addressed, such as deciding if a polyline with three bends is better represented using a single degree-4 curve or as two degree-3 curves joined at a point (which is called a *knot*). Equally important is the impact that such a decision has on the beauty of the drawing. Indeed, questions such as this are the driving force behind our search for a unified framework for characterizing graph drawings that use polylines and/or curves to represent edges.

On the topic of curve drawings themselves much less is known. There are several systems (including most general drawing packages) that allow for curves and polyline smoothing, but we have not been able to find much written on formal studies of curve drawings. A notable exception to this is work of Gansner, Koutsofios, North, and Vo [6] that addresses the issue of curve drawings for general graphs, considering aesthetic criteria such as the hierarchical structure of the underlying graph, the edge length, symmetry, and the number of edge crossings. Their analysis is not as formal as, say, the work of Kant [10] for polyline drawings, but Gansner *et al.* nevertheless provide evidence that their algorithm runs quickly and produces drawings that compare favorably with previous work. Unfortunately, the heuristics upon which their algorithm is based do not easily provide concrete statements about the properties of the drawings produced. For example, it seems that a drawing of a planar graph produced by this algorithm may have edge crossings. Thus, there is a need for algorithms for drawing graphs with curved edges so as to guarantee various aesthetic and complexity goals.

1.2 Our Results

In this paper we review and generalize several known aesthetic and complexity goals to a unified framework for drawing graphs with polyline and/or curve edges. Some of the novel aspects of this framework include a design goal we call *edge separation*, which is a curvilinear analogue to the concepts of edge resolution and angular resolution used for polyine drawings, as well as a concept we call *curvature minimization*, which provides a curvilinear analogue to the avoidance of sharp bends in polyline drawings.

In addition to the presentation of this framework, we provide algorithms that operate within it and for which we can make concrete performance claims. Specifically, we give an algorithm to draw an n-vertex planar graph in $O(n)$ time in an $O(n) \times O(n)$ integer grid using polyline edges that have at most two bends per edge and achieve an asymptotically-optimal worst-case angular resolution. Thus, we are able to achieve similar goals to a polyline drawing of Kant [10], but with only two bends per edge instead of three. More importantly, by limiting each polyline to two bends, we show how to extend our algorithm to draw an n-vertex planar graph in $O(n)$ time in an $O(n) \times O(n)$ grid so that each edge is drawn as a single cubic Bézier curve (i.e., with no knots) in a way that achieves good edge separation and has every vertex and control point placed at points with integer coordinates. Before we give the details of these algorithms, however, let us describe our unified framework for drawing graphs with polylines and curves.

2 A Framework For Drawing With Curves and Polylines

The goal of many graph drawing algorithms is to visually represent the essential properties of the graph in an intuitive and pleasing drawing.

2.1 Aesthetic Criteria

Based on this primary objective, we consider several formal aesthetic criteria for drawing graphs with edges represented as polylines and/or curves. Most of these criteria are not new, but are instead either direct translations of known aesthetic criteria to this combined curve/polyline framework or are generalizations of known polyline aesthetic criteria so that they also apply to curve drawings. In particular, we consider the following aesthetic criteria:

Aspect Ratio: If we consider the smallest rectangle that encloses a drawing, the aspect ratio is the ratio of the longer side of the rectangle to the shorter. In most cases, aspect ratios closer to 1 are more desirable.

Vertex Resolution: The vertex resolution is the minimum distance between any pair of vertices. Having good vertex resolution allows an observer to quickly distinguish vertices.

Edge Length: Edges should be as short as possible, so as to allow the eye to easily identify connected vertices.

Edge Separation: Edges should be separated so as to be easily distinguished. Our formalization of this notion is a combination of two commonly used graph drawing criteria—edge resolution and angular resolution—generalized to the curved setting.

We define an *offset region* for each edge e, which defines a region around e that should not contain any other edges or vertices. For each point p on e, the *offset* from p is defined as all points at distance at most $\delta_e(p)$ from p along a line perpendicular to e at p, where $\delta_e(p)$ is a function that depends on our drawing goals. (See Figure 1.)

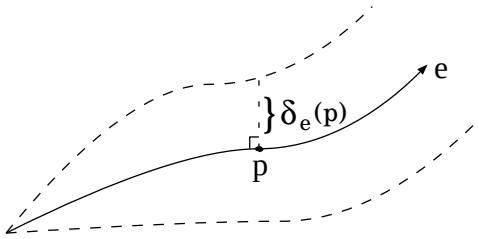

Figure 1: The offset region of an edge e.

For example, if we desire an angular resolution of at least $\theta > 0$ and edge resolution of at least $s > 0$, then we would define

$$\delta_e(p) = \min\{u \tan\theta, s\},$$

where u is the distance along e from the endpoint of e closer to p.

Thus, if e is a straight line segment, then the edge separation simply states that every edge nonincident to e is at least distance s away and every edge incident to e defines an angle of size at least θ. In this typical usage, we refer to θ as the *angular parameter* and s as the *separation parameter*.

Bend/Control-Point Minimization: For a polyline edge, we would like to have as few bends as possible, since many bends tend to make edges more difficult to follow. Analogously, as the bends in a polyline edge define the edge, so too control points define a spline or approximation curve. The more control points used to define a curve, the higher the degree of the curve or the larger the number of knots needed in a spline representation; hence, we wish to minimize the number of control points for curve edges.

Curvature Minimization: We would like edges to follow a relatively direct route between vertices, without sharp turns. The curvature of an edge is a measure of how quickly the edge bends. For polyline edges, this is simply the angle created at a bend. For curves, the curvature is measured as the rate of change of the angle of the tangent vector to the curve at any point. This, of course, can be computed directly, but a common approximation of curvature is to use the norm of the second derivative of the curve's defining equation.

These are not the only aesthetic criteria that one might wish to formalize, but they form a core set of common goals among wide classes of drawings.

2.2 Complexity Goals

In addition to such aesthetic criteria, there are also complexity issues for graph drawings, such as the following, which should also be considered:

Vertex Representational Complexity: Since graph drawing algorithms are typically implemented on computers, we should minimize the space needed to represent vertices. For example, many graph drawing algorithms impose the restriction that vertices be represented with integer coordinates, using $O(\log n)$ bits each. Such drawings are called *grid drawings*. Notice that this condition also forces vertex resolution to be at least unit distance.

Edge Representational Complexity: Similarly, we should consider the space needed to represent edges in a drawing. For example, in grid polyline or curve drawings we require that bend points and control points be represented with integer coordinates, using $O(\log n)$ bits each. In addition, by achieving good bend/control-point minimization, we are also reducing edge representational complexity.

Area: The area of a drawing is defined as the area of the smallest rectangle that encloses the drawing. We typically desire the area of a drawing to be small while not violating our aesthetic goals. Di Battista *et al.* [4] and Tamassia [14] summarize several known bounds on area-tradeoffs for various aesthetic properties for different types of drawings. For example, $\Theta(n^2)$ worst-case area is required for planar polyline grid drawings [2, 3, 10, 12].

Another interesting example of a tradeoff between the aesthetic criteria and the complexity measures is the angular resolution of the drawing. Clearly, the angular resolution of any graph drawing is at most $2\pi/d$, where d is the degree of the graph being drawn. But for a planar straight line grid drawing with $O(n^2)$ area, the angular resolution can be shown to be $O(\frac{1}{n^2})$ (e.g., see [14]). However, allowing the edges to be polylines, Kant [10] shows how to achieve an angular resolution of $\Theta(\frac{1}{d})$ in linear time with at most three bends per edge.

2.3 Characterizing Our Algorithms in this Framework

Operating within this unified framework, we give $O(n)$ time algorithms that build on the canonical-ordering approach of the algorithms of de Fraysseix, Pach, and Pollack [3] and Kant [10] to achieve the following results for grid drawings:

- each edge is a polygonal chain (cubic Bézier curve) with at most two bends (non-endpoint control points) per edge, such that each edge is monotonically increasing or decreasing in both x and y coordinates,
- vertices and bend (control) points are located at points in an $O(n) \times O(n)$ integer grid (which implies an $O(1)$ aspect ratio).
- the polyline drawing achieves an edge separation with separation parameter $s = 1$ and angular parameter θ that is $\Theta(1/d)$, where d is the degree of the graph; the curvilinear drawing algorithm achieves these bounds for its control curves.

We begin the presentation of the algorithm by first reviewing the canonical ordering of the vertices of a planar graph [3, 10].

3 Our Polyline Drawing Algorithm

Let G be an n-vertex plane graph of degree d. In addition, without loss of generality, we assume that G is maximal, since we can triangulate the faces of G without increasing the degree of any vertex in G by more than a constant factor. Recall that a maximal planar graph is a planar graph such that the addition of another edge destroys the planarity of the graph. In particular, consider any face of G that is not a 3-cycle, and choose an initial vertex p. Using p as a reference point, we have an upper path and a lower path leading out of p. We can begin with one of p's neighbors, say its lower neighbor, and add an edge from p's lower neighbor to its upper neighbor. We then continue from the upper neighbor to the next vertex on the lower path, and so on. This new path through the face triangulates the face, increasing the degree of each vertex by at most 2. (See Figure 2.) But each vertex is on $\Theta(d(v))$ faces, where $d(v)$ denotes the degree of vertex v, hence $d_{G'}(v) = \Theta(d_G(v))$. Thus, by obtaining an angular resolution of $O(\frac{1}{d})$ in G', we also obtain angular resolution of $O(\frac{1}{d})$ in G.

3.1 The Canonical Ordering

The following *canonical ordering* of the vertices of G is due to de Fraysseix, Pach, and Pollack [3] (and extendeded by Kant [10]). Let a, b, and c be the external vertices of G.

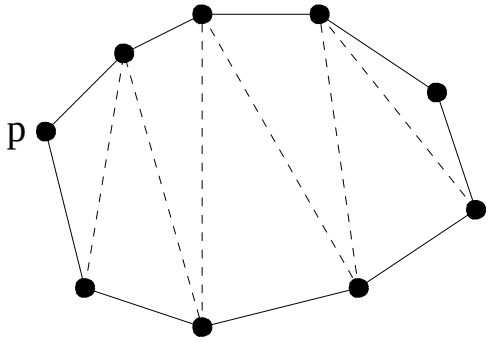

Figure 2: Triangulating a face of G.

A canonical ordering of the vertices of G (with respect to this embedding) is a labeling $v_1 = a, v_2 = b, v_3, \ldots, v_n = c$ such that:

- the graph G_k is biconnected, internally triconnected (the subgraph of G_k induced on the interior vertices of G_k is triconnected), and C_k contains the edge (v_1, v_2);
- vertex v_{k+1} is on C_{k+1} and has at least two neighbors in G_k on the path $C_k - (v_1, v_2)$, all consecutive,

where G_k is the subgraph of G induced on vertices $v_1, v_2, \ldots, v_k$ and C_k is the exterior face of G_k. We will refer to the vertices of C_k in order, meaning their clockwise order on C_k beginning with v_1.

It is easy to construct a labeling of G that satisfies the above conditions and thus is a canonical ordering as follows: Let the exterior face of G be C_n, consisting of vertices v_1, v_2, and v_n. Suppose that $v_n, v_{n-1}, \ldots, v_{k+1}$ have been defined. Then, there is a vertex $w \in C_k$, $w \notin \{v_1, v_2\}$, not incident to any chord of C_k. Set $v_k = w$. Our algorithm which draws G will place $v_1, v_2, \ldots, v_n$ in turn, according to this canonical ordering.

3.2 Our Algorithmic Approach

Our algorithm is a blending of de Fraysseix, Pach, and Pollack's ideas [3] and the algorithm that achieves angular resolution with three bends per edge due to Kant [10]. The novel aspect of our algorithm is in the development and use of a drawing control structure for each vertex, which we call the *join box*.

Let $V(G)$ be the vertex set of graph G, and let $d(v)$ denote the degree of vertex v. For each $v \in V(G)$, take a square around v with corners $d(v) - 1$ units above, below, left, and right of v. This is the *join box* associated with v (see Figure 3). This construction is quite simple, yet it is the key to being able to reduce the number of bends per edge to two.

Notice that if v is located at a lattice point (integer grid point), then there are $4d(v) - 4$ lattice points located on v's join box, called *join points*. We label the join points on the northwest side of v's join box from left to right $p_{v,1}, \ldots, p_{v,d(v)-1}$, beginning with the point located $d(v) - 2$ units to the left and one unit above v (see Figure 3). We label the join points on the northeast side of v's join box from right to left $q_{v,1}, \ldots, q_{v,d(v)-1}$, beginning with the point located $d(v) - 2$ units to the right and one unit above v. We label the join

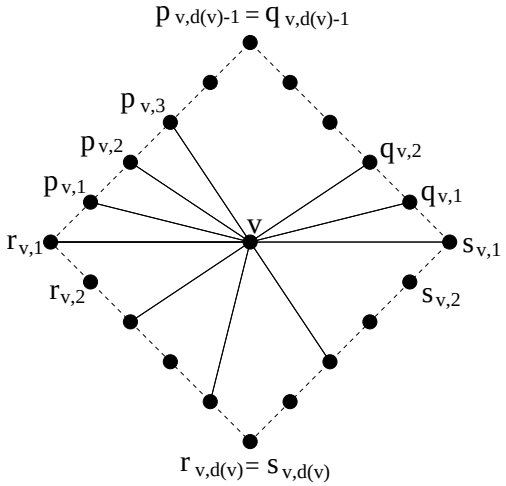

Figure 3: A vertex v's join box.

points on the southwest side of v's join box from left to right $r_{v,1}, \ldots, r_{v,d(v)}$, beginning with the point located $d(v) - 1$ units to the left of v. Finally, we label the join points on the southeast side of v's join box from right to left $s_{v,1}, \ldots, s_{v,d(v)}$, beginning with the point located $d(v) - 1$ units to the right of v.

The algorithm adds one vertex to the drawing of G during each phase so that the join points of the new vertex can connect to the join points of its neighbors via a straight line segment without any edge crossings. The join points will become the bends in the polyline edges of the graph, so that edge (u, v) will have two bends, one at a join point of u and the other at a join point of v.

The algorithm proceeds iteratively. At phase k of the algorithm, the k^{th} subgraph of G, G_k, has been embedded in the plane without edge crossings, while maintaining the following invariants:

1. The vertices and hence the join points of the k^{th} subgraph G_k are located at lattice points.

2. Let $w_1 = v_1, w_2, w_3, \ldots, w_m = v_2$ be the vertices of the exterior face C_k of G_k in order and let $x(w_i)$ be the x-coordinate of vertex w_i. Then $x(w_1) < x(w_2) < \ldots < x(w_m)$.

3. Let $[w_i, w_{i+1}]$ be an edge connecting join points of w_i and w_{i+1}. Then $[w_i, w_{i+1}]$ either has slope $+1$ or -1, $\forall i$, $1 \leq i < m$.

4. For each $v \in V(G_k)$, the join points $r_{v,1}$ and $s_{v,1}$ have been used and all of v's edges to its neighbors in G_k have been drawn. Also, the upper join points that have been used are consecutive beginning at $p_{v,1}$ and $q_{v,1}$ (as in Figure 3).

5. Each edge is monotonically increasing (decreasing) in both the x and y directions.

Notice that the contour of the exterior face C_k should appear as in Figure 4.

Now, consider when the algorithm is in phase $k + 1$, i.e., ready to construct G_{k+1} from G_k. First, note that as in the work of de Fraysseix, Pach, and Pollack [3], invariant (2)

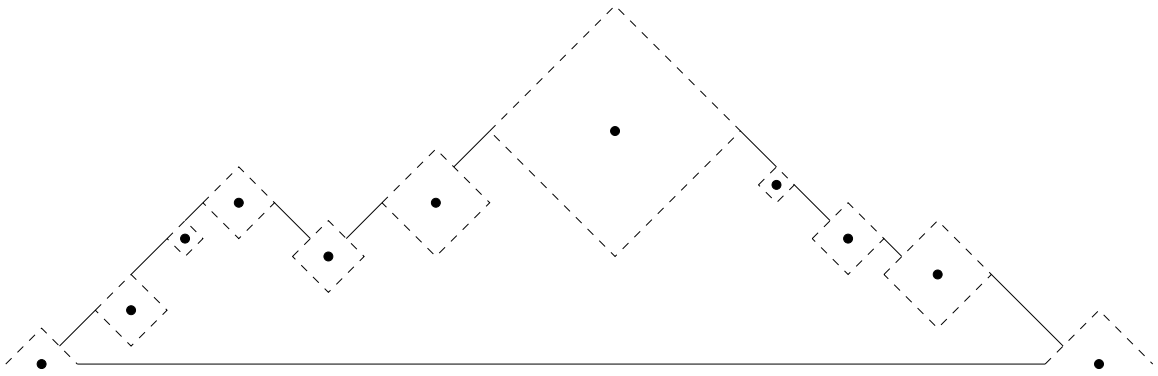

Figure 4: Appearance of C_k at any stage of the algorithm.

along with the fact that the sides of the join boxes have slopes of $+1$ and -1 imply that any two join points on the exterior face of G_k have an even Manhattan distance[1]. This property will allow us to initially place v_{k+1}. If we let $p(a,b)$ be the intersection of a line with slope $+1$ through point a and a line with slope -1 through point b, where $x(a) < x(b)$ then for any two join points, a and b, $p(a,b)$ is a lattice point. Also, notice that invariants (2) and (3) imply that the line segments from a to $p(a,b)$ and from b to $p(a,b)$ do not intersect G_k. So, we can add vertex v_{k+1} by finding its leftmost and rightmost neighbors on C_k, choosing suitable join points, a and b, of these neighbors, and initially placing the bottom of v's join box at $p(a,b)$. Unfortunately, this initial placement of v_{k+1} does not satisfy invariant (4). This issue is resolved by carefully shifting the vertices of the drawing of the k^{th} subgraph G_k to create space for v_{k+1}, and then inserting v_{k+1} along with edges to its neighbors on the outer face C_k, while maintaining all four invariants. Invariant (5) will be particularly useful when we replace the polyline edges by Bézier curves.

Let $w_1 = v_1, w_2, w_3, \ldots, w_m = v_2$ be the vertices of the outer face C_k of G_k in the order of their appearance from left to right, and let w_l and w_r be the leftmost and rightmost neighbors, respectively, of v_{k+1} on C_k. (Note that the neighbors of v_{k+1} in G_k form a contiguous sequence of vertices $w_l, w_{l+1}, \ldots, w_r$ on the exterior face C_k of G_k.) Also, let $q_{l,i}$ be the next unused join point on the right side of w_l's join box and let $p_{r,j}$ be the next unused join point on the left side of w_r's join box. We will initially place the bottom of v_{k+1}'s join box ($r_{k,d(k)} = s_{k,d(k)}$) at the lattice point $p(q_{l,i}, p_{r,j})$. Because the width of v_{k+1}'s join box is $2d(v_{k+1}) - 2$, the idea is to shift the part of G_k that is between w_l and w_r by $2d(v_{k+1}) - 2$ units to the right, along with v_{k+1}, and then shifting the part of G_k that is to the right of w_r an additional $2d(v_{k+1}) - 2$ units to the right. Having done this, we can now add the edges from the join points of v_{k+1} to join points of $w_l, w_{l+1}, \ldots, w_r$, and we say that v_{k+1} *covers* [2] $w_{l+1}, \ldots, w_{r-1}$ since they do not appear on the new outer face C_{k+1}. Shifting the subgraph G_k before inserting v_{k+1} allows us to maintain invariants (1)-(4), particularly invariants (3) and (4).

We can maintain invariant (5) by carefully selecting the join points of v_{k+1} to be used. We know that if v_{k+1} has only two neighbors in G_k, then we have monotonicity of the edges by property (3). If v_{k+1} has more than two neighbors, then there is a neighbor w_o with

[1]The Manhattan distance is the standard l_1 distance. In this case, $d(u,v) = |x(u) - x(v)| + |y(u) - y(v)|$.

$l+1 \leq o \leq r+1$ and $x(w_o) \leq x(v_{k+1}) < x(w_{o+1})$. So, to maintain property (5), let the bottom of v_{k+1}'s join box, $(r_{k,d(k)} = s_{k,d(k)})$ be the second bend point in edge (w_o, v_{k+1}). The remainder of the join points on the southwest side of v_{k+1}'s join box will go to $w_{l+1}, \ldots, w_{o-1}$ and the join points on the southeast side of v_{k+1}'s join box will go to $w_{o+1}, \ldots, w_{r-1}$. This will guarantee that edges $(w_l, v_{k+1}), \ldots, (w_o, v_{k+1})$ are all monotonically increasing and edges $(w_{o+1}, v_{k+1}), \ldots, (w_r, v_{k+1})$ are all monotonically decreasing. (We need only make sure that v_{k+1}'s join points are assigned to its neighbors in such a way that there are no edge crossings. That is, if w_i and w_{i+1} are assigned join points a and b respectively, then $x(a) < x(b)$.) Also, notice that by shifting, since we will only increase the x coordinate of any vertex v, and thus any vertices to the right of v, we do not destroy the monotonicity of the edges.

Thus, the key to the algorithm is in determining the sets of vertices to be shifted so that edge crossings cannot result from this shifting. These issues are dealt with in the following section.

4 The Details

We now need to address the issue of shifting, that is, how do we know which vertices (and join boxes) get shifted, and which do not. So, following the idea of de Fraysseix, Pach, and Pollack [3], we assume that for each vertex w_i on the exterior face C_k of G_k, we have already defined a subset $M_k(w_i) \subseteq V(G_k)$, called a *shifting set*, such that:

(a) $w_j \in M_k(w_i) \Leftrightarrow j \geq i$

(b) $M_k(w_1) \supset M_k(w_2) \supset \ldots \supset M_k(w_m)$

(c) For any nonnegative numbers $\Delta_1, \Delta_2, \ldots \Delta_m$, if we sequentially translate all vertices in $M_k(w_i)$ by distance Δ_i to the right $(i = 1, 2, \ldots, m)$, then the embedding of G_k remains a planar embedding. [Note: all points in $M_k(w_i) \setminus M_k(w_{i+1})$ are translated by $\Delta_1 + \Delta_2 + \ldots + \Delta_i$.]

In our algorithm, these Δ_i are relative distances that accumulate to determine the global position of each vertex. This issue is dealt with in more detail in the implementation section of the paper.

For $k = 3$, the bottom point of v_3's join box, $r_{3,d(v_3)} = s_{3,d(v_3)}$, is initially placed at the lattice point $(2d(v_1) - 2, 2)$ when v_1 is located at $(d(v_1) - 1, 0)$ and v_2 is located at $(2d(v_1) + d(v_2) - 3, 0)$. We then shift v_2 and v_3 by $2d(v_3) - 2$ units to the right, and then we shift v_2 and additional $2d(v_3) - 2$ units to the right. So, the initial positions of vertices v_1, v_2, and v_3 are $(d(v_1) - 1, 0)$, $(2d(v_1) + d(v_2) + 4d(v_3) - 7, 0)$, and $(2d(v_1) + 2d(v_3) - 4, d(v_3) + 1)$, respectively. See Figure 5 below.

Note that in C_3, $w_1 = v_1$, $w_2 = v_3$, and $w_3 = v_2$. In this case, conditions (a)-(c) are met by letting $M_3(v_1) = \{v_1, v_2, v_3\}$, $M_3(v_2) = \{v_2\}$, and $M_3(v_3) = \{v_2, v_3\}$.

Now, for the inductive step, suppose that G_k satisfies conditions (a)-(c), and we are adding vertex v_{k+1} to the graph. Again, suppose that w_l and w_r are v_{k+1}'s leftmost and rightmost neighbors on C_k respectively, and let $q_{l,i}$ and $p_{r,j}$ be the next unused join points on w_l and w_r. As described above, we initially place the bottom of v_{k+1}'s join box at $p(q_{l,i}, p_{r,j})$.

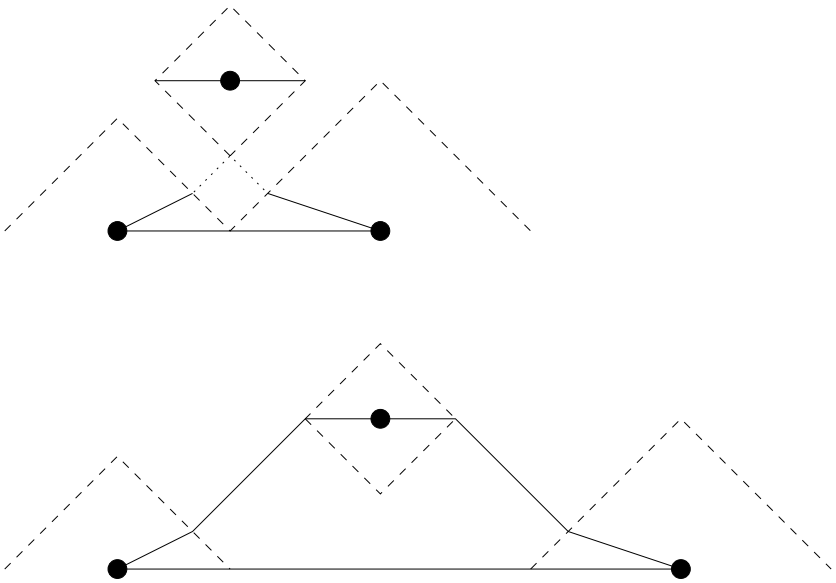

Figure 5: Initial and final layouts of G_3.

We now need to shift G_k so that v_{k+1} can be added to the graph while maintaining invariants (1)-(4). So, apply property (c) to G_k with $\Delta_{l+1} = 2d(v_{k+1}) - 2$, $\Delta_r = 2d(v_{k+1}) - 2$, and all other $\Delta_i = 0$. We then shift v_{k+1} by $2d(v_{k+1}) - 2$ units to the right and add edges from the join points of v_{k+1} to the join points of its neighbors on C_k. Notice that this specifically allow us to maintain invariants (3) and (4).

However, it first needs to be argued that v_{k+1} can "see" all of its neighbors on C_k. We say that a point can *see* another point if the line segment connecting the two points does not intersect the graph. Certainly invariants (3) and (4) imply that $r_{k,1}$ can see $q_{l,i}$ and $s_{k,1}$ can see $p_{r,j}$.

Lemma 1 *Each join point on the bottom of v_{k+1}'s join box can see each join point between $q_{l+1,d(l+1)-1}$ and $p_{r-1,d(r-1)-1}$ on the outer face C_k of G_k.*

Proof: Note that the shifting of v_{k+1} described above is the same as if we included it as a part of $M_k(w_{l+1})$ when it was initially placed at $p(q_{l,i}, p_{r,j})$. So, because the portion of C_k between v_{l+1} and v_{r-1} shifts rigidly, we need only argue that the join points of v_{k+1} can see their neighbors when v_{k+1} is initially placed. But this is true due a simple visibility argument, illustrated in Figure 6. Every point on the lower half of v's join box is located above both lines a (with slope -1) and b (with slope $+1$) which emanate from $q_{l+1,d(l+1)-1}$ and $p_{r-1,d(r-1)-1}$ respectively. ■

So, we have an embedding of G_{k+1}, where invariants (1)-(4) hold. We must now define the shifting sets for G_{k+1} and show that properties (a)-(c) are maintained.

The vertices of C_{k+1} are $w_1 = v_1, w_2, \ldots, w_l, v_{k+1}, w_r, \ldots, w_m = v_2$. For each member v of this sequence, we update the shifting sets to obtain $M_{k+1}(v) \subseteq V(G_{k+1})$, where:

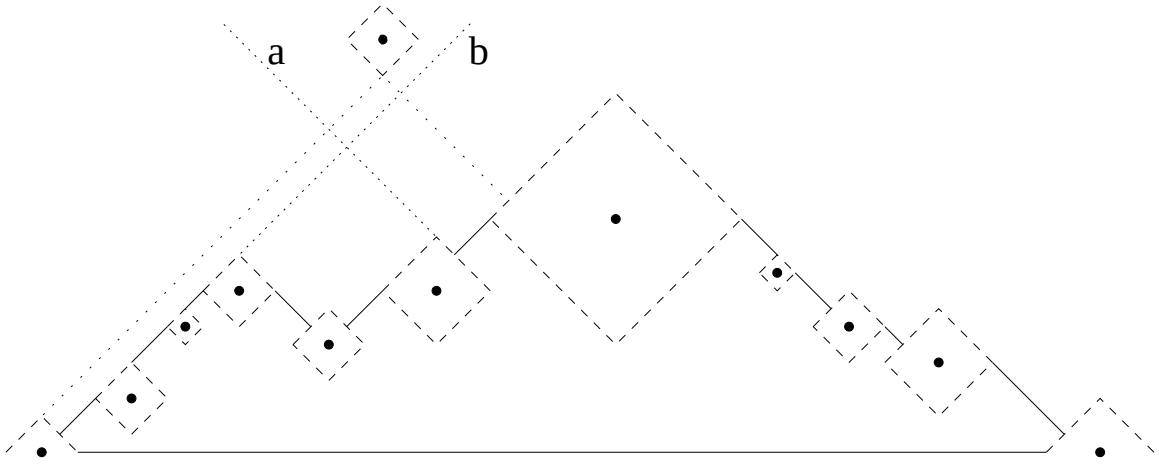

Figure 6: Join points of v_{k+1} are above both lines a and b.

$$\begin{aligned} M_{k+1}(w_i) &= M_k(w_i) \bigcup \{v_{k+1}\} && i \leq l \\ M_{k+1}(v_{k+1}) &= M_k(w_{l+1}) \bigcup \{v_{k+1}\} \\ M_{k+1}(w_j) &= M_k(w_j) && j \geq r \end{aligned}$$

Clearly, properties (a) and (b) hold. To verify property (c), fix a sequence of nonnegative integers $\Delta(w_1), \ldots, \Delta(w_l), \Delta(v_{k+1}), \Delta(w_r), \ldots, \Delta(w_m)$. For each v on C_{k+1}, sequentially shift the sets $M_{k+1}(v)$ by $\Delta(v)$ units to the right. (See Figure 7.)

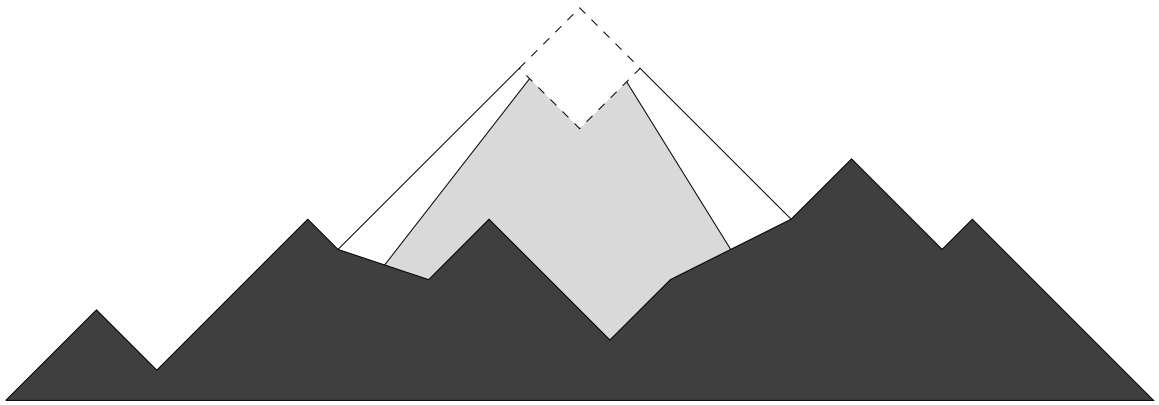

Figure 7: The graph after shifting.

Notice that after shifting, the darkly shaded area in Figure 7 is still a planar embedding by applying property (c) to G_k where $\Delta_1 = \Delta(w_1), \ldots, \Delta_l = \Delta(w_l), \Delta_{l+1} = \Delta(v_{k+1}) + 2d(v_{k+1}) - 2, \Delta_r = \Delta(w_r) + 2d(v_{k+1}) - 2, \Delta_{r+1} = \Delta(w_{r+1}), \ldots, \Delta_m = \Delta(w_m)$, and $\Delta_i = 0, \forall (l+2) \leq i \leq (r-1)$.

Also, note that the lightly shaded area has no edge crossings since it moves rigidly by a distance of $\Delta(w_1) + \Delta(w_2) + \ldots + \Delta(w_l) + \Delta(v_{k+1})$ units to the right.

Hence, after stage n of the algorithm, we have a planar embedding of $G_n = G$, satisfying invariants (1)-(4).

In fact, to get the desired edge separation, we can impose the additional condition that a join box does not intersect the convex hull of a nonincident edge.

In the base case, this condition holds, since, for example, v_1's join box will never intersect the convex hull of edge (v_2, v_3), because vertices are always shifted to the right. We can see that this condition still holds at the inductive step by noting that it holds for the darkly shaded region of Figure 7 by applying the inductive hypothesis to G_k and by noticing that the only new edges added are incident on v_{k+1} so that they all are above the darkly shaded region. Hence, v_{k+1}'s join box does not intersect any nonincident edges, and no other join boxes can intersect the convex hulls of the edges incident on v_{k+1}. Thus, the convex hulls of nonincident edges do not intersect, implying the nonincident edges themselves do not intersect, since we will be drawing with Bézier curves when we convert our embedding to a curve drawing.

In fact, because the vertices and bend points are located at integer coordinates and because the convex hull of each edge does not intersect any nonincident join box, the convex hulls of the edges have a separation of at least unit distance, implying that the edge separation of nonincident edges is at least unit distance.

It remains to analyze the algorithm.

4.1 Properties of the Embedding

First, it is clear that we have a planar polyline drawing of G, with at most two bends per edge. We now examine the angular resolution of the drawing of G and the size of the grid on which G is drawn.

Lemma 2 *The size of the minimum angle is $\frac{1}{d}$, where d is the maximum degree of any vertex of G.*

Proof: Let $v \in V(G)$ have degree d. Then, the minimum angle, Θ, occurs between a horizontal join segment, and a neighboring join segment. For $d \leq 5$, the lemma is easily verified. So, assume $d \geq 6$. The size of the angle is $\Theta = \arctan(\frac{1}{d-2})$. By the MacLaurin series expansion of arctan, we know that for $|x| < 1$, $\arctan(x) = x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \ldots \geq x - \frac{1}{3}x^3$. Since $\frac{1}{d-2} < 1$, we have $\Theta \geq \frac{1}{d-2} - \frac{1}{3}(\frac{1}{d-2})^3 > \frac{1}{d-2} > \frac{1}{d}$, completing the proof. ■

Lemma 3 *The size of the grid is at most* $(20n - 48) \times (10n - 24)$.

Proof: Since G is drawn entirely within the convex hull of vertices $\{v_1, v_2, v_3\}$, to find the length of the grid, we need the distance between $r_{1,1}$ and $s_{2,1}$, and to find the height, we need only find $p(r_{1,1}, s_{2,1})$. We know that v_1 never moves, so the length is simply the distance that v_2 is shifted to the right, which is bounded by:

$$\begin{aligned} \textstyle\sum_{i=1}^{n}(4d(v_i) - 4) &= 4(\textstyle\sum_{i=1}^{n} d(v_i) - \sum_{i=1}^{n} 1) \\ &= 4(6n - 12 - n) \\ &= 4(5n - 12) \\ &= 20n - 48 \end{aligned}$$

But this implies that the height of the grid is $10n - 24$, and the proof is complete. ■

4.2 Implementation Details and Running Time

Chrobak and Payne [2] give a linear time implementation of the algorithm of de Fraysseix, Pach, and Pollack [3] (which originally was published as an $O(n \log n)$ time algorithm) that is easily extended to our algorithm.

For completeness, let us give the details here. We may assume that G is a maximal planar graph that is embedded in the plane, and that we have a canonical ordering of the vertices of G. This is because we can find a planar embedding of G in linear time, for example, see the work of Hopcroft and Tarjan [8], and once we have a planar embedding of G, Kant [9] showed that it can be triangulated in $O(n)$ time. Finally, we can construct a canonical ordering in linear time (see [3, 10]).

We will view G_k as a forest consisting of trees, $T(w_1), \ldots, T(w_m)$, each rooted at a member of $C_k = \{w_1, \ldots, w_m\}$. The children of a node in $T(w_i)$ will be the vertices that w_i covers. Recall that if $w_l, w_{l+1}, \ldots, w_r$ are the neighbors of v_{k+1} in G_k, we say that v_{k+1} covers $w_{l+1}, \ldots, w_{r-1}$, since they were on the outer face C_k of G_k, but are not on the outer face C_{k+1} of G_{k+1}. But, as in the work of Chrobak and Payne [2], we will view this forest as a binary tree, T_k, associated with G_k and rooted at v_1, where each node in the tree represents a vertex v of G_k. The left child of v is the leftmost vertex that v covers, that is, the left child of v corresponds to w_{l+1} at the phase in the algorithm when v was added. The right child of v is v's neighbor to the right on the exterior face C_i of the graph G_i at some iteration i corresponding to the last time that v was on the exterior face. That is, v_{i+1} covers v. So, for example, v_1's right child will always be the next vertex on the exterior face at any iteration of the algorithm because v_1 appears on the exterior face of the graph at every phase. (See Figure 8 for an example of such a tree, with the labels of the vertices corresponding to a canonical ordering.) Just as we have G_k at any phase of the algorithm, and $G_n = G$, so too, we have T_k, with $T_n = T$.

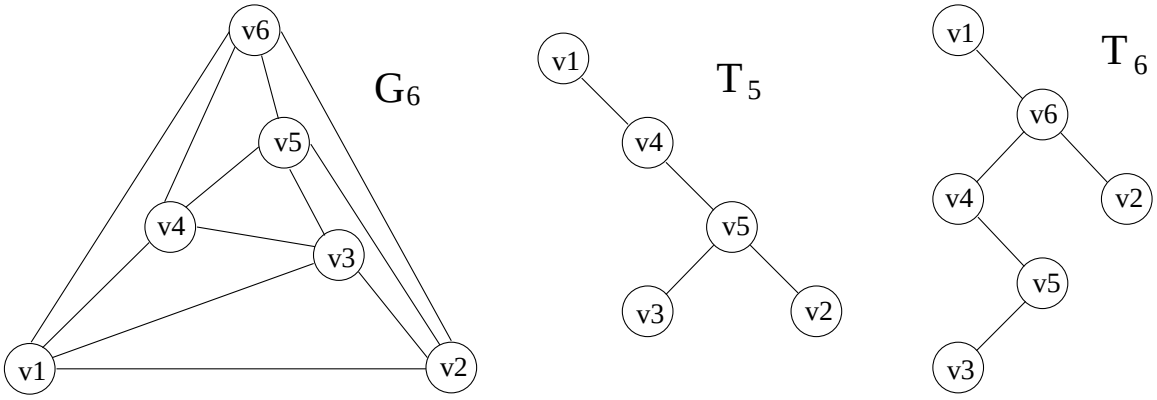

Figure 8: A graph G_6 and trees T_5 and T_6.

It is important to note a few interesting properties. First, C_k is the "right spine" of T_k, that is, C_k is v_1, v_1's right child, v_1's right child's right child, and so on. Also, the subtree of T_k rooted at w_i corresponds to the shifting set $M_k(w_i)$, and hence the left subtree of w_i corresponds to the set $M_k(w_i) \setminus M_k(w_{i+1})$.

The key to getting a linear implementation of the algorithm is to note that when adding

v_{k+1}, we do not need to know the exact positions of w_l and w_r, we only need to know their y-coordinates and their relative positions, ie $x(w_r) - x(w_l)$. In fact, what we really need is this information as relates to the join points $q_{l,i}$ and $p_{r,j}$, but this information is contained in the knowledge of the positions of the vertices themselves. From this, we can directly compute the y-coordinate of v_{k+1} and its position relative to w_l, that is $x(v_{k+1}) - x(w_l)$.

Chrobak and Payne [2] give the pseudocode to implement the straight line embedding algorithm of de Fraysseix, Pach, and Pollack. We modify this pseudocode to work for our algorithm.

For each vertex $v \neq v_1$, we define the *x-offset* of v as $\Delta x(v) = x(v) - x(w)$, where w is the parent of v in T (or T_k at the kth stage of the algorithm). More generally, if w is an ancestor of v, the x-offset between w and v is $\Delta x(w, v) = x(v) - x(w)$, exactly as in the implementation of Chrobak and Payne [2].

For each vertex $v \in T_k$ we store:

$$\begin{aligned}
\text{Left}(v) &= v\text{'s left child in } T \\
\text{Right}(v) &= v\text{'s right child in } T \\
\Delta x(v) &= \text{the } x\text{-offset of } v \text{ from its parent in } T \\
y(v) &= \text{the } y\text{-coordinate of } v \\
p(v) &= \text{the next unused upper left join point of } v \\
q(v) &= \text{the next unused upper right join point of } v
\end{aligned}$$

We need to maintain $p(v)$ and $q(v)$ so that we may determine the relative positions of new nodes as they are added to the graph. This is not necessary for de Fraysseix, Pach, and Pollack's algorithm, since they use straight line edges, whereas in our algorithm the join points determine the location of the vertex to be added.

Following the implementation of Chrobak and Payne [2], the algorithm consists of two phases. In the first phase, we add vertices one at a time, computing a vertex's x-offset, y-coordinate, and updating the offsets of one or two other vertices. The second phase consists of traversing T to compute the final x-coordinates of all the vertices.

We initialize v_1, v_2, and v_3 using the fact that the exterior face of G_3 is $C_3 = (v_1, v_3, v_2)$. We set the y-coordinates of each vertex to be their actual y-coordinates, as given in Section 4. Similarly, we use this initial placement to set the x-offsets of each vertex to be the difference in x-coordinates between the vertex and its predecessor on C_3, with the x-offset of v_1 set to $d(v_1) - 1$. To initialize the tree structure, set $\text{Right}(v_1) = v_3$, and $\text{Right}(v_3) = v_2$. All other pointers in the tree are set to *nil*. Finally, to initialize the next join points to be used, we note that the next available points are $p_{1,1}$ and $q_{1,2}$ for v_1, $p_{2,2}$ and $q_{2,1}$ for v_2, and $p_{3,1}$ and $q_{3,1}$ for v_3. This completes the initialization process.

We then repeat the following process for v_4 through v_n. Suppose that we are adding vertex v_{k+1}. Then, we must shift the sets $M_k(w_{l+1})$ and $M_k(w_r)$ by increasing both $\Delta x(w_{l+1})$ and $\Delta x(w_r)$ by $2d(v_{k+1}) - 2$. We next compute $\Delta x(w_l, w_r) := \Delta x(w_{l+1}) + \ldots + \Delta x(w_r)$. We can also directly compute $\Delta x(v_{k+1})$ and $y(v_{k+1})$ from the information at the nodes corresponding to w_l and w_r. We finally need to update $\Delta x(w_r) := \Delta x(w_l, w_r) - \Delta x(v_{k+1})$ and if $p + 1 \neq q$, then $\Delta x(w_{l+1}) := \Delta x(w_{l+1}) - \Delta x(v_{k+1})$.

It only remains to insert v_{k+1} into T_k. We do this by setting $\text{Right}(w_l) := v_{k+1}$, and $\text{Right}(v_{k+1}) := w_r$. Again, if $p + 1 \neq q$, then $\text{Left}(v_{k+1}) := w_{l+1}$ and $\text{Right}(w_{r-1}) := \text{nil}$. Otherwise, $\text{Left}(v_{k+1}) := \text{nil}$.

To compute the actual coordinates of the vertices, we traverse the tree, accumulating the x-offsets, exactly as in the *AccumulateOffsets* procedure in Chrobak and Payne's implementation [2]. The argument of the implementation's correctness also appears in the paper by Chrobak and Payne [2], and uses only the properties of T that were discussed earlier, namely that the subtree of T_k rooted at v is precisely $M_k(v)$.

Clearly, through each pass of the first phase of the implementation, the cost is bounded by $d(v)$, so summing over all vertices, the cost of the first phase is still $O(n)$. The second phase is merely a traversal of the tree, and hence only takes linear time. So, the overall algorithm is linear and we arrive at the following theorem:

Theorem 1 *There is a $O(n)$ time algorithm to draw a planar graph on a grid with the following properties:*

- *each edge is a polygonal chain with at most 2 bends per edge,*
- *each edge is monotonically increasing or decreasing in both its x and y coordinates,*
- *vertices and bend points are located at integer grid points,*
- *the size of the minimum angle created by two edges is at least $\frac{1}{d}$,*
- *the size of the grid is $O(n) \times O(n)$,*
- *there is at least unit distance between nonincident edges,*
- *and the drawing achieves edge separation with angular parameter $\theta = \frac{1}{d}$ and separation parameter $s = 1$.*

Figure 9 is a graph on 10 vertices drawn using our algorithm.

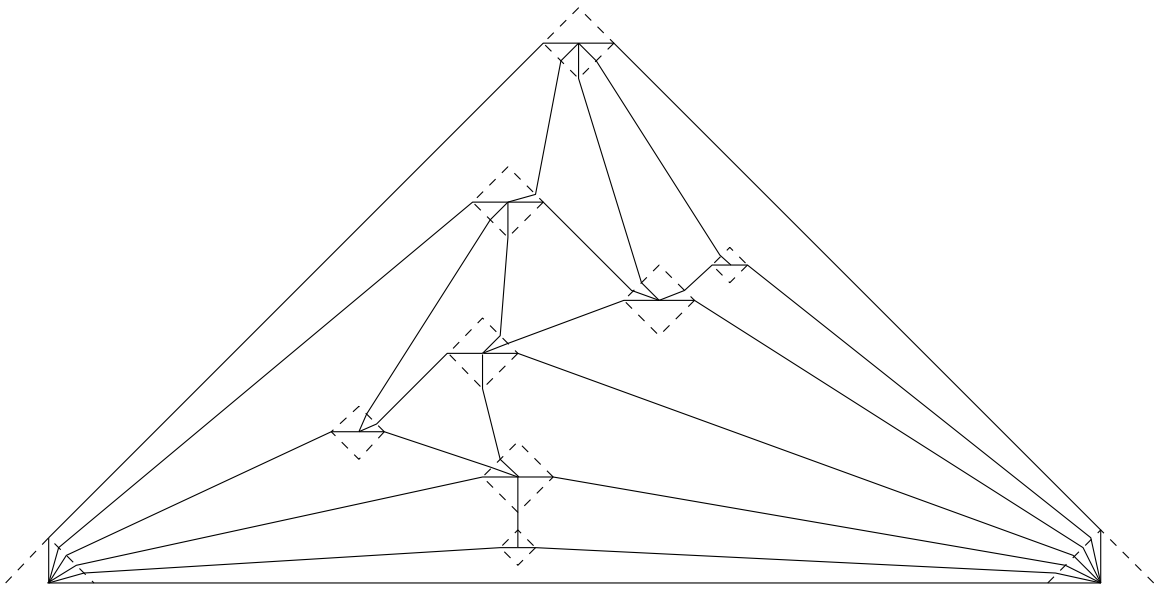

Figure 9: A graph on 10 vertices.

5 Drawing With Curves

We will now use some properties of the polyline grid drawing to show that the edges can be replaced by curves while still maintaining the desired aesthetic criteria. As mentioned, the type of approximating curve used in our algorithm is a cubic Bézier curve.

A cubic Bézier curve is defined by four control points V_0, V_1, V_2, and V_3. The curve interpolates V_0 and V_3, with the two segments (V_0, V_1) and (V_2, V_3) giving the tangents of the curve at the respective endpoints. The curve C is defined parametrically as $C = V_0(1-u)^3 + 3V_1u(1-u)^2 + 3V_2u^2(1-u) + V_3u^3$, where $u \in [0,1]$. A particularly useful property of Bézier curves is that the curve stays within the convex hull of the four control points.

We will argue that if we replace each polyline edge by a Bézier curve, where the control points of the curve are the vertices and bend points of the polyline edge, we do not introduce any edge crossings. In fact, we also are able to maintain a degree of edge separation (angular and edge resolution), as defined in the framework.

We deal with two cases, the case of nonincident edges and the case of incident edges. We have already argued that a join box does not intersect the convex hull of a nonincident edge, implying that we have unit separation of nonincident edges.

We now address the case of incident edges. Since the tangents of each curve simply correspond to the first segments of the polyline edges, we have angular resolution of $\frac{1}{d}$ in the planar curve grid drawing. Before arguing that incident edges do not cross, recall that all edges are monotonically increasing or decreasing. Consider edge e. Without loss of generality, the control points of e are $(0,0), (x_1, y_1), (x_2, y_2)$, and (x_3, y_3), with $0 \leq x_1 \leq x_2 \leq x_3$ and $0 \leq y_1 \leq y_2 \leq y_3$. Consider another edge incident at $(0,0)$, call it f. We want to examine the separation between e and f. When measuring edge separation, we want to consider the minimum distance between any pair of incident edges. Clearly, this distance would be achieved by parallel edges, if they were allowed in the graph. So, we will consider the case of parallel edges and show that we have a measure of edge resolution for parallel edges, implying that we have that same edge separation for incident edges.

So, to make e and f as close as possible, let the control points of f be $(0,0), (x_1+1, y_1-1), (x_2+1, y_2-1)$, and (x_3, y_3), as in Figure 10.

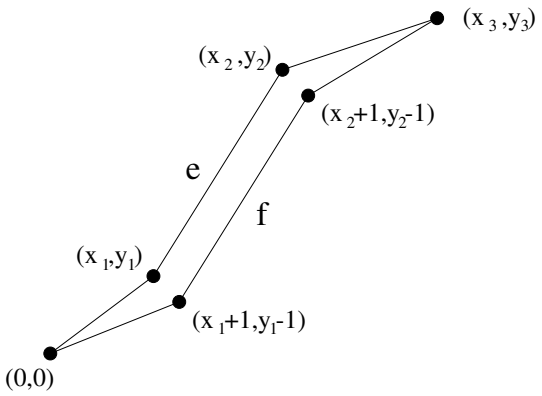

Figure 10: Parallel edges e and f.

The x and y coordinates of edge e are given by $x_e = 3u(1-u)^2x_1 + 3u^2(1-u)x_2 + u^3x_3$ and $y_e = 3u(1-u)^2y_1 + 3u^2(1-u)y_2 + u^3y_3$, respectively. Similarly, the x and y coordinates of edge f are given by $x_f = 3u(1-u)^2(x_1+1) + 3u^2(1-u)(x_2+1) + u^3x_3$ and $y_f = 3u(1-u)^2(y_1-1) + 3u^2(1-u)(y_2-1) + u^3y_3$, respectively. Taking differences, we have $x_f - x_e = 3u(1-u)$ and $y_f - y_e = -3u(1-u)$. This means that for any $u^* \in [0,1]$, we have a point $e(u^*)$ on e, and the corresponding point on f, $f(u^*)$, is $3u(1-u)$ units below and to the right of $e(u^*)$. For convenience, this is how we opt to initially measure the edge resolution of incident edges; ie, the distance between edges along a line of slope ± 1. So, the minimum distance between incident edges is at most $3u(1-u)/\sqrt{2}$. (We deal with monotonically increasing edges here, but the argument for decreasing edges is identical.)

This fact, coupled with the monotonicity of the polyline edges implies that converting the polyline edges to cubic Bézier curves does not introduce edge crossings. To see this, suppose that e and f cross. Then there is a point on f that is above some point on e. Call these points f^* and e^*, respectively. Then $x(f^*) = x(e^*)$ and $y(f^*) > y(e^*)$. Let $\hat{u}$ be the parameter value that gives this point f^*. Then, $e(\hat{u})$ is above and to the right of f^*. But if this is the case, then e must come back down from $e(\hat{u})$ to pass through the point e^*, contradicting the monotonicity of the edges.

See Figure 11 for an example of a curve drawing.

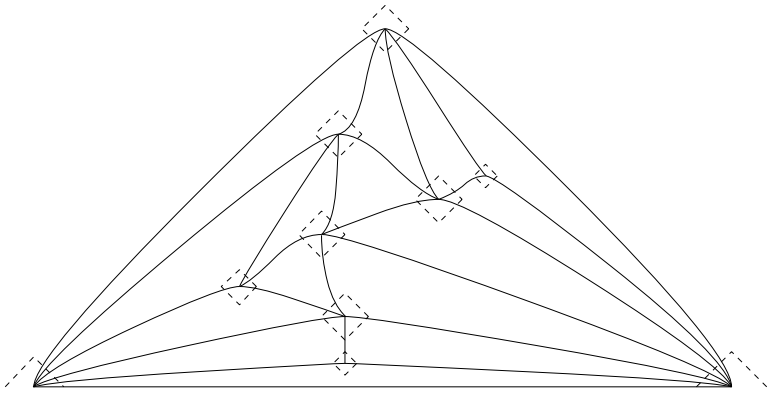

Figure 11: The graph from Figure 9 drawn with curve edges.

This gives the following theorem:

Theorem 2 *There is a $O(n)$ time algorithm to draw a planar graph on a grid with the following properties:*

- *each edge is a cubic Bézier curve,*
- *each edge is monotonically increasing or decreasing in both its x and y coordinates,*
- *vertices and control points are located at integer grid points,*
- *the size of the minimum angle created by two incident edges is at least $\frac{1}{d}$, as measured by the tangents of the curves,*
- *the size of the grid is $O(n) \times O(n)$,*

- *there is at least unit distance between nonincident edges,*
- *and the drawing has angular parameter* $\theta = \frac{1}{d}$ *and separation parameter* $s = 1$ *for the control curves.*

6 Comments And Open Problems

Our algorithm can easily be extended to triconnected graphs, just as Kant [10] extends the algorithm of de Fraysseix, Pach, and Pollack. When we initially place a chain of vertices, we simply locate the initial point based on the vertex of maximum degree in the chain. Then, we shift by the sum of the degrees of the vertices in the chain.

Having established a framework for drawing graphs with curves, there are any number of aesthetic criteria that can be redefined for curve drawings, leading to some directions of further research:

- Devise new algorithms to draw graphs with curves, operating under this general framework.
- Devise a dynamic algorithm for drawing graphs with curve edges.
- If one of the control points of the curve edges is far from the remaining three, the curvature of the edge can be quite large, in fact proportional to n. One may be able to devise an algorithm that also minimizes the curvature of the edges.
- Because there are roughly $4d(v)$ join points on v's join box and we only use $d(v)$ of them, we waste a great deal of space. There may be a way of cleverly reducing the sizes of the join boxes and thus the constants involved in the area of the drawing.
- Since we use the upper join points in order, often the join points near the top of the join box are not used. There may be some heuristics that allow for a better "spacing" of the upper join points once the final layout is constructed.

References

[1] T. M. Chan, M. T. Goodrich, S. R. Kosaraju, and R. Tamassia. Optimizing area and aspect ratio in straight-line orthogonal tree drawings. In S. North, editor, *Graph Drawing (Proc. GD '96)*, volume 1190 of *Lecture Notes Comput. Sci.*, pages 63–75. Springer-Verlag, 1997.

[2] M. Chrobak and T. Payne. A linear-time algorithm for drawing planar graphs. *Inform. Process. Lett.*, 54:241–246, 1995.

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

[4] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Comput. Geom. Theory Appl.*, 4:235–282, 1994.

[5] M. Formann, T. Hagerup, J. Haralambides, M. Kaufmann, F. T. Leighton, A. Simvonis, E. Welzl, and G. Woeginger. Drawing graphs in the plane with high resolution. *SIAM J. Comput.*, 22:1035–1052, 1993.

[6] E. R. Gansner, E. Koutsofios, S. C. North, and K. P. Vo. A technique for drawing directed graphs. *IEEE Trans. Softw. Eng.*, 19:214–230, 1993.

[7] A. Garg and R. Tamassia. Planar drawings and angular resolution: Algorithms and bounds. In *Proc. 2nd Annu. European Sympos. Algorithms*, volume 855 of *Lecture Notes Comput. Sci.*, pages 12–23. Springer-Verlag, 1994.

[8] J. Hopcroft and R. E. Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974.

[9] G. Kant. *Algorithms for Drawing Planar Graphs.* PhD thesis, Dept. Comput. Sci., Univ. Utrecht, Utrecht, Netherlands, 1993.

[10] G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16:4–32, 1996. (special issue on Graph Drawing, edited by G. Di Battista and R. Tamassia).

[11] S. Malitz and A. Papakostas. On the angular resolution of planar graphs. *SIAM J. Discrete Math.*, 7:172–183, 1994.

[12] W. Schnyder. Embedding planar graphs on the grid. In *Proc. 1st ACM-SIAM Sympos. Discrete Algorithms*, pages 138–148, 1990.

[13] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987.

[14] R. Tamassia. Graph drawing. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 44, pages 815–832. CRC Press LLC, Boca Raton, FL, 1997.