Optimal Algorithms for Two-Guard Walkability of Simple Polygons

B. Bhattacharya[*] A. Mukhopadhyay[†] G. Narasimhan[‡]

Abstract

A polygon P admits a *walk* from a boundary point s to another boundary point t if two guards can simultaneously walk along the two boundary chains of P from s to t such that they are always visible to each other. A walk is called a *straight* walk if no backtracking is required during the walk. A straight walk is *discrete* if only one guard is allowed to move at a time, while the other guard waits at a vertex. We present simple, optimal $O(n)$ time algorithms to determine all pairs of points of P which admit walks, straight walks and discrete straight walks. The chief merits of the algorithms are that these require simple data structures and do not assume a triangulation of P. Furthermore, the previous algorithms for the straight walk and the discrete straight walk versions ran in $O(n \log n)$ time even after assuming a triangulation.

1 Introduction

As a topological entity a simple polygon is merely a closed non-intersecting curve; as a metric entity its boundary structure can be extremely complex. Various computational problems related to simple polygons are the result of efforts to classify polygons with respect to their boundary structures. Among these are visibilty problems in their different incarnations, art-gallery or guard problems [10], which are fundamental to the understanding of polygon visibility. Many of these problems can be solved efficiently, if one assumes that a triangulation of the polygon is available. However, any algorithm that requires the notoriously complex linear-time triangulation algorithm of Chazelle [3] is simply impractical. In this paper, we show that it is possible to find efficient algorithms that avoid this requirement. In particular, we show how to solve various versions of the 2-guard walkability problem of Icking and Klein [8] in optimal linear time.

[*]School of Computing Science, Simon Fraser University, Burnaby, B.C., Canada, V5A 1S6. E-mail: binay@cs.sfu.ca;

[†]Department of Computer Science, University of Windsor, ON, Canada N9B 3P4; E-mail: asishm@cs.uwindsor.ca.

[‡]Math Sciences Dept., University of Memphis, Memphis TN. giri@msci.memphis.edu.

The 2-guard problem involves determining whether two guards can move along the boundary of a simple polygonal room from a common starting point s to a common destination point t, in opposite directions, while remaining in sight of each other at all times. If they can, then P is said to be 2-guard walkable (or, simply, *walkable*) with respect to (s, t). A polygon is said to be walkable if such a walk exists for some pair (s, t). Icking and Klein [8] introduced this class of problems, and showed that it is possible to test in $O(n \log n)$ time whether P is walkable with respect to a pair of points (s, t). Subsequently, a linear time solution was presented by Heffernan [7], assuming that a triangulation of the polygon is available. If a triangulation is not assumed the bound is $O(n \log n)$.

A stronger notion than walkability is that of straight walkability. A walkable polygon is said to be *straight walkable* if the guards are able to complete the walk, again under the condition of constant mutual visibility, without having to *backtrack* during their walk to ensure this. Tseng et al.[11] presented an $O(n \log n)$ algorithm to determine whether P is straight walkable, and within the same time bound showed how to generate all such pairs. An even stronger notion is that of discrete straight walkability [1]. A straight walkable polygon is said to be *discretely straight walkable* if only one of the guards is allowed to walk at a time, while the other remains stationary at a vertex. Testing discrete straight walkability and computing all discrete straight walkable pairs of points can be performed in $O(n \log n)$ time [9],

The **main results** of this paper are surprisingly simple, optimal linear-time algorithms that determine all pairs of points of P with respect to which the polygon P is: (i) walkable, (ii) straight walkable, and (iii) discretely straight walkable. While we improve the best known results for (discrete) straight walkability, the surprising element is that all our algorithms use only simple sweeps and elementary data structures.

2 Preliminaries

Let P be a simple polygon with n vertices. The open (closed) clockwise chain of P from u to v is denoted by $P(u, v)$ ($P[u, v]$). Half-open chains will be denoted by $P[u, v)$ or $P(u, v]$, depending on which end is open. For points p and q that belong to a boundary chain of P with distinct end-points, if p precedes q in counterclockwise(clockwise) order, we denote this by $p \leq_{ccw} q$ ($p \leq_{cw} q$). Every reflex vertex u (of P) determines two components, a clockwise component $P_{cw}(u)$, consisting of the chain $P[u, u_{cw}]$ and the bounding chord $\overline{uu_{cw}}$; and a counterclockwise component $P_{ccw}(u)$ a counterclockwise component $P_{ccw}(u)$, consisting of the chain $P[u_{ccw}, u]$ and the bounding chord $\overline{uu_{ccw}}$ (see Fig. 1). The bounding chords $\overline{uu_{cw}}$ and $\overline{uu_{ccw}}$ are called respectively the clockwise and the counterclockwise chords of u. A component is called *non-redundant* if it does not contain any other component.

A walk is said to be in an *s-deadlock* (*t-deadlock*) if two guards, both starting from s (t), have reached points from where neither can move forward without losing sight of each other. Points u and v illustrate this situation in Fig. 1 for two guards starting from s. Various cases of *s-deadlock* and *t-deadlock* are discussed in [11]. A walk with respect to a pair of points, (s,t), is deadlock free iff it is both s-deadlock free and t-deadlock free. Equivalently, a walk with respect to the pair (s,t) is s-deadlock (t-deadlock) free iff, for a walk originating at s (t), the moment a guard enters a component, say Q, the other guard must be already inside Q. We will use this latter formulation in designing our algorithms.

A polygon P is LR-visible [5] if there exist a pair of points s and t on P such that $P(s,t)$ and $P(t,s)$ are weakly-visible from each other. It was shown [8] that P is 2-walkable iff it is *LR-visible* and has a deadlock-free region. Bhattacharya and Ghosh [2] have given a simple linear time algorithm to determine if P is LR-visible. The algorithm also allows one to compute the shortest path tree of P from an arbitrary vertex. Thus if P is LR-visible, all its non-redundant components can be determined in linear time [4, 5]. Using these components, it is easy to compute in $O(n)$ time all possible pairs of boundary chains $(A_i, B_i), i = 0, \ldots, m$, such that for any $s \in A_i$ and any $t \in B_i$, P is LR-visible with respect to the pair of boundary points (s,t). The above pairs of chains can be determined with either all A_i's disjoint or with all the B_i's disjoint. The disjoint ones are reported in counterclockwise order. This latter output is the input to our algorithms in this paper.

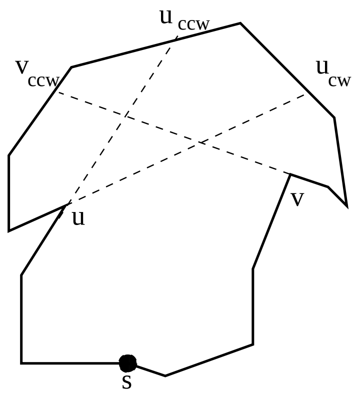

Figure 1: Components $P_{cw}(u)$ and $P_{ccw}(v)$ creating s-deadlock

Additional conditions are necessary for a 2-walkable polygon to be (discrete) straight walkable. A polygon is straight walkable (discrete straight walkable) if there exists a pair of points s and t on P such that P is 2-walkable with respect to s and t, and the chains $P(s,t)$ and $P(t,s)$ do not have a configuration called a "wedge" ("semi-wedge").

3 2-walkability

3.1 Overview

From the pairs of chains $(A_i, B_i), i = 0, \ldots, m$ that describe the LR-visible pairs of points of P, we choose a set of point pairs $(s_i, t_i) \in A_i \times B_i, i = 0, \ldots, m$. The points satisfy the property that $s_0 \leq_{ccw} s_1 \leq_{ccw} \cdots \leq_{ccw} s_m \leq_{ccw} t_0 \leq_{ccw} t_1 \leq_{ccw} \cdots \leq_{ccw} t_m \leq_{ccw} s_0$. We assume that the s_i's and the t_i's are not reflex vertices. We shall show in a later section how to choose such a set of point-pairs.

The main idea is to find a maximal deadlock-free region around each s_i and t_i by using a clockwise sweep by a clockwise guard and a counterclockwise sweep by a counterclockwise guard. This is achieved by a pair of "cooperating processes" termed as a *walk* where the two guards traverse the boundary of the polygon without losing sight of each other. In the i-th iteration, $walk_i$ can be described as follows: the counterclockwise guard, starting from s_i, is targeted to reach s_{i+1}, while the clockwise guard, also starting at s_i, is targeted to go as far as possible without crossing t_i. In particular, the clockwise guard begins to walk towards t_i, unless it encounters a reflex vertex v_l whose clockwise component $P_{cw}(v_l)$ does not contain the current position of the counterclockwise guard. Since any advancement of the clockwise guard would result in the loss of co-visibility, control passes to the counterclockwise guard who begins to walk from its current position in an attempt to enter this component. This walk of the counterclockwise guard can terminate in one of three different ways.

- It reaches its preassigned destination (s_{i+1}), in which case $walk_i$ is terminated after stacking some related parameters;
- It succeeds in entering $P_{cw}(v_l)$, in which case it waits, while the clockwise guard resumes advancing towards t_i;
- It encounters a reflex vertex v_r and $P_{ccw}(v_r)$ does not contain v_l in which case we report a deadlock. We now let the counterclockwise guard reach its target s_{i+1}, and terminate $walk_i$.

After we have generated all the walks, $walk_i$, $i = 0, \ldots, m$, the walk parameters left on the stack correspond to (s_i, t_i) pairs that permit s-deadlock free walks. We shall prove this claim in the next section.

3.2 Details of the Algorithm

To implement $walk_i$ described above, we need to be able to answer the following queries efficiently. Let the l and r be the current positions of the clockwise and the counterclockwise guards respectively. If l (r) is a reflex vertex, let l_{cw} (r_{ccw}) be the other endpoint of the bounding chords of $P_{cw}(l_i)$ ($P_{ccw}(r_i)$).

Visibility: Are $l, r \in P$ co-visible?

Clockwise containment: Is r in $P_{cw}(l)$?

Counterclockwise containment: Is l in $P_{ccw}(r)$?

Crossing-over event: Is $r \leq_{ccw} l_{cw}$?

Since (s_i, t_i) is an LR-visible pair, every component of P must contain one of s_i or t_i. Below, we will often appeal to this fact. In our description below, we differentiate between the cases: $i = 0$ and $i > 0$.

3.2.1 Implementing the queries for $walk_0$

Visibility query As seen below, when the guards are at location l_0 and r_0 during the walk process, we need to answer the visibility query. It may be noted that this is not a general visibility query since the two guards are monotonically traversing from point s_0 towards their destinations (s_1 and t_0). In order to answer this query efficiently (in constant amortized time), we preprocess P

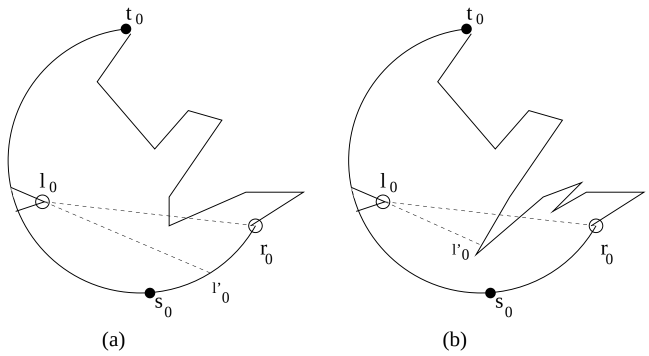

Figure 2: l_{cw} can lie in $P(r_0, s_0]$ or on $P[t_0, r_0)$.

with respect to the pair (s_0, t_0). The preprocessing involves computing the "restricted shortest paths" from s_0 and t_0 to every vertex on P. A restricted shortest path from s_0 to vertex $u \in P(s_0, t_0)$ (or, $v \in P(t_0, s_0)$) is the shortest path between s_0 and u (v) ignoring the effect of $P(v, s_0)$ ($P(u, s_0)$). This is a standard procedure that has been used in several papers (refer to [4, 2]) for exactly the same reason (i.e., answer visibility queries).

Clockwise containment query Let $\overline{l_0 l_{cw}}$ denote the bounding chord of the component $P_{cw}(l_0)$. We determine, in *constant time*, if $l_{cw} \in P(t_0, s_0)$. Note that this is easily determined by inspecting the last edges on the shortest paths from s_0 and t_0 to l_0, as these paths do not cross. If *not*, then $P_{cw}(l_0)$ contains r_0. If *yes*, as $P(r_0, l_0)$ does not intersect $\overline{l_0 r_0}$, we test if $P(t_0, r_0)$ intersects this segment. If it *does not*, we easily determine if $l_{cw} \in P(r_0, s_0)$. If it *does*, we resolve between the two cases shown in Fig. 2 as follows. We traverse the boundary chain from r_0 to t_0 to find the first vertex x from which l_0 is visible. Clearly, x is reflex. Since P is LR-visible with respect to the pair (s_0, t_0), this vertex also has the property that all counterclockwise components of the reflex vertices of $P[x, r_0)$ contain l_0.

To find x, the counterclockwise guard walks towards t_0, stopping at s_1 (= *CCW.limit*) or x, whichever comes earlier. This stop becomes the new r_0. In either case, the property of nonintersection of the segment $\overline{l_0 r_0}$ and the chain $P(r_0, l_0)$ is preserved.

Counterclockwise containment query Let $\overline{r_0 r_{ccw}}$ denote the counterclockwise chord of the component $P_{ccw}(r_0)$. We determine, in *constant time*, if $r_{ccw} \in P(s_0, t_0)$. If *not*, it follows from the LR-visibility of P that the component $P_{ccw}(r_0)$ contains s_0 and, *a fortiori,* l_0. If *yes* we check, in constant time again, if the ray $\overrightarrow{r_0 r_{ccw}}$ is directed into the pocket determined by the chain $P[r_0, l_0]$ and the chord $\overline{r_0 l_0}$. If *yes* then the component $P_{ccw}(r_0)$ contains l_0, otherwise not.

Crossing-over event query We must determine if the clockwise ray shot from l_0 hits r_0. In other words, the counterclockwise guard must find l_{cw} to effect the crossover. We know that $P[r_0, l_0]$ does not intersect the interior of $\overline{l_0 r_0}$. Also, $P[l_0, t_0]$ does not intersect the interior of $\overline{l_0 r_0}$. Hence $r_0 = l_{cw}$ iff a ray from r_0 in the direction of l_0 hits the chain $P[s_0, t_0]$.

We summarize the above discussions with the following claim:

Lemma: 3.1 *We can check in (amortized) constant time if* $r_0 \in P_{cw}(l_0)$, $l_0 \in P_{ccw}(r_0)$, *and if the crossing-over event has occurred, i.e., when the counterclockwise guard reaches* l_{cw}.

3.2.2 Implementing the queries for $walk_i$

We will now discuss how to answer the same queries for an arbitrary LR-pair (s_i, t_i), where $s_i \in P(t_0, s_0)$ and $t_i \in P(s_0, t_0)$. Let l_i and r_i denote the positions of the clockwise and counterclockwise guards respectively.

Clockwise containment query The following cases arise. **Case 1:** $l_i \in P[s_0, t_i]$ and $r_i \in P[t_0, s_i]$. This problem is the same as the clockwise containment problem for (s_0, t_0). **Case 2:** $l_i \in P[s_0, t_i]$ and $r_i \in P[t_i, t_0]$. In this case $P_{cw}(l_i)$ contains r_i since l_{cw} cannot lie on $P[l_i, t_0]$. **Case 3:** $l_i \in P[s_i, s_0]$ and $r_i \in P[t_i, t_0]$. Again, this case is very similar to Case 1. **Case 4:** $l_i \in P[s_i, s_0]$ and $r_i \in P[t_0, s_i]$. Clearly, if $l_{cw} \in P[s_0, t_0]$, $P_{cw}(l_i)$ does not contain r_i. Suppose this is not the case. As P is LR-visible with respect to (s_0, t_0), $l_{cw} \notin P[l_i, s_0]$. We know that $P(r_i, l_i)$ does not intersect $\overline{l_i r_i}$. Again, since P is LR-visible with respect to (s_0, t_0), $P[t_0, r_i)$ cannot intersect $\overline{l_i r_i}$. Hence, we can easily determine whether $l_{cw} \in P[r_i, l_i]$ or $l_{cw} \in P[t_0, r_i]$.

Counterclockwise containment query Again, this query is resolved by examining the following cases, given that $P_{cw}(l_i)$ does not contain r_i.

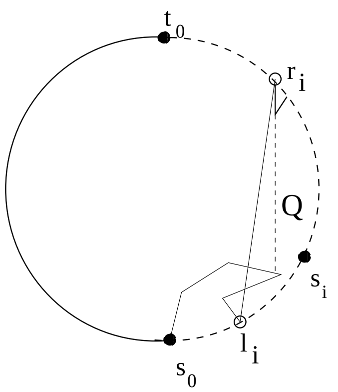

Figure 3: r_{ccw} cannot lie on $P[l_i, s_0]$

Case 1: $l_i \in P[s_0, t_i]$ and $r_i \in P[t_0, s_i]$. This is similar to the case for (s_0, t_0). **Case 2:** $l_i \in P[s_0, t_i]$ and $r_i \in P[t_i, t_0]$. In this case r_{ccw} must lie on $P[t_0, s_0]$, since P is LR-visible with respect to (s_0, t_0). Therefore, $P_{ccw}(r_i)$ contains l_i. **Case 3:** $l_i \in P[s_i, s_0]$ and $r_i \in P[t_i, t_0]$. Again, this case is very similar to Case 1. **Case 4:** $l_i \in P[s_i, s_0]$ and $r_i \in P[t_0, s_i]$. Clearly, if $r_{ccw} \in P[s_0, t_0]$ then $P_{ccw}(r_i)$ does not contain l_i. Suppose $r_{ccw} \in P[t_0, s_0]$. Suppose the ray from r_i towards r_{ccw} is directed towards the interior of the pocket determined by $P[r_i, l_i]$ and $\overline{l_i r_i}$. In this case, $r_{ccw} \notin P[l_i, s_0]$ as shown in Fig. 3, otherwise l_i is not visible from $P[s_0, t_0]$, contradicting the LR-visibility of P with respect to (s_0, t_0). If the ray is directed away from the pocket, then $r_{ccw} \in P[l_i, s_0]$ and so $P_{ccw}(r_i)$ does not contain l_i.

Crossing-over event query Again , we must detetmine if the ray shot from l_i hits r_i, i.e., if $l_{cw} = r_i$. **Case 1:** $l_i \in P[s_0, t_i]$ and $r_i \in P[t_0, s_i]$. This problem is the same as the crossing-over event problem for (s_0, t_0). **Case 2:** $l_i \in P[s_0, t_i]$ and $r_i \in P[t_i, t_0]$. This case cannot arise as l_{cw} must lie on $P[t_0, s_0]$. **Case 3:** $l_i \in P[s_i, s_0]$ and $r_i \in P[t_i, t_0]$. This problem is the same as the crossing-over event problem for (s_0, t_0). **Case 4:** $l_i \in P[s_i, s_0]$ and $r_i \in P[t_0, s_i]$. First we check if $l_{cw} \in P[s_0, t_0]$. If so, $r_i \neq l_{cw}$. Suppose $l_{cw} \in P[t_0, s_0]$. Since P is LR-visible with respect to (s_0, t_0), the first intersection of $P[t_0, l_i]$ in counterclockwise order with $\vec{l_i l_{cw}}$ is l_{cw}. Since $P(r_i, l_i)$ does not intersect $\overline{l_i r_i}$ in the interior, in our case $r_i = l_{cw}$. We summarize the above discussions as follows:

Lemma: 3.2 *We can check in (amortized) constant time if $r_i \in P_{cw}(l_i)$, $l_i \in P_{ccw}(r_i)$, and if the crossing-over event has occurred, i.e., when $r_i = l_{cw}$.*

Deadlock-Test The process $walk_i$ is described a slightly more general form in the procedure DEADLOCK-TEST as shown in Fig. 4. It accepts four arguments: the start points *CW.start* and *CCW.start* of the clockwise and counterclockwise guards, and their respective destinations, *CW.limit* and *CCW.limit*. The algorithm returns the maximal s-deadlock free chain $P[r, l]$ in $P[CCW.limit, CW.limit]$.

Several observations are in order. We note that if r_i and l_i are the current positions of the two guards, the boundary chain $P[r_i, l_i]$ does not obstruct their mutual visibility. This is easily seen from the invariance of the following two assertions during the movement of the guards from their start positions to the current positions.

- All the clockwise components generated by the reflex vertices on $P[s_i, l_i]$, with the possible exception of l_i, contain r_i.
- All the counterclockwise components generated by the reflex vertices on $P[r_i, s_i]$, with the possible exception of r_i, contain l_i.

In summary, we conclude with this theorem:

Theorem: 3.1 *Given a pair of points (s_i, t_i) with respect to which P is LR-visible,* DEADLOCK-TEST(s_i, s_i, t_i, s_{i+1}) *returns a pair of points (l_i, r_i); if $r_i = s_{i+1}$, then the chain $P(r_i, l_i)$ is deadlock-free, else $P(r_i, l_i)$ is a deadlock region with respect to a walk starting at s_i. The cost is linear in the number of vertices contained in the chain $P(r_i, l_i)$.*

3.3 Reporting all deadlock free pairs

We now show how to combine the results of the calls to procedure DEADLOCK-TEST from consecutive iterations. Suppose that after iterations i and $i +$

Algorithm DEADLOCK-TEST($CW.start, CCW.start, CW.limit, CCW.limit$)
Output A pair of points (l, r). If $r \neq CCW.limit$, $P[r, l]$ has an s-deadlock.
begin
  $l \leftarrow CW.start$; $r \leftarrow CCW.start$;
  { l (r) represents current position of clockwise (counterclockwise) guard }
  {The clockwise guard moves}
  Loop1: while (l is not a reflex vertex and $l <_{cw} CW.limit$) do
      $l \leftarrow next(l)$;
    if $CW.limit <=_{cw} l$ then
      Output (CW.limit,CCW.limit)
      { *$P[CCW.limit, CW.limit]$ has no s-deadlock.*};
      Return
    else {l is a reflex vertex.}
      if $P_{cw}(l)$ contains r then
        $l \leftarrow next(l)$;
        go to Loop1;
      else go to Loop2
  { l is a reflex vertex and r is not contained in $P_{cw}(l)$}
  {The counterclockwise guard moves}
  Loop2: while (r is not a reflex vertex and $CCW.limit <_{cw} r$ and $l_{cw} <_{cw} r$) do
      $r \leftarrow next(r)$;
    if $r <=_{cw} l_{cw}$ then
    { counterclockwise guard has entered the clockwise component at l;}
    { hand over control to clockwise guard }
      $r = l_{cw}$
      $l \leftarrow next(l)$;
      go to Loop1;
    else if {r = CCW.limit }
      Output (l, CCW.limit)
      Return
    else {r is reflex }
      if $P_{ccw}(r)$ contains l then
    { counterclockwise guard continues in Loop 2}
        $r \leftarrow next(r)$;
        go to Loop2;
      else {deadlock situation}
        Output (l, r)
        Return
end

Figure 4: Algorithm DEADLOCK-TEST

1, DEADLOCK-TEST indicate that $P[r_i, l_i]$ and $P[r_{i+1}, l_{i+1}]$ are deadlock free. Since $r_i = s_{i+1}$ and $l_{i+1} \leq_{ccw} s_{i+1}$, the two regions overlap and we can combine the results to claim that $P[r_{i+1}, l_i]$ is deadlock-free. Furthermore, in order to avoid either the clockwise guard or the counterclockwise guard from repeatedly going over any overlapping portions in different iterations (which will happen if $l_{i+1} \leq_{ccw} s_i$), the walks make use of information stored on a special *snapshot-stack* and skip over these regions during their walks.

Whenever an iteration reports a deadlock region, then the corresponding boundary chain of P is marked as *ineligible*. Future iterations will skip over this portion of the polygon, traversed by the clockwise guard.

The algorithm ALL-POINTS-PAIRS below implements the walks for all the iterations by making calls to DEADLOCK-TEST to determine all deadlock free regions.

ALL-POINTS-PAIRS needs to maintain lists of boundary objects not traversed by the clockwise guard yet, as well as a list of boundary objects labeled as "ineligible". Both lists are easily maintained as stacks.

Algorithm ALL-POINTS-PAIRS
Output Boundary chains that admit s-deadlock.
begin
  Set $i \leftarrow 0$; $s_{m+1} \leftarrow t_m$;
  while ($i \leq m$) do
      CW.start $\leftarrow$ CCW.start $\leftarrow s_i$; CW.limit $\leftarrow t_i$; CCW.limit $\leftarrow s_{i+1}$;
  Loop: $(l, r) \leftarrow$ DEADLOCK-TEST(CW.start, CCW.start, CW.limit, CCW.limit)
      If $r =$ CCW.limit then
          PUSH(snapshot-stack,l, r,CW.limit);
          $i \leftarrow$ index(CCW.limit);
      else
          Label the chain $P(r, l)$ as ineligible.
          if (snapshot-stack $\neq$ empty) then
              (a,b,c) $\leftarrow$ POP(snapshot-stack);
              if $P(b, a)$ does not contain l then
                  CW.start $\leftarrow l$; CCW.start $\leftarrow r$; Go to Loop;
              else
                  CW.start $\leftarrow$ a; CCW.start $\leftarrow r$; CW.limit $\leftarrow c$; Go to Loop;
          else $i \leftarrow$ index(CCW.limit);
  Traverse the ineligible boundary chains and remove all pairs whose associated s points lie on these parts. Report the non-deleted pairs as the pairs that allow s-deadlock free walk.
end

Since the boundary chains are traversed at most once in clockwise order and at most once in counterclockwise order and since the subroutine DEADLOCK-TEST outputs chain in time proportional to the size of the chain, we claim therefore:

Lemma: 3.3 *All the pairs of $(s_i, t_i), i = 0, 1, \ldots, m$ that allow s-deadlock free and t-deadlock free walks can be correctly determined in $O(n)$ time.*

Given the lists produced in Lemmas 3.3 it is easy determine the pairs of points with respect to which the polygon is 2-walkable. Therefore, we have:

Theorem: 3.2 *Given $(s_i, t_i), i = 0, 1, \ldots, m$ and P, it is possible to determine in optimal time the pairs of points among $(s_i, t_i), i = 0, 1, \ldots, m$ with respect to which P is 2-walkable.*

3.4 Reporting all pairs of bounding chains with respect to which P has no s-deadlock

[4, 5, 11] gave methods to determine the pairs of bounding chains $(A_i, B_i), i = 0, 1 \ldots m$ of P such that for any arbitrary $(s, t) \in (A_i, B_i)$, the polygon P is LR-visible. The pairs can be determined with either A_i's all disjoint or B_i's all disjoint. We assume all A_i's to be disjoint when they are tested for s-deadlock and assume all B_i's to be disjoint when they are tested for t-deadlock.

We describe our method for s-deadlock only. Our objective is to identify the parts A'_i and B'_i of A_i and B_i respectively such that for any arbitrary $(s, t) \in (A'_i, B'_i)$, the polygon P is s-deadlock free. Let us consider the pair (A_i, B_i). In order to test (A_i, B_i) for s-deadlock, we replace this pair (A_i, B_i) by a set of points $(s_j, t_j), j = 0, 1, \ldots$ where $s_0, s_1, \ldots$ belong to A_i and $t_0, t_1, \ldots$ belong to B_i such that s-deadlock free chains corresponding to (A_i, B_i) can be determined once we know the status of $(s_j, t_j)'$s for s-deadlock.

Let $v_1, v_2, \ldots, v_k$ be the reflex vertices on $A_i = p_{cw}(a_i, a'_i)$. Let $v_0 = a_i$ and $v_{k+1} = a'_i$. Let $A_{i_j} = P(v_{j-1}, v_j), j = 1, 2, \ldots, k+1$. Let (s_j, t_j) be any arbitrary pair in (A_{i_j}, B_i). We can show that

Lemma: 3.4 *P is s-deadlock free for chain (A_{i_j}, B_i) if and only if P is s-deadlock free for the pair (s_j, t_j).*

From the above lemma we see that we can replace each chain pair by point pairs whose number is one more than than the number of reflex vertices in A_i. Since A'_is are disjoint, the total number of points generated is at most n. Also it should be noted that s's are all non-reflex points. Thus we have the two theorems below.

Theorem: 3.3 *Given LR-visible chain pairs $(A_i, B_i), i = 1, \ldots, m$, where all A_i's are disjoint, all s-deadlock-free chain pairs can be determined in $O(n)$ time.*

Theorem: 3.4 *All 2-walkability pairs of an arbitrary polygon P can be determined in optimal linear time.*

4 Straight Walkability

Our input is a set chain-pairs (A_i, B_i) such that each chain-pair (A_i, B_i) admits 2-walkability. We outline an algorithm to determine if P is straight walkable and, if so, determine the chain-pairs which admit such walkability.

The concept of a wedge is important for determining whether P is straight walkable [8]. A wedge is a chain $P[v, u]$, defined by two reflex vertices u and v such that $P_{ccw}(u)(P_{cw}(v))$ contain $v(u)$ and the clockwise chord $\overline{uu}_{cw}$ and the counterclockwise chord $\overline{vv}_{ccw}$ intersect. A wedge is called non-redundant if it does not contain any other wedge or a component. Otherwise, it is called redundant.

Let W be any wedge in P. Tseng at al [11] showed that

Lemma: 4.1 *If P is straight walkable with respect to (s, t) then either $s \in W$ or $t \in W$.*

Therefore P is not straight walkable if it has at least three or more disjoint wedges. To determine if P is straight walkable we proceed along the following lines. Let P be 2-walkable with respect to the point pair (s, t). We compute all the non-redundant wedges in stages. First, all the non-redundant wedges containing s or t are computed. Then we compute all the non-redundant wedges contained in $P(s, t)$ and $P(t, s)$, keeping track of the number of disjoint wedges computed so far. We stop if three disjoint wedges are detected. We can implement the above ideas using some of the techniques developed in Section 3. Hence

Theorem: 4.1 *We can determine all pairs of chains with respect to which P is straight walkable in linear time.*

5 Discrete Straight Walkability

The problem of determining the chain-pairs with respect to which P is discretely straight walkable can also be determined in a similar manner.

The concept of a semi-wedge [9] is important here. A semi-wedge is a chain $P[v, u]$, defined by two reflex vertices u and v such that $P_{ccw}(u)(P_{cw}(v))$ contain $v(u)$ and the clockwise chord $\overline{uu}_{cw}$ (counterclockwise chord $\overline{vv}_{ccw}$), while u_{ccw} and v_w belong to the relative interior of some edge. A semi-wedge is called non-redundant if it does not contain any other semi-wedge or wedge. Otherwise, it is called redundant.

Analogous to the result by [11], if W is a semi-wedge of P, [9] showed that

Lemma: 5.1 *If P is discretely straight walkable with respect to (s, t) then either $s \in W$ or $t \in W$.*

This implies that if P is not discretely straight walkable with respect to any (s, t) pair if it contains three disjoint non-redundant semi-wedges.

Our algorithm proceeds in a similar manner: determine three disjoint semi-wedges with respect to an arbitrary (s, t) pair. Details are provided in Appendix C. The claims are summarized in the following theorem:

Theorem: 5.1 *We can determine all pairs of chains with respect to which P is discretely straight walkable in linear time.*

6 Conclusions

In this paper, we present surprisingly simple and efficient (optimal, linear-time) algorithms for many 2-guard problems. We believe that this work will help to bridge the gap between impractical linear-time algorithms for visibility problems and practical efficient implementations.

References

[1] E. M. Arkin, M. Held, J. S. B. Mitchell, S. S. Skiena. Hamiltonian Triangulations for Fast Rendering. *Visual Computer*, 12:429-444, 1996.

[2] B. K. Bhattacharya and S. K. Ghosh. Characterizing LR-visible polygons and related problems. *Proc. of CCCG 1998; (accepted to CGTA, 2000).*

[3] B. Chazelle. Triangulating a simple polygon in linear time. *Proc. of the 31-st Annual IEEE FOCS* 220-229, 1990.

[4] G. Das, P. J. Heffernan and G. Narasimhan. Finding all weakly-visible chords of a polygon in linear time. *Nordic J. of Comput.* 1:433-457, 1994.

[5] G. Das, P. J. Heffernan and G. Narasimhan. LR-visibility in polygons. *Computational Geometry: Theory and Applications* 7:37-57, 1997.

[6] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear time algorithms for visibility and shortest path problems inside a triangulated simple polygon. *Algorithmica*, 2:209–233, 1987.

[7] P. Heffernan. An optimal algorithm for the two-guard problem. *Intl. Journal on Comp. Geometry and Applications* 6:15-44, 1996.

[8] C. Icking and R. Klein. The two guards problem. *Intl. Journal on Comp. Geometry and Applications* 2:257-285, 1992.

[9] G. Narasimhan. On Hamiltonian Triangulations of Simple Polygons. *Intl. Journal on Comp. Geometry and Applications* 9(3):261-275, 1999.

[10] J. O'Rourke. Art Gallery Theorems and Algorithms. Oxford University Press, New York, NY, 1987.

[11] L.H. Tseng, P. Heffernan and D.T. Lee. Two-guard walkability of simple polygons. *Intl. Journal on Comp. Geometry and Applications* 8(1):85-116, 1998.