Short and Smooth Polygonal Paths

James Abello[1] and Emden Gansner[2]

[1] Communication Information Systems Research, AT&T Labs-Research, USA
abello@research.att.com
[2] Information Analysis and Display Research, AT&T Labs-Research, USA
erg@research.att.com

Abstract. Automatic graph drawers need to compute paths among vertices of a simple polygon which besides remaining in the interior need to exhibit certain aesthetic properties. Some of these require the incorporation of some information about the polygonal shape without being too far from the actual shortest path. We present an algorithm to compute a locally convex region that "contains" the shortest Euclidean path among two vertices of a simple polygon. The region has a boundary shape that "follows" the shortest path shape. A cubic Bezier spline in the region interior provides a "short and smooth" collision free curve between the two given vertices. The obtained results appear to be aesthetically pleasant and the methods used may be of independent interest. They are elementary and implementable. Figure 7 is a sample output produced by our current implementation.

1 Introduction

The problem of finding collision free paths has been studied in robotics, VLSI layout and computational geometry. A host of shortest path based methods have been proposed in the literature. In some cases, curvature constraints are imposed and in others the physical constraints of robot cars or manipulators are incorporated ([5], [6], [10], [11], [12], [16], [18]).

A different flavor of path routing is required by automatic graph drawers specially in applications where the nodes are represented by single connected shapes. In this case, once nodes are positioned the edges need to be placed and the layouts are forced to use some form of curved edges to avoid collision with non-incident nodes ([7], [17]). The described environment can be modeled as a simple polygon P containing a collection of disjoint simple polygonal holes, corresponding to the node obstacles. The general edge placement problem consists in drawing "natural-looking" curves between vertices in this environment. Arguably, natural curves avoid obstacles, stay close to a shortest path, do not turn too sharply, and avoid unnecessary inflections [4]. Robotics physical constraints such as robot size, mass, acceleration, or turning radius do not seem to have a clear interpretation in the context of natural-looking curves in graph drawings. Recently, a heuristic has been proposed that produces curves which satisfy some of the criteria mentioned above [4]. Unfortunately, the obtained curves are forced

C. L. Lucchesi, A. V. Moura (Eds.): LATIN'98, LNCS 1380, pp. 151–162, 1998.

to touch the obstacles that lie on the shortest path and are not guaranteed to be completely contained in the available free space. The authors of [4] comment on the difficulty of producing an implementable solution that uses in an efficient manner the available free space while keeping the curve in the proximity of the shortest path. We offer an algorithmic solution to both of these problems for the case of simple polygons without holes. Namely, for a given pair of vertices x, y of a simple polygon P, the algorithm produces a "smooth" curve that is completely contained in the interior of P and that lies on the proximity of the shortest path from x to y. The methods used are visibility based and may be of independent interest. They are elementary and implementable. Figure 7 is a sample output produced by our current implementation of the algorithm. Finally, it is important to notice that the apparently more general edge placement problem that considers polygons with holes in their interior can be handled using similar methods to the ones presented here for simple polygons without holes. We explain the overall idea in Sect. 4.

The second section of the paper contains the relevant definitions. The third section is the bulk of the paper. It contains a description of the algorithm and the main arguments justifying its correctness. The general problem, closing remarks and further exploration avenues are presented in Sect. 4.

2 Problem Statement

2.1 Definitions

We consider simple polygons on the plane with an implicit counterclockwise labeling of the vertices from 1 to n. Given a pair of non-consecutive vertices x and y on the boundary of P, consider their counterclockwise boundary successors $next(x)$ and $next(y)$ respectively. The boundary is thus divided into two closed chains $[next(x), y]$ and $[next(y), x]$. Denote by $CSP[x, y]$ the ordered shortest path from x to y in the counterclockwise direction. Notice that $CSP[x, y]$ is not necessarily the shortest interior Euclidean path from x to y which we denote by $SP[x, y]$. It shall be clear then that any two nonconsecutive vertices x and y determine a subpolygon $R[x, y]$ whose counterclockwise boundary is $\{x, CSP[next(x), y], next(y), CSP[next(y), x]\ \}$. The shortest Euclidean path $SP[x, y]$ is nowhere exterior to $R[x, y]$ (see Fig. 1). There are however other subpolygons "similar" to $R[x, y]$ that not only contain $SP[x, y]$ but that provide in its proximity as much space as allowed by the boundary of the input polygon. Our objective is then to find one such subpolygon on which we can draw a smooth curve that is "close" to $SP[x, y]$. We refer to this subpolygon as the drawing region.

2.2 Visibility Notions

Two polygon vertices are called *visible* if the open line segment between them is completely contained in the interior of the polygon or if they are consecutive on

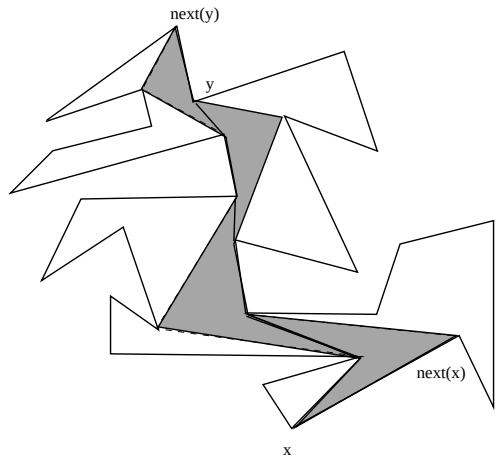

Fig. 1. The shortest Euclidean path $SP[x, y]$ is nowhere exterior to the subpolygon $R[x, y]$ determined by two nonconsecutive points x and y.

the polygon boundary. The set of vertices visible from a vertex x is denoted by $N(x)$. The *visibility graph* of a polygon is the graph whose vertices correspond to the vertices of the polygon and edges correspond to *visible* pairs of polygon vertices. It can be computed optimally [8]. A very large subclass of these graphs has been studied in ([1], [2]) and a good overview of related research can be found in [13].

If x and y are two *visible* vertices, (x smaller than y in the implicit counterclockwise order), let $fr(x, y)$ denote the first vertex to the right of y (if any) that is visible from x. Similarly, let $fl(x, y)$ be the first to the left of y (if any) that is visible from x. The computation of $fr(x, y)$ and $fl(x, y)$ is a fundamental operation in many visibility based algorithms. It is useful then to let $T(x, y)$ denote the time taken to compute either $fr(x, y)$ or $fl(x, y)$ for a given pair of visible vertices x and y in a polygon P. With this in mind, for a subset S of vertices of a polygon P, let

$T(S) = \max\{\ T(x, y)$: both x and y are visible vertices of $S\}$.

One can define similar notions in terms of other computational resources but we will not discuss these issues any further here.

3 The Algorithm

3.1 Algorithm Overview

Our general method can be viewed in three stages with the first two interleaved for efficiency purposes as follows.

```
Stage I.  Find Local Pentagons

  For every ordered triple of consecutive vertices (i,j,k) in
    SP[x,y] where i and k are both  different from x and y do
    {
     Compute an auxiliary point  on each of the directed rays
     r(i,j) and  r(k,j)  and   call them a(i,j)  and   a(k,j)
     respectively. The   interior   of  the   pentagon  with
     vertices i, j, k, a(i,j), a(k,j) must be contained in P.
    }

Stage II.  Find the boundary of the drawing region

  For  every four consecutive  vertices  (i,j,k,l) in SP[x,y]
    where i and l are both different from x and y do
    {
     Use the local pentagons  obtained in the  previous stage
     to obtain from  them  local heptagons and  octagons that
     surround the shortest path.
    }

Stage III. Find  a curve from x  to y in  the interior of the
           drawing region that approximates SP[x,y].
```

3.2 Stage I: How to Find the Local Pentagons?

The first thing to notice is that if i, j and k are three consecutive vertices on $SP[a, b]$ then, by the Jordan closed curve theorem, the sequence $(i, j, k, [k, i])$ forms a subpolygon of P. This guarantees the existence of points on the directed rays $r(i, j)$ and $r(k, j)$ within this subpolygon. The main idea is to find, in an interleaved fashion, a sequence of points $fe(k, j)$ on the ray $r(k, j)$ which are visible from i and a corresponding sequence $fe(i, j)$ on the ray $r(i, j)$ which are visible from k. These two sequences satisfy the additional requirement of being mutually visible. The farthest points in these two sequences are called the auxiliary points $a(k, j)$ and $a(i, j)$ and they form with i, j, and k a pentagon whose interior is contained in the interior of P. The sequences are determined by intersections of the rays $r(k, j)$ and $r(i, j)$ with either, visibility rays emanating from the vertices i and k or with certain special polygon boundary edges. We describe next the major steps involved in these computations. For clarity of exposition we present just the procedure *rauxpoint (P,i,j,k)* in charge of the computation of $a(k, j)$. Switching the roles of i and k and substituting $fr(i, j)$ for $fl(k, j)$ we obtain the symmetric procedure *lauxpoint(P,i,j,k)* that computes $a(i, j)$. As previously indicated, interleaving these two procedures we obtain a method to compute the auxiliary points. As a final notational convenience we let *closest*(j, S) denote those vertices in a set S that are closest in Euclidean distance to a vertex j.

Computation of the Right Auxiliary Points. Assume that (i, j, k) are three consecutive vertices on $SP[x, y]$ which form a left turn and recall the definition of $fr(i, j)$ and $fl(k, j)$ given in Sect. 2. It is a well known fact , proved using standard visibility arguments, that there exists a unique boundary edge of P which serves as the base of a funnel subpolygon with apex i that passes through the vertices j and $fr(i, j)$. We denote such boundary edge by $fbase(i, j, fr(i, j))$ (see Fig. 2). Similarly, there is a funnel subpolygon associated with k, j and $fl(k, j)$. The computation depends on the position of $fr(i, j)$ and $fl(k, j)$ with respect to the lines $l(k, j)$ and $l(i, j)$ respectively. In one case, the intersection of the funnel bases $fbase(i, j, fr(i, j))$ and $fbase(k, j, fl(k, j))$ with the rays $r(k, j)$ and $r(i, j)$ are used in determining the auxiliary points. In the other, the procedure is called recursively to look for other suitable funnel bases on a smaller polygon P' whose interior is contained in P. Figs. 2, 3 and 4 illustrate the main cases that need to be considered by the procedure *rauxpoint(P,i,j,k)*.

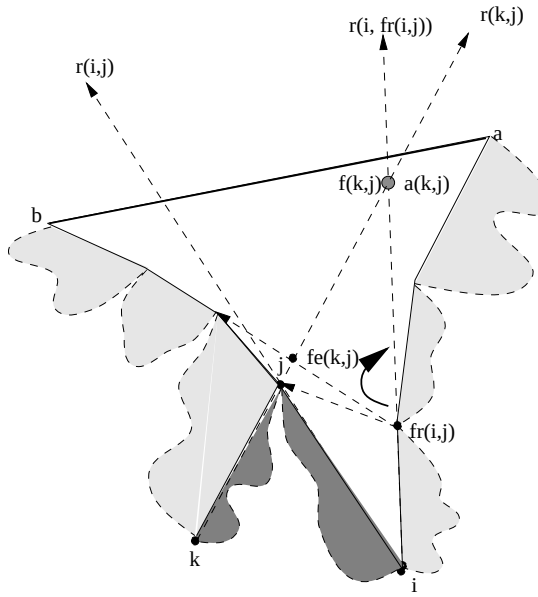

Fig. 2. The ray $r(i, fr(i, j))$ intersects the ray $r(k, j)$ and i and $fr(i, j)$ lie on the same side of the line $l(k, j)$

procedure rauxpoint(P,i,j,k)

```
If  (r(i,   fr(i,j)) intersects the ray r(k,j)
     then label  such intersection f(k,j)
     else f(k,j) := infinity;
```

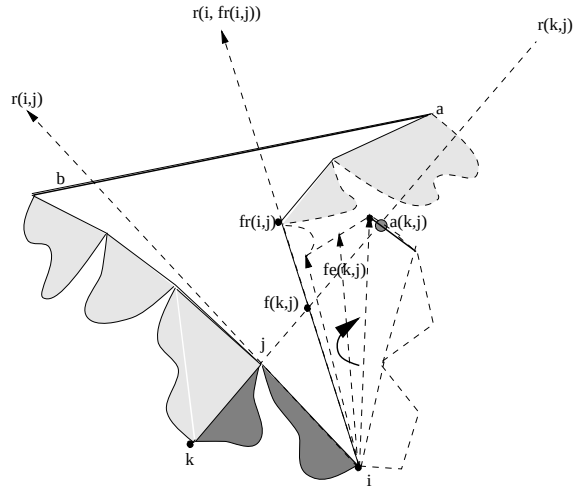

Fig. 3. The ray $r(i, fr(i,j))$ intersects the ray $r(k,j)$ but i and $fr(i,j)$ do not lie on the same side of the line $l(k,j)$

```
If (i and fr(i,j) are not on different sides of the line l(k,j))
    then { [a,b] :=   fbase(i,j,fr(i,j));
           p1 :=  [a,b] intersection with r(k,j);
           a(k,j) := closest(j,{p1, f(k,j)})
         }
    else {
           Let P' be the polygon with boundary
             (i, f(k,j), j, fr(i,j), clockwisechain[fr(i,j),i]);
           a(k,j) := rauxpoint(P',i,f(k,j),j);
         }
```

Lemma 1. *The auxiliary points a(i,j) and a(k,j) are computed by interleaving the procedures rauxpoint(P,i,j,k) and lauxpoint(P,i,j,k). These points together with i, j and k form a pentagon whose interior is completely contained in the interior of P (Fig. 5).*

Proof. In the case that i and $fr(i,j)$ do not lie on different sides of the line $l(k,j)$, straightforward properties of their associated funnel guarantee that the *procedure rauxpoint* computes a point $a(k,j)$ on the ray r(k,j) which is visible from i. In the remaining case, the interior of the polygon P' with boundary $(i, f(k,j), j, fr(i,j), clockwisechain[fr(i,j), i])$ is contained in the interior of P and the shortest path in P' from i to j goes trough $f(k,j)$. This allow us to apply the procedure recursively to P'. Similar analysis is true for the *procedure lauxpoint* when the points involved are k and $fl(k,j)$ and the computed point

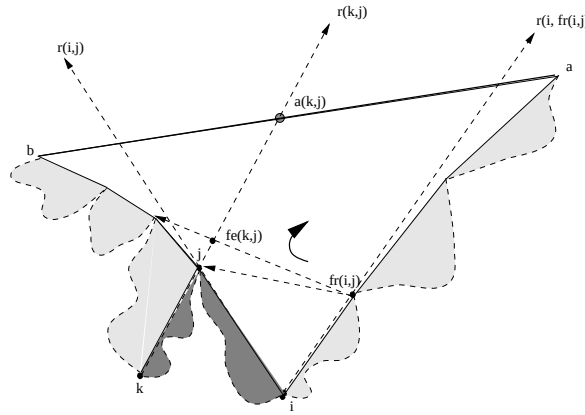

Fig. 4. When the ray $r(i, fr(i, j))$ does not intersect the ray $r(k, j)$, the funnel base intersected with the ray $r(k, j)$ determines the auxiliary point $a(k, j)$

is $a(i, j)$ on the ray $r(i, j)$. Finally, the invariant maintained by the interleaved execution of these two procedures insure that the interior of the pentagon with vertices $i, j, k, a(i, j)$ and $a(k, j)$ is completely contained in the interior of P. □

Complexity of Finding the Auxiliary Points. In order to speed up the computation of the auxiliary points and assuming that we are interested in answering a series of shortest path queries it is worthwhile to precompute besides the visibility graph of P an auxiliary bipartite graph *AV(Vertices of P, Boundary edges of P)*. *AV* gives for every vertex v of P, the ordered sequence of boundary edges seen by v. This graph was implicitly defined in [2] and an efficient algorithm for its computation has been recently proposed in [14]. It is of interest to notice that the visibility graph of P determines completely the auxiliary graph *AV* but not conversely. In any case the complexity is still *O(Visibility graph of P)*. With these two graphs at hand, it follows from the proof of the previous Lemma, that the auxiliary points $a(i, j)$ and $a(k, j)$ can be found in $O(|N(k)|)$ and $O(|N(i)|)$ respectively.

3.3 Stage II. How to Find the Boundary of the Drawing Region?

Given four consecutive vertices (i, j, k, l) in $SP[x, y]$ for which the auxiliary points $a(i, j), a(k, j), a(j, k)$ and $a(l, k)$ have been computed, it is necessary to check if they are "compatible" with each other in the sense of determining a "locally convex" ordered set of points. Insuring that this compatibility requirement is satisfied amounts to a case analysis that depends on how the triangles $\{j, a(i, j), a(k, j)\}$ and $\{k, a(j, k), a(l, k)\}$ are positioned with respect to each

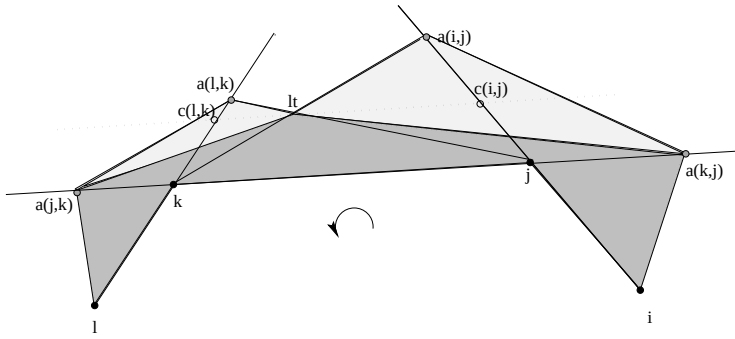

Fig. 5. The local pentagons around (i, j, k) and (j, k, l) described in Lemma 1 and the heptagon mentioned in one case of the proof of Lemma 2.

other. The proof of the following two lemmas can be turned into a procedure *findregion(i,j,k,l)* that computes the desired locally convex regions.

A Useful Property of Successive Quadruples of Points in Shortest Euclidean Paths.

Lemma 2 (The concave case). *Let (i,j,k,l) denote four consecutive points in a shortest Euclidean path between two vertices of a simple polygon. Assume also that the path from i to j to k to l is concave. Under these conditions the auxiliary points computed by the procedures of the previous section satisfy the following property:*

There exists a point lt (called hereafter local top) contained in the union of the two triangles with vertices $(k, j, a(i, j))$ and $(k, j, a(l, k))$ which form with the vertices $i, j, k, l, a(j, k)$ and $a(k, j)$, a heptagon whose interior is completely contained in the interior of P (Fig. 5).

Proof. (Sketch.) The interior of the triangles $(j, k, a(i, j))$ and $(j, k, a(l, k))$ can not contain vertices of P because that contradicts the way that $a(i, j)$ and $a(l, k)$ were chosen by the procedures of the previous section. The result depends completely on the relative position of these two triangles and on which side of the directed line $l(k, j)$ resides the intersection of the lines $l(i, j)$ and $l(l, k)$. In some cases, the local top is defined to be either the intersection of the segment $[j, a(l, k)]$ with the segment $[k, a(i, j)]$ or the intersection of the segment $[j, a(i, j)]$ with the segment $[k, a(l, k)]$. In the remaining cases, the local top is defined to be either the intersection of the ray $r(a(j, k), a(l, k))$ with the segment $[a(k, j), a(i, j)]$ or the intersection of the ray $r(a(k, j), a(i, j))$ with the segment $[a(j, k), a(l, k)]$. The described choice of the local top together with visibility considerations determines the desired heptagon (Fig. 5). □

We point out that in many cases, depending on the local geometry, the local heptagons can be enlarged without increasing the overall complexity. These local optimizations will be explained in detail somewhere else.

Similar analysis to the one presented in the previous lemma give us the following result.

Lemma 3 (The non-concave case). *If (i, j, k, l) are four consecutive vertices on a shortest Euclidean path between two vertices of a polygon P which do not constitute a concave path then they together with the auxiliary vertices form an octagon whose interior is completely contained in the interior of P (Fig. 6).* □

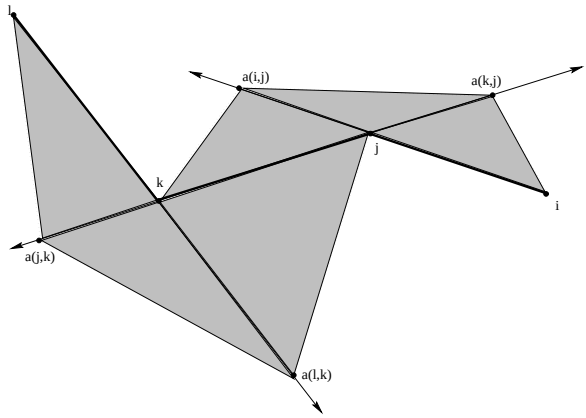

Fig. 6. The *octagon* associated with a non-concave subpath i, j, k, l as described in Lemma 3.

Complexity of Finding the Locally Convex Region. The computations described in the previous two lemmas are all local. We repeat them sliding a window of size four over SP(x,y). Some care is necessary in preserving the local convexity during the incremental region computation. This amounts to the maintenance of the region boundary in a data structure that allow us to answer ray shooting queries efficiently. At most a logarithmic cost is incurred here. Figure 7 is a sample output of our current implementation.

3.4 Stage III. Embed a Smooth Curve Within the Computed Drawing Region

The collection of points computed in the previous stage defines a subpolygon that contains the shortest Euclidean path from x to y. This subpolygon is the region on which the shortest path approximating curve will be drawn. As one approach to embed a smooth curve in its interior we can limit ourselves to cubic Bezier splines [3]. This family of curves is general enough to give pleasing results,

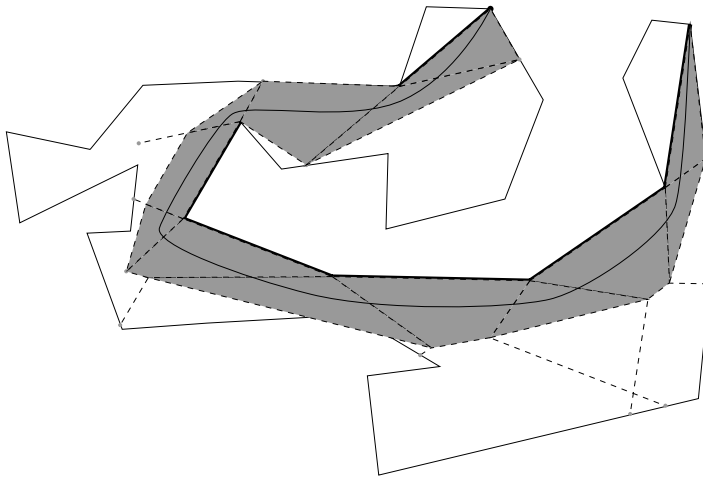

Fig. 7. Sample output produced by the current implementation. The drawing region is highlighted

is computational simple, and most importantly for our purposes, satisfies the convex hull property, i.e., a Bezier curve determined by four points lies within the convex hull of those four points. To exploit this property we observe that the drawing region computed in the previous stage can be viewed as a chain of triangles and convex polygons. Each convex polygon contains two vertices on the shortest path where bends occur. We can pick these points as the first and last control points p_0 and p_3. The control point p_1 is chosen to lie on the ray based at p_0 that bisects the two edges of the polygon meeting at p_0, and in the interior of the polygon. The control point p_2 is chosen similarly with respect to p_3. For aesthetic reasons, it is desirable that p_1 and p_2 roughly equally divide the distance between p_0 and p_3, and that path p_0, p_1, p_2, p_3 mimic any change in curvature of the shortest path from p_0 to p_3. For example, if the bends on the shortest path at p_0 and p_3 have the same sign of curvature, we expect p_2 not to cross the ray (p_0, p_1) and p_1 not to cross the ray (p_3, p_2). This solution is adequate, but causes the curve to coincide with the shortest path on turns and makes no use of the additional space provided by the triangles anchored at the bends. We can improve the situation by generalizing this technique. Namely, using the triangle at each bend, we pick p_0 and p_3 to lie on the two sides of the triangle meeting at the bend, each one-third along the side away from the bend. The change in angle at the bend can be divided proportionally between p_0 and p_3, defining rays based at these points on which we can pick p_1 and p_2 inside the polygons and satisfying the needed local convexity or concavity properties.

Another approach is to use subdivision methods like Chaikin's algorithm [3]. We are currently experimenting with these techniques and the obtained results will be reported in the final paper version. For now, we let *O(SP*$[x, y]$*, Drawing Region)* denote the complexity of embedding a "smooth" curve in a drawing region.

4 Overall Complexity and Concluding Remarks

A polygonal hole is a simple planar polygon together with its interior. In the general edge placement problem, the collection H of polygonal holes are disjoint and we assume that they are completely contained in the interior of a large bounding simple polygon P. The forbidden space *forb(H)* is the union of the interiors of the polygons in H and the free configuration space *free(P,H)* is equal to $P \backslash forb(H)$. Given two points x and y in *free(P,H)*, any Euclidean shortest path from x to y is a polygonal path whose inner vertices are vertices of H. This path can be constructed in $O(hlog(h))$ time, where h is the total number of boundary edges of the polygons in H ([9]). Assuming that we are answering a collection of shortest path queries, it is more effective to precompute the visibility graph of $P \cup H$ where two vertices are considered visible if the segment joining them is completely contained in *free(P,H)*. This computation can be done in $O(h'log(h') + k)$ where h' is the number of vertices in $P \cup H$ and k is the number of edges in its visibility graph([8]). When x and y are not vertices of $P \cup H$ we consider the extended visibility graph of $(H \cup P)^*$ = vertices of $(H \cup P) \cup \{x, y\}$. In either case, after having a shortest path from x to y we can proceed to compute in its vicinity a locally convex region on which a spline curve will be drawn. The methods discussed here are adaptable to this more general case but their correctness proofs become more intricate. Due to space limitations we defer the details to the journal version of this report.

The overall complexity of the proposed approach is *O(Visibility graph of vertices in free space)* + *O(SP*$[x, y]$*, Drawing Region)*. The second term depends on the quality of the desired approximation and we believe the first term is optimal in the amortized sense. An interesting related issue for further exploration is how to deal with the presence of collinearities. Currently, we can prove that, under the real arithmetic model, most of the steps to determine the drawing region are insensitive to the presence of collinearities. However, in certain situations where the shortest path contains a subsequence of "almost" collinear points, the resulting drawing region may exhibit subregions which are locally very small. To overcome this difficulty we have developed a parametrized version of the algorithm. The basic idea is to reapply the algorithm to those internal convex segments of the obtained boundary region as long as they satisfy a prespecified curvature constraint. This version, produces "better" results in the sense that it depends more heavily on the topology of the input polygon that on the presence of collinearities.

Acknowledgements. Thanks to Peter Eades and to Sandra Sudarsky for suggestions that helped to improve the readability of the paper.

References

1. J. Abello, O. Egecioglu, K. Kumar, "Visibility Graphs of Staircase Polygons and the Weak Bruhat Order I: From Polygons to Maximal Chains," *Discrete and Computational Geometry,* Vol. 14, No. 3, 1995, pp. 331-358.
2. J. Abello, K. Kumar, "Visibility Graphs and Oriented Matroids," In R. Tamassia, I. Tollis, editors, *Symposium on Graph Drawing GD'94, Princeton*, *Lecture Notes in Computer Science,* Vol. 894, 1994, pp. 147-158.
3. R. Bartels, J. Beatty, B. Barsky, An Introduction to Splines for Use in Computer Graphics and Geometric Modeling, Morgan Kaufman, Los Altos, California, 1987.
4. D. P. Dobkin, E. Gansner, E. Koutsofios, S. C. North, "Implementing a general-purpose edge router, " To appear in, *Proceedings of the Symposium on Graph Drawing GD'97, Rome*, Sept. 1997.
5. L. E. Dubis. "On Curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents, " *Amer. J. Math.,* 79:497-516, 1957.
6. S. Fortune and G. Wilfong, "Planning constraint motion," In *Proc. 20th Annu. ACM Sympos. Theory Comput.,* pp. 445-459, 1988.
7. E. R. Gansner, E. Koutsofios, S.C. North and K.P. Vo, "A technique for drawing directed graphs, " *IEEE Transactions on Software Engineering*, March 1993.
8. S. K. Ghosh and D. M. Mount, "An output-sensitive algorithm for computing visibility graphs", *Siam J. Computing,* 20(5):888-910, 1991.
9. J. Hershberger and S. Suri, "Efficient computation of Euclidean shortest paths in the plane," In *Proc. 34th Annu. IEEE Sympos. Found. Comput. Sci.,* pp. 508-517, 1993.
10. Y. Kanayama and B. I. Hartman, "Smooth local path planning for autonomous vehicles, " In *Proc. IEEE Intl. Conf. on Robotics and Automation*, Vol. 3, pp. 1265-1270, 1989.
11. J.C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, Boston, 1991.
12. J. P. Laumond, "Finding collision free smooth trajectories for a non-holonomic mobile robot, " In *Proc. IEEE Intl. Join Conf. on Artificial Intelligence*, pp. 1120-1123, 1987.
13. J. O'Rourke, "The Computational Geometry Column," *SIGACT News*, 1992.
14. J. O'Rourke, I. Streinu, "Pseudo Visibility Graphs," *Proc. ACM Symposium on Computational Geometry*, June 1997, France.
15. F. Preparata, M. Shamos, Computational Geometry: An Introduction, Springer Verlag, NY, 1995.
16. J. A. Reeds and L. A. Shepp, "Optimal paths for a car that goes both forward and backwards, " *Pacific Journal of Mathematics,*, 145(2), 1990.
17. G. Sander, M, Alt, A, Ferdinand, and R. Wilhelm, "Clax, a visualized Compiler, " In F. J. Brandenburg, editor, *Symposium on Graph Drawing GD'95*, Vol. 1027 of *Lecture Notes in Computer Science*, pp. 459-462, 1996.
18. J. T. Schwartz and M. Sharir, "Algorithmic motion planning in robotics," In J. van Leeuwen, editor, *Algorithms and Complexity*, Vol A of *Handbook of Theoretical Computer Science*, pp. 391-430. Elsevier, Amsterdam, 1990.