Journal of Graph Algorithms and Applications
http://jgaa.info/
vol. 7, no. 4, pp. 311–333 (2003)

Polar Coordinate Drawing of Planar Graphs with Good Angular Resolution

Christian A. Duncan
Department of Computer Science
University of Miami
Coral Gables, FL 33124
http://www.cs.miami.edu/
duncan@cs.miami.edu

Stephen G. Kobourov
Department of Computer Science
University of Arizona
Tucson, AZ 85721
http://www.cs.arizona.edu/
kobourov@cs.arizona.edu

Abstract

We present a novel way to draw planar graphs with good angular resolution. We introduce the polar coordinate representation and describe a family of algorithms for constructing it. The main advantage of the polar representation is that it allows independent control over grid size and bend positions. We first describe a standard (Cartesian) representation algorithm, CRA, which we then modify to obtain a polar representation algorithm, PRA. In both algorithms we are concerned with the following drawing criteria: angular resolution, bends per edge, vertex resolution, bend-point resolution, edge separation, and drawing area.

The CRA algorithm achieves 1 bend per edge, unit vertex and bend resolution, $\sqrt{2}/2$ edge separation, $5n \times \frac{5n}{2}$ drawing area and $\frac{1}{2d(v)}$ angular resolution, where $d(v)$ is the degree of vertex v. The PRA algorithm has an improved angular resolution of $\frac{\pi}{4d(v)}$, 1 bend per edge, and unit vertex resolution. For the PRA algorithm, the bend-point resolution and edge separation are parameters that can be modified to achieve different types of drawings and drawing areas. In particular, for the same parameters as the CRA algorithm (unit bend-point resolution and $\sqrt{2}/2$ edge separation), the PRA algorithm creates a drawing of size $9n \times \frac{9n}{2}$.

Communicated by: P. Mutzel and M. Jünger;
submitted June 2002; revised December 2002.

The work by S. G. Kobourov was partially supported by NSF grant ACR-0222920.

1 Introduction

In the area of planar graph drawing there has been considerable interest in algorithms that produce readable drawings [3]. Among the many properties which contribute to the readability of planar graphs, edge smoothness, vertex resolution, bend-point resolution, angular resolution, and edge separation are of great importance. Edges are often drawn as straight-line segments connecting two vertices. An edge can also be drawn as a sequence of straight-line segments, in which case the smallest number of bends is desirable. An edge may also be drawn as a smooth curve. These three types of edges generally provide aesthetically pleasing drawings.

1.1 Definitions

A graph drawing has good *vertex resolution* if vertices cannot get arbitrarily close to one another, that is, if vertices are well distributed in the drawing. As a result, a great deal of research has been concentrated on graph drawing algorithms which place vertices on the integer grid such that the *drawing area* is proportional to the number of vertices n of the graph, typically $O(n) \times O(n)$. If there are bends in the edges, then the bend-points are also placed on the integer grid. The *bend-point resolution* of a graph refers to the minimum distance between two bends. The *edge separation* of a graph refers to the minimum distance between two edges that are sufficiently away from their endpoints (since incident edges can get arbitrarily close to each other near their common endpoint).

A graph drawing has good *angular resolution* if adjacent edges cannot form arbitrarily small angles. This is achieved by ensuring that the edges emanating from a given vertex "fan out" evenly around the vertex. Note, however, that good angular resolution cannot always be achieved while simultaneously guaranteeing straight-line edges and small sub-exponential drawing area [10]. By introducing bends in the edges, however, we can guarantee both good resolution and small drawing area.

1.2 Previous Work

Garg and Tamassia [6] consider the problem of drawing with good angular resolution, and Kant [9] shows how to create drawings with angular resolution of $\Theta(1/d(v))$ in an $O(n) \times O(n)$ area grid, using edges with at most three bends each. Gutwenger and Mutzel [8] describe an improved algorithm with better constant factors which produces very aesthetically pleasing drawings in a $(2n-5) \times (3n/2 - 7/2)$ grid with at least $2/d(v)$ angular resolution using at most three bends per edge. The algorithm of Goodrich and Wagner [7] requires one less bend per edge and guarantees angular resolution of $\Theta(1/d(v))$ for each vertex v, but at the expense of larger area, $(20n-48) \times (10n-24)$. Cheng, Duncan, Goodrich, and Kobourov [1] improve the above algorithm so that every edge has at most one bend while the angular resolution is $\Theta(1/d(v))$ for each vertex v and maximum area is $30n \times 15n$.

1.3 Our Results

We first present a new Cartesian representation algorithm (CRA) which improves the bounds of previous algorithms. In particular, CRA guarantees 1 bend per edge, unit vertex resolution, unit bend-point resolution, $\sqrt{2}/2$ edge separation, $5n \times \frac{5n}{2}$ drawing area, and $\frac{1}{2d(v)}$ angular resolution, where $d(v)$ is the degree of vertex v.

We then present a novel polar representation algorithm (PRA). The PRA algorithm also guarantees $\frac{\pi}{4d(v)}$ angular resolution, 1 bend per edge, and unit vertex resolution. The bend-point resolution and edge separation are parameters that can be modified to achieve different types of drawings and drawing areas. In particular, for the same parameters as the CRA algorithm (unit bend-point resolution and $\sqrt{2}/2$ edge separation), the PRA algorithm creates a drawing of size $9n \times \frac{9n}{2}$. Note that in some situations the vertex resolution is more important than the bend-point resolution or the edge separation. In such situations, all of the previous algorithms perform poorly since they are designed to maintain constant resolution particularly between vertices and bend-points. Using the PRA algorithm, we can relax the bend-point resolution constraints and get significant improvements.

The PRA algorithm relies on a novel approach for representing bends and vertices. Traditionally, vertices and bend-points are restricted to lie on integer grid coordinates. One reason for this is that the points are defined by a pair of integers. In this way, all operations on the points (for example, shifting) are performed with integer arithmetic. At the drawing stage, the integer coordinates are mapped to pixels on the screen.

Another reason for placing vertices and bend-points on integer grid coordinates is that this approach guarantees good vertex resolution, good bend-point resolution, and good edge separation [1, 7, 8, 9]. Rather than insisting that bend-points lie on integer grid coordinates, we propose an alternative approach which allows bend-points to be located on a grid represented by polar coordinates. We call this a *polar representation* approach because both the vertices and the bend-points are represented using polar coordinates.

At the exact moment of drawing the graph onto the screen, an algorithm using polar representation requires a rounding calculation to determine the exact pixel location for the bend-points. Note, however, that the traditional approach also uses a rounding calculation for scaling from the integer grid space to the pixel space.

The main advantage of using a polar representation is that it allows us to independently control grid size and bend positions. Polar coordinates allow us to specify different vertex resolution, bend-point resolution, and edge separation. We achieve this added flexibility at the expense of slightly increased storage for the graph representation. A Cartesian representation requires exactly two integers for each point while the polar representation requires up to five integers per point.

Both of our algorithms assume that the graph is a fully triangulated, undirected, planar graph. If the graph were not fully triangulated, one can still

solve the problem by fully triangulating the graph, embedding this new graph, and then removing the inserted edges. Approached properly, this scheme incurs at most a constant factor decrease in angular resolution as the modified degree of a vertex can triple in size, e.g. fully-triangulating a path. As a result, for the remainder of this paper when we say "graph" we mean a fully triangulated, undirected, planar graph. We leave it as an open exercise to modify the algorithm to work more effectively for general undirected planar graphs.

In Section 2 and Section 3 we present the Cartesian Representation Algorithm (CRA) and argue its correctness. CRA is an improved version of the algorithm from Cheng *et al* [1] for drawing with good angular resolution. In Section 4 we introduce the concept of embedding graphs using a polar coordinate system and then present the Polar Representation Algorithm (PRA) which is a modification of the CRA.

2 The CRA Algorithm

The Cartesian Representation Algorithm is a natural extension of some previous algorithms that guarantee good angular resolution [9, 8, 7, 1]. In our algorithm the vertices of the graph are inserted sequentially by their canonical ordering, generating subgraphs $G_1, G_2, \ldots, G_n$. The canonical ordering [5] for a planar graph G orders the vertices of G so that they can be inserted one at a time without creating any crossings. We define G_k at step i to be the graph induced by vertices $1, 2, \ldots, k$. From our ordering, we shall see that G_1, G_2, and G_3 are basic graphs, a vertex, a line, and a triangle respectively. Graph G_{k+1} is created from G_k by inserting the next vertex v_{k+1} in the canonical order. Before we show the details of our algorithm we need several definitions. Following the notation of [5], let $w_1 = v_1, w_2, \ldots, w_m = v_2$ be the vertices of the exterior face C_k of graph G_k in order. For a particular subgraph G_k with $k > 2$ and vertex v_{k+1}, we refer to w_l and w_r as the leftmost and rightmost neighbors of v_{k+1} on C_k, see Fig. 1. We also say that v_{k+1} *dominates* w_i for $l < i < r$. That is these vertices on C_k are no longer on C_{k+1}.

When referring to vertices and points, we often need to use the (current) coordinates of the vertices and points on the grid. Let $v(x)$ and $v(y)$ represent the x and y coordinates of some vertex v.

2.1 Vertex Regions

In the immediate vicinity of every vertex there are two types of regions: *free regions* and *port regions*. The free and port regions alternate around the vertex, see Fig. 2(a). For each free region there is at most one edge passing through it to v. Each port region is bounded by a line segment with a number of ports and every edge inside the port region passes through a unique port. The number of ports in a port region is as small as possible. We define the six regions around v based on rays extending at certain angles or slopes from v. For convenience, we

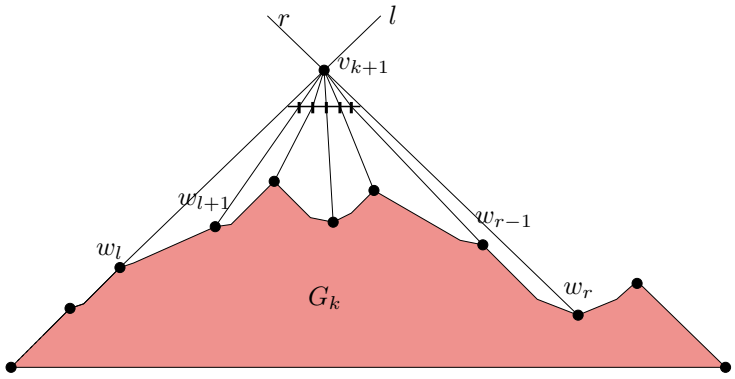


Figure 1: Graph G_{k+1} after inserting v_{k+1}. The shaded part is G_k. Vertices w_l and w_r are the leftmost and rightmost neighbors of v_{k+1}. The horizontal line segment below v_{k+1} is the middle port region through which all the edges (v_{k+1}, w_i), $l < i < r$, are routed.

assume that 0° is pointing in the vertical direction. As illustrated in Figure 2(a), the six regions around v are defined as follows:

- Free region M^f: between $-45°$ and $45°$
- Free region R^f: between 90° and 135°
- Free region L^f: between $-135°$ and $-90°$
- Port region M^p: between L^f and R^f
- Port region L^p: between L^f and M^f
- Port region R^p: between R^f and M^f

The algorithm draws each edge in E, except the initial edge (v_1, v_2), by "routing" it through a port of one of the two vertices in a fashion similar to Cheng *et al* [1]. Each edge consists of two connected edge segments. One edge segment, the *port edge segment*, connects a vertex with one of its ports. The other segment, the *free edge segment*, connects a vertex to one of its neighbor's ports. For example, for an edge $e = (u, v)$, if we route e through the leftmost port in u's middle port region M^p, we would draw two line segments, see Fig. 2(b): the *port edge segment* would pass from u to the port, and the *free edge segment* would pass from the port to v. This method of construction guarantees that the free edge segments always pass through free regions and that each port transmits at most one port edge segment.

We perform our construction in incremental stages, where each stage corresponds to the insertion of a new vertex. Observe that at each stage, for every vertex v except those on the external face, $w_1 = v_1, w_2, \ldots, w_m = v_2$, there

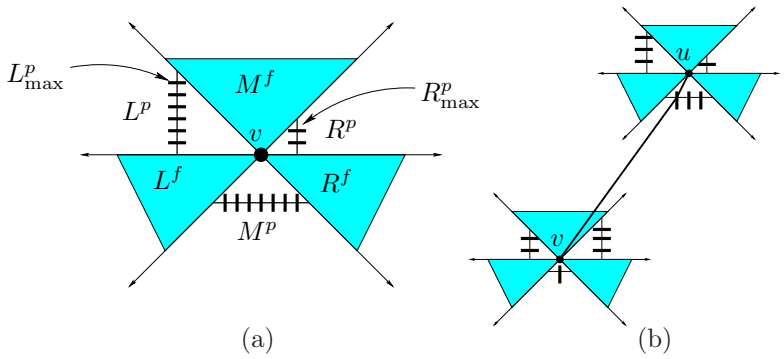


Figure 2: Vertex regions and edge routing: the number of ports along each port region is determined by the number of edges that need to be routed through that port region. (a) The area around a vertex v is divided into 6 regions. The free regions are shaded and at most one free edge segment goes through each one of them. All the port segments use ports in the port regions of v. (b) Routing an edge $e = (u, v)$, where the port edge segment connects u to one of its ports and the free edge segment connects the port to v, going through one of v's free regions.

are exactly three free edge segments. The remaining edges are connected to v via port segments. These remaining edges can be grouped into three classes based on which port region they are routed through, L^p, R^p, or M^p. Count the number of edges in each of these groups and let $d_l(v)$ be the number of port edge segments using port region L^p. Similarly, define $d_r(v)$ and $d_m(v)$ to be the number of port segments using port regions R^p and M^p. Observe that in the final stage, there are exactly three vertices on the exterior face, v_1, v_2, v_n, and then $\sum_{v \in V}(d_l(v) + d_r(v) + d_m(v)) = |E| - 1$. That is, for every edge, there is a corresponding port and free edge segment, except for the edge (v_1, v_2). This initial edge is only a single free (horizontal) edge segment. We could also, of course, remove the port edge segments for the final external face as well and thus the summation would be $|E| - 3$.

For a vertex v we define the *maximal right port* $R^p_{\max}$ as follows. Let v have coordinates (v_x, v_y). Then the $R^p_{\max}$ of v has coordinates $(v_x + d_r(v) + 1, v_y + d_r(v))$ if $d_r(v) > 0$ and (v_x, v_y) otherwise. We define the *maximal left port* $L^p_{\max}$ of v in a similar fashion, see Fig. 2(a).

2.2 Invariants of the CRA Algorithm

By design, our algorithm is incremental with n stages, where each stage corresponds to the insertion of the next vertex in the canonical order. Thus it is natural to define several key invariants to be maintained at every stage. The four invariants below are similar in flavor to those of Cheng *et al* [1] except that here we do not need to maintain any joint boxes.

1. All vertices and ports have integer coordinates.
2. Let $w_1 = v_1, w_2, \ldots, w_m = v_2$ be the vertices of the exterior face C_k of G_k in order. Then $w_1(x) < w_2(x) < \ldots < w_m(x)$.
3. The free edge segment of edge $e = (w_i, w_{i+1})$, $0 < i < m$, has slope ± 1 and e's port edge segment goes through a maximal port.
4. For every vertex v there is at most one (free) edge segment crossing each of its free regions. All other edge segments are port edge segments.

2.3 Vertex Shifting

In the algorithms that maintain good angular resolution with the aid of vertex joint boxes [1, 7], every time a new vertex is inserted, already placed vertices need to be shifted a great deal so that the joint box can fit amongst them. The amount of shifting required is typically of the order of the degree of the vertex. Invariably this leads to large constants behind the $O(n) \times O(n)$ area, e.g. $(20n-48) \times (10n-24)$ in [7] and $30n \times 15n$ in [1]. In our algorithm we never need to shift any vertex by more than five grid units allowing us to draw G in a $5n \times \frac{5n}{2}$ grid. When a new vertex v is inserted, we must create enough space so that the leftmost w_l and rightmost w_r neighbors of v can "see" v through their respective maximal port regions. Note that the previous $R^p_{\max}$ port of w_l and $L^p_{\max}$ of w_r were used at an earlier stage. Thus, we must create an additional port along the R^p region of w_l. Similarly, additional space is necessary along the L^p region of w_r.

In order to create more space we need to move w_l and w_r. We also have to ensure that the four invariants and the planarity of the graph are maintained. This is achieved by shifting the "shifting set" of the vertex as well as the vertex itself. Using the definition of de Fraysseix *et al* [5], define the *shifting set* $M_k(w_i)$ for a vertex w_i on the external face of G_k to be a subset of the vertices of G such that:

1. $w_j \in M_k(w_i)$ iff $j \geq i$
2. $M_k(w_1) \supset M_k(w_2) \supset \ldots \supset M_k(w_m)$
3. Let $\delta_1, \delta_2, \ldots, \delta_m > 0$; 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 planar.

These shifting sets can be defined recursively. Let w_l and w_r be the leftmost and rightmost neighbors of v on C_k. Then construct $M_{k+1}(w_i)$ recursively as follows:

$$M_{k+1}(w_i) = M_k(w_i) \cup v_{k+1}, \text{ for } i \leq l,$$
$$M_{k+1}(v_{k+1}) = M_k(w_{l+1}) \cup v_{k+1},$$
$$M_{k+1}(w_j) = M_k(w_j), \text{ for } j \geq r.$$

For convenience, define a *right-shift of m units* for a vertex w_i as shifting $M_k(w_i)$ by m units to the right so that all ports for every vertex in $M_k(w_i)$ also shift except the ports in the L^p region of w_i. Define a *left-shift of m units* for vertex w_i as shifting $M_k(w_{i+1})$ by m units to the right so that all ports for every vertex in $M_k(w_{i+1})$ also shift including the ports in the R^p region of w_i.

2.4 CRA Overview

The CRA algorithm constructs the graph one vertex at a time, by creating the graphs $G_1, G_2, \ldots, G_n$. Constructing G_i, $1 \leq i \leq 3$ is straightforward (see Figure 4(a)), so assume that G_k, for $k \geq 3$, has been constructed with exterior face $C_k = (v_1 = w_1, w_2, \ldots, w_m = v_2)$. Suppose we have embedded G_k with exterior face C_k. To construct G_{k+1}, let v_{k+1} be the next vertex in the canonical ordering and recall that w_l and w_r are, respectively, the leftmost and rightmost neighbors of v_{k+1} on the exterior face C_k.

Recall that $d_r(w_l)$ is the current number of port edge segments using R^p of w_l, and that $d_l(w_r)$ is the current number of port edge segments using L^p of w_r. There are two cases to consider:

- **case (a)** $d_r(w_l) = 0$, see Fig. 3(a).
- **case (b)** $d_r(w_l) > 0$, see Fig. 3(b).

In case (a) perform a left-shift of 2 units on w_l in order to free space for a port in the R^p region of w_l. In case (b) perform a left-shift of 1 unit on w_l. Similarly, if $d_l(w_r) = 0$ then perform a right-shift of 2 units on w_r. Otherwise perform a right-shift of 1 unit on w_r.

Insert v_{k+1} at the intersection of lines l and r, where l is the line with slope $+1$ through w_l's maximal right port and r is the line with slope -1 through w_r's maximal left port, see Fig. 1. In the case where lines l and r do not intersect in a grid point it suffices to shift all the elements in $M_k(w_r)$ one additional unit to the right.

The edges from v_{k+1} to w_l and w_r are routed through w_l's maximal right port and w_r's maximal left port, respectively. The remaining edges go from v_{k+1} to vertices w_i, $l < i < r$.

Before placing the M^p region of v_{k+1} it is necessary to ensure that there are enough ports on it that can be used to connect v_{k+1} to $w_{l+1}, w_{l+2}, \ldots, w_{r-1}$. The M^p region is a horizontal line segment with $1, 3, \ldots, 2m-1$ ports when the line segment is respectively $1, 2, \ldots, m$ grid units below v_{k+1}. To allocate enough space then we simply locate the horizontal line segment $\lceil (r-l)/2 \rceil$ units below v_{k+1}.

As shown in the next section the M^p region can be placed correctly, that is, placed so that it does not lie below any of the vertices w_i, $l < i < r$. We now need to route the edges from v_{k+1} to w_i. In the case where $r - l$ is an even amount there are exactly enough ports for each of the vertices, so the routing is simple, the first (leftmost) port goes to w_{l+1} and the last (rightmost) port to w_{r-1}. If it is odd, there is one extra port. Ideally, we would simply skip

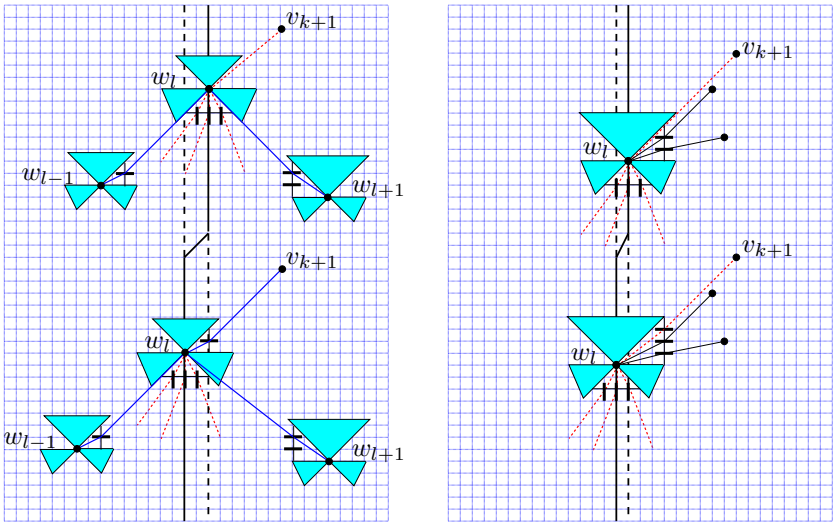


Figure 3: Adding the current vertex v_{k+1}. Here w_l is the leftmost neighbor of v_{k+1} on the exterior face of G_k. (a) If $d_r(w_l) = 0$, then we need to shift w_l two grid units to the left. (b) If $d_r(w_l) > 0$, then it suffices to shift w_l only one unit to the left. Note that the shifting set $M_k(w_l)$ also shifts with w_l.

the rightmost (or leftmost) port. However, it is possible that this would force the last edge (among others) to have the free edge segment be outside the valid region. Therefore, we proceed as follows, assign leftmost port to w_{l+1}, then w_{l+2}, and so on until either all are assigned or one vertex, w_a, has a free edge segment that is outside of the free region. We then assign ports from rightmost port to w_{r-1}, then w_{r-2}, until w_a is assigned. Note this is identical to simply skipping one port and continuing left. In the next section, we shall show that this correctly routes the edges. That is, all edges go through ports and the free edge segments lie in free regions.

It is important to point out that in the interest of saving space, being as compact as possible, we allow free edge segments to initially have length zero. That is a vertex w_i can actually lie on a port of another vertex v. This is not a problem so long as the port is used only to route an edge between v and w_i. During shifting, the vertex and the port are treated separately. That is, they are not necessarily confined to be in the same location. See Figure 4.

3 Correctness of the Algorithm

The algorithm works correctly if all four invariants are maintained. We show that free edge segments always remain in free edge regions and that there is at most one free edge segment per free region. We then need to bound the

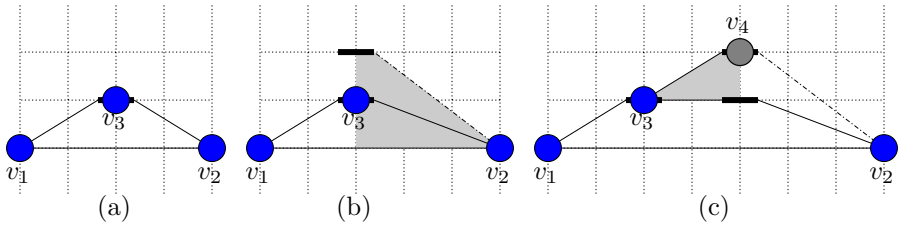


Figure 4: After initial settings of v_1, v_2, and v_3, inserting v_4 to share edges with v_2 and v_3. (a) Initial configuration. Notice that v_3 lies on both $R^p_{\max}$ of v_1 and $L^p_{\max}$ of v_2. (b) Shifting vertex v_2 one unit to the right. The shaded region indicates the L^p region of v_2. Notice that v_2's port did not move but a new port was inserted above it, the new $L^p_{\max}$ of v_2. (c) Shifting vertex v_3 two units to the left. The shaded region indicates the R^p region of v_3. Notice that v_2's port did not move and so now v_3 does not overlap that port though it still overlaps v_1's port. The new vertex v_4 is placed at the intersection of the diagonals from v_3's $R^p_{\max}$ port and v_2's $L^p_{\max}$ port. In this case, they are the same point and once again v_4 overlaps both maximal ports.

drawing area required by the algorithm and show that good angular resolution is maintained. Finally, we have to bound the number of bends created and analyze the running time.

Lemma 1 *If a free edge segment lies in a free region in G_k, then it remains in the free region in G_{k+1}.*

Proof: The initial edge (v_1, v_2) is treated as a special edge. It is a free edge segment that is not connected to any ports. As the vertices shift this edge remains horizontal and thus remains inside its free regions. It can also be seen that the lemma holds from G_2 to G_3.

For $k \geq 3$, we must consider how a free edge segment could "move". When inserting vertex v_{k+1}, the graph G_k changes by performing shifts (right or left shifts). These shifts move vertices and ports and possibly cause edge segments to change slope. Let e be a free edge segment in G_k which connects two vertices w and v via one of w's port regions. The slope of e determines whether e lies inside a free region of v or not. Therefore, we need to prove that upon shifting either the slope of e does not change or that the change does not allow e to leave the free region.

There are a few important points to remember about shifting before we proceed with the proof. First, shifting is only done with the shifting sets of vertices on C_k and for any such operation, all vertices and ports which move are shifted the same amount. In addition, all ports are shifted along with their respective vertices except for certain ports belonging to vertices on C_k. Second, if a vertex $w \in G_k$ is not in C_k then it must have been previously dominated by another vertex $v_{k'}, k' \leq k$. At that time, w is added to the shifting set of

$M_{k'}(v_{k'})$. From the recursive construction of the shifting sets, for all $k \geq k'$ and any vertex $w_i \in C_k$, $v_{k'} \in M_k(w_i)$ if and only if $w \in M_k(w_i)$. That is, $v_{k'}$ shifts if and only if w shifts.

Recall that there are three types of free regions: M^f, L^f, R^f. Let us first assume that e lies in the M^f region of v. Edge segments in the M^f regions are created by a vertex w dominating another vertex v. But from the arguments above, v must then belong to the shifting set of w and so both v and w are always shifted together. Since e connects to w's M^p region, the port always shifts with v and w. Therefore, the slope of e cannot change and it must remain in M^f.

As the two remaining cases are symmetric to each other, without loss of generality, let us now assume that e lies in the L^f region of v. This implies that the slope of e is between 0 and +1. Edges lying in the L^f region are created by neighboring vertices on some prior external face. That is, at some previous stage $k' \leq k$, we connected $v_{k'} = v$ and $w = w_l \in C_{k'}$. In this situation, e is routed from a port in R^p of w_l to $v_{k'}$. If we define $v_{k'}, w_l, w_r$ in the usual manner, for all $k \geq k'$ and all $w \in C_k$, if $w_l \in M_k(w)$ then $v_{k'} \in M_k(w)$. That is, if w_l shifts then so must $v_{k'}$. Note that $v_{k'}$ can shift without w_l shifting. As for w_l's R^p region, it is possible that the port region shifts without w_l but only on a left shift of w_l. In this case, $v_{k'}$ again still shifts with the R^p region of w_l. Therefore, shifting affects the slope of e only if $v_{k'}$ shifts and w_l's R^p region does not. Since $v_{k'}$ moves farther away from w_l's port region, the slope of e becomes more horizontal (approaches 0). Consequently, e still remains within the L^f region of v. ■

In order to prove our next main lemma (Lemma 3) we need to present a few smaller issues describing the relationship between the vertices and the lines of slope ±1. Each of these lemmas relies on the fact that G_k maintains the key invariants as described in Section 2.2.

Definition 1 *Let v be a vertex in G_k. Define v^+ (respectively) v^-) to be the line of slope +1 (resp. -1) passing through v.*

Property 1 *Suppose we are given a graph G_k maintaining the key invariants. For any $w_i \in C_k$, the region above w_i and between w^+ and w^- is empty of any vertices in G_k.*

Note, this property comes directly from the fact that the external face has free edge segments of slope ±1 and that $x(w_1) < x(w_2) < \ldots < x(w_m)$. See Figure 1.

Lemma 2 *Suppose we are given a graph G_k maintaining the key invariants. For any $w_i, w_j \in C_k$ and $i \leq j$, let p_i and p_j be any two points on w_i^+ and w_j^+ such that $p_i(y) = p_j(y)$. Then $p_j(x) - p_i(x) \geq j - i$. By symmetry, if we use w_i^- and w_j^-, then we still have $p_j(x) - p_i(x) \geq j - i$.*

Proof: See Figure 5 for a simple example. We shall prove this lemma inductively. It is certainly true for the case when $i = j$. So, let us assume the lemma

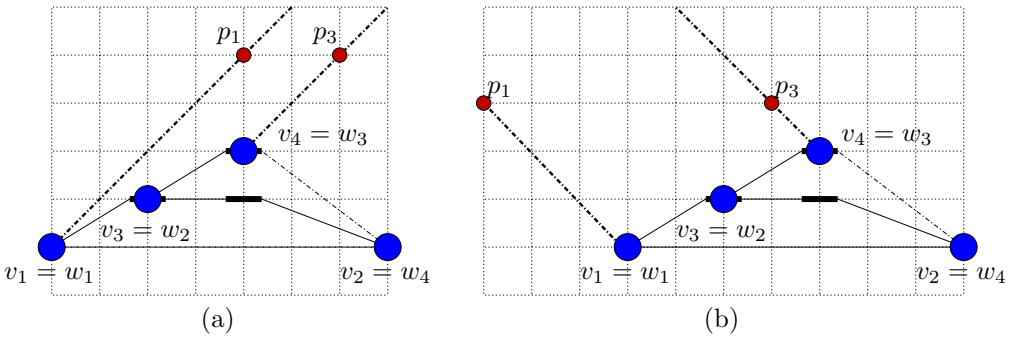


Figure 5: A simple example between two vertices on C_k, w_1 and w_3. (a) Points p_1 and p_3 are on w_1^+ and w_3^+ respectively. Notice that $p_3(x) - p_1(x) = 2 = 3 - 1$. (b) Same scenario except now points are on w_1^- and w_3^-.

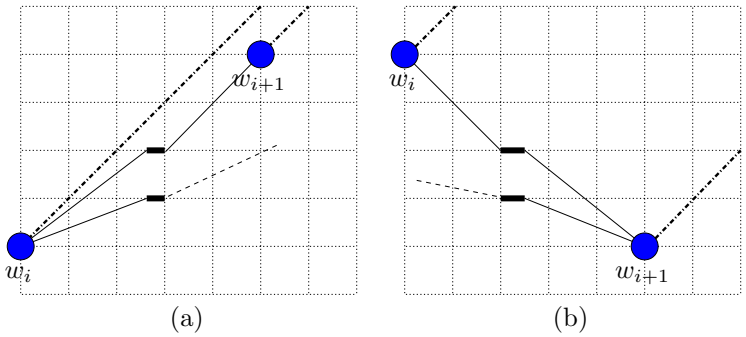


Figure 6: Notice the relationship between w_i, w_{i+1}, and their respective lines of slope $+1$ when w_{i+1} is (a) increasing and (b) decreasing.

holds for all $j' < j$. There are two possibilities for w_j, either $w_j(y) > w_{j-1}(y)$, increasing along the external face, or $w_j(y) < w_{j-1}(y)$, decreasing along the external face. In the first case, recall that the port edge segment connecting w_j to w_{j-1} must go through the $R^p_{\max}$ of w_{j-1}. Therefore, w_j^+ is shifted over one unit and hence $p_j(x) - p_{j-1}(x) = 1$. In the second case, the connection is through the $L^p_{\max}$ of w_j and therefore $p_j(x) - p_{j-1}(x) \geq 3$. See Figure 6. From our assumption then, we have

$$p_j(x) - p_i(x) = (p_j(x) - p_{j-1}(x)) + (p_{j-1}(x) - p_i(x)) \geq 1 + (j - 1 - i) = j - i.$$

To see the symmetric argument, notice that if we flip the graph about the y-axis, we have the same problem. ■

Corollary 1 *Suppose we are given a graph G_k maintaining the key invariants.*

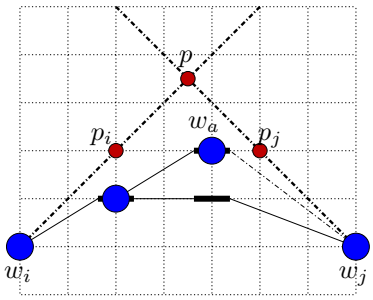


Figure 7: The intersection, p, of w_i^+ and w_j^- and its relationship with some w_a between w_i and w_j. Notice the distances between $p_i(x)$, $p_j(x)$, and $w_a(x)$.

For any $w_i, w_j \in C_k$ and $i \leq j$, let p be the intersection of w_i^+ and w_j^-. Then $p(y) \geq \max_{i \leq a \leq j}(w_a(y)) + (j-i)/2$.

Proof: Let w_a be any vertex with $i \leq a \leq j$. Let p_i and p_j be any two points on w_i^+ and w_j^- such that $p_i(y) = p_j(y) = w_a(y)$. That is, we are looking at points on the horizontal line passing through w_a. For notation, let $p_a = w_a$ which by definition is on both w_a^+ and w_a^-.

Since p_i and p_j satisfy the assumptions of Lemma 2, we can see that $p_j(x) - p_i(x) \geq j - i$. Let us now look at p, the intersection of w_i^+ and w_j^-. See Figure 7. From the above inequality and the fact that $p_i(y) = p_j(y) = w_a(y)$ we have

$$p(y) = p_i(y) + (p_j(x) - p_i(x))/2 \geq w_a(y) + (j-i)/2.$$

Therefore, the corollary holds for the maximum of all $w_a(y)$. ■

Lemma 3 *Every free edge segment passes through a free region which contains no other edges.*

Proof: From Lemma 1, we know that once a free edge segment lies within a free region it remains inside. Therefore, we only need to be concerned about ensuring that free edge segments are initially routed through a free region. This, of course, happens only with edges extending from a new vertex $v = v_{k+1}$.

For $k \geq 2$, when v is inserted there are two types of new edges added: the *outside edges* between v and the outside neighbors, w_l and w_r, and the *inside edges* between v and the inside neighbors w_i where $l < i < r$. In both cases the new edge is routed through a port creating one free edge segment and one port edge segment. A free edge segment of an outside edge has slope either $+1$ or -1 by construction; therefore it lies inside the free regions L^f and R^f of vertex v. Since v is a new vertex, there are no other segments inside these two free regions.

Dealing with the inside edges is more complex. We first need to show that there is sufficient space between the vertices on the exterior face of G_k and the

new vertex v. Second, we need to show that v has enough ports in its middle port region M^p for each of the vertices on G_k that it is connected to. Third, we need to show that the free edge segments of the inside edges initially lie inside their respective free regions. We shall show that for every inside neighbor w_i, $l < i < r$,

- vertex w_i lies on or below M^p, the middle port region for v, and
- we can assign a unique port along the M^p port region of v, such that the edge segment connecting w_i to that port fits inside w_i's middle free region M^f

The first part is fairly easy, we chose the middle port region M^p to be $\lceil (r-l)/2 \rceil$ units below v. That is, the y-coordinate of M^p is $v(y) - \lceil (r-l)/2 \rceil$. Recall that when inserting v, it is placed at the intersection p of w_l^+ and w_r^-, unless such an intersection is not on a grid point, in which case w_l is shifted left one unit to place p on a grid point. Note that in actuality, w_l and w_r are also shifted one or two units to make the connection fall on a port but the end result is that v is located at the intersection of w_l^+ and w_r^- prior to shifting. From Corollary 1, then we know that $v(y) = p(y) \geq \max_{l \leq a \leq r}(w_a(y)) + (r-l)/2$ and it follows that all inside neighbors w_i, $l < i < r$, lie completely below (or on) the M^p port region. Note that it is only possible for one inside neighbor w_a (the maximum vertex) to actually lie directly on the port region.

We now show that our assignment strategy from Section 2.4 properly routes edges through free regions. First note that if $r-l$ is even, then there are exactly $r-l-1$ ports on M^p and if it is odd there are exactly $r-l$ ports. As there are two cases, let us look at the odd case, which has one "extra" port and is a bit trickier to prove. The other case follows a nearly identical (though simpler) argument. The assignment is done in two phases, a left to right assignment, $w_{l+1}, w_{l+2}, \ldots, w_{a-1}$, for some vertex w_a followed by a right to left assignment, $w_{r-1}, w_{r-2}, \ldots w_a$. The vertex w_a is defined to be the first time in the left to right assignment where the free edge segment in the routing would lie outside the free region. We call this the *skip* vertex because it essentially skips one port. Since there are exactly $r-l$ ports for $r-l-1$ vertices, there can only be one possible "skip".

Let w_i be one of the vertices routed. If $i < a$, the edge e connecting w_i to v is routed through the $(i-l)^{\text{th}}$ port, p_i. Otherwise, e is routed through the $(i-l+1)^{\text{th}}$ port. Let p_l be the intersection of w_l^+ with the port region. And, let p_r be the intersection of w_r^- with the port region. Then we know that if $i < a$,

$$\begin{aligned} p_i(x) - p_l(x) &= i - l \\ p_r(x) - p_i(x) &= r - i + 1 \end{aligned} \tag{1}$$

where the $+1$ term comes because $r-l$ is odd. If $i \geq a$,

$$\begin{aligned} p_i(x) - p_l(x) &= i - l + 1 \\ p_r(x) - p_i(x) &= r - i. \end{aligned} \tag{2}$$

The free edge segment of e lies in a free region only if its slope is between -1 and $+1$. If we let p_i^+ and p_i^- be the intersection of w_i^+ and w_i^- with the port region, then e's free edge segment is in a free region if and only if $p_i(x)$ lies on or between $p_i^+(x)$ and $p_i^-(x)$. Applying Lemma 2, we know

$$p_i^+(x) - p_l(x) \geq i - l, \text{ and} \tag{3}$$
$$p_r(x) - p_i^-(x) \geq r - i. \tag{4}$$

Because w_i lies on or below the port region, we know that

$$p_i^+(x) \geq p_i^-(x) \tag{5}$$

Given that w_a is the first vertex which lies outside of the free region in the first phase, we know that, if $i < a$, w_i's edge segment must lie in a free region. Let us then look at the case where $i = a$. Combining Equations (2), (3), and (4), we see that

$$\begin{aligned} p_a^+(x) - p_l(x) &\geq a - l \\ &= p_a(x) - p_l(x) - 1 \Rightarrow \\ p_a^+(x) + 1 &\geq p_a(x) \\ p_r(x) - p_a^-(x) &\geq r - a \\ &= p_r(x) - p_a(x) \Rightarrow \\ p_a(x) &\geq p_a^-(x). \\ p_a^-(x) &\leq p_a(x) \leq p_a^+(x) + 1 \end{aligned} \tag{6}$$

Notice that $p_a(x)$ is (on or) between $p_a^-(x)$ and $p_a^+(x)$ except for the case when $p_a^+(x) + 1 = p_a(x)$, i.e. p_a lies one unit to the left of p_a^+.

So, let us assume that p_a does not lie (on or) between the two slopes. Therefore, $p_a^+(x)+1 = p_a(x)$. Now, since w_a is the skip vertex we know that the port q lying just to the left of p_a is free. Since $p_a(x) = q(x) + 1$, we substitute in to Equation (6) yielding

$$p_a^-(x) - 1 \leq q(x) \leq p_a^+(x). \tag{7}$$

But, q was not a valid port so it must not lie (on or) between w^- and w^+. The only possibility is that $q(x) = p_a^-(x) - 1$ which implies that $p_a^+(x) + 1 = p_a(x) = q(x)+1 = p_a^-(x)$. This in turn immediately implies that $p_a^+(x) < p_a^-(x)$ which contradicts Equation (5). Hence, p_a must lie between w_a^- and w_a^+, more precisely,

$$p_a^-(x) \leq p_a(x) \leq p_a^+(x). \tag{8}$$

Let us now look at the case where $i > a$. Observe that since the ports are assigned consecutively $p_i(x) - p_a(x) = i - a$. Applying Lemma 2 and Equation (8), for w_a and w_i, we see that

$$\begin{aligned} p_i^+(x) - p_a^+(x) &\geq i - a \\ &= p_i(x) - p_a(x) \\ &\geq p_i(x) - p_a^+(x) \Rightarrow \\ p_i^+(x) &\geq p_i(x). \end{aligned} \tag{9}$$

Applying Equations (2) and (4), we see that

$$\begin{aligned} p_r(x) - p_i^-(x) &\geq r - i \\ &= p_r(x) - p_i(x) \Rightarrow \\ p_i(x) &\geq p_i^-(x). \end{aligned}$$

Therefore p_i lies between w_i^- and w_i^+ and the edge to w_i is properly routed.

The argument for the case when $r - l$ is even is identical except one does not have to deal with the issue of a skip vertex. Therefore, we know that all free edge segments are properly routed through free regions. ■

Lemma 4 *If G_k maintains invariants one through four, then G_{k+1} maintains invariants one through four.*

Proof: By definition of the shifting set, invariants one and two hold, see [7]. By construction of the algorithm, invariant three holds as well. Also by construction every edge, except (v_1, v_2), inserted has a port edge segment and a free edge segment. By lemmas 1 and 3 invariant four also holds. ■

Lemma 5 *The angular resolution for vertex $v \in G$ as produced by the algorithm is $1/2d(v)$, where $d(v)$ is the degree of vertex v.*

Proof: The worst angle is achieved between a free edge segment for some edge f and a port edge segment for some edge e, where f is located at the boundary of its free region and e is the neighboring port edge segment. There are six possible cases but the argument is the same for all of them, so without loss of generality consider the case in Fig. 8. Let v be the vertex and $d(v) = d$ be its degree. Also let s and t be the lengths as shown in Fig. 8. Let θ be the angle between f and e, and x the number of ports as shown in the figure. Note that all vertices have at least one edge connected to them via free edge segments.[1] So, the number of ports, x, in any port region is at most $d-1$. From the figure, observe that $\tan(\theta) = t/(s-t)$ and hence $\arctan(t/(s-t)) = \theta$. But

$$\frac{t}{s-t} = \frac{\sqrt{2}/2}{\sqrt{2}(x+1) - \sqrt{2}/2} = \frac{1}{2x+1}$$

Using the Maclaurin expansion for $\arctan(y)$, where $y < 1$ we have

$$\arctan(y) = y - y^3/3 + y^5/5 - ... > y - y^3/3 > y - y^2/(y+1) = y/(y+1) = 1/(1+1/y)$$

Here, the last inequality comes from the fact that for $0 < y < 1$, $y^3/3 < y^3/(y+1) < y^2/(y+1)$. Since $0 < x \leq d-1$ and $0 < 1/(2x+1) < 1$, this yields

$$\theta = \arctan(1/(2x+1)) > 1/(1+2x+1) = 1/(2x+2) \geq 1/2d.$$

Therefore, the angular resolution is strictly greater than $1/2d$. ■

[1] In fact, all but the three external vertices have three free edge segments connected to them and it is a simple matter to make the external vertices have two free edge segments connected to them.

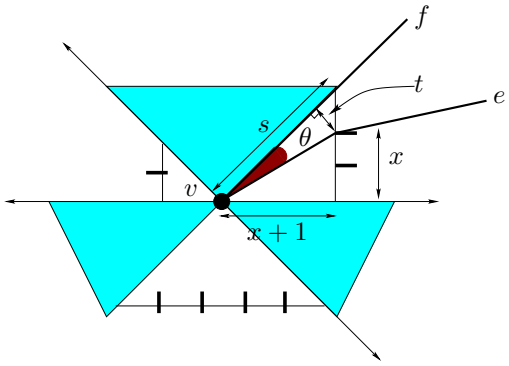


Figure 8: The minimum angle between two edges adjacent to vertex v is proportional to the degree d of the vertex. Using our algorithm the angle cannot be smaller than $1/2d$.

Theorem 1 *For a given planar graph G, the algorithm produces in $O(n)$ time a planar embedding with grid size $5n \times 5n/2$, using at most one bend. The angular resolution for every vertex v of G is $1/2d(v)$.*

Proof: Since every edge has only two segments, there can be at most one bend per edge. Chrobak and Payne [2] show how to implement the algorithm of De Fraysseix, *et al.* [5] in linear time. Their approach can be easily extended to our algorithm. By invariants three and four and by lemma 5 the angular resolution is at most $1/2d(v)$.

It remains to show that the drawings produced by the algorithm fit on the $5n \times 5n/2$ grid. Every time we insert a vertex v_k, we increase the grid size by at most 5 units, which implies that the width of the drawing is at most $5n$. The final drawing fits inside an isosceles triangle with sides of slope $0, +1, -1$. The width of the base is $5n$ and so the height is less than $5n/2$. ■

4 The PRA Algorithm

In this section, we introduce a novel approach to represent bends and vertices. Rather than insisting that bends lie on integer grid coordinates, we propose an alternative approach which allows bends to be located on a grid represented by polar coordinates. Using a polar representation allows us to independently control the grid size and edge bend positions. We begin by considering the polar representation in general and then present the PRA algorithm that uses the new approach.

A point p in the polar grid system is represented by a set of integers. For the vertices we only need two integers (p_x, p_y). For the bend-points we may need up to five integers. We shall see in the PRA algorithm that these five integers

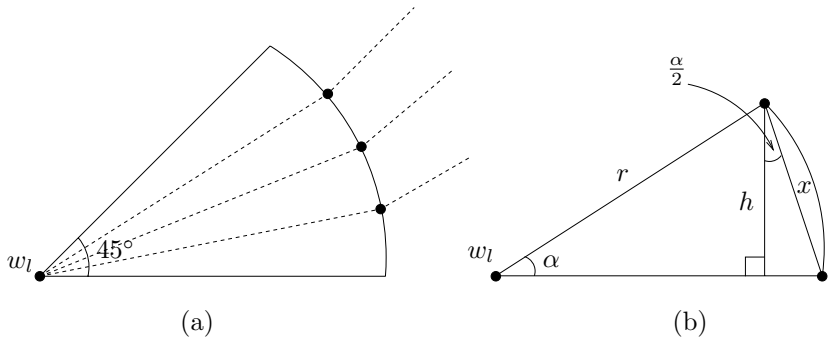


Figure 9: Vertex w_l is the left-most neighbor of the next vertex v_{k+1} along the exterior face of G_k. The $d_r(w_l)$ ports of w_l are evenly spaced on the arc of a circle of radius $2d_r(w_l)$ bounded by the middle free region M^f and the right free region R^f. (a) An example of the layout for the R^p region with $d_r(w_l) = 3$. (b) The distance x between two adjacent ports or a port and an adjacent free region can be computed given the radius of the circle and the angle between the edges connecting the ports to w_l: $x = 2r \sin \frac{\alpha}{2}$.

need not be explicitly stored for every bend-point. In general, a bend-point is given by:

- (p_x, p_y), the origin of the polar system
- p_r, the radius of the circle around the origin (p_x, p_y)
- p_d and p_n, the angle (p_θ) of the circle where the point is located, i.e., $p_\theta = 2\pi p_n / p_d$. For convenience, we consider $p_\theta = 0$ to be the vertical direction.

The PRA algorithm places vertices at integer grid coordinates, thus guaranteeing unit vertex resolution. As it is based on the CRA algorithm it also uses only 1 bend per edge. The main difference in the two algorithms is in the placement of the bend-points. In the PRA algorithm, bend-points will be placed on a circle around the vertex (rather than on a straight-line segment). Therefore, the origin, (p_x, p_y) for each bend-point need not be explicitly stored – it suffices to store the origin of the vertex that the bend-point is associated with. Similarly, groups of bend-points around a given vertex will have the same radius and hence each of the bend-points need not explicitly store p_r. Since the points will be evenly spaced in a port region, the values for p_θ need also not be explicitly stored for each bend-point.

Consider the leftmost neighbor, w_l, of the next vertex in the canonical order, v_{k+1}. The ports are evenly spaced in the R^p region for w_l, Fig. 9(a). The length of the straight-line segment separating two bend-points or a bend-point and an adjacent free region can be computed as follows. Consider the example in Fig. 9(b). We would like to compute the length x in terms of the radius of

the circle and the angle between the two line segments connecting consecutive ports to w_l. From basic trigonometry, the angle between h and x is $\alpha/2$. We can express h in terms of r and α: $h/r = \sin\alpha$ and we can express x in terms of h and α: $h/x = \cos\alpha/2$. Combining the two expressions we obtain

$$x = \frac{h}{\cos\frac{\alpha}{2}} = \frac{r\sin\alpha}{\cos\frac{\alpha}{2}} = \frac{2r\sin\frac{\alpha}{2}\cos\frac{\alpha}{2}}{\cos\frac{\alpha}{2}} = 2r\sin\frac{\alpha}{2}.$$

Assume we have inserted $v_1, v_2, \ldots, v_k$ and have a drawing of G_k with exterior face C_k. Consider inserting the next vertex v_{k+1} in the canonical order. Let w_l and w_r be the leftmost and rightmost neighbors of v_{k+1} on the exterior face C_k. Define f_b and f_e to be the *bend-point resolution* and *edge separation* respectively. Observe that in the standard Cartesian representation algorithms $f_b = 1$ and $f_e = \sqrt{2}/2$. Let $d_r(w_l)$, respectively $d_l(w_r)$, be the number of port edge segments using R^p of w_l, respectively L^p of w_r. When inserting v_{k+1}, the degrees for w_l and w_r affect the amount of shifting necessary to ensure proper resolution. As the cases for $d_r(w_l)$ and $d_l(w_r)$ are symmetrical, we shall concentrate on $d_r(w_l)$. There are two cases to consider:

- **case (a)** $d_r(w_l) = 0$ prior to insertion
- **case (b)** $d_r(w_l) \geq 1$ prior to insertion

In **case (a)** we insert the first edge in the port region R^p between the two free regions R^f and M^f of w_l. We place the port in the middle of the arc of a circle connecting R^f and M^f. Since there are no other bends yet in R^p we are only concerned with maintaining the edge separation. We need to place the port sufficiently away from the vertex w_l. Consider the relationship between the radius of the circle and the edge separation, see Fig. 9.

The edge separation $f_e = x = 2r\sin\frac{\alpha}{2}$. But since there is only one port and it is in the middle of the arc, $\alpha = \pi/8$. We are interested in the radius necessary to achieve the edge separation f_e which is given by

$$r = \frac{f_e}{2\sin\frac{\alpha}{2}} = \frac{f_e}{2\sin\frac{\pi}{16}} < \frac{4f_e}{\sqrt{2}} = 2\sqrt{2}f_e.$$

Since we maintain that the vertices are at integer coordinates and the radii are also integers, then the minimum radius required in case (a) is

$$r < \lceil 2\sqrt{2}f_e \rceil.$$

In **case (b)** we insert an additional port in the port region R^p which already has at least one port. In this case, we must ensure that both the edge separation f_e and bend-point resolution f_b are preserved. In this case the radius required is given by:

$$\max\{\lceil \frac{f_e}{2\sin\frac{\pi}{8(d_r(w_l)+1)}} \rceil, \lceil \frac{f_b}{2\sin\frac{\pi}{8(d_r(w_l)+1)}} \rceil\}.$$

Algorithm	f_v	f_b	f_e	drawing area	resolution
CRA	1	1	$\sqrt{2}/2$	$5n \times 5n/2$	$1/2d(v)$
PRA1	1	1	$\sqrt{2}/2$	$9n \times 9n/2$	$\pi/4d(v)$
PRA2	1	1/2	1/2	$7n \times 7n/2$	$\pi/4d(v)$

Table 1: Fixing specific values for the vertex resolution f_v, bend-point resolution f_b, and edge separation f_e allows us to compare the PRA and CRA algorithms.

Typically, $f_b \geq f_e$, so we can assume that the bend-point resolution determines the radius in case (b). Using this together with the fact that $\sin\alpha > 0.97\alpha$ for $\alpha < \pi/8$, the minimum radius required is

$$r < \lceil \frac{f_b}{2\sin\frac{\pi}{8(d_r(w_l)+1)}} \rceil < \lceil \sqrt{2} f_b (d_r(w_l)+1) \rceil$$

Summing over all vertices in the graph, the sum of the radii used for the right port regions, R, yields:

$$R = \sum_{v_i \in V : d_r(v_i)=1} \lceil 2\sqrt{2} f_e \rceil + \sum_{v_i \in V : d_r(v_i)>1} \lceil \sqrt{2} f_b (d_r(v_i)+1) \rceil. \quad (10)$$

With R we bounded the number of shifts required because of "right" neighbors. Similarly, we can define L, the shifts necessary due to "left" neighbors:

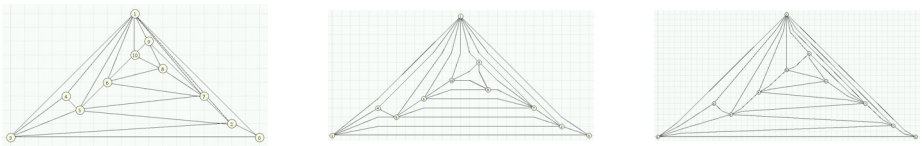
$$L = \sum_{v_i \in V : d_l(v_i)=1} \lceil 2\sqrt{2} f_e \rceil + \sum_{v_i \in V : d_l(v_i)>1} \lceil \sqrt{2} f_b (d_l(v_i)+1) \rceil. \quad (11)$$

L and R bound the number of shifts required due to left and right neighbor visibility. Note, however, that if we shift by the minimum amount required by the f_e and f_b parameters, the location of the next vertex v_{k+1} may not be at integer coordinates. We can guarantee that v_{k+1} is placed on the integer grid by performing some additional shifts. By shifting at most 3 more units, we are guaranteed to find an integer location for v_{k+1}. Then the total shifting required is at most $L + R + 3n$. Since the final drawing fits inside an isosceles right-angle triangle, the total area required for the drawing is $(L + R + 3n) \times (\frac{L+R+3n}{2})$.

In order to compare the PRA algorithm to the CRA algorithm, we evaluate equations 11 and 10 using two sets of parameters, Table 1. In all three cases the algorithms guarantee at most one bend per edge. The PRA algorithms place all the vertices on grid points and each bend-point is determined by at most five integer polar coordinates.

5 Conclusion and Open Problems

In this paper we present two algorithms for drawing planar graphs with good angular resolution while maintaining small drawing area. Other drawing criteria optimized by the algorithms include number of bends, vertex resolution,



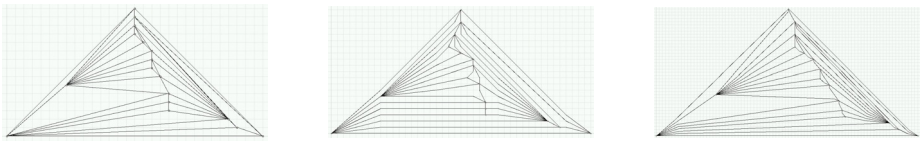
(a) (b) (c)

Figure 10: A graph with 11 vertices drawn using (a) the canonical ordering on the 10×19 grid; (b) the CRA algorithm on the 14×29 grid; (c) the PRA algorithm on the 23×45 grid.

bend-point resolution, and edge separation. The first algorithm, CRA, is a traditional algorithm in which vertices and bend-points are represented using Cartesian coordinates. It improves on the best known simultaneous bounds for the six drawing criteria. In the PRA algorithm vertices and bend-points are represented using polar coordinates. It is based on the CRA algorithm but allows for independent control over the grid size and bend positions.

Using a polar coordinate representation yields slightly worse area bounds compared to the CRA algorithm, see Fig 10 and Fig. 11. We believe, however, that the PRA approach is more promising. The angular resolution of the PRA algorithm is better and it provides greater control over the drawing process.

The PRA bounds presented in this paper can be further improved. Using two integers to represent the radius (similar to the way the angles are currently represented) will most likely result in smaller drawing area. Our current estimates indicate that certain (small) values of edge separation and bend-point resolution yield grids of size $4n \times 2n$. It is likely that when using only one bend per edge, the best angular resolution will be achieved for vertex regions in which each of the port and free regions have angles $\pi/3$ rather than a combination of $\pi/4$ and $\pi/2$. The biggest challenge, however, to the success of the PRA algorithm deals with the three potential shifts needed to align a new vertex onto an integer grid. If we can reduce this bottleneck, we feel that the PRA algorithm can significantly surpass the bounds of the CRA algorithm.



(a) (b) (c)

Figure 11: A graph with 17 vertices drawn using (a) the canonical ordering on the 16×31 grid; (b) the CRA algorithm on the 21×41 grid; (c) the PRA algorithm on the 43×85 grid.

Acknowledgments

A preliminary version of this paper appeared in the Proceedings of the 9th Symposium on Graph Drawing [4].

References

[1] C. C. Cheng, C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Drawing planar graphs with circular arcs. *Discrete and Computational Geometry*, 25:405–418, 2001.

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

[3] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, Englewood Cliffs, NJ, 1999.

[4] C. A. Duncan and S. G. Kobourov. Polar coordinate drawing of planar graphs with good angular resolution. In *Proc. 9th Symposium on Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, pages 407–421, 2002.

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

[6] A. Garg and R. Tamassia. Planar drawings and angular resolution: Algorithms and bounds. In *Proc. 2nd European Symposium on Algorithms*, pages 12–23, 1994.

[7] M. T. Goodrich and C. G. Wagner. A framework for drawing planar graphs with curves and polylines. In *Proc. 6th Symposium on Graph Drawing*, volume 1547 of *Lecture Notes in Computer Science*, pages 153–166, 1998.

[8] C. Gutwenger and P. Mutzel. Planar polyline drawings with good angular resolution. In *Proc. 6th Symposium on Graph Drawing*, volume 1547 of *Lecture Notes in Computer Science*, pages 167–182, 1998.

[9] 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).

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