# Approximate Shortest Routes for Frontier Visibility under Limited Visibility

Xia Chen[1], Fajie Li[1], and Reinhard Klette[2]

[1] College of Computer Science and Technology, Huaqiao University
Xiamen, Fujian, China
[2] Computer Science Department, The University of Auckland
Auckland, New Zealand
xiaoyu@hqu.edu.cn, li.fajie@hqu.edu.cn, r.klette@auckland.ac.nz

**Abstract.** Consider a simple polygon $P$ and a point $s$ on the frontier $\partial P$ of $P$. For any real $\delta > 0$ there exists a shortest path $\rho$ inside of $P$ such that $s$ is on the path $\rho$, and for each point $p$ in $\partial P$, there exists a point $q$ in $\rho$ at Euclidean distance less than or equal $\delta$ to $p$ such that the line segment $pq$ is in $P$. Such an optimum path $\rho$ is called a *shortest route for $\partial P$ visibility under $\delta$-visibility* that starts at point $s$ on $\partial P$. We provide an approximation algorithm (which belongs to the class of rubberband algorithms) for finding such a path $\rho$ in $\mathcal{O}(n^2)$ run time, where $n$ is the number of vertices of a given simple polygon $P$. The run time does not depend on $\delta$ or on the start point $s$.

## 1 Introduction

In 1973, Victor Klee proposed the following *Art Gallery Problem*: How many (non-moving) guards are needed in a polygon such that each point in the polygon can be seen by at least one of those guards [5]? In 1988, Wei-Pang Chin and Simeon Ntafos proposed the *Watchman Route Problem* (WRP): Find a shortest route inside of a simple polygon such that each point in the polygon can be seen from at least one point on the route [2]. Since then, the WRP and its variants have attracted much interest in computational geometry; for example, see [1, 3, 9, 10, 12, 16].

The proposed algorithms are typically under the assumption that the guard or watchman has infinite visibility. In the real world, it is more reasonable to assume finite visibility for a person or a robot. That is, viewing of a guard/watchman is limited by Euclidean distance $\delta > 0$. Such a constrained visibility is called $\delta$-visibility, and it was first proposed by Shin [15]. Later, Ntafos proposed the WRP under limited visibility [11]. References [6, 7] presented linear-time algorithms for computing a $\delta$-kernel or an edge visibility polygon. References [1, 3, 14] considered static point distribution under $\delta$-visibility.

In this paper, we present an approximate rubberband algorithm for computing a shortest route $\rho$ for frontier $\delta$-visibility for a given simple polygon $P$ and a start point $s$ such that for each point $p$ in $\partial P$, there exists at least a point $q$

in $\rho$ such that the line segment $pq$ is in $P$, with $d_e(p, q) \leq \delta$ (where $d_e$ is the Euclidean metric), and $\rho$ passes through $s$.

The rest of the paper is organized as follows: In Section 2, we present definitions and lemmas used in this paper. We describe our algorithm in Section 3 and analyse the time complexity and approximation factor in Section 4. Some experimental results will be presented in Section 5. Section 6 concludes the paper.

## 2 Preliminaries

In this paper, $P$ denotes a simple polygon (i.e. a 2-dimensional region bounded by a simple polyline). Let $S_1$ and $S_2$ be two subsets of $P$. Let $v$ be a vertex of $P$, and $e$ an edge of $P$. By $d_e(v, e)$ we denote the Euclidean distance between $v$ and $e$, that is, $d_e(v, e) = \min\{d_e(v, u) : u \in e\}$.

**Definition 1.** *$S_2$ is $S_1$-*visible *if, for each point $p \in S_2$, there exists at least a point $q \in S_1$ such that the line segment $pq$ is in $P$. If $S_2$ is $S_1$-visible and $S_1$ is $S_2$-visible, then we say that $S_1$ and $S_2$ are* visible from each other*. If there do not exist non-empty subsets $S'_1 \subseteq S_1$ and $S'_2 \subseteq S_2$ such that $S'_1$ and $S'_2$ are visible from each other, then we say that $S_1$ and $S_2$ are* not visible from each other*. Otherwise, we say that $S_1$ and $S_2$ are* partially visible from each other*.*

If $S_1$ and $S_2$ are visible from each other then they are also partially visible from each other.

**Definition 2.** *Let $\delta > 0$. If for each point $p \in S_2$, there exists at least a point $q \in S_1$ such that $d_e(p, q) \leq \delta$ and $pq$ is in $P$, then we say that $S_2$ is $\delta$-*visible *from $S_1$.*

The main algorithm in this paper is for computing an approximate shortest route $\rho$ such that the frontier $\partial P$ of $P$ is $\delta$-visible from $\rho$

In Definition 2, if $S_1$ is a singleton that contains a vertex $v_i$ of polygon $P$, then the set of points which are $\delta$-visible from $v_i$ is also called the *$\delta$-visible region* of $v_i$, denoted by $V_i^\delta$:

$$V_i^\delta = \{p : pv_i \subseteq P \wedge d_e(p, v_i) \leq \delta\}$$

Let $\langle v_0, v_1, v_2, \ldots, v_{n-1}\rangle$ be the sequence of all vertices of the simple polygon $P$ describing $\partial P$ in counterclockwise order.

**Lemma 1.** *If $p_i \in V_i^\delta$ then the edge $v_iv_{i+1}$ is $\delta$-visible from $p_ip_{i+1}$.*

*Proof.* Let line segment $v_ix$ (or $v_{i+1}x'$) be perpendicular to $p_ip_{i+1}$ at $x$ (or $x'$) (see Figure 1). It is clear that $d_e(v_i, x) \leq d_e(v_i, p_i) \leq \delta$ and $d_e(v_{i+1}, x') \leq d_e(v_{i+1}, p_{i+1}) \leq \delta$. If both $x$ and $x'$ are in between $p_i$ and $p_{i+1}$, then for each $q$ on the edge $v_iv_{i+1}$, the Euclidean distance between $q$ and $p_ip_{i+1}$ must be less than $\delta$. If $x$ or $x'$ is not in between $p_i$ and $p_{i+1}$, then it is still true that for each $q$ on the edge $v_iv_{i+1}$, the Euclidean distance between $q$ and $p_ip_{i+1}$ must be less than $\delta$.


**Fig. 1.** Illustration for Lemma 1

For each vertex $v_i$ of the simple polygon $P$, we compute its $\delta$-visible region $V_i^\delta$. We also call each visible region a *cage*. By Lemma 1, a shortest route $\rho$ for $\delta$-visibility from $\partial P$ is a a shortest route that passes through (that is, visits) the start point $s$ and each $\delta$-cage $V_i^\delta$ in order $\langle V_0^\delta, V_1^\delta, V_2^\delta, \ldots, V_{n-1}^\delta \rangle$.

If $v_i$ is a reflex vertex (i.e., its internal angle is greater than 180°), let $v_j$ and $v_k$ be two non-reflex vertices such that $v_j$, $v_i$ and $v_k$ are located around $\partial P$ counterclockwise, and there are no other non-reflex vertices between $v_j$ (or $v_i$) and $v_i$ (or $v_k$). Let $p_j \in V_j^\delta$ and $p_k \in V_k^\delta$, $\rho(p_j, p_k)$ the Euclidean shortest path between $p_j$ and $p_k$ inside of $P$, and $\partial P(v_j, v_k)$ the section of $\partial P$ from $v_j$ to $v_k$ counterclockwise.

**Lemma 2.** *$\partial P(v_j, v_k)$ is $\delta$-visible from $\rho(p_j, p_k)$.*

*Proof.* If $j = i-1 \mod n$ and $j = i+1 \mod n$, then there may be the following two cases:

Case 1. $V_j^\delta$ and $V_k^\delta$ are non-visible. In this case, $\rho(p_j, p_k)$ is a polyline consisting of three points $p_{i-1}$, $v_i$ and $p_{i+1}$ (see Figure 2). It is clear that $v_{i-1}v_i$ (or $v_i v_{i+1}$) is $p_{i-1}v_i$ (or $v_i p_{i+1}$) $\delta$-visible.

Case 2. $V_j^\delta$ and $V_k^\delta$ are partially visible (see Figure 3). By Lemma 1, $v_{i-1}v_{i+1}$ is $\delta$-visible from $p_{i-1}p_{i+1}$. Note that $\partial P(v_j, v_k)$ is inside the polygon $v_{i-1}v_{i+1}p_{i+1}p_{i-1}$, thus, $\partial P(v_j, v_k)$ is $\delta$-visible from $\rho(p_j, p_k)$.

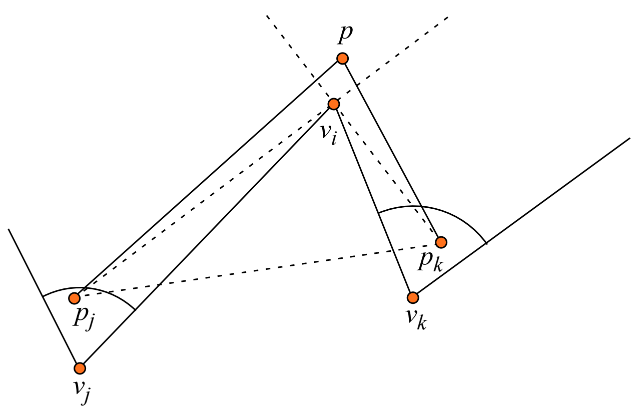


**Fig. 2.** Illustration for Case 1 of Lemma 2.

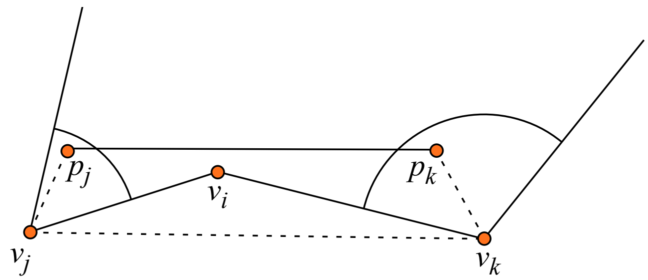


**Fig. 3.** Illustration for Case 2 of Lemma 2.

Analogously, if $j \neq i-1 \mod n$ or $j \neq i+1 \mod n$, then the lemma is still correct.

By Lemmas 1 and 2, a shortest route $\rho$ for $\delta$-visibility from $\partial P$ is a a shortest route that passes through the start point $s$ and each $\delta$-cage $V_i^\delta$ in order, and $\rho$ must pass through each reflex vertex. Thus, we do not need to compute the $\delta$-cage $V_i^\delta$ for each reflex vertex $v_i$.

Let $v_i$ be a non-reflex vertex of the simple polygon $P$. If $P \backslash V_i^\delta$ is a simply connected region, then $V_i^\delta$ is called a *Type 1* $\delta$-visible region. $v_i$ is called *Type 1* non-reflex vertex. Otherwise, $V_i^\delta$ s called a *Type 2* $\delta$-visible region. $v_i$ is called *Type 2* non-reflex vertex.

For example, in Figure 4, $V_j^\delta$ is a Type 1 $\delta$-visible; both $V_i^\delta$ and $V_k^\delta$ are Type 2 $\delta$-visible regions. It is clear that for each Type 2 $\delta$-visible region $V_i^\delta$, there exists a maximal $0 < \delta_i \leq \delta$ such that the $\delta_i$-visible region of $v_i$ is a Type 1 $\delta_i$-visible region, denoted by ${V'}_i^\delta$. In Figure 4, ${V'}_i^\delta$ and ${V'}_k^\delta$ are Type 1 $\delta$-visible regions. It is clear that ${V'}_i^\delta \subset V_i^\delta$ and ${V'}_k^\delta \subset V_k^\delta$ in Figure 4 while in Figure 5, ${V'}_i^\delta = V_i^\delta$.

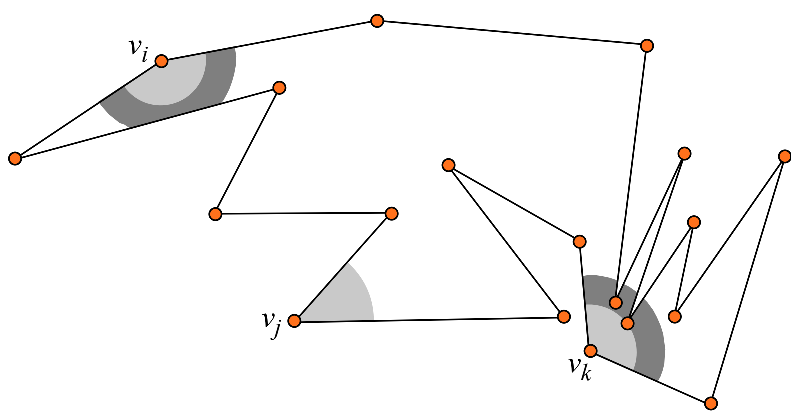


**Fig. 4.** Examples of $\delta$-visible regions of Type 1 or 2.

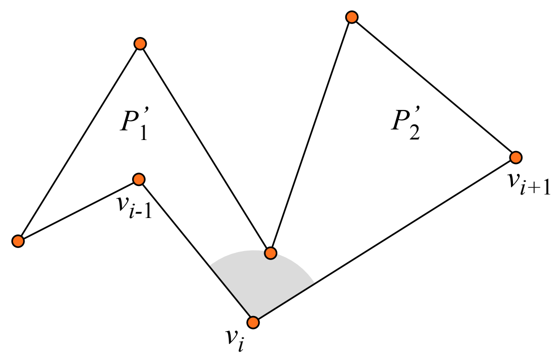


**Fig. 5.** Illustration for Lemma 3.

**Lemma 3.** *For each vertex $v_i$, if its $\delta$-visible region $V_i^\delta$ is a Type 2 $\delta$-visible region, then each shortest route $\rho$ for $\partial P$ visibility under $\delta$-visibility passes through ${V'}_i^\delta$.*

*Proof.* By the definition of Type 2 $\delta$-visible region, $P \backslash V_i^\delta$ is not a simply connected region. Thus, $\rho$ must pass through ${V'}_i^\delta$. Otherwise, $\rho$ must not enter a simply connected subregion $P'$ of $P$ (See Figure 5). Thus, each edge of $P'$ is not $\delta$-visible from $\rho$. This is a contradiction to $\rho$ being a shortest route for $\delta$-visibility from $\partial P$.

For each reflex vertex $v$, if there does not exist a $\delta$-visible region $V_i^\delta$ of vertex $v_i \neq v$ such that $v \in V_i^\delta$, then $v$ is called a *Type 1* reflex vertex. Otherwise, $v$ is called a *Type 2* reflex vertex.

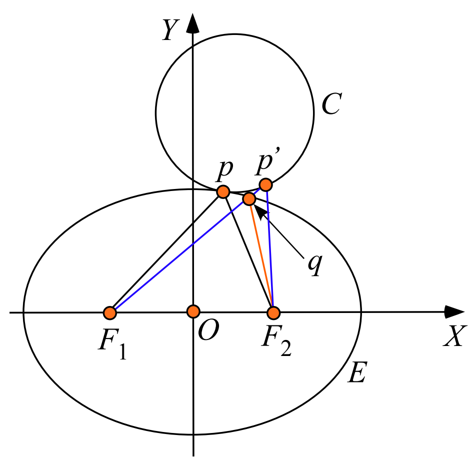


**Fig. 6.** Illustration for Lemma 4.

**Lemma 4.** *Let $F_1$ and $F_2$ be the foci of an ellipse $E$. If $E$ intersects a cycle $C$ with a single point $p$, then for each point $p'$ in $C$, $d_e(F_1, p') + d_e(F_2, p') \geq d_e(F_1, p) + d_e(F_2, p)$ (See Figure 6).*

*Proof.* In Figure 6, let line segment $F_1p'$ intersects the ellipse $E$ at point $q$, then we have that $d_e(F_1, p') + d_e(F_2, p') \geq d_e(F_1, q) + d_e(F_2, q) = d_e(F_1, p) + d_e(F_2, p)$.

## 3 Algorithm

At first we describe a preprocessing step; see Fig. 7. Let $G = [V, E, w]$ be an undirected weighted graph, where $V$ is the set of vertices of the simple polygon $P$; for every two vertices $u$, $v \in V$, $u$ and $v$ is connected by an edge $uv \in E$ iff $u$ and $v$ are visible; the weight of $uv$ is defined as $w(uv) = d_e(u, v)$.

The main algorithm in Fig. 8 is now for computing a shortest route $\rho$ such that $\rho$ starts at $s$, then passes through a sequence of $\delta$-cages in order.

Our algorithm follows general rubberband algorithm (RBA) design principles; see [8, 9, 13] for RBAs.

First we create an initial route $\rho_0 = \langle p_0, p_1, p_2, \ldots, p_{m-1} \rangle$. Then we enter a loop as follows: for every three consecutive vertices $p_{i-1}, p_i, p_{i+1}$ of the route, we update $p_i$ by replacing it by an optimal point $q_i$ such that

$$d_e(p_{i-1}, q_i) + d_e(p_{i+1}, q_i) = \min\{d_e(p_{i-1}, q) + d_e(p_{i+1}, q)\}$$

where $q$ is in a line segment or a section of a circle, for $i = 1, 2, \ldots, m$, and indices $\mod m$. We terminate the loop when the difference in length between the current route and the previous route is sufficiently small (that is, it is less than or equals an accuracy parameter $\varepsilon > 0$).

The algorithm consists of two major steps. In the initial step, we create an initial route. We start at $s$, scan the sequence $\langle v_0, v_1, \ldots, v_{n-1} \rangle$ of vertices of $P$

**Procedure 1** (Compute type of a non-reflex vertex)
*Input:* $\delta > 0$, $G$ and a non-reflex vertex $v_i$ of $P$.
*Output:* the type of $v_i$.

Let $N(v_i)$ be the set of neighbours of $v_i$ in $G$.
**for** each $v_j \in N(v_i) \backslash \{v_{i-1}, v_{i+1}\} = N'(v_i)$ **do**
  **if** $d_e(v_j, v_i) < \delta$ **then**
    Report "$v_i$ is Type 2".
  **else**
    **if** $v_j, v_{j+1} \in N'(v_i)$ and $d_e(v_i, v_j v_{j+1}) < \delta$ **then**
      Report "$v_i$ is Type 2".
    **end if**
  **end if**
**end for**
Report "$v_i$ is Type 1".

**Fig. 7.** Computation of the type of a non-reflex vertex.

**Algorithm 1** (Label vertices and compute $\delta$-cages)
*Input:* $\delta > 0$, the start point $s$, the simple polygon $P$.
*Output:* For each vertex $v_i$ of $P$, label it as $Rv_1$ (i.e., reflex, Type 1), $Rv_2$ (i.e., reflex, Type 2), $NRv_1$ (i.e., non-reflex, Type 1), or $NRv_2$ (i.e., non-reflex, Type 2). Compute $\delta$-cage $V_i^\delta$ if $v_i$ is labelled as $NRv_1$.

1: Label the start point $s$ as $Rv_0$.
2: Start from $s$, let the sequence of vertices of $P$ counterclockwise as $V = \{v_0, v_1, v_2, \ldots, v_{n-1}\}$.
3: **for** each $i \in \{0, 1, \ldots, n-1\}$ **do**
4:   **if** $v_i$ is a reflex vertex **then**
5:     Label $v_i$ as $Rv_1$ (This label will be updated if it is Type 2, see Line 13).
6:   **else**
7:     **if** $v_i$ is a Type 2 non-reflex vertex **then**
8:       Label $v_i$ as $NRv_2$.
9:     **else**
10:       Label $v_i$ as $NRv_1$.
11:       Compute $\delta$-cage $V_i^\delta$.
12:       **if** $V_i^\delta$ contains a reflex vertex $v_j$ **then**
13:         Label (or update the label of) $v_j$ as $Rv_2$.
14:       **end if**
15:     **end if**
16:   **end if**
17: **end for**

**Fig. 8.** Labelling of vertices and computation of $\delta$-cages.

counterclockwise. If the current vertex $v_i$ is labelled as $Rv_1$ (i.e., Type 1 reflex vertex) or $NRv_2$ (i.e., Type 2 non-reflex vertex), then add $v_i$ into a queue $Q\rho_0$ of current vertices of $\rho_0$ (the first element of $Q\rho_0$ is $s$); else if it is labelled as $Rv_2$ (i.e., Type 2 reflex vertex), then ignore it; else it must be labelled as $NRv_1$ (i.e., Type 1 non-reflex vertex), then compute a point $v_i'$ in the arc portion of the frontier of $\delta$-visible region $V^\delta$ (see the explanation after Lemma 4). Then let $v_i'$ be a vertex of $\rho_0$ (i.e., add $v_i'$ into a queue $Q\rho_0$). By Lemma 4, $v_i'$ can be computed by the latest vertex added in the queue $Q\rho_0$ of current vertices of $\rho_0$, the arc portion and another point $u$ that can be defined as follows: if next scanned vertex $v_{i+1}$ is labelled as $Rv_1$ or $NRv_2$, then let $u$ be $v_{i+1}$; else if next scanned vertex $v_{i+1}$ is labelled as $Rv_2$, then ignore it and scan next vertex of $P$; else next scanned vertex $v_{i+1}$ must be labelled as $Rv_1$, then let $u$ be a point in the perpendicular bisector of the edge $v_iv_{i+1}$ (see Figure 9).

For each point $v$ in the queue $Q\rho_0$ of vertices of $\rho_0$, put the second label of it as follows: If $v = v_i$ is labelled as $Rv_1$ or $NRv_2$, then put its second label as $r1_i$; else if $v$ is taken in a perpendicular bisector, then put its second label as $r2$; else if $v = v_i$ is taken in an arc portion of the frontier of $\delta$-visible region $V_i^\delta$, then put its second label as $r3_i$ (or $r3_{i1}$ and $r3_{i2}$ if there are two such vertices).

According to Section 2, the route $\rho_0$ must pass through each $\delta$-cage in order. Thus, each vertex with second label $r3$ maps to a cage. It is possible that an edge

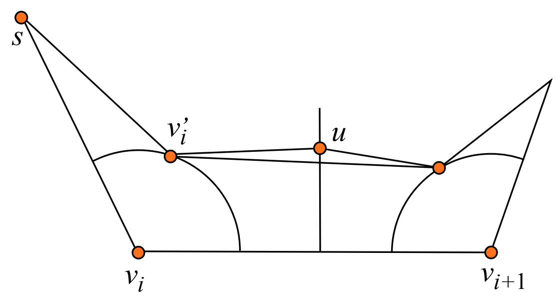


**Fig. 9.** Illustration for taking a point $u$ in a perpendicular bisector of the edge $v_i v_{i+1}$.

of the route may intersect two vertices with a cage. The vertices with second labels $r1$ or $r2$ must be inside of $P$ and the total number and coordinates (i.e., the location) of them may be changed in the iteration step that is described below.

In the iteration step, we update the current route by decreasing its length in each iteration. For every three continuous vertices of the route, we fix the first and third vertices, update the second one so as to obtain a shorter portion of the route. We start at the initial route $\rho_0 = \langle p_0, p_1, p_2, \ldots, p_{m-1} \rangle$. For every three continuous vertices $p_i$, $p_{i+1}$ and $p_{i+2}$, *Case 1*, $p_{i+1}$'s second label is $r3_k$ (or $r3_{k1}$, or $r3_{k2}$). If line segment $p_i p_{i+2}$ intersects $\delta$-cage $V_k^\delta$ with one point or two points, then update $p_{i+1}$ by replacing it by the single intersection point or any one of the two intersection points and keep its second label as $r3_k$ (or $r3_{k1}$, or $r3_{k2}$). Otherwise, line segment $p_i p_{i+2}$ does not intersect $\delta$-cage $V_k^\delta$ at all. Then we compute an optimal point in the arc portion of the frontier of $\delta$-cage $V_k^\delta$. We update $p_{i+1}$ by replacing it by this optimal point and keep its second label as $r3_k$. *Case 2*, $p_{i+1}$'s second label is $r1$ or $r2$. If $p_i$, and $p_{i+2}$ are visible, then delete $p_{i+1}$ from the set of vertices of current route $\rho$. Otherwise, compute an optimal point $p$ in the $\partial P$ such that $d_e(p_{i-1}, p) + d_e(p_{i+1}, p)$ is minimal and update $p_{i+1}$ by replacing it by $p$ and keep its second label the same as $p_{i+1}$'s second label. Repeat the iteration step until the difference of the length of current route and the length of the previous route is sufficiently small (i.e., less than or equals an accuracy parameter $\varepsilon_0 > 0$).

The initial route may not be completely contained in $P$, but the output route is completely in $P$.

## 4 Analysis

This section analyses the time complexity of our algorithm. The visibility graph $G = [V, E, w]$ can be computed in $\mathcal{O}(|V|log|V| + |E|)$ time [4]. It is clear that Procedure 1 can be computed in $\mathcal{O}(n)$ time. Thus, the types of all vertices of the simple polygon $P$ can be computed in $\mathcal{O}(n^2)$ time, where $n$ is the number of $P$. Algorithm 1 can be computed in $\mathcal{O}(n)$ time. Thus, the total preprocessing time is $\mathcal{O}(n^2)$.

The main algorithm is an example of a rubberband algorithm that runs in $\kappa(\varepsilon)\mathcal{O}(n)$ (see, for example, [9, 13]), where $\kappa(\varepsilon) = \frac{L_0 - L}{\varepsilon}$, and $L_0$ or $L$ are the length of the initial or the output route, respectively. Our approximation algorithm obtains an upper bound $L_u$ of the length of the true route. We may employ a convex polygon with $m-1$ vertices to approximate each cage and apply the algorithm for solving the Safari Route Problem to obtain a lower bound $L_l$ of the length of the true route (the time complexity is $\mathcal{O}((n+m)^2 log(n+m))$; see [3]). Thus, our algorithm has an approximation factor of $L_u/L_l$. Experimental results in the next section show that the route obtained by our algorithm is very close to the true route after 200 iterations for the given example.

## 5 Experimental Results

Table 1 shows the results obtained by the main algorithm when the input simple polygon $P$ is a regular $n$-gon. Results indicate that the route is very close to the true route after only 200 iterations.

| $n$ | $L_0$ | *iterations* | $\delta_0$ | $\delta_{50}$ | $\delta_{100}$ | $\delta_{200}$ |
|---|---|---|---|---|---|---|
| 1000 | 14969·9626681692 | 554 | 1·0000633331 | 1·0000008158 | 1·0000001081 | 1·0000000031 |
| 2000 | 29969·9438111304 | 945 | 1·0000352022 | 1·0000011979 | 1·0000003088 | 1·0000000370 |
| 5000 | 74969·9324651109 | 1980 | 1·0000154133 | 1·0000010049 | 1·0000004172 | 1·0000001248 |
| 10000 | 149969·92867794 | 3364 | 1·0000080563 | 1·0000007009 | 1·0000003510 | 1·0000001456 |

**Table 1.** The $n$s are the numbers of vertices of a regular polygon $P$. The $L_0$s are the lengths of initial routes. By *iterations* we list the numbers of iterations to obtain true routes. By $\delta_i = L_i/L$ we characterize the $i$-the iteration, where $L_i$ and $L$ are the length of the route obtained in the $i$-th iteration or of the true route, respectively. We only show values for $i = 0, 50, 100, 200$.

Table 2 shows results obtained for different values of $\delta$. The input is the simple polygon shown in Figure 10.

| $\delta$ | $L_0$ | *iterations* | $L_0/L$ |
|---|---|---|---|
| 1 | 98·7989122759 | 17 | 1·0000279637 |
| 1·5 | 92·7823178547 | 21 | 1·0000933844 |
| 2 | 86·9251557575 | 25 | 1·0002145610 |
| 2·5 | 81·2466385573 | 29 | 1·0003975345 |
| 3 | 75·7705619909 | 34 | 1·0006375863 |

**Table 2.** Column *iterations* are the minimum numbers of iterations sufficient to obtain the true routes. $L_0$ or $L$ are the lengths of initial or true routes, respectively.

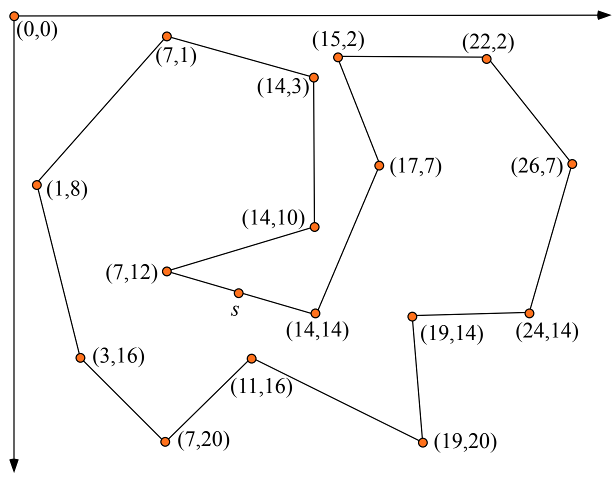


**Fig. 10.** An example of a non-convex simple polygon as used in our experiments.

## 6 Conclusion

In this paper we presented an approximate (rubberband-type) algorithm for computing a shortest route $\rho$ for $\delta$-visibility of the frontier $\partial P$ of a given simple polygon $P$, a start point $s$, and $\delta > 0$. The algorithm has a run time in $\mathcal{O}(n^2)$, where $n$ is the number of vertices of polygon $P$. Experiments indicate that an approximate route is very close to the true after a relatively small number of iterations.

## References

1. S. Bespamyatnikh. An $\mathcal{O}(nlogn)$ algorithm for the Zoo-keeper's problem, *Computational Geometry: Theory and Applications*, 2003.
2. W. Chin and S. Ntafos. Optimum watchman routes, *Inf. Processing Letters*, **28**:39–44, 1988.
3. M. Dror, A. Efrat,A. Lubiw, J. S. B. Mitchell. Touring a Sequence of Polygons, In Proc. *ACM Symposium Theory Computation*, 2003.
4. S. K. Ghosh, D. M. Mount. An output-sensitive algorithm for computing visibility graphs, *SIAM J. Computing*, 888–910, 1991.
5. R. Honsberger. *Mathematical Gems II*, Mathematical Association of America, 1976.
6. S. H. Kim. Visibility algorithms under distance constraint, Ph.D. Thesis, Dept. of Computer Science, KAIST, 1994.
7. S. H. Kim, J. H. Park, S. H. Choi, S. Y. Shin, K.-Y. Chwa. An optimal algorithm for finding the edge visibility polygon under limited visibility, *Information Processing Letters*, **53**:359–365, 1995.
8. F. Li and R. Klette. Exact and approximate algorithms for the calculation of shortest paths. IMA Minneapolis, Report 2141 on www.ima.umn.edu/preprints/oct2006, 2006.

9. F. Li, and R. Klette. Watchman route in a simple polygon with a rubberband algorithm. In Proc. *Canadian Conf. Computational Geometry*, pages 1–4, Winnipeg, Canada, 2010.
10. C. Mata, J. Mitchell. Approximation algorithms for geometric tours and network design problems, In Proc. *Annual Symposium Computational Geometry*, 1995.
11. S. Ntafos. Watchman routes under limited visibility, *Comput. Geometry Theory Application*, **1**:149–170, 1992.
12. S. Ntafos, L. Gewali. External watchman routes, *The Visual Computer*, **8**:474–483, 1994.
13. X. Pan, F. Li, and R. Klette. Approximate shortest path algorithms for sequences of pairwise disjoint simple polygons. In Proc. *Canadian Conf. Computational Geometry*, pages 175–178, Winnipeg, Canada, 2010.
14. S. Roya, D. Bardhanb, S. Dasc, Base station placement on boundary of a convex polygon, *J. Parallel Distributed Computing*, **68**:265–273, 2008.
15. S. Y. Shin. Computational Geometry, Course Notes, Dept. of Computer Science, KAIST, 1987.
16. X. Tan. A 2-approximation algorithm for the zookeeper's problem, *Information Process. Letters* **100**:183–187, 2006.