Available online at www.sciencedirect.com
SCIENCE DIRECT®

Computational Geometry ●●● (●●●●) ●●●–●●●

Computational Geometry Theory and Applications

www.elsevier.com/locate/comgeo

Polygonal path simplification with angle constraints

Danny Z. Chen [a,1], Ovidiu Daescu [b,*,2], John Hershberger [c], Peter M. Kogge [a], Ningfang Mi [b], Jack Snoeyink [d,3]

[a] *Department of Comp. Sci. and Eng., University of Notre Dame, Notre Dame, IN 46556, USA*
[b] *Department of Comp. Sci., Univ. of Texas at Dallas, Richardson, TX 75083, USA*
[c] *Mentor Graphics, 8005 S.W. Boeckman Road, Wilsonville, OR 97070, USA*
[d] *Department of Comp. Sci., Univ. of North Carolina at Chapel Hill, Chapel Hill, NC 27599, USA*

Received 7 April 2004; received in revised form 18 September 2004; accepted 18 September 2004

Communicated by T. Asano

Abstract

We present efficient geometric algorithms for simplifying polygonal paths in $\mathbb{R}^2$ and $\mathbb{R}^3$ that have angle constraints, improving by nearly a linear factor over the graph-theoretic solutions based on known techniques. The algorithms we present match the time bounds for their unconstrained counterparts. As a key step in our solutions, we formulate and solve an *off-line ball exclusion search* problem, which may be of interest in its own right.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Path simplification; Angle constraint; Computational geometry; Off-line search

* Corresponding author.
E-mail addresses: dchen@cse.nd.edu (D.Z. Chen), daescu@utdallas.edu (O. Daescu), john_hershberger@mentor.com (J. Hershberger), kogge@cse.nd.edu (P.M. Kogge), nxm024100@utdallas.edu (N. Mi), snoeyink@cs.unc.edu (J. Snoeyink).
[1] Chen's research was supported in part by the National Science Foundation under Grants CCR-9623585 and CCR-9988468.
[2] Daescu's research was supported in part by the National Science Foundation under Grant CCF-0430366.
[3] Snoeyink's research was supported in part by the National Science Foundation under Grants CCF-9988742 and ITR-0076984.

0925-7721/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.comgeo.2004.09.003

1. Introduction

We consider a common problem of simplifying a polygonal path or chain P in $\mathbb{R}^2$ or $\mathbb{R}^3$ by another polygonal path P' formed by an ordered subsequence of the vertices of P such that P' remains "close to" P. We add the additional constraint that any two consecutive line segments of P' are subject to some angle constraint. This constraint arises in general cartographic simplification, and in simplification that is specific to applications of robotics and vehicle routing. The question was first raised to us in the setting of airplane routing, in order to simplify flight paths without introducing sharp turns (limiting the maximum turn angle). On the other hand, one of the general simplification heuristics is to eliminate vertices with gradual turns (limiting the minimum turn angle). Below in this section, we first precisely define the problems considered, survey related work, and outline our results.

1.1. Problem definition

We define the turn angle for two consecutive segments $\overline{p_h p_i}$ and $\overline{p_i p_k}$ of P' as follows: let *ray*$(b|a)$ be the ray that extends the line segment $\overline{ab}$ from b to infinity and does not contain $\overline{ab}$. That is, *ray*$(b|a)$ is collinear with $\overline{ab}$ but extends from b away from a. The *turn angle* between $\overline{p_h p_i}$ and $\overline{p_i p_k}$ is defined as the minimum angle one needs to rotate *ray*$(p_i|p_h)$ around p_i to overlap with the line segment $\overline{p_i p_k}$.

For a line segment $\overline{ab}$ and a real number $\varepsilon \geqslant 0$, called the *tolerance*, we define the *error tolerance region* $R_\varepsilon(\overline{ab})$ as the set of points whose Euclidean distance from $\overline{ab}$ is at most ε. Based on these definitions, we formulate two problems, depending on whether small or large turn angles are disallowed on P'.

Given an arbitrary polygonal path $P = (p_1, p_2, \ldots, p_n)$ of n vertices, in $\mathbb{R}^2$ or $\mathbb{R}^3$, where any two consecutive vertices p_i, p_{i+1} on P are connected by the line segment $\overline{p_i p_{i+1}}$, for $1 \leqslant i < n$, find another polygonal path $P' = (p_1 = p_{i_1}, p_{i_2}, \ldots, p_{i_m} = p_n)$ of m vertices ($m < n$), satisfying the following conditions:

(1) The integer indices satisfy $1 = i_1 < i_2 < \cdots < i_{m-1} < i_m = n$.
(2) For every $j = 1, 2, \ldots, m-1$, the subpath $P_{i_j, i_{j+1}} = (p_{i_j}, p_{i_j+1}, \ldots, p_{i_{j+1}})$ of P is entirely contained in the error tolerance region $R_\varepsilon(\overline{p_{i_j} p_{i_{j+1}}})$, for a given tolerance $\varepsilon \geqslant 0$.
(3.min) The turn angle for any two consecutive line segments $\overline{p_h p_i}$ and $\overline{p_i p_k}$ on P' is at least a specified value δ, with $0 \leqslant \delta(\overline{p_h p_i}) < \pi/2$. This is the *min turn angle* case as illustrated in Fig. 1 for $\mathbb{R}^2$.
(3.max) The turn angle for any two consecutive line segments $\overline{p_h p_i}$ and $\overline{p_i p_k}$ on P' is at most a specified value δ, with $\pi/2 \leqslant \delta(\overline{p_h p_i}) < \pi$. This is the *max turn angle* case.

The problem version of limiting the *min turn angle* models the situation in which turns of small angles are eliminated—this gives simplifications that better preserve the character of the original line by making sure that all turns are justified or make all course corrections for a vehicle be greater than a given mechanical accuracy. The problem version of limiting the *max turn angle* models the situation in which a robot or vehicle cannot make a very sharp turn (e.g., a car or airplane). It will become clear later on that the two problem versions are related to each other so that the solution for one version also solves the other. Hence, we mainly discuss the min turn angle problem.

The problem is a generalization of a well-studied polygonal path simplification problem [2,4,5,7,8, 11–13,16,20–26], in which neither of the third condition above is considered. Our algorithms report a

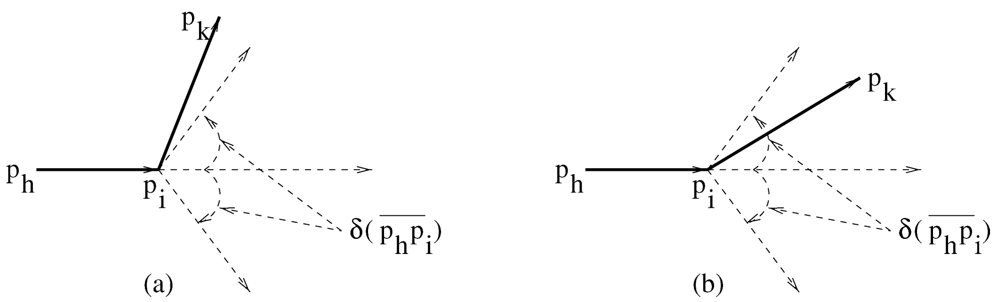

Fig. 1. Illustrating the angle constraint condition for the *min turn angle* case in $\mathbb{R}^2$: (a) edges $\overline{p_h p_i}$ and $\overline{p_i p_k}$ can be consecutive in the path P', and (b) edges $\overline{p_h p_i}$ and $\overline{p_i p_k}$ cannot be consecutive in P'.

path satisfying the angle constraints, if one exists, or otherwise report that no such path exists. (Although the input path P always satisfies the first two constraints, it need not satisfy the third one.)

A path simplification problem has two coupled parameters: m, the number of vertices of P', and ε, the tolerance. When ε is made smaller, then m tends to become larger, and ε tends to be larger when m is made smaller. This trade-off between m and ε gives rise to two different optimization problems that we will consider:

(1) *Min-# problem*: Given a polygonal path P and a real number $\varepsilon > 0$, find an ε-simplification path P' with the smallest number of vertices (given ε, minimize m).
(2) *Min-ε problem*: Given a polygonal path P and an integer $m < n$, find a simplification path P' with at most m vertices that minimizes the error ε between P' and P.

Different error criteria have been used for simplifying polygonal paths (e.g., see [7,8,11]). The error criterion used in [20,22,23] and this paper, called the *tolerance zone* criterion [7], is one of the most natural definitions. Under this criterion, if a subpath $P_{i_j,i_{j+1}} = (p_{i_j}, p_{i_j+1}, \ldots, p_{i_{j+1}})$ of P is completely contained in the ε-tolerance region of the line segment $R_\varepsilon(\overline{p_{i_j} p_{i_{j+1}}})$, then we say that $\overline{p_{i_j} p_{i_{j+1}}}$ is an ε-simplifying line segment for $P_{i_j,i_{j+1}}$. The path P' is an ε-simplification of P if each line segment $\overline{p_{i_j} p_{i_{j+1}}}$ of P' is an ε-simplifying line segment for $P_{i_j,i_{j+1}}$, for $j = 1, 2, \ldots, m-1$.

Other commonly used error criteria include the *infinite beam* criterion [11,16,22,26] and the *uniform measure* criterion [4]. Under the *infinite beam* criterion, the ε-tolerance zone of a line segment $\overline{p_i p_j}$ is the region consisting of the set of points that are at distance no larger than ε from the line $L(\overline{p_i p_j})$ supporting $\overline{p_i p_j}$. In $\mathbb{R}^2$, for monotone paths, under the *uniform measure* criterion, the simplification error between a line segment $\overline{p_{i_j} p_{i_{j+1}}}$ of P' and the corresponding subpath $P_{i_j,i_{j+1}}$ of P is defined as $\max\{d(p_k, \overline{p_{i_j} p_{i_{j+1}}}) \mid i_j \leqslant k \leqslant i_{j+1}\}$, where $d(p_k, \overline{p_{i_j} p_{i_{j+1}}})$ denotes the vertical distance between p_k and $\overline{p_{i_j} p_{i_{j+1}}}$.

1.2. Previous work

A number of results for the polygonal path simplification problem, under various error criteria, have been presented by Imai and Iri [20–22], Melkman and O'Rourke [23], and Toussaint [26]. Imai and Iri [22] formulated the problem in terms of graph theory: construct a model of an unweighted directed acyclic graph for path simplification, and then use breadth-first search to compute a shortest path in this graph. This has later been exploited by most of the algorithms devoted to the problem [4,7,8,11,16]. A notable exception, for the planar case, is the work of Agarwal and Varadarajan [4], which uses a divide and conquer approach to achieve an $O(n^{4/3+\delta})$ time and space complexity, where $\delta > 0$ is an arbitrarily small constant. However, their algorithms work only for the L_1 distance metric and do not extend to higher dimensions. The most popular heuristic method that is used in path simplification, the recursive simplification heuristic of Douglas and Peucker [14], can be implemented in $O(n \log^* n)$ time in $\mathbb{R}^2$ [19], but does not guarantee an optimal solution. If the vertices of the simplifying path are not required to be a subset of the vertices of the input path, then faster algorithms are possible [17,18,21]. Approximate solutions for the min-# problem have also been considered. In [2], near-linear time algorithms are proposed for computing a simplifying path with vertices among those of P. Other somewhat related problems (e.g., off-line ball inclusion testing [7], off-line range searching [9]) have been studied recently.

Solutions to some subdivision simplification problems are also based on polygonal path simplification. In [13], polygonal path simplification has been used to simplify a planar subdivision S with N vertices and M extra points in $O(N(N+M)\log N)$ time. If a minimum size simplification is sought, the problem becomes NP-hard [17]. Unless P = NP, one cannot obtain in polynomial time a simplification within a factor of $n^{1/5-\delta}$ of an optimal solution, for any $\delta > 0$ [15].

1.3. Our results

While there are known results on polygonal path simplification in $\mathbb{R}^2$ and $\mathbb{R}^3$, *without* angle constraints (e.g., [7,11]) or on curvature-constrained geometric paths (mainly in $\mathbb{R}^2$, e.g., [1]), we are not aware of any published work on the specific problems we consider. However, one may use known graph-theoretic techniques [21] to reduce the problem to that of computing shortest paths in a graph with $O(n^2)$ vertices and $O(n^3)$ edges.

We present efficient algorithms for solving the polygonal path simplification problem with angle constraints in $\mathbb{R}^2$ and $\mathbb{R}^3$, with time bounds matching those of the best known path simplification algorithms without angle constraints [7,8,11]. The algorithms we present improve by nearly a linear factor in the time bound over the possible solutions based on graph-theoretic techniques mentioned above.

The running times of our min-# algorithms are $O(n^2)$ in $\mathbb{R}^2$ and $O(n^2 \log n)$ in $\mathbb{R}^3$. The time bounds of our min-# algorithms crucially depend on how fast we can solve some special instances of a certain 1-dimensional (for $\mathbb{R}^2$) or 2-dimensional (for $\mathbb{R}^3$) off-line search problem which we refer to as the *off-line ball exclusion search* (OLBES) problem (more on the OLBES problem in Section 2).

We develop efficient data structures that solve the general 1-dimensional and 2-dimensional OLBES problems in $O(n \log n)$ time. We also show that for the special instance of the 1-dimensional OLBES problem that results from the $\mathbb{R}^2$ version of the path simplification problem with angle constraints, we can reduce the time bound from $O(n \log n)$ to $O(n)$. Our OLBES data structures can also handle on-line point queries in $O(\log n)$ time each. Further, our solutions can be easily extended to other types of objects (such as bounded convex objects with boundary described by a constant number of polynomial functions of maximum degree bounded by a small constant), and can be applied to a class of geometric paths and other related problems.

Using techniques similar to those for the unconstrained case [7,11], the min-ε problem in $\mathbb{R}^2$ and $\mathbb{R}^3$ can be solved in time $O(n^2 \log n)$ and $O(n^2 \log^3 n)$, respectively.

The rest of the paper is organized as follows. In Section 2 we show how to reduce the min-# problem to solving $O(n)$ OLBES problems. In Sections 3 and 4 we develop efficient data structures and discuss our algorithms for solving the $\mathbb{R}^2$ and $\mathbb{R}^3$ OLBES problems, respectively. In Section 5 we give some remarks on the on-line query version. We conclude the paper in Section 6.

2. Algorithmic paradigm: reduction to OLBES

In this section, we explain our algorithmic approach for angle-constrained path simplification. To solve the unconstrained polygonal path simplification problems in $\mathbb{R}^2$ and $\mathbb{R}^3$, the known solutions [7,8] build a directed acyclic graph $G_P = (V_P, E_P)$ for P, where V_P is the vertex set of P and the $O(n^2)$ edges of E_P are all valid simplifying segments for their corresponding subpaths of P, and compute a shortest path from p_1 to p_n in G_P. For the angle-constrained versions, the main difficulty is how to compute the desired shortest paths in G_P.

2.1. *Overview of the algorithmic approach*

As the previous algorithms for the unconstrained min-# problem (e.g., [7,8]), we divide the min-# problem into two subproblems, as follows:

(1) Build an $O(n^2)$ size directed acyclic graph $G_P = (V_P, E_P)$ for P, such that E_P consists of all ε-simplifying segments.
(2) Compute a shortest path from p_1 to p_n in G_P, satisfying the angle constraint, if such a path exists, or otherwise report that no solution exists.

We solve the first subproblem by applying the best known iterative min-# algorithms for the unconstrained version. Those algorithms compute the set of ε-simplifying segments in $O(n^2)$ time in $\mathbb{R}^2$ [8,11] and $O(n^2 \log n)$ time in $\mathbb{R}^3$ [7].

To compute a shortest p_1-to-p_n path in G_P, we use dynamic programming as the main technique. This enables us to formulate as a key subproblem a special off-line range search problem: Given n weighted balls of arbitrary radii and n points, for each point p find the minimum-weight ball that does not contain p. One can use standard circular range search techniques to solve this problem (e.g., in $\mathbb{R}^2$ using range search queries would result in $O(n^{1+\varepsilon})$ time algorithms, where $\varepsilon > 0$ is an arbitrarily small constant [3]). We exploit the special properties of this range search problem to achieve better time bounds:

OLBES (*Off-Line Ball Exclusion Search*): Given a sequence $\mathcal{E} = (e_1, e_2, \ldots, e_n)$ such that each e_i, $i = 1, 2, \ldots, n$, is either a ball B_i of arbitrary radius or a point p_i, for every point $p_k \in \mathcal{E}$, find the smallest-index ball $B_j \in \{e_1, e_2, \ldots, e_{k-1}\}$ such that $p_k \notin B_j$, or report no such ball exists.

As we will see later, for path simplification in two dimensions, the OLBES balls are arcs on $\mathbb{S}^1$. For path simplification in three dimensions, the balls are disks on $\mathbb{S}^2$.

2.2. *The reduction*

Suppose we are given a polygonal path $P = (p_1, p_2, \ldots, p_n)$ in $\mathbb{R}^2$ or $\mathbb{R}^3$ and an error value $\varepsilon > 0$, and we want to find a minimum size path $P' = (p_{i_1} = p_1, p_{i_2}, \ldots, p_{i_m} = p_n)$ which satisfies the angle constraint condition under the tolerance zone criterion. As mentioned earlier, we first construct an $O(n^2)$ size directed acyclic graph $G_P = (V_P, E_P)$ for P that contains all valid simplifying segments for the unconstrained problem, by using the algorithms in [7,11]. Note that it is fairly straightforward to compute an angle-constrained shortest path in G_P in $O(n^3)$ time by applying the same idea used in computing shortest paths with turn penalties [6]: construct the *dual graph* G'_P of G_P in which the edges of G_P become nodes and two nodes of G'_P are connected by an edge in G'_P if they correspond to a possible turn in G_P. It is easy to see that the dual graph G'_P is also a directed acyclic graph and that it has $O(n^2)$ vertices and $O(n^3)$ edges. The dual graph has no turn penalties and shortest paths in G'_P can be computed by standard techniques.

To obtain a faster solution, we compute an angle-constrained shortest p_1-to-p_n path in G_P by a dynamic programming algorithm. Let $ACSP_j(k)$ denote the angle-constrained shortest path from p_1 to p_k in G_P, with $1 < k \leqslant n$, such that the last edge of $ACSP_j(k)$ is $\overline{p_j p_k}$. Suppose at the end of iteration i ($i \geqslant 1$), $ACSP_j(k)$ is available for every $j = 1, 2, \ldots, i$ and every $k = 2, 3, \ldots, n$ such that $j < k$. For example, in Fig. 2, there are two available shortest paths $ACSP_j(k)$ and $ACSP_{j'}(k)$ with the last edges $\overline{p_j p_k}$ and $\overline{p_{j'} p_k}$, respectively. At iteration $i + 1$, from the available $ACSP_j(i + 1)$'s, $j = 1, 2, \ldots, i$, we compute $ACSP_{i+1}(k)$ for every $k = i + 2, i + 3, \ldots, n$. Dynamic programming enables us to compute $ACSP_{i+1}(k)$, for $k = i + 2, i + 3, \ldots, n$, using batched off-line computation.

At iteration $i + 1$, we have (at most) i available shortest paths $ACSP_j(i + 1)$, with the last edge $\overline{p_j p_{i+1}}$, where $j < i + 1$ and $\overline{p_j p_{i+1}}$ is an incoming edge to p_{i+1} in G_P. To decide if an outgoing edge $\overline{p_{i+1} p_k}$ of p_{i+1}, where $i + 1 < k$, can succeed $\overline{p_j p_{i+1}}$ to extend an angle-constrained path in G_P, we should

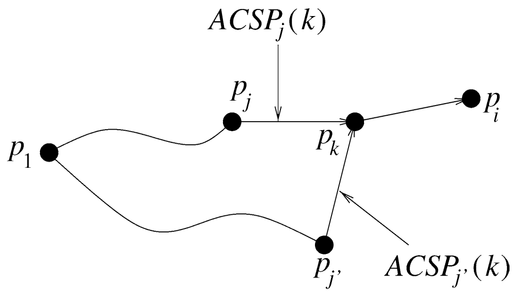

Fig. 2. An example of two available shortest paths $ACSP_j(k)$ and $ACSP_{j'}(k)$.

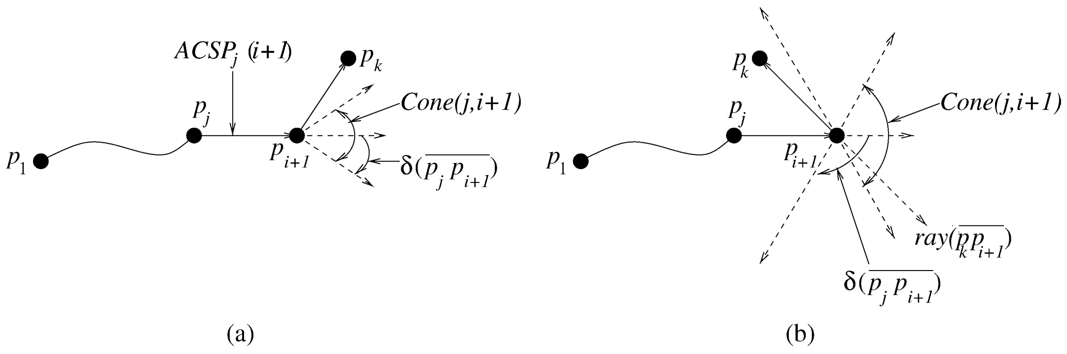

Fig. 3. The cone $Cone(j, i + 1)$ when (a) $\delta(\overline{p_j p_{i+1}}) < \frac{\pi}{2}$ and (b) $\delta(\overline{p_j p_{i+1}}) \geqslant \frac{\pi}{2}$.

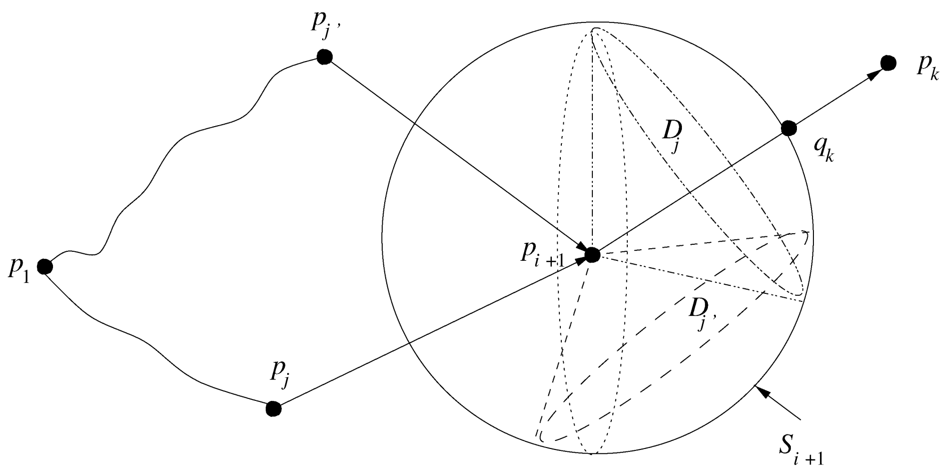

Fig. 4. Illustrating the reduction to the OLBES problem.

check if the turn angle between $\overline{p_j p_{i+1}}$ and $\overline{p_{i+1} p_k}$ is no smaller (or no larger) than the specified value δ. Based on our definition of the angle constraint condition, the set of directions at p_{i+1} which make an angle δ defines a cone of directions at p_{i+1}. For the *min turn angle* constraint, angle $\delta < \pi/2$ is acute, as depicted in Fig. 3(a). We define the cone $\mathit{Cone}(j, i+1)$ as the cone of directions at p_{i+1}, and then $\overline{p_{i+1} p_k}$ can succeed $\overline{p_j p_{i+1}}$ if and only if $\mathit{ray}(\overline{p_{i+1} p_k})$ is not contained in the cone $\mathit{Cone}(j, i+1)$. For the *max turn angle* constraint, angle $\delta \geqslant \pi/2$ is obtuse, and we can derive the same constraint on an opposite ray. Consider the set of directions that do not satisfy the angle constraint. These directions define a cone $\mathit{Cone}'(j, i+1)$ with an acute angle at p_{i+1}. $\mathit{Cone}(j, i+1)$ is the cone of directions that are the opposite of the directions in $\mathit{Cone}'(j, i+1)$. Then, $\overline{p_{i+1} p_k}$ can succeed $\overline{p_j p_{i+1}}$ if and only if $\mathit{ray}(p_{i+1}|p_k)$ is not contained in the cone $\mathit{Cone}(j, i+1)$. It should be clear now that, after this slight modification, an algorithm for solving the min turn angle case also solves the max turn angle case of the problem. Thus, we discuss only the min turn angle case in what follows.

Let S_{i+1} denote the unit sphere ($\mathbb{S}^1$ or $\mathbb{S}^2$) with center at p_{i+1} and let D_j denote the disk on S_{i+1} obtained by intersecting $\mathit{Cone}(j, i+1)$ with S_{i+1}. We associate with D_j a weight w_j equal to the length of the shortest path $\mathit{ACSP}_j(i+1)$ along the corresponding incoming edge $\overline{p_j p_{i+1}}$ to p_{i+1}. At iteration $i+1$, by intersecting each cone $\mathit{Cone}(j, i+1)$ $(j < i+1)$ with S_{i+1}, we have (at most) i weighted disks D_h of different radii. Let q_k be the intersection point of S_{i+1} with $\mathit{ray}(\overline{p_{i+1} p_k})$. At iteration $i+1$, by intersecting S_{i+1} with the ray corresponding to an outgoing edge $\overline{p_{i+1} p_k}$ of p_{i+1} $(i+1 < k)$, we have (at most) $n-i-1$ points q_r on S_{i+1}. For every q_r, we then find the minimum weight disk D_h such that $q_r \notin D_h$ (see Fig. 4 for an example in $\mathbb{R}^3$).

The problem above can be reduced to a special case of the OLBES problem. We first sort the (at most) i disks D_h in the order of nondecreasing weights and place the ordered disk sequence into an initially empty set $\mathcal{E}$. We then attach the (at most) $n-i-1$ points q_r at the end of $\mathcal{E}$. Thus, we obtain a sequence $\mathcal{E} = (e_1, e_2, \ldots, e_n)$ of disks and points such that the first i objects of the sequence are disks ordered by their weights and the remaining objects of the sequence are points. For a point $q_r \in \mathcal{E}$, finding the minimum weight disk not containing that point corresponds to finding the smallest index disk $D_h \in \mathcal{E}$ such that $q_r \notin D_h$.

S0925-7721(04)00139-7/FLA AID:777 Vol.●●●(●●●)
COMGEO:m2 v 1.32 Prn:21/01/2005; 13:33 cgt777 [DTD5] P.8 (1-15)
by:Gi p. 8

Lemma 1. *For min or max turn angle constraints, an angle-constrained p_1-to-p_n shortest path in G_P can be computed in $O(nT(n))$ time, where $T(n)$ is the time for solving a special OLBES problem of size $O(n)$.*

Proof. This is done by using the above dynamic programming algorithm for computing the angle-constrained p_1-to-p_n shortest path in G_P. Since the algorithm has $O(n)$ steps and in each step we have a special OLBES problem, there are $O(n)$ OLBES problems to solve, each of which is of size $O(n)$. □

Thus, in order to obtain the claimed time bounds for the path simplification problem with angle constraints, it remains to show how to efficiently solve the corresponding instances of the OLBES problem.

3. The 1-dimensional OLBES problem

In Section 2, we have reduced the problem of simplifying a polygonal path with angle constraints to the problem of solving $n-1$ individual instances of the off-line ball exclusion search problem. In this section, we discuss the solution for the 1-dimensional OLBES problem.

Theorem 1. *Given a sequence $\mathcal{E}$ of n 1-dimensional disks and n query points on the real line, we can determine the OLBES answers for all the query points in $\mathcal{E}$ in $O(n \log n)$ time.*

Proof. We build a complete binary tree T such that each leaf of T is associated with a disk D_j in $\mathcal{E}$ (the first leaf for D_1, the second leaf for D_2 and so on). Note that each 1-dimensional disk is an interval on the real line. We go up the tree T in a bottom-up fashion, and at each internal node v we compute and store the common intersection of the intervals associated with all leaf descendants of v. Since the common intersection at v can be computed in constant time from the common intersections associated with v's left and right children, the overall computation in T takes $O(n)$ time. Obviously, the common intersection at the root of T may be empty.

We sort the points (real values) in $\mathcal{E}$ increasingly and then search for the desired disks for all the $O(n)$ query points q_k at once, starting at the root of T. At the root u of T, we select the query points that do not fall in the interval stored at the root. We partition these points into two sets, based on inclusion in the interval stored at the left child u_l of the root: the points that do not fall in this interval are placed in the subset *left*(u), for u_l (the root of the left subtree); a point that falls in the interval is placed in the subset *right*(u), for the right child u_r of the root (the root of the right subtree), if its index is larger than the index of the rightmost leaf of the subtree rooted at u_l, otherwise it is dropped (since it is inside all disks in $\mathcal{E}$ preceding it). The index of the rightmost leaf of the subtree of T rooted at v, for all $v \in T$, can be computed in $O(n)$ time by a simple traversal of T. Obtaining *right*(u) (and then *left*(u)) reduces to extracting a sorted subsequence from a sorted sequence. Thus, the computation at each level of T can be performed in $O(n)$ time and the 1-D OLBES problem can be solved in $O(n \log n)$ time. □

Lemma 2. *For a sequence $\mathcal{E}$ of n 1-dimensional disks and n query points on the real line, such that* (i) *the left and right endpoints of the disks form sorted sequences and* (ii) *the points appear after all the disks in $\mathcal{E}$ and are sorted by their values, we can find the answers for all the points in $\mathcal{E}$ in $O(n)$ time.*

Proof. We solve this special instance of the OLBES problem in a different way. To find the minimum index disk that does not contain the point q_k, for each of the points $q_k \in \mathcal{E}$, we use the following transformation to a *value-index* coordinate system. Each endpoint of an interval $\mathcal{I}_j$ is mapped to a point with coordinates (x_j, w_j) where x_j is the value of the endpoint on the real axis and w_j is the index of the interval in $\mathcal{E}$. Each point $q_k \in \mathcal{E}$ is mapped to a point $(x_k, 0)$ in the *value-index* plane. Thus, we obtain three sets of points in *value-index* coordinates, each of which is sorted by the value coordinate: the set L_j for the left endpoints, the set R_j for the right endpoints, and the set Q_j corresponding to the points in $\mathcal{E}$.

For a point $q_k \in Q_j$ let $\mathit{min\_right}(q_k)$ denote the minimum value of the indices of the endpoints in R_j which are on or to the left of the vertical line at q_k. We set $\mathit{min\_right}(q_k)$ to zero if there is no right endpoint to the left of q_k. Then, $\mathit{min\_right}(q_k)$, for each $q_k \in Q_j$, can be computed in $O(n)$ time by first merging the two sequences corresponding to the value coordinates of the points in R_j and Q_j and then performing a left-to-right scan on the resulting sequence. Similarly, let $\mathit{min\_left}(q_k)$ denote the minimum value of the indices of the endpoints in L_j which are on or to the right side of the vertical line at q_k. We set $\mathit{min\_left}(q_k)$ to zero if there is no left endpoint to the right of q_k. Then, $\mathit{min\_left}(q_k)$, for each $q_k \in Q_j$, can be computed in $O(n)$ time by first merging the two sequences corresponding to the value coordinates of the points in L_j and Q_j and then performing a right-to-left scan on the resulting sequence. A minimum index disk that does not contain q_k corresponds to $\min\{\mathit{min\_left}(q_k), \mathit{min\_right}(q_k)\}$. If the value is zero then there is no disk. □

We next outline some properties of the reduction to the OLBES problem for the special case when we have the same angle constraint for all incoming edges at p_{i+1}, which gives equal-radius disks on S_{i+1}.

Lemma 3. *Consider the iteration $i+1$ of the dynamic programming algorithm, for any i such that $1 \leqslant i < n-1$, and assume that the incoming edges and the outgoing edges of G_P at p_{i+1} form two sorted sets, respectively. Assume also that all incoming edges at p_{i+1} have the same angle constraint with respect to the outgoing edges. Then*:

(1) *the points corresponding to the outgoing edges appear in sorted order on the boundary of S_{i+1}, and*
(2) *the endpoints of the disks on S_{i+1} form two sorted sequences.*

Proof. Since the outgoing edges at p_{i+1} are sorted, their intersections with S_{i+1} appear in the same sorted order. The 1-dimensional disks on S_{i+1} have centers at the intersection points of S_{i+1} with the rays through p_{i+1} corresponding to incoming edges. Since these edges are sorted around p_{i+1}, the center points are also sorted on the boundary of S_{i+1}. Then, the left and right endpoints of the disks, respectively, form two sorted sequences on S_{i+1}. □

From Lemma 3, it follows that by requiring the same angle constraint for all incoming edges at p_{i+1}, the sorted order of the left and right endpoints of the disks corresponds to the sorted order of the centers of the disks. Since G_P can be viewed as a planar geometric graph, the sorted order of the incoming and outgoing edges for all vertices in G_P can be obtained in $O(n^2)$ time by standard line arrangement traversal (of the dual line arrangement of the points p_i, $i = 1, 2, \ldots, n$). Once the incoming edges at p_{i+1} are sorted, the sorted order of the centers of the disks (and thus the sorted order of their left and right endpoints, respectively) can be obtained in linear time. For the rest of this section, for the OLBES problem at p_{i+1}, we assume that the incoming and outgoing edges are available in sorted order around p_{i+1}.

In what follows, we describe our solution for the special instance of the 1-dimensional OLBES problem. Suppose that at iteration $i+1$, we have i cones $Cone(j, i+1)$ and $n-i-1$ points p_k, where each cone $Cone(j, i+1)$ is defined by two rays emanating from p_{i+1} and it is weighted by the length of the corresponding shortest path $ACSP_j(i+1)$. For each cone $Cone(j, i+1)$, we obtain a circular arc $\mathcal{A}_j$ by intersecting $Cone(j, i+1)$ with the circle C_{i+1} with the center point p_{i+1} (the planar correspondent of S_{i+1}). The circular arc $\mathcal{A}_j$ is associated with the weight $ACSP_j(i+1)$. For each point p_k, we obtain a query point q_k by intersecting the ray emanating from p_i and going through p_k with the circle C_{i+1}. As discussed above, these circular arcs and points form the sequence $\mathcal{E}$. We sort the circular arcs by weight. Since there are at most i weight values, each of which is an integer no larger than i, the circular arcs can be sorted by weight in $O(i)$ time. The circular arcs, in sorted order of weight, followed by the query points (sorted by the angle at p_{i+1}) form the sequence $\mathcal{E}$.

Lemma 4. *At iteration i, $1 \leqslant i < n$, we can find an angle-constrained shortest path $ACSP_i(k)$ for every $k > i$ in $O(n)$ time.*

Proof. We build a complete binary tree T_a such that each leaf of T_a is associated with a circular arc $\mathcal{A}_j$ in $\mathcal{E}$, as in the proof of Theorem 1. Since by the definition of the angle constraint no circular arc is larger than an open semicircle, the common intersection of any two arcs is either empty or consists of one circular arc. Then, the computation in T_a requires $O(n)$ time. We also build a balanced binary search tree T_q on the set of points in $\mathcal{E}$, based on their angle value around p_i. In the query procedure, we use an arc in T_a as a query input. We start with the arc stored at the root of T_a and use the endpoints of this arc to form two search paths in T_q that isolate the points $q_k \in T_q$ that are on the arc. These points are eliminated from T_q. Since each search path has length $O(\log n)$, this computation can be performed in $O(\log n)$ time. We next partition the points in T_q into two subtrees, based on inclusion in the interval $\mathcal{A}_l$ stored at the left child u_l of the root. The points that do not fall in the interval $\mathcal{A}_l$ are placed in the tree T_{u_l}, for u_l. The other points are placed in the tree T_{u_r}, for the right child u_r of the root. T_{u_l} and T_{u_r} are obtained from T_q in $O(\log n)$ time. Note that the points in one of these two trees may be initially available in two subtrees (see Fig. 5). To obtain a single tree we treat this case as a standard delete operation at the root of a balanced binary search tree.

Let k be the number of points in T_{u_l} and let $n-k$ be the number of points in T_{u_r}. We proceed recursively by querying the points in T_{u_l} with the arc stored at the left child of u_l and querying the points in T_{u_r} with the arc stored at the left child of u_r. We now analyze the running time of this procedure. We perform $O(n)$ queries, corresponding to the arcs in T_a. A query can be performed in $O(\log(|T_q|))$ time, where $|T_q|$ denotes the number of points in T_q. After the first query, the points are placed in T_{u_l} and T_{u_r}, of size k and $n-k$, respectively. Then, the time to query T_{u_l} and T_{u_r} is $\log k + \log(n-k)$ which is maximized when $k = \frac{n}{2}$. This gives the recurrence $T(n) = 2T(\frac{n}{2}) + O(\log n)$ whose solution is $T(n) = O(n)$. $\square$

Theorem 2. *The min-# problem with angle constraints in $\mathbb{R}^2$ can be solved in $O(n^2)$ time using $O(n^2)$ storage.*

Proof. There are $O(n)$ iterations in the dynamic programming algorithm and in each iteration we spend $O(n)$ time to compute $ACSP_j(i+1)$'s, as specified in Lemma 4, resulting in a total of $O(n^2)$ time. $\square$

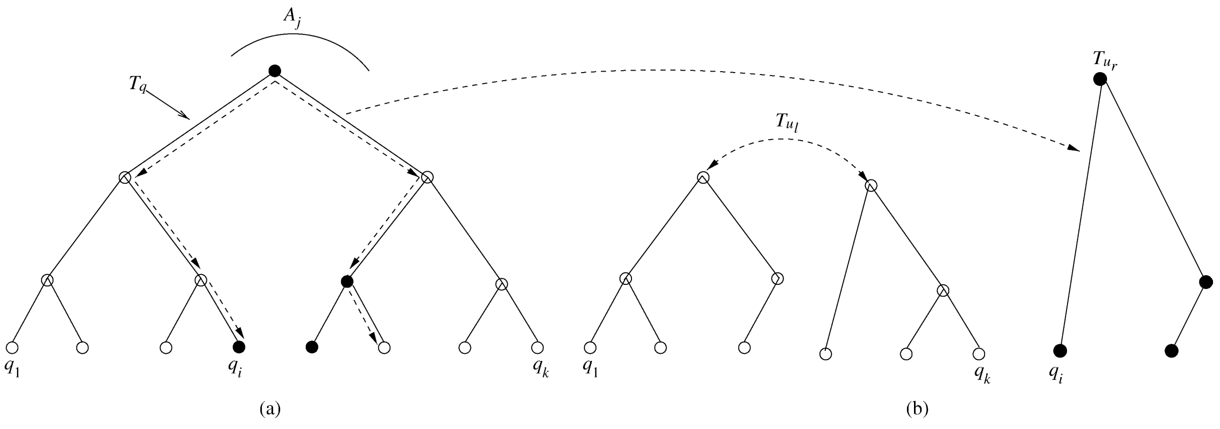

Fig. 5. (a) Querying T_q with an arc $\mathcal{A}_j \in T_a$ and (b) the resulting trees T_{u_l} and T_{u_r}.

Theorem 3. *The min-ε problem with angle constraints in $\mathbb{R}^2$ can be solved in $O(n^2 \log n)$ time using $O(n^2)$ storage.*

Proof. The solution is similar to that for the unconstrained version. The set of $O(n^2)$ possible simplification errors can be computed in $O(n^2 \log n)$ time [11]. By using binary search on the set of $O(n^2)$ possible simplification errors, at each step of the search running the (angle-constrained version of the) min-# algorithm, the min-ε problem with angle constraint in $\mathbb{R}^2$ can be solved in $O(n^2 \log n)$ time, matching the best bound for the unconstrained version. □

4. The 2-dimensional OLBES problem

Recall that the 2-dimensional OLBES problem that results from our modeling of the min-# problem with angle constraints is a special case of the general 2-dimensional OLBES problem. In this special instance the points in $\mathcal{E}$ appear after all the disks in $\mathcal{E}$. Specifically, if $\mathcal{E}$ has k disks and $n-k$ points, then the first k elements of $\mathcal{E}$ are disks and the remaining $n-k$ elements of $\mathcal{E}$ are points. Also, recall that the disks of $\mathcal{E}$ are spherical disks on $\mathbb{S}^2$. However, since each disk is less than a semi-sphere we can treat the problem as one on disks in $\mathbb{R}^2$ (details on this treatment are provided in [7]).

Clearly, the min-# problem with angle constraints in $\mathbb{R}^3$ can be solved in $O(n^2 \log n)$ time if we can solve the 2-dimensional OLBES problem in $O(n \log n)$ time. In what follows we first prove how to solve this special case and then show how to modify the solution to handle the general case.

We build a complete binary tree T such that each leaf of T is associated with a disk D_j in $\mathcal{E}$ (the first leaf for D_1, the second leaf for D_2 and so on). We go up the tree T in a bottom-up fashion, and at each internal node v we compute and store the common intersection of the disks associated with all leaf descendants of the subtree of T rooted at v. Using the algorithm in [7], we can compute $I(v)$ for each node v in T in a total of $O(n \log n)$ time. Essentially, the algorithm in [7] uses merge-like operations to compute the common intersection at v from the common intersections stored at the two children of v. Clearly, since the intersection of disks is a convex region, the upper and lower chains can be maintained in sorted order and the merging at each level can be performed in linear time, resulting in $O(n \log n)$ time

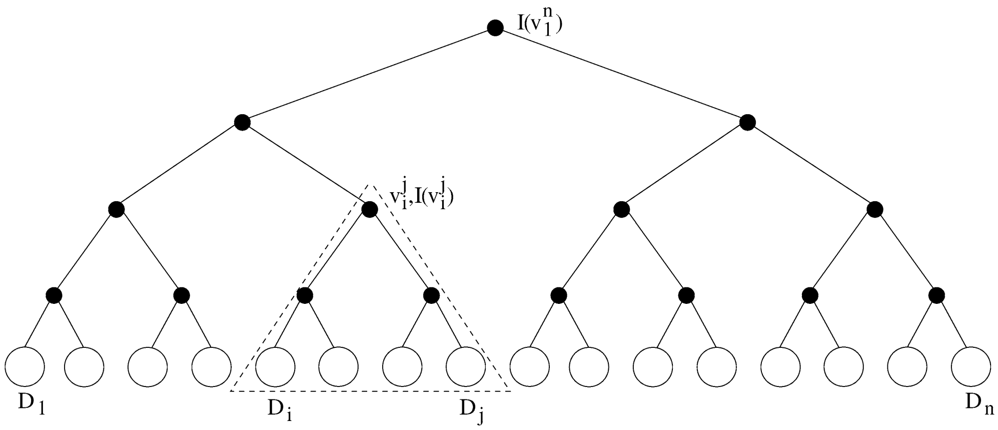

Fig. 6. Illustrating T and $I(v_i^j)$ for the 2-D OLBES problem.

over the $O(\log n)$ levels. If $I(v_i^j)$ denotes a representation of the common intersection stored at internal node v_i^j, e.g., $I(v_1^i)$ represents the common intersection of $\{D_1, D_2, \ldots, D_i\}$ and $I(v_{i+1}^n)$ represents the common intersection of $\{D_{i+1}, D_{i+2}, \ldots, D_n\}$, then the common intersection of all the disks is denoted by $I(v_1^n)$ and it is stored at the root of the tree T (see Fig. 6). Obviously, $I(v_1^n)$ may be empty.

We sort the points in $\mathcal{E}$ by x-coordinate and then search for the desired disks for all the $O(n)$ query points q_r at once, starting at the root of T. Observe that the searching for the smallest-index disk D_h that does not contain a query point q_r traverses a root-to-leaf path in T. At the root of T, we select the query points that do not fall inside the common intersection stored at the root. We partition those points into two sets, based on inclusion in the common intersection at the left child of the root. With the points that are not in the common intersection at the left child of root we proceed on the left subtree of the root; with the remaining points we proceed on the right subtree. Since the computation at each level of T can be done altogether in $O(n)$ time, by merge-like operations, this special 2-D OLBES problem can be solved in $O(n \log n)$ time. To solve the general 2-D OLBES problem we observe that we only need the following simple change. For each internal node $v \in T$, store the index in $\mathcal{E}$ of the rightmost leaf of the subtree rooted at v, which can be done in $O(n)$ time by a bottom-up traversal of T. If a point in the set for the right subtree has its index smaller than the index in $\mathcal{E}$ of the rightmost leaf in the left subtree, then we drop this point (since it is inside all disks preceding it).

Theorem 4. *Given a sequence $\mathcal{E}$ of n disks and n query points, we can determine the OLBES answers for all query points in $\mathcal{E}$ in $O(n \log n)$ time.*

Combining the dynamic programming approach and the reduction to 2-D OLBES problem in Section 2 with the solution for the 2-D OLBES problem, we obtain:

Theorem 5. *The min-# problem with angle constraints in $\mathbb{R}^3$ can be solved in $O(n^2 \log n)$ time using $O(n^2)$ storage.*

Proof. There are $O(n)$ iterations in the dynamic programming algorithm and in each iteration we spend $O(n \log n)$ time to compute $ACSP_j(i+1)$'s, as it follows from Theorem 4, for a total of $O(n^2 \log n)$ time. □

Theorem 6. *The min-ε problem with angle constraints in $\mathbb{R}^3$ can be solved in $O(n^2 \log^3 n)$ time using $O(n^2)$ storage.*

Proof. The solution is again similar to that for the unconstrained version. We apply the parametric search approach used in [7] for the unconstrained version. The only difference is that, in the decision step of the parametric search, the sequential algorithm for the unconstrained min-# problem is replaced with that for the angle-constrained version. Since the two min-# algorithms have the same time bounds we obtain the claimed time. □

5. Remarks

We can modify our solutions for the OLBES problem to solve the on-line (query) version: given a sequence $\mathcal{E} = (e_1, e_2, \dots, e_n)$ of balls in $\mathbb{R}^1$ or $\mathbb{R}^2$, preprocess $\mathcal{E}$ such that for a query point q, one can efficiently find the smallest index ball in $\mathcal{E}$ that does not contain q.

We have the following results.

Theorem 7. *A sequence $\mathcal{E}$ of n 1-dimensional balls can be preprocessed in $O(n)$ time into a data structure of size $O(n)$, so that for a query point q, one can find in $O(\log n)$ time the smallest index ball in $\mathcal{E}$ that does not contain q.*

Proof. Follows from the proof of Theorem 1. □

Theorem 8. *A sequence $\mathcal{E}$ of n 2-dimensional balls can be preprocessed in $O(n \log n)$ time into a data structure of size $O(n \log n)$, such that for a query point q, one can find in $O(\log n)$ time the smallest index ball in $\mathcal{E}$ that does not contain q.*

Proof. Consider the data structure designed for the 2-D OLBES problem. Traversing a root-to-leaf path in the tree T to identify the leftmost disk that does not contain the point q seemingly would take $O(\log^2 n)$ time: there are $O(\log n)$ levels and at each visited node v on the path we spend $O(\log n)$ time to decide if q lies or not in the disk intersection $I(v)$ stored at v.

To reduce the query time to $O(\log n)$, we use the *fractional cascading* technique [10] on T. Given a directed graph $G = (V, E)$ such that each node v contains a sorted list $L(v)$, the fractional cascading problem is to construct an $O(n)$ space data structure, where $n = |V| + |E| + \sum_{v \in V} |L(v)|$, such that given a path $\{v_1, v_2, \dots, v_m\}$ in G and an arbitrary element x, one can locate x efficiently in each $L(v_i)$. In [10], Chazelle and Guibas give an $O(n)$ time algorithm to construct a fractional cascading data structure, from a graph G as above, with a search time of $O(\log n + m \log d(G))$, where $d(G)$ is the maximum degree of any node in G.

In our case, we let T correspond to the graph G. We maintain each $I(v)$ like in [7]. An internal node v can be described as a vertical visibility map: a collection of intervals on the x-axis, each enhanced with pointers to the two circles that contribute the arcs bounding $I(v)$ above and below.

Consider first the location structure $L(v)$ for the map at node v. We build a simple balanced tree on the endpoints of the (x-axis) intervals at v; at a leaf of this tree, we know the interval and we can compare the point with the two arcs associated with the interval to decide if it is in the intersection $I(v)$.

Fractional cascading is performed only on comparisons with x-coordinates, which allows us to easily pass subsets of x-coordinates up the tree T and insert them in the structures $L(v)$. This is the easiest form of fractional cascading (there is a single comparison in a single linearly ordered space), so that the first lookup takes $\mathrm{O}(\log n)$ time to get the interval, but each lookup afterwards takes $\mathrm{O}(1)$ time apiece, for a total of another $\mathrm{O}(\log n)$ time.

We have $\sum_{v\in T}|L(v)| = \mathrm{O}(n\log n)$, $m = \mathrm{O}(\log n)$ and $d(T) = \mathrm{O}(1)$. Thus, we obtain the claimed space and preprocessing and query times. □

6. Conclusions

We have presented efficient algorithms for solving the polygonal path simplification problem with angle constraints in $\mathbb{R}^2$ and $\mathbb{R}^3$, whose time bounds match those of the best known path simplification algorithms without angle constraints [7,8,11]. Our algorithms improve by nearly a linear factor in the time bound over the possible solutions based on standard graph-theoretic techniques.

To solve the problem, we have formulated an interesting (more general) off-line search problem referred to as the *off-line ball exclusion search* (OLBES), and have developed efficient data structures that solve the OLBES problem in $\mathrm{O}(n\log n)$ time. Our solutions can be easily extended to other types of objects, and can be applied to a class of geometric paths and other related problems.

Acknowledgement

The authors would like to thank the anonymous reviewers for their helpful comments and suggestions.

References

[1] P.K. Agarwal, T. Biedl, S. Lazard, S. Robbins, S. Suri, S. Whitesides, Curvature constrained shortest paths in a convex polygon, in: Proc. 14th ACM Symp. on Comp. Geom., 1998, pp. 392–401.
[2] P.K. Agarwal, S. Har-Peled, N. Mustafa, Y. Wang, Near-linear time approximation algorithms for curve simplification, in: Proc. of 10th Annual European Sympos. Algorithms, 2002, pp. 29–41.
[3] P.K. Agarwal, J. Matoušek, Dynamic half-space range reporting and its applications, Algorithmica 13 (1995) 325–345.
[4] P.K. Agarwal, K.R. Varadarajan, Efficient algorithms for approximating polygonal chains, Discrete Comput. Geom. 23 (2000) 273–291.
[5] H. Alt, J. Blomer, M. Godau, H. Wagener, Approximation of convex polygons, in: Proc. 10th Coll. on Autom., Lang. and Prog. (ICALP), 1990, pp. 703–716.
[6] J. Anez, T. De La Barra, B. Perez, Dual graph representation of transport networks, Transportation Res. B 30 (3) (1996) 209–216.
[7] G. Barequet, D.Z. Chen, O. Daescu, M.T. Goodrich, J. Snoeyink, Efficiently approximating polygonal paths in three and higher dimensions, Algorithmica 33 (2) (2002) 150–167.
[8] W.S. Chan, F. Chin, Approximation of polygonal curves with minimum number of line segments or minimum error, Internat. J. Comput. Geom. Appl. 6 (1) (1996) 59–77.

S0925-7721(04)00139-7/FLA AID:777 Vol.●●●(●●●)
COMGEO:m2 v 1.32 Prn:21/01/2005; 13:33
cgt777
[DTD5] P.15 (1-15)
by:Gi p. 15

[9] B. Chazelle, Lower bounds for off-line range searching, Discrete Comput. Geom. 17 (1997) 53–65.

[10] B. Chazelle, L.J. Guibas, Fractional cascading: I. A data structuring technique, Algorithmica 1 (3) (1986) 133–162.

[11] D.Z. Chen, O. Daescu, Space-efficient algorithms for approximating polygonal curves in two dimensional space, Internat. J. Comput. Geom. Appl. 13 (2) (2003) 95–112.

[12] L.P. Cordella, G. Dettori, An $O(n)$ algorithm for polygonal approximation, Pattern Recogn. Lett. 3 (1985) 93–97.

[13] M. de Berg, M. van Kreveld, S. Schirra, Topologically correct subdivision simplification using the bandwidth criterion, Cartography and GIS 25 (1998) 243–257.

[14] D.H. Douglas, T.K. Peucker, Algorithms for the reduction of the number of points required to represent a digitized line or its caricature, Canadian Cartographer 10 (2) (1973) 112–122.

[15] R. Estkowski, J.S.B. Mitchell, Simplifying a polygonal subdivision while keeping it simple, in: Proc. 17th ACM Symposium on Computational Geometry, 2001, pp. 40–49.

[16] D. Eu, G.T. Toussaint, On approximation polygonal curves in two and three dimensions, CVGIP: Graphical Models and Image Processing 56 (3) (1994) 231–246.

[17] L.J. Guibas, J.E. Hershberger, J.S.B. Mitchell, J.S. Snoeyink, Approximating polygons and subdivisions with minimum link paths, Internat. J. Comput. Geom. Appl. 3 (4) (1993) 383–415.

[18] S.L. Hakimi, E.F. Schmeichel, Fitting polygonal functions to a set of points in the plane, CVGIP: Graphical Models and Image Processing 53 (2) (1991) 132–136.

[19] J. Hershberger, J. Snoeyink, Cartographic line simplification and polygon CSG formulae in $O(n \log^* n)$ time, in: Proc. 5th International Workshop on Algorithms and Data Structures, 1997, pp. 93–103.

[20] H. Imai, M. Iri, Computational-geometric methods for polygonal approximations of a curve, Computer Vision, Graphics and Image Processing 36 (1986) 31–41.

[21] H. Imai, M. Iri, An optimal algorithm for approximating a piecewise linear function, J. Inform. Process. 9 (3) (1986) 159–162.

[22] H. Imai, M. Iri, Polygonal approximations of a curve-formulations and algorithms, in: Computational Morphology, North-Holland, Amsterdam, 1988, pp. 71–86.

[23] A. Melkman, J. O'Rourke, On polygonal chain approximation, in: Computational Morphology, North-Holland, Amsterdam, 1988, pp. 87–95.

[24] B.K. Natarajan, On comparing and compressing piecewise linear curves, Technical Report, Hewlett Packard, 1991.

[25] B.K. Natarajan, J. Ruppert, On sparse approximations of curves and functions, in: Proc. 4th Canadian Conference on Computational Geometry, 1992, pp. 250–256.

[26] G.T. Toussaint, On the complexity of approximating polygonal curves in the plane, in: Proc. IASTED International Symp. on Robotics and Automation, Lugano, Switzerland, 1985.