An optimal algorithm for the separating common tangents of two polygons
Abrahamsen, Mikkel

Published in:
Leibniz International Proceedings in Informatics

DOI:
10.4230/LIPIcs.SOCG.2015.198

Publication date:
2015

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (APA):
An Optimal Algorithm for the Separating Common Tangents of Two Polygons

Mikkel Abrahamsen

Department of Computer Science, University of Copenhagen
Universitetsparken 5
DK-2100 Copenhagen Ø
Denmark
miab@di.ku.dk

Abstract
We describe an algorithm for computing the separating common tangents of two simple polygons using linear time and only constant workspace. A tangent of a polygon is a line touching the polygon such that all of the polygon lies to the same side of the line. A separating common tangent of two polygons is a tangent of both polygons where the polygons are lying on different sides of the tangent. Each polygon is given as a read-only array of its corners. If a separating common tangent does not exist, the algorithm reports that. Otherwise, two corners defining a separating common tangent are returned. The algorithm is simple and implies an optimal algorithm for deciding if the convex hulls of two polygons are disjoint or not. This was not known to be possible in linear time and constant workspace prior to this paper.

An outer common tangent is a tangent of both polygons where the polygons are on the same side of the tangent. In the case where the convex hulls of the polygons are disjoint, we give an algorithm for computing the outer common tangents in linear time using constant workspace.

1 Introduction
The problem of computing common tangents of two given polygons has received some attention in the case where the polygons are convex. For instance, it is necessary to compute outer common tangents of disjoint convex polygons in the classic divide-and-conquer algorithm for the convex hull of a set of \( n \) points in the plane by Preparata and Hong [12]. They give a naïve linear time algorithm for outer common tangents since that suffices for an \( O(n \log n) \) time convex hull algorithm. The problem is also considered in various dynamic convex hull algorithms [5, 8, 11]. Overmars and van Leeuwen [11] give an \( O(\log n) \) time algorithm for computing an outer common tangent of two disjoint convex polygons when a separating line is known, where each polygon has at most \( n \) corners. Kirkpatrick and Snoeyink [9] give an \( O(\log n) \) time algorithm for the same problem, but without using a separating line. Guibas et al. [7] give an \( \Omega(\log^2 n) \) lower bound on the time required to compute an outer common tangent of two intersecting convex polygons, even if it is known that they intersect in at most two points. They also describe an algorithm achieving that bound.

* Research partly supported by Mikkel Thorup’s Advanced Grant from the Danish Council for Independent Research under the Sapere Aude research career programme.
Touissaint [13] considers the problem of computing separating common tangents of convex polygons and notes that the problem occurs in problems related to visibility, collision avoidance, range fitting, etc. He gives a linear time algorithm. Guibas et al. [7] give an $O(\log n)$ time algorithm for the same problem.

All the here mentioned works make use of the convexity of the polygons. If the polygons are not convex, one can use a linear time algorithm to compute the convex hulls before computing the tangents [6, 10]. However, if the polygons are given in read-only memory, it requires $\Omega(n)$ extra bits to store the convex hulls. In this paper, we also obtain linear time while using only constant workspace, i.e. $O(\log n)$ bits. For the outer common tangents, we require the convex hulls of the polygons to be disjoint. There has been some recent interest in constant workspace algorithms for geometric problems, see for instance [1, 2, 3, 4].

The problem of computing separating common tangents is of special interest because these only exist when the convex hulls of the polygons are disjoint, and our algorithm detects if they are not. Thus, we also provide an optimal algorithm for deciding if the convex hulls of two polygons are disjoint or not. This was to the best of our knowledge not known to be possible in linear time and constant workspace prior to our work.

1.1 Notation and some basic definitions

Given two points $a$ and $b$ in the plane, the closed line segment with endpoints $a$ and $b$ is written $ab$. When $a \neq b$, the line containing $a$ and $b$ which is infinite in both directions is written $L(a, b)$.

Define the dot product of two points $x = (x_0, x_1)$ and $y = (y_0, y_1)$ as $x \cdot y = x_0y_0 + x_1y_1$, and let $x^\perp = (-x_1, x_0)$ be the counterclockwise rotation of $x$ by the angle $\pi/2$. Now, for three points $a$, $b$, and $c$, we define $T(a, b, c) = \text{sgn}((b - a) \cdot (c - b))$, where $\text{sgn}$ is the sign function. $T(a, b, c)$ is 1 if $c$ is to the left of the directed line from $a$ to $b$, 0 if $a$, $b$, and $c$ are collinear, and $-1$ if $c$ is to the right of the directed line from $a$ to $b$. We see that

$$
T(a, b, c) = T(b, c, a) = T(c, a, b) = -T(c, b, a) = -T(b, a, c) = -T(a, c, b).
$$

We also note that if $a'$ and $b'$ are on the line $L(a, b)$ and appear in the same order as $a$ and $b$, i.e., $(b - a) \cdot (b' - a') > 0$, then $T(a, b, c) = T(a', b', c)$ for every point $c$.

The left half-plane LH(a, b) is the closed half plane with boundary $L(a, b)$ lying to the left of directed line from $a$ to $b$, i.e., all the points $c$ such that $T(a, b, c) \geq 0$. The right half-plane RH(a, b) is just LH(b, a).

Assume for the rest of this paper that $P_0$ and $P_1$ are two simple polygons in the plane with $n_0$ and $n_1$ corners, respectively, where $P_k$ is defined by its corners $p_k[0], p_k[1], \ldots, p_k[n_k - 1]$ in clockwise or counterclockwise order, $k = 0, 1$. Indices of the corners are considered modulo $n_k$, so that $p_k[i]$ and $p_k[j]$ are the same corner when $i \equiv j \pmod{n_k}$.

We assume that the corners are in general position in the sense that $P_0$ and $P_1$ have no common corners and the combined set of corners $\bigcup_{k=0,1} \{p_k[0], \ldots, p_k[n_k - 1]\}$ contains no three collinear corners.

A tangent of $P_k$ is a line $\ell$ such that $\ell \not\subset P_k$ and $P_k$ are not disjoint and such that $P_k$ is contained in one of the closed half-planes defined by $\ell$. The line $\ell$ is a common tangent of $P_0$ and $P_1$ if it is a tangent of both $P_0$ and $P_1$. A common tangent is an outer common tangent if $P_0$ and $P_1$ are on the same side of the tangent, and otherwise the tangent is separating. See Figure 1.

For a simple polygon $P$, we let $\mathcal{H}(P)$ be the convex hull of $P$. The following lemma is a well-known fact about $\mathcal{H}(P)$. 

SoCG’15
Lemma 1. For a simple polygon $P$, $\mathcal{H}(P)$ is a convex polygon and the corners of $\mathcal{H}(P)$ appear in the same cyclic order as they do on $P$.

The following lemma states folklore properties of tangents of polygons.

Lemma 2. A line is a tangent of a polygon $P$ if and only if it is a tangent of $\mathcal{H}(P)$. Under our general position assumptions, the following holds: If one of $\mathcal{H}(P_0)$ and $\mathcal{H}(P_1)$ is completely contained in the other, there are no outer common tangents of $P_0$ and $P_1$. Otherwise, there are two or more. There are exactly two if $P_0$ and $P_1$ are disjoint. If $\mathcal{H}(P_0)$ and $\mathcal{H}(P_1)$ are not disjoint, there are no separating common tangents of $P_0$ and $P_1$. Otherwise, there are exactly two.

2 Computing separating common tangents

In this section, we assume that the corners of $P_0$ and $P_1$ are both given in counterclockwise order. We prove that Algorithm 1 returns a pair of indices $(s_0, s_1)$ such that the line $L(p_0[s_0], p_1[s_1])$ is a separating common tangent with $P_k$ contained in $\text{RHP}(p_1-k[s_1-k], p_k[s_k])$ for $k = 0, 1$. If the tangent does not exist, the algorithm returns NULL. The other separating common tangent can be found by a similar algorithm if the corners of the polygons are given in clockwise order and ‘−’ is changed to ‘−’ in lines 3 and 10.

The algorithm traverses the polygons in parallel one corner at a time using the indices $t_0$ and $t_1$. We say that the indices $(s_0, s_1)$ define a temporary line, which is the line $L(p_0[s_0], p_1[s_1])$. We update the indices $s_0$ and $s_1$ until the temporary line is the separating common tangent. At the beginning of an iteration of the loop at line 2, we traverse one corner $p_u[t_u]$ of $P_u$, $u = 0, 1$. If the corner happens to be on the wrong side of the intermediate line, we make the temporary line pass through that corner by updating $s_u$ to $t_u$ and we reset $t_1-u$ to $s_1-u+1$. The reason for resetting $t_1-u$ is that a corner of $P_1-u$ which was on the correct side of the old temporary line can be on the wrong side of the new line and thus needs to be traversed again.

We show that if the temporary line is not a separating common tangent after each polygon has been traversed twice by the loop beginning at line 2, then the convex hulls of the polygons are not disjoint. Therefore, if a corner is found to be on the wrong side of the line defined by
Figure 2 Algorithm 1 running on two polygons $P_0$ and $P_1$. The corners $p_k[s_k^{(i)}]$ are marked and labeled as $s_k^{(i)}$ for the initial values $s_k^{(0)}$ and after each iteration $i$ where an update of $s_k$ happens. The segments $p_0[s_0^{(i)}]p_1[s_1^{(i)}]$ on the temporary line are dashed.

Algorithm 1: SeparatingCommonTangent($P_0, P_1$)

1. $s_0 \leftarrow 0$; $t_0 \leftarrow 1$; $s_1 \leftarrow 0$; $t_1 \leftarrow 1$; $u \leftarrow 0$
2. while $t_0 < 2n_0$ or $t_1 < 2n_1$
   3. if $T(p_{1-u}[s_{1-u}], p_u[s_u], p_u[t_u]) = 1$
   4. $s_u \leftarrow t_u$
   5. $t_{1-u} \leftarrow s_{1-u} + 1$
   6. $t_u \leftarrow t_{u} + 1$
   7. $u \leftarrow 1 - u$
8. for each $u \leftarrow \{0, 1\}$
   9. for each $t \leftarrow \{0, \ldots, n_u - 1\}$
10. if $T(p_{1-u}[s_{1-u}], p_u[s_u], p_u[t]) = 1$
11. return NULL
12. return $(s_0, s_1)$

$(s_0, s_1)$ in the loop beginning at line 8, no separating common tangent can exist and NULL is returned. Let $s_k^{(i)}$ be the value of $s_k$ after $i = 0, 1, \ldots$ iterations of the loop at line 2. We always have $s_k^{(0)} = 0$ due to the initialization of $s_k$. See Figure 2.

Assume that $s_0$ is updated in line 4 in iteration $i$. The point $p_0[s_0^{(i)}]$ is in the half-plane $LHP(p_1[s_1^{(i-1)}], p_0[s_0^{(i-1)}])$, but not on the line $L(p_1[s_1^{(i-1)}], p_0[s_0^{(i-1)}]).$ Therefore, we have the following observation.

Observation 3. When $s_k$ is updated, the temporary line is rotated counterclockwise around $s_{1-k}$ by an angle less than $\pi$.

Assume in the following that the convex hulls of $P_0$ and $P_1$ are disjoint so that separating common tangents exist. Let $(r_0, r_1)$ be the indices that define the separating common tangent such that $P_k$ is contained in $RHP(p_{1-k}[r_{1-k}], p_k[r_k]),$ i.e., $(r_0, r_1)$ is the result we are going to prove that the algorithm returns.

Since $\mathcal{H}(P_k)$ is convex, the temporary line always divides $\mathcal{H}(P_k)$ into two convex parts. If we follow the temporary line from $p_{1-k}[s_{1-k}]$ in the direction towards $p_k[s_k]$, we enter
\( \mathcal{H}(P_k) \) at some point \( x \) and thereafter leave \( \mathcal{H}(P_k) \) again at some point \( y \). We clearly have \( x = y \) if and only if the temporary line is a tangent to \( \mathcal{H}(P_k) \), since if \( x = y \) and the line was no tangent, \( \mathcal{H}(P_k) \) would only be a line segment. The part of the boundary of \( \mathcal{H}(P_k) \) counterclockwise from \( x \) to \( y \) is in \( \text{RHP}(p_{i-k}[s_1-k], p_k[s_k]) \) whereas the part from \( y \) to \( x \) is on \( \text{LHP}(p_{i-k}[s_1-k], p_k[s_k]) \). We therefore have the following observation.

\textbf{Observation 4.} Let \( d \) be the index of the corner of \( \mathcal{H}(P_k) \) strictly after \( y \) in counterclockwise order. There exists a corner \( p_k[t] \) of \( P_k \) such that \( \mathcal{T}(p_{i-k}[s_1-k], p_k[s_k], p_k[t]) = 1 \) if and only if \( \mathcal{T}(p_{i-k}[s_1-k], p_k[s_k], p_k[d]) = 1 \).

Let \( c_k \) be the index of the first corner of \( \mathcal{H}(P_k) \) when following \( \mathcal{H}(P_k) \) in counterclockwise order from \( y \), \( c_k = 0, \ldots, n_k - 1 \). If \( y \) itself is a corner of \( \mathcal{H}(P_k) \), we have \( p_k[c_k] = y \). By observation 4 we see that \( \mathcal{T}(p_{i-k}[s_1-k], p_k[s_k], p_k[c_k]) \geq 0 \) with equality if and only if \( p_k[c_k] = p_k[s_k] = y \). Let \( c_k(0) \) be \( c_k \) when only line \( 1 \) has been executed. Consider now the value of \( c_k \) after \( i = 1, 2, \ldots \) iterations of the loop at line 2. Let \( c_k(i) \) be \( c_k \) and add \( n_k \) to \( c_k(i) \) until \( c_k(i) \geq k \). This gives a non-decreasing sequence of indices \( c_k(0), c_k(1), \ldots \) of the first corner of \( \mathcal{H}(P_k) \) in \( \text{LHP}(p_{i-k}[s_1-k], p_k[s_k]) \). Actually, we prove in the following that we need to add \( n_k \) to \( c_k(i) \) at most once before \( c_k(i) \geq k \). If \( r_k \neq c_k(0) \) we add \( n_k \) to \( r_k \). Thus we have \( 0 = s_k(0) \leq c_k(0) \leq r_k < 2n_k \).

The following lemma intuitively says that the algorithm does not “jump over” the correct solution and it expresses the main idea in our proof of correctness.

\textbf{Lemma 5.} After each iteration \( i = 0, 1, \ldots \) and for each \( k = 0, 1 \) we have \( 0 \leq s_k(i) \leq c_k(i) \leq r_k < 2n_k \).

\textbf{Proof.} We prove the lemma for \( k = 0 \). From the definition of \( r_0 \), we get that \( 0 = s_0(0) \leq c_0(0) \leq r_0 < 2n_0 \). Since the sequence \( s_0(0), s_0(1), \ldots \) is non-decreasing, the inequality \( 0 \leq s_k(i) \) is true for every \( i \).

Now, assume inductively that \( s_0(i-1) \leq c_0(i-1) \leq r_0 < 2n_0 \) and consider what happens during iteration \( i \). If neither \( s_0 \) nor \( s_1 \) is updated, the statement is trivially true from the induction hypothesis, so assume that an update happens.

By the old temporary line we mean the temporary line defined by \((s_0(i-1), s_1(i-1))\) and the new temporary line is the one defined by \((s_0(i), s_1(i))\). The old temporary line enters \( \mathcal{H}(P_0) \) at some point \( x \) and exits at some point \( y \) when followed from \( p_1[s_1(i-1)] \). Likewise, let \( v \) be the point where the new temporary line exits \( \mathcal{H}(P_0) \) when followed from \( p_1[s_1(i)] \). The point \( x \) exists since the convex hulls are disjoint.

Assume first that the variable \( u \) in the algorithm is 0, i.e., a corner of the polygon \( P_0 \) is traversed. In this case \( s_1(i-1) = s_1(i) \).

We now prove \( s_0(i) \leq c_0(i) \). Assume that \( p_0[s_0(i-1)] \neq p_0[c_0(i-1)] \). The situation is depicted in Figure 3. In this case \( \mathcal{T}(p_1[s_1(i-1)], p_0[s_0(i-1)], p_0[c_0(i-1)]) = 1 \). Hence, the update happens when \( p_0[c_0(i-1)] \) is traversed or earlier, so \( s_0(i) \leq c_0(i-1) \leq c_0(i) \). Assume now that \( p_0[s_0(i-1)] = p_0[c_0(i-1)] \). We cannot have \( c_0(i) = c_0(i-1) \) since \( \mathcal{T}(p_1[s_1(i-1)], p_0[s_0(i-1)], p_0[c_0(i-1)]) = -\mathcal{T}(p_1[s_1(i-1)], p_0[s_0(i-1)], p_0[c_0(i-1)]) = -1 \). Therefore, \( c_0(i) > c_0(i-1) \). Consider the corner \( p_0[c'] \) on \( \mathcal{H}(P_0) \) following \( p_0[c_0(i-1)] \) in counterclockwise order, \( c' > c_0(i-1) \). Due to the minimality of \( c' \), we have \( c' \leq c_0(i) \). By observation 4, \( \mathcal{T}(p_1[s_1(i-1)], p_0[s_0(i-1)], p_0[c']) = 1 \). Therefore, \( s_0 \) must be updated when \( p_0[c'] \) is traversed or earlier, so \( s_0(i) \leq c' \leq c_0(i) \).

For the inequality \( c_0(i) \leq r_0 \), consider the new temporary line in the direction from \( p_1[s_1(i-1)] \) to \( p_0[s_0(i-1)] \). We prove that \( v \) is in the part of \( \mathcal{H}(P_0) \) from \( y \) counterclockwise to \( r_0 \). The point \( p_0[s_0(i)] \) is in the polygon \( Q \) defined by the segment \( xy \) together with the part of
\[ \mathcal{H}(P_0) \] from \( y \) counterclockwise to \( x \). Therefore, the new temporary line enters and exits \( Q \). It cannot exit through the segment \( xy \), since the old and new temporary lines intersect at \( p_1[s_1^{(i-1)}] \), which is in \( \mathcal{H}(P_0) \). Therefore, \( v \) must be on the part of \( \mathcal{H}(P_0) \) from \( y \) to \( x \). If \( r_0 \) is on the part of \( \mathcal{H}(P_0) \) from \( x \) counterclockwise to \( y \), then \( v \) is on the part from \( y \) to \( r_0 \) as we wanted.

Otherwise, assume for contradiction that the points appear in the order \( y,p_0[r_0],v,x \) counterclockwise along \( \mathcal{H}(P_0) \), where \( p_0[r_0] \neq v \neq x \). The endpoints of the segment \( p_1[s_1^{(i-1)}]x \) are on different sides of the tangent defined by \( (r_0,r_1) \), so the segment intersects the tangent at a point \( w \). The part of \( \mathcal{H}(P_0) \) from \( p_0[r_0] \) to \( x \) and the segments \( xw \) and \( wp_0[r_0] \) form a simple polygon \( \mathcal{C} \), see Figure 3 for an example. The new temporary line enters \( \mathcal{C} \) at the point \( v \), so it must leave \( \mathcal{C} \) after \( v \). The line cannot cross \( \mathcal{H}(P_0) \) after \( v \) since \( \mathcal{H}(P_0) \) is convex. It also cannot cross the segment \( xw \) at a point after \( v \) since the old and the new temporary line cross before \( v \), namely at \( p_1[s_1^{(i-1)}] \). The tangent defined by \( (r_0,r_1) \) and the new temporary line intersect before \( v \) since the endpoints of the segment \( p_1[s_1^{(i-1)}]v \) are on different sides of the tangent. Therefore, the line cannot cross the segment \( wp_0[r_0] \) at a point after \( v \). Hence, the line cannot exit \( \mathcal{C} \). That is a contradiction.

Therefore, \( v \) is on the part of \( \mathcal{H}(P_0) \) from \( y \) to \( p_0[r_0] \) and hence the first corner \( p_0[c_0^{(i)}] \) of \( \mathcal{H}(P_0) \) after \( v \) must be before or coincident with \( p_0[r_0] \), so that \( c_0^{(i)} \leq r_0 \).

Assume now that \( u = 1 \) in the beginning of iteration \( i \), i.e., a corner of the other polygon \( P_1 \) is traversed. In that case, we have \( s_0^{(i)} = s_0^{(i-1)} \leq c_0^{(i-1)} \leq c_0^{(i)} \), and we need only prove \( c_0^{(i)} \leq r_0 \). Observation 3 gives that \( v \) is in the part of \( \mathcal{H}(P_0) \) from \( y \) to \( x \), since the new temporary line is obtained by rotating the old temporary line counterclockwise around \( p_0[s_0^{(i-1)}] \) by an angle less than \( \pi \). That \( v \) appears before \( p_0[r_0] \) on \( \mathcal{H}(P_0) \) counterclockwise from \( y \) follows from exactly the same arguments as in the case \( u = 0 \). This completes the proof.

\begin{lemma}

If the temporary line is different from the tangent defined by \( (r_0,r_1) \), then \( T(p_0[s_0],p_1[s_1],p_1[r_1]) = 1 \) or \( T(p_1[s_1],p_0[s_0],p_0[r_0]) = 1 \).

\end{lemma}

\textbf{Proof.} Assume not. There are points of the temporary line on each side of the tangent because it is separating, so the temporary line and the tangent cross each other in a point \( a \).
The point $a$ is on the segment $p_0[r_0]p_1[r_1]$, since otherwise $p_0[r_0]$ and $p_1[r_1]$ would be on the same side of the temporary line, so $T(p_0[r_0], p_1[s_1], p_1[r_1]) = 1$ or $T(p_1[s_1], p_0[s_0], p_0[r_0]) = 1$. Choose a point $d_R$ on the temporary line in $RHP(p_0[r_0], p_1[r_1])$ which is so far away from $a$ that all intersections between the line and the polygons are on the same side of $d_R$ as $a$. Choose $d_L$ in a similar way in $LHP(p_0[r_0], p_1[r_1])$. We have $-1 = T(p_0[r_0], p_1[r_1], d_R) = T(p_0[r_0], a, d_R) = -T(d_R, a, p_0[r_0])$, so the supports must appear in the order $s_0, s_1$ when traveling along the temporary line from $d_R$ towards $a$ for $T(p_1[s_1], p_0[s_0], p_0[r_0]) \leq 0$ to hold.

We also have that $p_0[s_0]$ is on the segment $ad_L$ since $p_0[s_0] \in LHP(p_0[r_0], p_1[r_1])$ and $p_1[s_1]$ is on the segment $ad_R$ since $p_1[s_1] \in RHP(p_0[r_0], p_1[r_1])$. Hence, the order of the supports from $d_R$ towards $a$ is $s_1, s_0$. That is a contradiction. ◀

We are now ready to prove that Algorithm 1 has the desired properties.

**Theorem 7.** If the polygons $P_0$ and $P_1$ have separating common tangents, Algorithm 1 returns a pair of indices $(s_0, s_1)$ defining a separating common tangent such that $P_k$ is contained in $RHP(p_{i-k}[s_{1-k}], p_k[s_k])$ for $k = 0, 1$. If no separating common tangents exist, the algorithm returns NULL. The algorithm runs in linear time and uses constant workspace.

**Proof.** Assume first that separating common tangents do not exist. Then the test in line 10 makes the algorithm return NULL due to some corner $p_n[t]$ on the wrong side of the temporary line.

Assume now that separating common tangents do exist and that the temporary line is not the desired tangent. Without loss of generality, we may assume that $T(p_1[s_1], p_0[s_0], p_0[r_0]) = 1$ by Lemma 6. Lemma 5 gives that $p_0[r_0]$ will be traversed if no other update of $s_0$ or $s_1$ happens. Therefore, an update happens before the loop at line 2 finishes. We conclude that when the loop finishes, the pair $(s_0, s_1)$ defines the separating common tangent as stated.

When an update happens in iteration $i$ of the loop at line 2, the sum $s_0 + s_1$ is increased by a value which is at least $\frac{s_{k-1}}{4}$, where $j \geq 0$ was the previous iteration where an update happened. Inductively, we see that the number of iterations is always at most $2(s_0 + s_1) + t_0 - s_0 + t_1 - s_1 \leq 2(t_0 + t_1) \leq 4(n_0 + n_1)$. ◀

3 Computing outer common tangents

In this section, we assume that two polygons $P_0$ and $P_1$ are given such that their convex hulls are disjoint. We assume that the corners $p_0[0], \ldots, p_0[n_0 - 1]$ of $P_0$ are given in counterclockwise order and the corners $p_1[0], \ldots, p_1[n_1 - 1]$ of $P_1$ are given in clockwise order. We say that the orientation of $P_0$ and $P_1$ is counterclockwise and clockwise, respectively. We prove that Algorithm 2 returns two indices $(s_0, s_1)$ that define an outer common tangent such that $P_0$ and $P_1$ are both contained in $RHP(p_0[s_0], p_1[s_1])$.

As in the case of separating common tangents, we define $s_k^{(i)}$ as the value of $s_k$ after $i = 0, 1, \ldots$ iterations of the loop at line 2 of Algorithm 2. See Figure 4. For this algorithm, we get a slightly different analogue to Observation 3:

**Observation 8.** When $s_k$ is updated, the temporary line is rotated around $s_{1-k}$ in the orientation of $P_{1-k}$ by an angle less than $\pi$.

Let $y$ be the point where the temporary line enters $H(P_k)$ when followed from $p_{1-k}[s_{1-k}]$ and $x$ the point where it exits $H(P_k)$. We have the following analogue of Observation 4.

**Observation 9.** Let $d$ be the index of the corner of $H(P_k)$ strictly after $y$ following the orientation of $P_k$. There exists a corner $p_k[t]$ of $P_k$ such that $T(p_0[s_0], p_1[s_1], p_k[t]) = 1$ if and only if $T(p_0[s_0], p_1[s_1], p_k[d]) = 1$. 


Also, let the indices \( P \) be the lines defined by the indices \( (s_0^{(i)}, s_1^{(i-1)}) \) and \( (s_0^{(i)}, s_1^{(i)}) \), respectively. Let \( y \) and \( x \) be the points where the old temporary line enters and exits \( \mathcal{H}(P_0) \) followed from \( p_1[s_1^{(i-1)}] \), respectively, and let \( v \) be the point where the new temporary line enters \( \mathcal{H}(P_0) \). The points \( y \) and \( v \) exist since the convex hulls of \( P_0 \) and \( P_1 \) are disjoint.

---

**Algorithm 2:** OuterCommonTangent\( (P_0, P_1) \)

1. \( s_0 \leftarrow 0; \ t_0 \leftarrow 1; \ s_1 \leftarrow 0; \ t_1 \leftarrow 1; \ u \leftarrow 0 \)
2. \textbf{while} \( t_0 < 2n_0 \) or \( t_1 < 2n_1 \) \textbf{do}
3. \hspace{1em} \textbf{if} \( \mathcal{T}(p_0[s_0], p_1[s_1], u[t_u]) = 1 \)
4. \hspace{2em} \( s_u \leftarrow t_u \)
5. \hspace{2em} \( t_{1-u} \leftarrow s_{1-u} + 1 \)
6. \hspace{2em} \( t_u \leftarrow t_u + 1 \)
7. \hspace{2em} \( u \leftarrow 1 - u \)
8. \textbf{return} \( (s_0, s_1) \)

---

**Lemma 10.** After each iteration \( i = 0, 1, \ldots \) and for each \( k = 0, 1 \) we have

\[
0 \leq s_k^{(i)} \leq c_k^{(i)} \leq r_k < 2n_k.
\]

**Proof.** Assume \( k = 0 \) and the induction hypothesis \( s_0^{(i-1)} \leq c_0^{(i-1)} \leq r_0 \). The inequality \( s_0^{(i)} \leq c_0^{(i)} \) can be proven exactly as in the proof of lemma 5. Therefore, consider the inequality \( c_0^{(i)} \leq r_0 \) and assume that an update happens in iteration \( i \).

Let the old temporary line and the new temporary line be the lines defined by the indices \( (s_0^{(i-1)}, s_1^{(i-1)}) \) and \( (s_0^{(i)}, s_1^{(i)}) \), respectively. Let \( y \) and \( x \) be the points where the old temporary line enters and exits \( \mathcal{H}(P_0) \) followed from \( p_1[s_1^{(i-1)}] \), respectively, and let \( v \) be the point where the new temporary line enters \( \mathcal{H}(P_0) \). The points \( y \) and \( v \) exist since the convex hulls of \( P_0 \) and \( P_1 \) are disjoint.

---

**Figure 4** Algorithm 2 running on two polygons \( P_0 \) and \( P_1 \). The corners \( p_k[s_k^{(i)}] \) are marked and labeled as \( s_k^{(i)} \) for the initial values \( s_k^{(0)} \) and after each iteration \( i \) where an update of \( s_k \) happens. The segments \( p_0[s_0^{(i)}], p_1[s_1^{(i)}] \) on the temporary line are dashed.
Assume first that the variable \( u \) in the algorithm equals 0 when the update happens. We prove that \( v \) is in the part of \( \mathcal{H}(P_0) \) from \( y \) to \( p_0[r_0] \) following the orientation of \( P_0 \), which is counterclockwise. The point \( p_0[s_0^{(i)}] \) is in the simple polygon \( Q \) bounded the part of \( \mathcal{H}(P_0) \) from \( y \) counterclockwise to \( x \) and the segment \( xy \). Therefore, the new temporary line must enter \( Q \) to get to \( p_0[s_0^{(i)}] \). It cannot enter through \( xy \), since the old and new temporary line cross at \( p_1[s_1^{(i-1)}] \) which is not in \( \mathcal{H}(P_0) \) by assumption. Therefore, it must enter through the part of \( \mathcal{H}(P_0) \) from \( y \) to \( x \), so \( v \) is in this part. If \( r_0 \) is not in the part of \( \mathcal{H}(P_0) \) from \( y \) to \( x \), it is clearly true that \( v \) is in the part from \( y \) to \( p_0[r_0] \). Otherwise, assume for contradiction that the points appear on \( \mathcal{H}(P_0) \) in the order \( y, p_0[r_0], v, x \) and \( p_0[r_0] \neq v \neq x \). Let \( \ell_0 \) be the half-line starting at \( p_0[r_0] \) following the tangent away from \( p_1[r_1] \), and let \( \ell_1 \) be the half-line starting at \( x \) following the old temporary line away from \( p_1[s_1^{(i-1)}] \). The part of \( \mathcal{H}(P_0) \) from \( p_0[r_0] \) to \( x \) and the half-lines \( \ell_0 \) and \( \ell_1 \) define a possibly unbounded area \( A \) outside \( \mathcal{H}(P_0) \), see Figure 5. We follow the new temporary line from \( p_1[s_1^{(i-1)}] \) towards \( v \). The point \( p_1[s_1^{(i-1)}] \) is not in \( A \) and the new temporary line exits \( A \) at \( v \) since it enters \( \mathcal{H}(P_0) \) at \( v \), so it must enter \( A \) somewhere at a point on the segment \( p_1[s_1^{(i-1)}]v \). It cannot enter through \( \mathcal{H}(P_0) \) since \( \mathcal{H}(P_0) \) is convex. It cannot enter through \( \ell_0 \) since \( v \) and \( p_1[s_1^{(i-1)}] \) are on the same side of the outer common tangent. It cannot enter through \( \ell_1 \) since the old and new temporary line intersect in \( p_1[s_1^{(i-1)}] \), which is not in \( A \). That is a contradiction, so \( v \) is on the part of \( \mathcal{H}(P_0) \) from \( y \) to \( p_0[r_0] \). Hence, the first corner after \( y \) is coincident with or before \( p_0[r_1] \), i.e., \( \ell_0^{(i)} \leq r_0 \).

Assume now that \( u = 1 \) in the beginning of iteration \( i \) so that a corner of the polygon \( P_1 \) is traversed. Observation 8 gives that \( v \) is on the part of \( \mathcal{H}(P_0) \) from \( y \) counterclockwise to \( x \). It follows that \( v \) appears before \( p_0[r_0] \) on \( \mathcal{H}(P_0) \) counterclockwise from \( y \) from exactly the same arguments as in the case \( u = 0 \).

We have the following equivalent of Lemma 6 which, however, has a different proof.

**Lemma 11.** If the temporary line is different from the tangent defined by \((r_0, r_1)\), then \( T(p_0[s_0], p_1[s_1], p_0[r_0]) = 1 \) or \( T(p_0[s_0], p_1[s_1], p_1[r_1]) = 1 \).

**Proof.** Assume not. The points \( p_0[s_0] \) and \( p_1[s_1] \) are both in \( RHP(p_0[r_0], p_1[r_1]) \). Therefore, the temporary line cannot be parallel with the tangent, since in that case we would have \( T(p_0[s_0], p_1[s_1], p_0[r_0]) = 1 \). Let \( a \) be the intersection point between the tangent and the temporary line. The point \( a \) cannot be in the interior of the segment \( p_0[r_0]p_1[r_1] \),
We have described an algorithm for computing the separating common tangents of two simple polygons in linear time using constant workspace. We have also described an algorithm for computing outer common tangents using linear time and constant workspace when the convex hulls of the polygons are disjoint. Figure 6 shows an example where Algorithm 2 does not work when applied to two disjoint polygons with overlapping convex hulls. In fact, if there was no bound on the values \( t_0 \) and \( t_1 \) in the loop at line 2, the algorithm would update \( s_0 \) and \( s_1 \) infinitely often and never find the correct tangent. An obvious improvement is to find an equally fast and space efficient algorithm which does not require the convex hulls to be disjoint. An algorithm for computing an outer common tangent of two polygons, when such one exists, also decides if one convex hull is completely contained in the other. Together with the algorithm for separating common tangents presented in Section 2, we would have an optimal algorithm for deciding the complete relationship between the convex hulls: if one is contained in the other, and if not, whether they are disjoint or not. However, keeping in mind that it is harder to compute an outer common tangent of intersecting convex polygons.
than of disjoint ones [7], it would not be surprising if it was also harder to compute an outer
common tangent of general simple polygons than simple polygons with disjoint convex hulls
when only constant workspace is available.

References

1. M. Abrahamsen. An optimal algorithm computing edge-to-edge visibility in a simple poly-
gon. In Proceedings of the 25th Canadian Conference on Computational Geometry, CCCG,
2. T. Asano, K. Buchin, M. Buchin, M. Korman, W. Mulzer, G. Rote, and A. Schulz. Memory-
constrained algorithms for simple polygons. Computational Geometry: Theory and Applic-
4. L. Barba, M. Korman, S. Langerman, and R.I. Silveira. Computing the visibility polygon
and Computation, ISAAC, volume 7014 of Lecture Notes in Computer Science, pages 70–
structure for convex hulls. International Journal of Computational Geometry & Applica-
8. J. Hershberger and S. Suri. Applications of a semi-dynamic convex hull algorithm. BIT
Proceedings of the 4th International Workshop on Algorithms and Data Structures, WADS,
10. A.A. Melkman. On-line construction of the convex hull of a simple polyline. Information
12. F.P. Preparata and S.J. Hong. Convex hulls of finite sets of points in two and three
13. G.T. Toussaint. Solving geometric problems with the rotating calipers. In Proceedings of
the IEEE Mediterranean Electrotechnical Conference, MELECON, pages A10.02/1–4, 1983.