Computational Geometry 48 (2015) 443–452

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo

Weak visibility queries of line segments in simple polygons

Danny Z. Chen [a], Haitao Wang [b,*]

[a] *Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA*
[b] *Department of Computer Science, Utah State University, Logan, UT 84322, USA*

A R T I C L E   I N F O

Article history:
Received 30 January 2013
Accepted 26 January 2015
Available online 9 February 2015

Keywords:
Visibility polygons
Weak visibility from line segments
Visibility queries
Zone theorem

A B S T R A C T

Given a simple polygon P in the plane, we present new data structures for computing the weak visibility polygon from any query line segment in P. We build a data structure in $O(n)$ time and space that can compute the visibility polygon for any query line segment s in $O(k \log n)$ time, where k is the size of the visibility polygon of s and n is the number of vertices of P. Alternatively, we build a data structure in $O(n^3)$ time and space that can compute the visibility polygon for any query line segment in $O(k + \log n)$ time. In order to develop these data structures, we obtain many other results that may be interesting in their own right.

1. Introduction

Given a simple polygon $\mathcal{P}$ of n vertices in the plane, two points in $\mathcal{P}$ are *visible* to each other if the line segment joining them lies in $\mathcal{P}$. For a line segment s in $\mathcal{P}$, a point p is *weakly visible* (or *visible* for short) to s if s has at least one point that is visible to p. The *weak visibility polygon* (or *visibility polygon* for short) of s, denoted by $Vis(s)$, is the set of all points in $\mathcal{P}$ that are visible to s. The *weak visibility query problem* is to build a data structure for $\mathcal{P}$ such that $Vis(s)$ can be computed efficiently for any query line segment s in $\mathcal{P}$.

This problem has been studied before. Bose et al. [3] built a data structure of $O(n^3)$ size in $O(n^3 \log n)$ time that can compute $Vis(s)$ in $O(k \log n)$ time for any query, where k is the size of $Vis(s)$. Throughout this paper, we always let k denote the size of $Vis(s)$ for any query line segment s. Bygi and Ghodsi [4] gave an improved data structure with the same size and preprocessing time as that in [3] but its query time is $O(k + \log n)$. Aronov et al. [1] proposed a smaller data structure of $O(n^2)$ size with $O(n^2 \log n)$ preprocessing time and $O(k \log^2 n)$ query time. Table 1 gives a summary. If the problem is to compute $Vis(s)$ for a single segment (not queries), then there is an $O(n)$ time algorithm [12].

1.1. Our contributions

In this paper, we present two new data structures whose performances are also given in Table 1. Our first data structure, which is built in $O(n)$ time and $O(n)$ space, can compute $Vis(s)$ in $O(k \log n)$ time for any query segment s. Comparing with the data structure in [1], our data structure reduces the query time by a logarithmic factor and uses much less preprocessing time and space.

The preprocessing time and size of our second data structure are both $O(n^3)$, and each query takes $O(k + \log n)$ time. Comparing with the result in [4], our data structure has less preprocessing time. In addition, our solution, which is based

* Corresponding author.
E-mail addresses: dchen@cse.nd.edu (D.Z. Chen), haitao.wang@usu.edu (H. Wang).

http://dx.doi.org/10.1016/j.comgeo.2015.02.001
0925-7721/© 2015 Elsevier B.V. All rights reserved.

Table 1
A summary of the data structures. The value k is the size of $Vis(s)$ for any query segment s.

Data structure	Preprocessing time	Size	Query time
[3]	$O(n^3 \log n)$	$O(n^3)$	$O(k \log n)$
[4]	$O(n^3 \log n)$	$O(n^3)$	$O(k + \log n)$
[1]	$O(n^2 \log n)$	$O(n^2)$	$O(k \log^2 n)$
Our Result 1	$O(n)$	$O(n)$	$O(k \log n)$
Our Result 2	$O(n^3)$	$O(n^3)$	$O(k + \log n)$

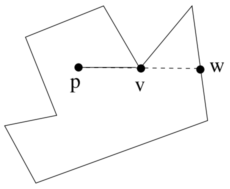

Fig. 1. Illustrating a window $\overline{vw}$ of p.

on the approach in [3], is simpler than that in [4]. Further and more importantly, our techniques explore many geometric observations on the problem that may be useful elsewhere. In particular, we prove a tight combinatorial bound for the "zone" in a line segment arrangement contained in a simple polygon, as follows, which is interesting in its own right.

Let S be a set of (possibly intersecting) line segments in a simple polygon $\mathcal{P}$ such that both endpoints of each segment of S are on $\partial\mathcal{P}$ (i.e., the boundary of $\mathcal{P}$). Let $\mathcal{A}$ be the arrangement formed by the segments in S and the edges of $\partial\mathcal{P}$. For any line segment s in $\mathcal{P}$ (whose endpoints need not be on $\partial\mathcal{P}$), the *zone* of s, denoted by $Z(s)$, is defined to be the set of faces of $\mathcal{A}$ that intersect s. For each edge of any face in $\mathcal{A}$, it either lies on a segment of S or lies on $\partial\mathcal{P}$. Let Λ be the number of edges of the faces in $Z(s)$ each of which lies on a segment of S. We want to find a good upper bound for Λ. By using the zone theorem for the general line segment arrangement [9], we can easily obtain $\Lambda = O(|S|\alpha(|S|))$, where $\alpha(\cdot)$ is the inverse Ackermann function [14]. In this paper, we prove a tight bound $\Lambda = O(m)$, where $m \le |S|$ is the number of segments in S each of which contains at least one edge of the faces in $Z(s)$. An immediate application of this result is that we obtain an efficient query algorithm for our second data structure. Since combinatorial bounds on arrangements are fundamental, this result may find other applications as well.

The rest of this paper is organized as follows. In Section 2, we review some geometric structures and an algorithmic scheme that will be used by the query algorithms of both our data structures. We will also give a "ray-rotating" data structure in Section 2, which is needed by our first data structure in Section 3. In Sections 3 and 4, we present our first and second data structures, respectively. As a by-product of our second data structure, the combinatorial bound of the zone mentioned above is also given in Section 4. Section 5 concludes the paper.

2. Preliminaries

In this section, we review some geometric structures and discuss an algorithmic scheme that will be used by the query algorithms of both our data structures given in Sections 3 and 4. We will also give a "ray-rotating" data structure in Section 2.1, which is needed by our first data structure in Section 3.

For simplicity of discussion, we assume no three vertices of $\mathcal{P}$ are collinear; we also assume for any query segment s, s is not collinear with any vertex of $\mathcal{P}$ and each endpoint of s is not collinear with any two vertices of $\mathcal{P}$. As in [1,3], our approaches can be easily extended to the general situation.

Denote by $\partial\mathcal{P}$ the boundary of $\mathcal{P}$. The visibility graph of $\mathcal{P}$ is a graph whose vertex set consists of all vertices of $\mathcal{P}$ and whose edge set consists of edges defined by all visible pairs of vertices of $\mathcal{P}$. Here, two adjacent vertices on $\partial\mathcal{P}$ are considered visible to each other. Throughout the paper, we use K to denote the size of the visibility graph of $\mathcal{P}$. Note that $K = O(n^2)$ and $K = \Omega(n)$. The visibility graph can be computed in $O(K)$ time [15].

We discuss the *visibility decomposition* of $\mathcal{P}$ [1,3]. Consider a point p in $\mathcal{P}$ and a vertex v of $\mathcal{P}$. Suppose the line segment $\overline{pv}$ is in $\mathcal{P}$, i.e., p is visible to v. Suppose v is a reflex vertex. If we extend $\overline{pv}$ along the direction from p to v, then we will stay inside $\mathcal{P}$. Let w be the point on the boundary of $\mathcal{P}$ that is hit first by our above extension of $\overline{pv}$ (e.g., see Fig. 1). We call the line segment $\overline{vw}$ the *window* of p. The point p is called the *defining point* of the window and the vertex v is called the *anchor vertex* of the window. It is well known that the boundary of the visibility polygon of the point p consists of parts of $\partial\mathcal{P}$ and the windows of p [1,3]. If the point p is a vertex of $\mathcal{P}$, then as in [1], the window $\overline{vw}$ is called a *critical constraint* of $\mathcal{P}$, and p is called the *defining vertex* of the critical constraint. For example, in Fig. 2, both $\overline{up_u}$ and $\overline{vp_v}$ are critical constraints; for $\overline{up_u}$, its anchor vertex is u and its defining vertex is v, and for $\overline{vp_v}$, its anchor vertex is v and defining vertex is u. It is easy to see that the total number of critical constraints is $O(K)$ because each critical constraint corresponds to a visible vertex pair of $\mathcal{P}$ and a visible vertex pair corresponds to at most two critical constraints.

As in [1,3], we represent the visibility polygon $Vis(s)$ of a segment s by a cyclic list of the vertices and edges of $\mathcal{P}$ in the order in which they appear on the boundary of $Vis(s)$, and we call such a list the *combinatorial representation* of $Vis(s)$ [1].