# Shortest Paths in Convex and Simple Weighted Polygons

Christian Chenier* Jorge Urrutia*

## Abstract

Consider a polygon $P$ and two points $p, q \in P$. Suppose that to move from $p$ to $q$, we can travel along the edges of $P$ or through the interior of $P$. Assume that the speed at which we can travel along the edges of $P$ is one unit per second, and the travel speed through the interior of $P$ is $1/s$ units per second ($s > 1$). The problem consists of finding the shortest path between $p$ and $q$. We solve this problem in $O(n)$ time for convex polygons. For simple polygons, we show two algorithms. The first takes $O(n \log n)$ query time with $O(nE \log n)$ preprocessing (where $E$ is the number of edges in the visibility graph of $P$) and $O(n^2)$ space, and the second algorithm runs in $O(E \log n)$ query using $O(E)$ preprocessing and space.

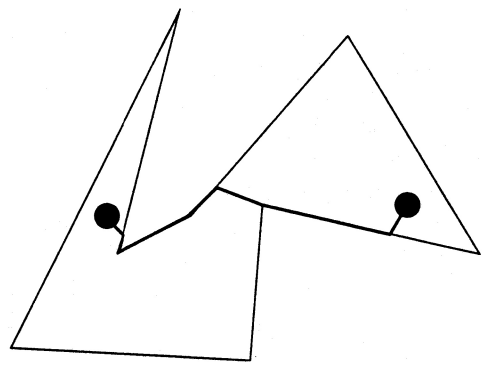

Figure 1: A simple example

Keywords: shortest path, polygon, weighted region

## 1 Introduction

Suppose that a traveler living in a polygonal world represented by a simple polygon $P$ wants to move from a location $p$ to another location $q$ and he may do so by driving his jeep on highways built along the edges of $P$, or taking shortcuts through fields. Our problem is to find the fastest route from $p$ to $q$.

The speed of travel on the boundary is assumed one unit while the speed in the interior is $1/s$ where $s > 1$. In this paper we present algorithms to solve this problem for the cases when $P$ is a simple polygon or a convex polygon. For the case where $P$ is a simple polygon, we give two algorithms, The first one finds the shortest path from $p$ to $q$ in $O(n \log n)$ time, assuming $O(nE \log n)$ preprocessing where $E$ is the number of edges of the visibility graph of $P$, and $O(n^2)$ space. The second one has a query time of $O(E \log n)$ with $O(E)$ preprocessing and space. For convex polygons, we solve the problem in $O(n)$ time. Our algorithms return a shortest path (there can be more than one path with the same low cost). For an example, see Figure 1.

A simpler version of this problem has been solved for years: when the costs of traveling inside and on the boundary are equal ($s = 1$), ie. the problem of finding the shortest path inside the polygon. In that case there is a unique shortest path which can be found in linear time [Guib87]. Notice also that if $s < 1$ then the resulting shortest path is the same as when $s = 1$ except that we never actually travel on the edges, but at distance $\epsilon$ from them ($\epsilon$ is an arbitrarily small constant).

A more complex version of this problem has been looked at by Mitchell and Papadimitriou [Mitc87, Mitc91] where traveling is done in a polyhedral terrain in which each of its faces has an associated travel cost. However, the time complexity associated with this problem is extremely high ($O(n^8 L)$ where $L$ is the precision). This problem allows multiple shortest paths with equal cost.

The problem studied here has applications in Geographic Information Systems (GIS). Here the polygon's boundary represents part of a road network, and the inside is a field, or other terrain which can be traveled on. This terrain is considered to have a uniform travel cost. This kind of algorithm can be used for any kind of travel (foot, all terrain vehicle, horse, dog sled, ...) where a road is defined as a narrow and long surface on which the traveling costs less than on the surrounding terrain. This problem is especially important in emergency situations but it can also be used for a range of traveling problems.

*Computer Science Dept., University of Ottawa, Ottawa, Ontario, K1N 6N5

This paper is structured as follows: in Section 2, some basic properties of shortest paths are obtained. In Section 3 we study the case when $P$ is a simple polygon. Then, in Section 4 we study the case when $P$ is convex. Proofs for some lemmas can be found in the full paper.

## 2 Preliminaries

In this section we see definitions and lemmas which will be useful through the paper. A polygon will be assumed to have its vertices labeled $v_0, \ldots, v_{n-1}$ in such a way that $v_i$ is adjacent to $v_{i+1}$, $i = 0, \ldots, n-1$, addition taken mod$n$.

An ear of a polygon $P$ is a triangle with vertices $v_{i-1}$, $v_i$, $v_{i+1} \in P$ such that $v_{i-1}$ is visible from $v_{i+1}$. The angle of an ear is the angle between the two edges of the polygon which form the ear. Convex polygons with $n$ vertices have $n$ ears, and in general, any polygon has at least two ears.

If we assume that traveling one unit of distance within the interior of $P$ takes $s$ seconds while traveling the same distance on a edge of $P$ takes one second, it is natural to assign the following weight to a line segment contained in $P$:

Given two points $x$, $y \in P$ joined by a line segment $l(x, y)$ totally contained in $P$, we associate to $l(x, y)$ a weight $w(l(x, y)) = |l(x, y)| * s$ if the interior of $l(x, y)$ is totally contained in the interior of $P$; if both $x$ and $y$ are contained in an edge of $P$ then $w(l(x, y)) = |l(x, y)|$ where $|l(x, y)|$ is the Euclidean distance between $x$ and $y$. If $l(x, y)$ is totally contained in the interior of $P$, it will be called a shortcut; this follows the natural intuition that we will travel from $u$ to $v$ through the interior of $P$ only if it guarantees us a shorter traveling time than traveling along the edges of $P$.

Consider two points $p$ and $q$ of $P$, and a rectilinear path $H(p, q)$ from $p$ to $q$ contained in $P$, that is a chain of line segments starting at $p$ and ending at $q$. Clearly $H(p, q)$ can be decomposed into a set of $k$ line segments $l_1, \ldots, l_k$ such that the interior of each $l$ is totally contained in the interior of $P$ or it is totally contained in an edge of $P$. If the interior of $l_j$ is contained in the interior of $P$, $l_j$ will be called a shortcut of $H(p, q)$.

We now associate to $H(p, q)$ a weight:

$$W(H(p, q)) = w(l_1) + \ldots + w(l_k)$$

. We say that $H(p, q)$ is an optimal $p$-$q$ path if for any other rectilinear chain $H'(p, q)$ from $p$ to $q$, we have $W(H(p, q)) \leq W(H'(p, q))$.

We now give some lemmas that describe basic properties of optimal $p$-$q$ paths. The following result was shown in [Mitc91]:

**Lemma 1** *If a shortcut of an optimal p-q path has an endpoint on an edge e of P, then the shortcut and e meet at an angle $\alpha = \sin^{-1}(1/s)$ to the perpendicular of the edge, where $1/s$ is the cost of travel inside the polygon. $(0 < \alpha < \pi/2)$.*

Note that this follows Snell's Law of Refraction.

We define edge-to-edge shortcuts to be shortcuts with both endpoints contained in the interior of edges of $P$. The next lemma is fundamental to our paper:

**Lemma 2** *Given any two points $p$, $q \in P$, there always exists an optimal p-q path with no edge-to-edge shortcuts.*

Proof: Let $l$ be an edge-to-edge shortcut of a shortest $p$-$q$ path . We now show that there is an equally expensive $p$-$q$ path which does not include $l$. Let $e_1$ and $e_2$ be the edges of $P$ containing the endpoints of $l$. By Lemma 1, $l$ forms angles of size $\pi/2-\alpha$ with $e_1$ and $e_2$. Two cases arise:

1. The angle formed by the lines containing $e_1$ and $e_2$ is equal to $2\alpha$.
2. $e_1$ and $e_2$ are parallel.

For case 1, assume w.l.o.g. that $l$ is horizontal and that the lines containing $e_1$ and $e_2$ intersect below $l$. Consider the diagonal $l'$ of P parallel to $l$ and below it, with both endpoints in $e_1$ and $e_2$ such that $l'$ contains a vertex of $P$. Then it is easy to see that traveling from one endpoint of $l$ to the other along $l$ is equally expensive as traveling between them along $e_1$, $l'$ and $e_2$. This generates another shortest path from $p$ to $q$ avoiding $l$.

A similar analysis can be done for the case when $e_1$ and $e_2$ are parallel.

To simplify the algorithm and the proof, we only consider the paths which have at least one endpoint at a vertex of the polygon. This is permitted since the cost of traveling between edges which have a difference in angle of $2\alpha$, or between parallel edges, is the same wherever we take the shortcut. To get an equally expensive path, we therefore move one endpoint of the shortcut to a vertex. The path length will remain unchanged. We therefore do not need to consider edge-to-edge shortcuts.

**Lemma 3** *If the angle of an ear is less than $2\alpha$ and neither p nor q are contained in the ear, no optimal p-q path will enter the interior of the ear, so we may as well delete it.*

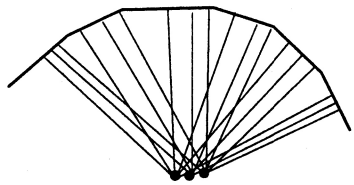

Figure 2: $O(n^2)$ shortcuts to edges

Proof: As shown in the proof of Lemma 2, it is equally expensive to go around on an ear of angle $2\alpha$ than to cut across it. It is easy to see that if the angle is smaller than $2\alpha$, it is preferable to cut across the ear. Obviously, when the angle is larger, it is less expensive to travel along the edge.

# 3 The Simple Polygon Case

In this section we sketch two algorithms to find shortest paths between two points $p$ and $q$ in a simple polygon $P$ with $n$ vertices. Our first algorithm uses $O(E)$ preprocessing and space, and $O(E \log n)$ query time, where $E$ is the number of edges in the $\alpha$-visibility graph of $P$. Our second algorithm uses $O(nE \log n)$ preprocessing, $O(n^2)$ space, and answers queries in $O(n \log n)$ time.

The visibility graph is $\Omega(n^2)$ in size. In Figure 2 we see that the $n/2$ bottom vertices each have $n/2$ visible edges on top, yielding $O(n^2)$ edges in all. Notice that we have only shown the path perpendicular to the edge. A path very similar to this is possible in the case where the cost of traveling inside the polygon is extremely high.

Given an angle $\alpha$, the $\alpha$-visibility graph of a polygon $P$ is defined as follows: The vertices of the $\alpha$-visibility graph are the vertices of $P$ together with points $p$ in the interior of edges of $P$ such that there is a vertex $v$ of $P$ visible from $p$ and the line segment joining $p$ to $v$ intersects the edge of $P$ containing $p$ at angle $\pi/2-\alpha$. Two vertices $u$ and $v$ of the $\alpha$-visibility graph are connected if they are vertices of $P$ that are visible from each other or if $u$ is an interior point to an edge of $P$ and $v$ is a vertex of $P$ visible from $u$ such that the angle between the line connecting $u$ to $v$ and the edge of $P$ containing $u$ is $\pi/2-\alpha$. It easy to see that the $\alpha$-visibility graph of $P$ can be obtained in $O(E)$ time, using [Her87], where $E$ is the number of edges of the $\alpha$-visibility graph.

From Lemma 2, we obtain the following result :

**Lemma 4** *Given any two points p, q on the boundary of P, there is a shortest p-q path contained in the edges of the $\alpha$-visibility graph of P.*

Consider two points $p$ and $q$ and the shortest path $H(p,q)$ connecting them. Three types of shortest paths must be considered:

1. $H(p,q)$ is a line segment connecting $p$ to $q$.
2. $H(p,q)$ goes through an edge of $P$ but does not visit any vertex of $P$.
3. $H(p,q)$ visits at least one vertex of $P$.

It now follows:

**Lemma 5** *In case 3 above, there exists a shortest path $H(p,q)$ from $p$ to $q$ such that all of the edges of this path except possibly the first and the last are contained in the $\alpha$-visibility graph of $P$. Moreover if $p$ (resp. $q$) is an interior point of $P$ the edge of $H(p,q)$ containing $p$ (resp. $q$) joins $p$ to a vertex of $P$ or intersects an edge of $P$ at an angle $\pi/2-\alpha$.*

From this we have :

**Theorem 1** *Finding a shortest path between p and q can be done in $O(E \log n)$ using $O(E)$ preprocessing.*

Proof: As described before, it is not hard to see that cases 1 and 2 can be solved in linear time. Case 3 however requires more work. In the preprocessing part, we calculate the $\alpha$-visibility graph of $P$. This can be done in $O(E)$ time. By Lemma 4 and Lemma 5 there is a shortest path from $p$ to $q$ contained in the graph obtained from the $\alpha$-visibility graph by adding to it the shortcuts from $p$ and $q$ to the boundary of $P$. This can be done in linear time. Our result now follows by applying Dijkstra's algorithm to the resulting graph.

We now show how to answer shortest path queries in $O(n \log n)$ time, at the expense of increasing the preprocessing time to $O(nE \log n)$ and using $O(n^2)$ space. As in the previous theorem, cases 1 and 2 can be easily dealt with, so we concentrate on case 3.

To solve case 3 we use the following strategy: for every vertex $v$ of $P$ we calculate the shortest path between $p$ and $v$ and the shortest path between $v$ and $q$. We report the smallest such path.

We now show how to calculate the length of the shortest path between any point $p \in P$ and a vertex $v$ of $P$ in $O(\log n)$ time, using $O(E \log n)$ preprocessing.

The next lemma follows easily from Lemma 1 and Lemma 2.

**Lemma 6** *The shortest path spanning tree from v to all vertices of P can be done in $O(E \log n)$ where $E$ is the size of the alpha visibility graph.*

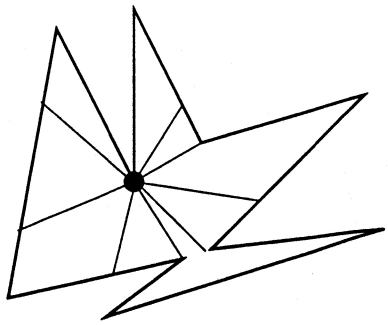

Figure 3: All possible shortcuts

This shortest path will be called $T(v)$

It is easy to see that $T(v)$ induces a partition $\Pi_i'(P,v)$ into $O(n)$ regions $R_1,\ldots,R_s$. For each $R_i$ let $F(R_i)$ be the point in $R_i$ closest to $v$. We store the distance between $F(R_i)$ and $v$. We then further subdivide each $R_i$ into $O(|R_i|)$ regions, $R_{i,1}$, ..., $R_{i,t}$, $1 \leq i \leq s$ in such a way that for every $q \in R_{i,j}$, $1 \leq j \leq t$, we can calculate the distance from $q$ to $F(R_i)$ in constant time. This produces a partitioning $\Pi_i(P,v)$ into $O(n)$ regions.

To calculate the distance between any point $p \in P$ and $v$ we proceed as follows:

1. locate the region $R_{i,j}$ of $\Pi_i(P,v)$ containing $p$
2. find the distance of $p$ to $F(R_i)$
3. find the distance from $p$ to $v$

Step 1 can be done in $O(\log n)$ time and steps 2 and 3 in constant time.

**Lemma 7** *Given the $\alpha$-visibility graph of $P$, we can answer $p$-$v$ distance queries in $O(\log n)$ time, using $O(E \log n)$ preprocessing .*

We are now ready to give our algorithm to find the shortest path between two points $p, q \in P$.

Preprocessing:

1. Calculate the $\alpha$-visibility graph.
2. For every vertex $v$ of $P$ calculate $\Pi_i'(P,v)$

Query: Given $p$ and $q$, for every $v \in P$ find the distance between $p$ and $v$ and the distance between $v$ and $q$, and report the shortest. Clearly for every vertex this can be carried out in $O(\log n)$ time.

We now have the following theorem:

**Theorem 2** *Using $O(n^2)$ space and $O(nE \log n)$ preprocessing, we can answer queries of the type $d(p,q)$ in $O(n \log n)$ time.*

# 4 The Convex Case

In the case where $P$ is convex, a linear time algorithm is possible. We give the algorithm here along with a proof of optimality.

**Theorem 3** *For $P$ convex, the shortest path from $p$ to $q$ can be found in optimal $O(n)$ time.*

In linear amortized time, we can recursively trim ears which have angle $\beta < 2\alpha$, unless $p$ or $q$ lies inside the triangle defined by the ear. We get an irreducible polygon with respect to $p$ and $q$. Note that this irreducible polygon contains a combination of shortcut edges and sections of the original polygon's edges. It is obvious that trimmed ears are not needed since (by Lemma 3) it is more expensive to travel within them than to cut across them.

We then consider the shortcuts from $p$ and from $q$ to all edges. No shortcut-to-shortcut edges of the irreducible polygon is possible, for obvious reasons. Shortcuts can meet the polygon at vertices or on a polygon's edge, at angle $\pi/2 - \alpha$. Each edge will have at most 4 intersections with these shortcuts (two associated to $p$ and two to $q$), yielding $O(n)$ shortcuts in total.

It is along two of the shortcuts found above that the shortest path will start and end. We also know that the shortest path will only leave the irreducible polygon at the beginning and at the end of the shortest path. We must travel along the irreducible polygon's boundary in a single direction. Assume w.l.o.g that we will need to travel in a counterclockwise direction. We consider the subset of shortcuts from $p$ that meet the irreducible polygon at an angle appropriate to counterclockwise travel. We label the cost of travel along the shortcut to the intersection with the irreducible polygon. We then start at an arbitrary intersection $I_1$, and travel counterclockwise, carrying the cost of travel from $I_1$ to the next intersection, $I_2$, and so on. We keep the minimum cost for each $I_j$. We may need to travel twice around the polygon in order to get the shortest path to $I_1$. We repeat this procedure for clockwise travel from $q$.

Let $w$ be the union of all vertices of the irreducible polygon and the intersection points of the shortcuts to that polygon. If the shortest path travels through $w$, a vertex of the irreducible polygon, then the shortest path is of length $d(p,w)+d(w,q)$, which is known, assuming a particular direction of travel. What remains to do is to find the minimum over all $w$, in both direction of travel and to finally compare to the straight path from $p$ to $q$.

This algorithm is optimal since even if the length of the shortest path is small, we may need to compare

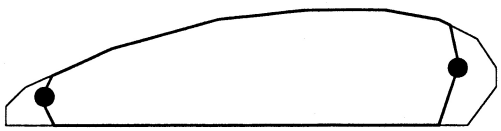

Figure 4: Verifying a path of length $O(n)$ is necessary

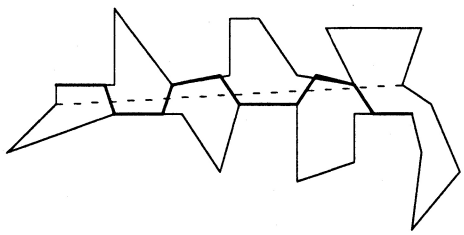

Figure 5: $O(n)$ crossings on a dividing line

it to a linear size path before determining that it is in fact the shortest possible path. This is obvious from looking at Figure 4.

If $p$ (or $q$) is known in advance, a construction similar to a Voronoi diagram is possible. This requires preprocessing but brings the query time down to $O(\log n)$.

When both $p$ and $q$ are unknown in advance, we conjecture that it is possible to build a data structure which would be some combination of two Voronoi diagrams, yielding $O(n^2)$ regions, which would results in $O(\log n)$ query time, after preprocessing.

# 5 Conclusion

A common approach when calculating shortest paths is using a triangulation to determine possible shortcuts. In the problem studied here, there is no obvious way an arbitrary triangulation can be used. For example, a given triangulation edge could be crossed $O(n)$ times (see Figure 5). It is therefore hard to divide the polygon into simpler sub-cases.

Our algorithm for the simple polygon can be applied to non-simple polygons with little modification. Since the path must lie at all time within the polygon's boundary, we can divide the polygon at the points where edges cross each other. We get a subproblem which consists of finding a path from $p$ to the first such point and so on until we enter the last region (where $q$ lies). This approach can be useful for a parallel implementation.

All of the algorithms that were presented in this paper work, without modification, for polygons with holes.

# References


[Guib87] Guibas, L. J., J. Hersberger, D. Leven, M. Sharir, and R. E. Tarjan, Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons, *Algorithmica*, Vol. 2, 1987, 209-233.

[Her87] Hersberger, J., Finding the visibility graph of a simple polygon in time proportional to its size, *Annual ACM Symposium on Computational Geometry*, 1987.

[Mitc87] Mitchell, Joseph S. B., and Christos H. Papadimitriou, The Weighted Region Problem (Extended Abstract). *Proceedings of the 3rd ACM Symposium on Computational Geometry*, Waterloo, 1987, 30-38

[Mitc91] Mitchell, Joseph S. B., and Christos H. Papadimitriou, The Weighted Region Problem: Finding Shortest Paths Through a Weighted Planar Subdivision. *Journal of the ACM*, Vol. 38 No. 1, January 1991, 18-73.