Shortest Paths in Weighted Polygons

by

Christian Chénier

A thesis
presented to the University of Ottawa
in fulfilment of the
thesis requirement for the degree of
Master of Computer Science

Department of Computer Science,
University of Ottawa,
Ottawa, Ontario, Canada

ISBN 0-612-15709-1

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

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 algorithm runs in $O(E \log n)$ time using $O(E)$ space (where E is the size of the visibility graph of P). The second algorithm has two variations which both require $O(nE \log n)$ preprocessing (where E is the number of edges in the visibility graph of P). The first variation takes $O(n \log n)$ query time and $O(n^2 \log n)$ space. The second variation has a query time of $O(n \log^2 n)$ but uses $O(n^2)$ space. For the orthogonal case, we give a $O(E)$ time and space algorithm.

Keywords: shortest path, polygon, weighted region, orthogonal, convex

Acknowledgements

I would like to thank my thesis supervisor, Jorge Urrutia, for his advice, patience and confidence in my work, as well as for the countless hours he spent helping me.

I thank Margaret Urrutia for reading this thesis.

I would like to thank my wife, Julie Cantin, for her love, her support, and her encouragement.

Financial assistance from the Department of Computer Science at the University of Ottawa is gratefully acknowledged.

Contents

List of Figures

Chapter 1

Introduction

1.1 Shortest Paths

Shortest paths were first studied in Graph Theory for solving the problem of finding the least expensive path between pairs of vertices of a graph G. In this problem, a cost is associated to each edge of G and we seek to find the path connecting two selected vertices of G such that the sum of the edges along the path is minimized. Dijkstra [Dijk59] obtained an $O(E \log n)$ algorithm to solve this problem.

More specific to computational geometry, there was work on finding the shortest path between two vertices, p and q in a polygon P. Because of the results already known from graph theory, the first approach was to calculate the visibility graph of P, which requires $O(n^2)$ time, and then to apply Dijkstra's algorithm to it. That method was not optimal.

An optimal linear time algorithm using triangulations of polygons was

developed in [Lee84]. This algorithm first triangulates a polygon, and then uses this triangulation to find the shortest path between two nodes in linear time. Theoretically, the triangulation step can be done in linear time [Chaz87]. However, it is well known that the existing linear algorithm for triangulation is not practical to implement. (Easy-to-implement algorithms run in $O(n \log n)$ time.) Nevertheless, this approach of finding the shortest path is the best known. The general idea used here is to first find the set of triangulated regions within which the shortest path is to pass. Once that is known, we can draw a path through the center of each of these regions. This path can then be refined by simply *stretching* it. Algorithmically, this is done by starting at p and finding the shortest path to the two vertices of the triangulated region which connects to the rest of the path. This is done for every triangulated region until we reach q.

Our discussion of shortest paths continues in Chapter 2.

1.2 Main Problem

In this thesis, we study the following variation to the shortest path problem. Suppose that a traveller 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 roads 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 to be one unit while the speed in the interior is $1/s$ where $s > 1$. Note that the Euclidian shortest

path problem presented in the previous section is a sub-case of our problem, when $s = 1$.

The problem studied in this thesis should be called the weighted polygon shortest path problem. To avoid cumbersome notation, we will simply refer to it as the *shortest path* problem. In order to differentiate our problem from the well known shortest path problem, we will call the later the *Euclidian shortest path* problem. Because the shortest path is the path on which travel requires less time, we also refer to it as the least expensive path.

Notice 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 ϵ from them (where ϵ is an arbitrarily small constant). Therefore, the case where $s = 1$ is not interesting to study.

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 travelled 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 travelling costs less than on the surrounding terrain. The results of this problem can be especially important in emergency situations but they could also be used for a range of travelling problems.

Our problem is different from the simpler problem of finding the Euclidian shortest path in a polygon, as discussed above: Both algorithms mentioned for it are not applicable to our problem. First, as we will see later, the shortest

path in this problem does not necessarily follow any visibility edge. This prevents us from using the first algorithm which used Dijkstra's algorithm since this algorithm relies on the visibility graph. Secondly, a triangulation of P may not provide any useful information as to which region the path follows. We will show later that the shortest path can cross a triangulated region a linear number of times. Therefore, the second algorithm is also not usable in this problem.

In this thesis we present algorithms to solve this problem for the cases when P is a simple polygon, an orthogonal polygon, or a convex polygon. For the case where P is a simple polygon, we give two algorithms. The first one runs in $O(E \log n)$ time and requires $O(E)$ space, where E is the number of edges of the visibility graph of P. For the second one, preprocessing is done in $O(nE \log n)$ time, and we show two types of queries: the first finds the shortest path from p to q in $O(n \log n)$ time, using $O(n^2 \log n)$ space, and the second has a query time of $O(n \log^2 n)$ but utilizes only $O(n^2)$ space. For P convex we show an optimal algorithm which runs in $O(n)$ time. For the orthogonal case, we were able to use structural properties of orthogonal polygons to develop an $O(E)$ time and space algorithm. Our algorithms return a shortest path (there could be more than one path with the same low cost). For an example of a typical shortest path, see Figure 1.1. For an example where many paths from p to q have the same weight, see Figure 1.2.

A more complex version of this problem has been looked at by Mitchell and Papadimitriou [Mitc87b, Mitc91] where travelling is done in a polyhedral terrain in which each of its faces has an associated travel cost. However, the

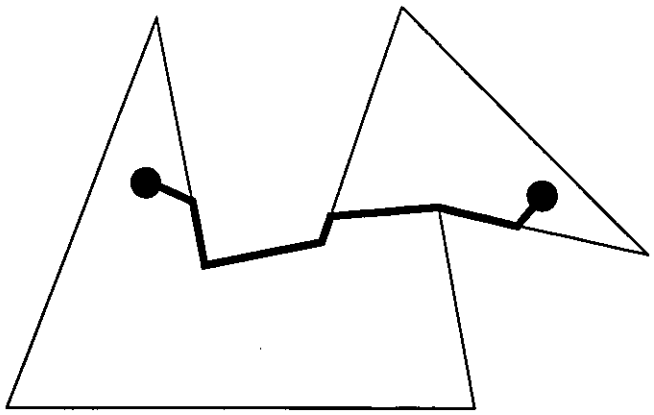

Figure 1.1: A simple example

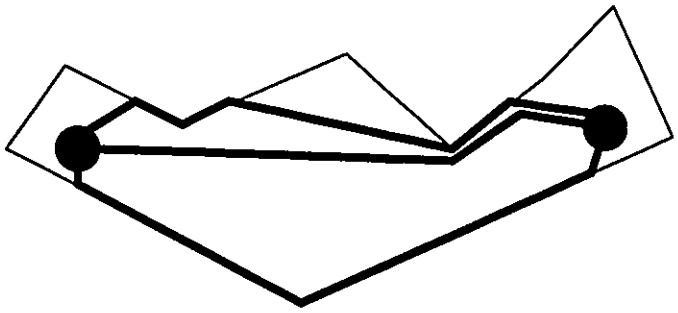

Figure 1.2: Many paths with the same weight

time complexity associated with this problem is extremely high ($O(n^8L)$ where L is the precision). This problem allows multiple shortest paths with equal cost. Details about this, as well as ideas borrowed from these authors can be found in Chapter 3.

Another chapter (Chapter 4) will be spent explaining basic lemmas, definitions, and observations that will be used for the main part of the thesis.

One concept we will use throughout the thesis is the idea of visibility. This includes the study of visibility graphs. These concepts are explained in Chapter 5

Then, in Chapter 6 we will go over basics of spanning trees, and shortest path spanning trees. With this tool, we will solve the shortest path problem from a single source. These concepts will be especially important when discussing the simple polygon case (Chapter 7).

Finally, we will get to the core of the problem, with Chapter 7 explaining the two algorithms for the simple polygon case. Then in Chapter 8 we will solve the problem for orthogonal polygons. Finally, we will study the case when P is convex in Chapter 9.

Parts of this thesis appear in [Chen95].

Chapter 2

Other Shortest Path Problems

Various types of shortest paths have been studied for travel within a polygon. The simplest of them, the Euclidian shortest path was discussed in Chapter 1. An example of an Euclidian shortest path can be seen in Figure 2.1.

Another popular and useful shortest path problem is the link distance problem (also called *bend number*). This consists of finding a path with the least number of turns. This is useful if the cost of turning is high. Applications exist in robotics where, in order for a robot to turn, it must first come to a complete stop, then turn and restart. If we assume that the robot's speed of travelling is fast, then the number of times it changes direction can have a great impact on the overall travelling time. Another application is communications where a repeater is needed each time the signal needs to change direction. It is obvious here that minimizing the number of repeaters is of great importance. Thus minimizing the number of turns may in some applications be the most important factor.

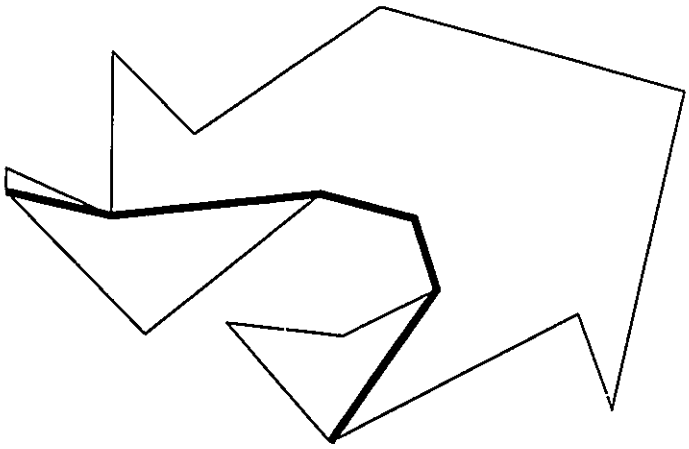

Figure 2.1: Euclidian shortest path

In Figure 2.2, one can see an example where the Euclidian shortest path would contain $O(n)$ turns, but the link distance is only one.

There exist linear time algorithms to solve the link distance problem in a simple polygon [Suri86]. The algorithm shown in that paper uses linear time triangulation. The dual graph of the triangulation is computed so that the algorithm (and the path) follows it. Visibility is also computed for vertices of P to provide *chords* (edges of the visibility graph which are not edges of P). The path will only change direction on one of these chords. The algorithm simply follows the dual graph and computes the next chord. Notice that this algorithm finds a path with the smallest link distance. However, there could exist a shorter length path with the same link distance.

Non-Euclidian metrics are often studied in Computational Geometry, and the shortest path problem is no exception. A popular problem is to consider

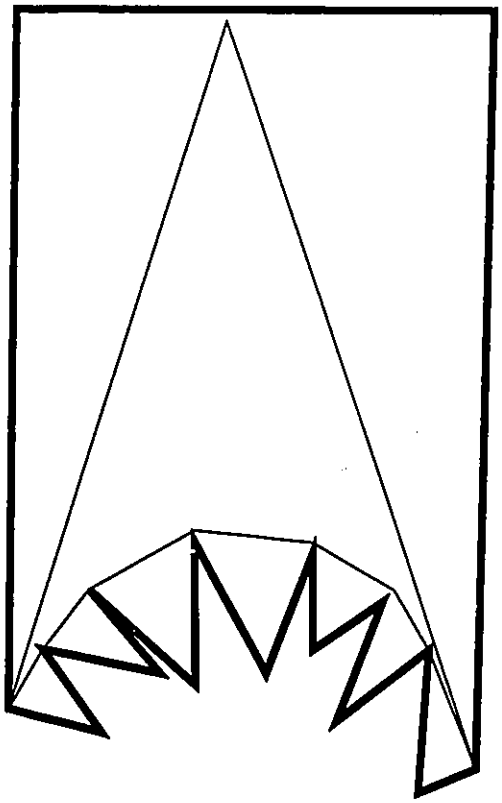

Figure 2.2: The link distance is one

orthogonal link distance (also called the *Manhattan*, L_1, or *rectilinear* link distance), which may be more realistic when dealing with simple robots when travel is only possible along one of the axes at a time. Also, this metric can be used to simulate travel within a city where all streets are orthogonal.

This problem is solved for orthogonal polygons in [Berg91, Schu93]. First [Berg91] showed that there exists a shortest path which also has the smallest possible link distance. That paper shows many related algorithms, including a linear time approximation of the solution. The main results from that paper were improved in [Schu93]. The improvements lie in storage and preprocessing: both go from $O(n \log n)$ to linear. Query time remains the same. In both papers, queries on the link number require constant time for vertices of P and $O(\log n)$ for arbitrary points. Reporting the shortest path requires the same time complexities, plus the length of the path. In [Schu93], the data structure uses a polygonal subdivision of P and its dual tree, which reminds us of the approach used in [Suri86] discussed above.

Of course, countless other shortest path problems within a polygon's boundary exist. For example, see [Espi91, McDo92, Wynt93].

Other types of shortest path problems exist: within graphs [Dijk59], on polygonal terrains [Mitc91], on triangulated irregular networks (TINs) and on polyhedrons [Chen90, Mitc87a], in $\mathbb{R}^2$ around polygonal obstacles [Stor94, Arki91], among weighted obstacles [Leè90] ....

Among those, one which relates to this work is the shortest path on a polygonal terrain. This problem occurs when one is given a terrain with varying characteristics to travel on. Each polygon represents a region within

which travel cost is homogeneous. The solution consists not only of finding the shortest path within these regions, but also which region to travel through. This problem is therefore much more complex that the one presented in this thesis, and its best known algorithm is also very much complex: $O(n^8L)$ where L is the precision [Mitc91]. But since that problem still relates to the one studied here, we will study it in more detail in the next chapter.

Chapter 3

Refraction

A well known law of Physics is Snell's law of refraction, by Willebrord Snell (1591–1627). This law comes from the problem of light travelling through various transparent mediums (eg. air, glass, water, ...). When observing light travel, one notices that part of the light is reflected when it enters another medium (ie. water reflecting the sun) and some light enters the other medium. The angle at which light travels through the second medium is different from the angle at which it travelled through the first medium. We can see the angles at which light reflects in Figure 3.1, where

$$\frac{\sin\theta_2}{\sin\theta_1} = \frac{v_2}{v_1} = constant$$

Refraction is also found in some calculus problems to find a shortest path when travelling from one medium to another of different density. This idea was used by Mitchell and Papadimitriou [Mitc87b, Mitc91] to solve a shortest path problem. The problem they looked at consisted of finding

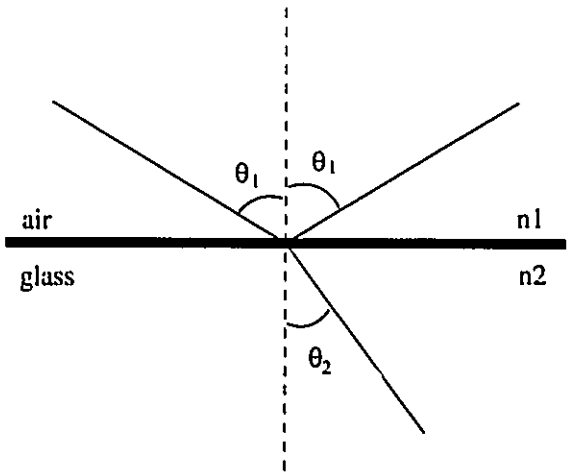

Figure 3.1: Snell's Law of Refraction

a shortest path in a polygonal terrain where the cost of travelling in each region has a weight. In general, they considered a triangulated terrain in which a different cost of travel is associated with each triangular region. This problem is much more complex than the one studied here since it involves a great number of contiguous polygons. Their main result is to prove that there exists a polynomial algorithm to solve this problem. They do this by giving an algorithm which has a complexity of $O(n^8L)$ where L is the bit precision. That algorithm uses $O(n^4)$ space.

To achieve this result, they apply the law of refraction to the shortest path problem. Generally, this applies when we travel from one region to another region with a different weight. The region is considered to be the medium, and the path taken is the same as the path a ray of light would take. They also show that one of the regions could be long and narrow (ie.

a road). In this case, when the path is refracted from one region onto the road, it travels some distance on the road and is later refracted onto one of the neighbouring regions. Note that when applying these laws of Physics to shortest path problems, the path is always fully refracted: no part of it is reflected.

Here we give a very brief outline of the somewhat complex algorithm used by Mitchell and Papadimitriou. The approach is to apply the continuous Dijkstra technique [Mitc87a] to solve the Discrete Geodesic Problem. They try to simulate a *wavefront* from the starting point p. This wavefront is a sweep line l where all points on l are at equal distance to p. When the wavefront hits an edge, some changes occur in the direction of the path (called *event*), due to refraction. They show that there are $O(n^4)$ events, which are put in a priority queue. An event is taken out of the priority queue to find new events. The space used by the algorithm is bounded by the number of events, and is therefore $O(n^4)$ in size. Searches which require $O(n^4 L)$ time are done on the data, yielding the total running time of $O(n^8 L)$. L is shown to have a logarithmic bound, related to the precision of the result. The authors state that regardless of the high complexity, their algorithm is still practical since average case performance is much faster than the worst case.

We use these results on refraction in this thesis. For our purposes, we only need to look at refraction onto a road. Moreover, once the path is on a road, it can only be refracted onto the same region it came from since our problem contains only one polygon. Of course the road can be composed of many contiguous line segments. This and other details will be discussed in

Figure 3.2: Refraction onto and off a line segment

later chapters. Figure 3.2 shows an example of a path refracted onto and off a road composed of a single line segment.

Notice that the complexity of the algorithm by Mitchell and Papadimitriou is much higher than the results shown in this thesis. This is due to a number of problems which arise when considering many regions with different weights. On top of calculating the shortest path within a region, they need to compute what regions the path must travel through and where the path should cross the boundary between each regions.

Chapter 4

Definitions and General Lemmas

The purpose of this chapter is to provide basic miscellaneous definitions and lemmas which will be useful for the algorithms and proofs in this thesis.

4.1 Special Properties

Let $Q = v_1, \ldots, v_n$ be a set of n different points on the plane. A closed polygonal chain R of Q is the sequence of line segments joining v_i to v_{i+1}, $i = 1, \ldots, n$, where $v_{n+1} = v_1$.

The line segments connecting v_i to v_{i+1}, $i = 1, \ldots, n$ will be called the edges of R. R is called simple if no two of its edges intersect, other than at a common endpoint. A polygon is a closed bounded region of the plane bounded by a simple polygonal chain.

Throughout this thesis algorithms and proofs rely on the fact that polygons are simple.

The polygon P in which we will try to compute the shortest path will be assumed to have its vertices labelled, in the counterclockwise direction, $v_0, \ldots, v_{n-1}$ 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}. Visibility will be discussed in more detail in Chapter 5, but for now let us state that two points are visible from each other if the line segment that joins them is totally contained in P. 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 exactly n ears, and in general, simple polygons have at least two ears [Meis75].

A vertex v of P is said to be reflex if the interior angle in P at v is greater than π. Non-convex polygons have at least one reflex vertex.

4.2 Shortest Paths and Shortcuts

Let us re-state the main problem. Given a polygon P and a cost of travel within the boundary of the polygon of s per unit distance. The cost of travel on the boundary of P is 1. Find the shortest path from two points $p, q \in P$.

The main principle behind the shortest path problem is that we want to know when to travel on the edges of P or when we should travel within the interior of P. Travel on the edges of P costs less (we can travel fast).

However it may be that the shortest path through the interior of P is less expensive. In that case, we would travel slowly, but the Euclidian distance to travel would be much smaller, yielding an overall lower cost.

It is easy to notice that the shortest path between two points $p, q \in P$ will be composed of a number of segments. Some of these segments will be on edges of P whereas others will lie in the interior of P.

If we assume that travelling one unit of distance within the interior of P takes s seconds while travelling 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 one edge of P then its weight is $w(l(x, y)) = |l(x, y)|$ where $|l(x, y)|$ is the Euclidean distance between x and y.

Consider two points p and q of P, and a polygonal chain $H(p, q)$ from p to q contained in P, that is composed of a chain of line segments starting at p and ending at q. (We will see later that the shortest path follows this description.) 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.

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 chain $H'(p,q)$ from p to q, we have $W(H(p,q)) \leq W(H'(p,q))$.

We now give a formal definition of shortcuts. Consider two points, $p, q \in P$ and a shortest path from p to q. In general this path consists of line segments which are either contained in the interior of P or are on an edge of P.

Definition 1 *We will call shortcuts all possible segments s_i of a polygonal chain (eg. the shortest path) where for every i, s_i lies entirely (except for endpoints) within the interior of the polygon. We also call shortcuts all edges contained in the interior of P and which respect the law of refraction because they are potential shortcuts, and there is no way to differentiate them from* real *shortcuts without computing all shortest paths first.*

We will see in detail later how to determine if an edge respects the law of refraction. Shortcuts are the basic building blocks of shortest paths. We will use the concept of shortcuts often in this thesis.

4.3 Shortcut Properties

We start by solving a simple shortest path problem from p to q. We are given start and end points p and q (see Figure 4.1), where q lies on a line e, and p lies below e and to the left of q. Cost of travel on e is 1, and cost of travel elsewhere is s, where $s > 1$. We want to find the least expensive path from p to q.

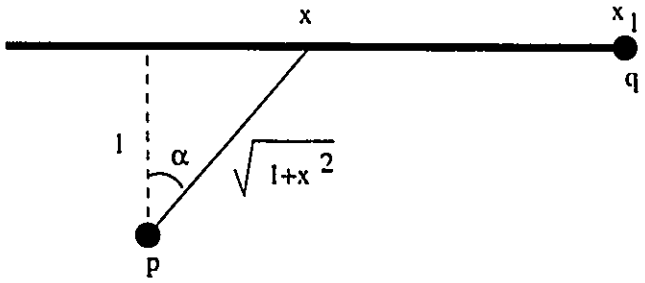

Figure 4.1: A simple weighted shortest path problem

We now solve that problem and show that the angle at which the shortest path follows the law of refraction, as shown in Chapter 3. We define the cost of travel from p to q as

$$F(x) = s\sqrt{x^2+1} + x_1 - x$$

where $s\sqrt{x^2+1}$ is the cost of travel from p to x, and $x_1 - x$ is the cost of travel from x to q. We look for the shortest path, so we must minimize that function. So we consider its derivative:

$$F'(x) = \frac{sx}{\sqrt{x^2+1}} - 1$$

By setting $F'(x)$ to 0, we get:

$$s = \frac{\sqrt{x^2+1}}{x}$$

It is obvious from this that x is a function of s (recall that s is a constant).

Notice that the resulting shortest path is composed of two line segments: one on e and another one in space. We will call the latter one a shortcut

because, as in real life, it is a path off the main road (ie. off the line e), and it is a shorter path. 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 travelling time than travelling along the edges of P.

Going back to Figure 4.1, we notice that:

$$\sin\alpha = \frac{x}{\sqrt{1+x^2}}$$

We therefore have $s = \frac{1}{\sin\alpha}$. This leads us to a lemma that describes basic properties of optimal p-q paths:

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 ($0 < \alpha < \pi/2$), or the shortcut meets e at one of its endpoints.*

If a shortcut is contained in an ear of P defined by v_1, v_2, and v_3, one can easily see two ways to get from v_1 to v_3: to travel along a shortcut to bypass v_2 or to travel on the edges of the ear. In the latter case, we will say that we are travelling around (the boundary of) a.

Shortcuts have a number of properties. If an end of a shortcut falls on an edge e of P, it must do so at a given angle α to the perpendicular to e (see Lemma 1), unless it meets e at an endpoint. If the end of the shortcut lies on a vertex v of P, the angle between the shortcut and the edges of P which are connected to v must each be greater than $\pi/2 - \alpha$. So no shortcut will leave a vertex of P if the angle of P at this vertex is less than $\pi - 2\alpha$. This

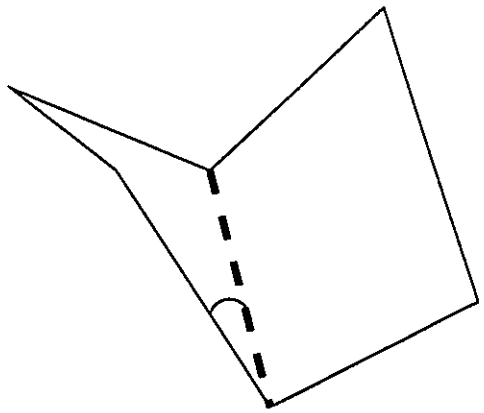

Figure 4.2: A vertex-to-vertex shortcut not to be considered

property is violated in Figure 4.2, and the shortcut shown is not a valid one. In this case, the vertex-to-vertex α-visibility is not considered since it goes against the law of refraction.

Observation 1 *Two shortcuts which belong to the shortest path cannot meet within the interior of P.*

It is trivial to see, with basic Euclidian geometry, that if a shortcut were to meet another in the interior of P, then it would be easy to make the shortest path shorter by adding a line segment to eliminate the intersection of the two shortcuts (see Figure 4.3).

If we cannot add a line segment from the beginning of the first shortcut to the end of the second shortcut because they are not visible from one another, then there exists at least one vertex v of P which stands between them. In that case the shortest path may be composed of two (or more) shortcuts, going through v.

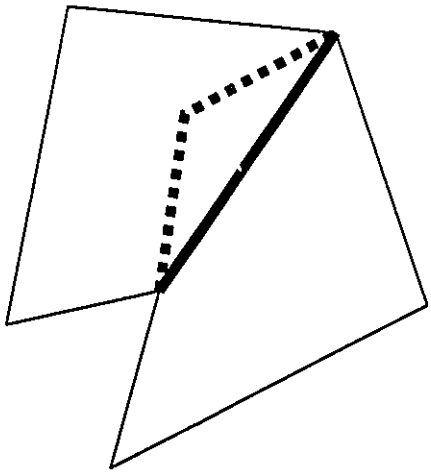

Figure 4.3: Shortcuts are composed of a single line segment

4.4 Special Shortcut Cases

In this section, we show a way to limit the number of shortcuts which need to be studied to a finite quantity. Indeed, as we will see, there exist cases where an infinite number of shortcuts are possible. We will show that we need only consider some of these shortcuts.

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 this thesis:

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.

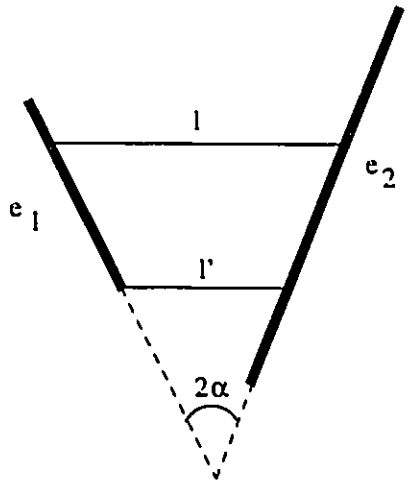

Figure 4.4: Edge-to-edge shortcut in a wedge of angle 2α

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 with e_2. Only two cases can arise:

1. The angle formed by the lines containing e_1 and e_2 is equal to 2α.

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 (see Figure 4.4). Consider another shortcut l' parallel to l and below it, with both endpoints in e_1 and e_2 such that l' contains a vertex of P (an endpoint of e_1 or e_2). Then it is easy to see that travelling along l is equally expensive to travelling from one endpoint of l, through part of e_1, through l' and finally through part of e_2 to get to the other endpoint of l. This generates another shortest path from p to q avoiding l.

The proof for this is trigonometric. First, let us assume that e_1 and e_2 meet at a vertex. This is done to simplify the calculations and, as we will see later, the results hold just as well if e_1 and e_2 don't meet. Also, we assign a length of 1 to the distance between the intersection of e_1 and e_2 and both endpoints of l. Through trigonometry we get the following equation:

$$l^2 = 1^2 + 1^2 - 2(1)(1)\cos 2\alpha = 2 - 2\cos 2\alpha \tag{4.1}$$

We also know that:

$$\cos 2\alpha = \cos^2 \alpha - \sin^2 \alpha$$

and:

$$\cos^2 \alpha = 1 - \sin^2 \alpha$$

so we get:

$$\cos 2\alpha = 1 - 2\sin^2 \alpha \tag{4.2}$$

By substituting Equation 4.2 into Equation 4.1, we get:

$$l^2 = 2 - 2(1 - 2\sin^2 \alpha) = 4\sin^2 \alpha$$

$$l = 2\sin \alpha \tag{4.3}$$

We have already shown that $\alpha = \sin^{-1}(1/s)$, so we have:

$$\sin \alpha = \frac{1}{s} \tag{4.4}$$

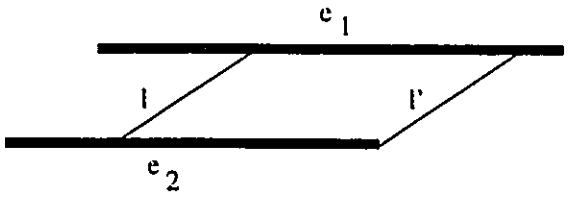

Figure 4.5: Edge-to-edge shortcut between two parallel lines

So, finally, by substituting Equation 4.4 into Equation 4.3, we get:

$$l = \frac{2}{s}$$

$$ls = 2$$

This shows that going around (cost is 2) is equal to taking the shortcut (which has cost equal to its length times the cost of travelling inside: s). Because this holds for the case when e_1 and e_2 meet, and because the distance between that meeting point and l is arbitrary (set to 1 above), it is easy to see that any shortcut parallel to l will give another equally expensive shortest path. This, of course, holds for cases when e_1 and e_2 do not meet.

A similar analysis can be done for the case when e_1 and e_2 are parallel (see Figure 4.5). Assuming that e_1 and e_2 are horizontal, one notices immediately that moving l to the left or to the right does not change the cost of travel. So again, there exists l', parallel to l, where one endpoint of l' lies on an endpoint of e_1 or e_2. (Actually, there exist two such lines, l' and l''.)

In both special cases described above, there is an equivalent cost shortcut with one endpoint on a vertex of P. Note that in some special cases, there

may be an obstacle in the way of l'. In that case, we would consider l'' which is still parallel to l, but which touches one vertex of the obstacle. (This is done by sliding l down until it comes in contact with the obstacle.) In that case, both endpoints of l'' lie in the middle of an edge, but we simply break l'' into two parts where it touches the obstacle. Since the contact point to the obstacle is a vertex, this creates two vertex-to-edge shortcuts, which have equivalent cost to l.

To simplify the algorithm and the proof, we will only consider the paths which have at least one endpoint at a vertex of the polygon. This is permitted since the cost of travelling between edges which have a difference in angle of 2α, 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, which greatly simplifies the algorithms.

4.5 Cutting-Off Regions

In this section we show how to simplify polygon P by trimming away parts of it.

We know that the shortest path will alternate between series of edges of P and single shortcuts. We have already seen that two shortcuts cannot be visited in sequence without travelling on an edge or vertex of P. On the other hand, consecutive edges of P can be visited, but not when the interior

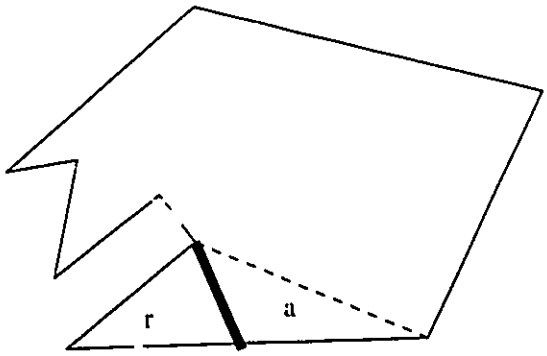

Figure 4.6: Cutting away part of an ear

angle between them is less than 2α, as shown in the following lemma:

Lemma 3 *If the angle of an ear a is less than 2α then there exists a region r of a which the shortest path never travels through if neither p nor q are contained in r. Therefore any shortest path from p to q is entirely contained in $P - r$.*

Proof: First, we must define r. Let v_1,v_2, and v_3 be the vertices which make the ear. Let e_1 be the polygon edge from v_1 to v_2 and e_2 be the edge from v_2 to v_3. Let us shoot a ray l from v_1 (resp. v_3) so that it hits the infinite line defined by e_2 (resp. e_1) at angle $\pi/2 - \alpha$. Two possible rays could be shot but we only consider the one that goes away from v_2 on e_2 (resp. e_1). The region r is the triangular region which is separated from the remainder of P by l, as in Figure 4.6. Note that r is always included in a, and it can have any size from very small to all of a.

In Figure 4.7, we see an example where r is the entire ear. The two vertex-to-edge shortcuts (shown with dashed lines) only fall on the extensions of e_1

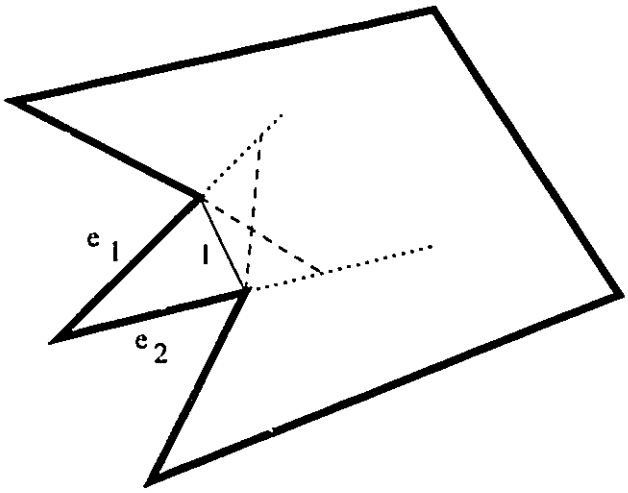

Figure 4.7: Cutting away the entire ear

and e_2. In that case, there exists a vertex-to-vertex shortcut l which cuts the entire ear.

In Section 4.4 we saw that it is equally expensive to go around on an ear of angle 2α than to cut across it. It is straightforward to deduce that if the angle is larger than 2α, it is preferable to travel along the edges. Similarly, when the angle is smaller it will be less expensive to cut across the ear on a shortcut. In these cases, there is no cost advantage to travelling within the region which was cut by the shortcut.

Lemma 3 holds for regions of P other than ears. This concept will be discussed further for orthogonal polygons (see Chapter 8) and for convex polygons (see Chapter 9).

Chapter 5

Visibility Graphs

5.1 Visibility

Visibility is a property which can be studied in various situations. In our case, we study visibility within a polygon P. We recall that two points $a, b \in P$ are said to be visible to each other if the line segment which joins a and b is entirely contained in the interior of P. Otherwise a is not visible from b, and vice-versa. Visibility is also often studied for a family of objects (lines or polygons) on the plane [Asan86, Welz85]. In that case the complexity of the algorithm to compute the visibility graph is generally higher than the complexity of the algorithm to solve the same problem within a polygon because the objects are not connected. Of course, visibility can also be studied in higher dimensions.

Often, visibility is computed from a fixed object. For example one may want to know which vertices of P are visible from a, or what visibility region

within P is visible from a. Visibility from a point can be found in linear time [ElGi81, Lee83].

Likewise we can calculate visibility from an edge e of P [Avis81, Lee86]. In this case we can consider weak visibility [Sack90] or strong visibility. Regions which are weakly visible from e are regions which are visible from at least one point on e. For a region to be strongly visible, it must be visible from all points of e. Much more can be said about visibility. For details and more problems see [ORou87].

A visibility graph is a data structure which is built from knowledge about visibility, in our case, within a polygon. To construct it, we first consider one vertex v_1 of P and find all edges of P which are weakly visible from v_1. We store weakly visible edges in memory. We then continue on with another vertex v_2 and calculate its visibility region. The resulting visibility graph is a graph which links all vertices of P to weakly visible edges.

Lemma 4 *Visibility graphs can be built in time $O(E)$ where E is the number of visibility edges.*

Proof of this is shown in [Hers89] where linear time triangulation yields a $O(E)$ algorithm. Note that E is at least linear in size, and its upper bound is $O(n^2)$. The worst case occurs when P is convex.

The algorithm developed by Hershberger is as follows: First, P is triangulated and then a shortest path spanning tree (see Chapter 6), which they call *shortest path map* is computed for a vertex v_1 of P, in linear time, using visibility algorithms from [Guib87]. This map is later updated for v_2 in time

proportional to the differences between the map for v_1 and the map for v_2. Building the map for all v_i is shown to be proportional to the total number of differences between consecutive maps. It is also shown that each difference between consecutive maps corresponds to a new visibility edge. Therefore the building of the maps takes $O(E)$ time.

It is interesting to note that the algorithm in [Hers89] uses shortest paths to compute visibility, and in this thesis we will use these visibility results to compute another type of shortest path. This two-way relationship shows the strength of the link between the two concepts. More on this relationship can be found in [Hers87].

5.2 α-Visibility

We start this section by defining α-visibility graphs:

Definition 2 *Let s be a constant where $s \geq 1$ and $\alpha = 1/\sin s$. The α-visibility graph is a weighted graph which includes all possible shortcuts within P as well as all the edges of P.*

For an example of an α-visibility graph, see Figure 5.1.

The visibility graph of a polygon is $\Omega(n^2)$ in size. In Figure 5.2 we see an α-visibility graph in which 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

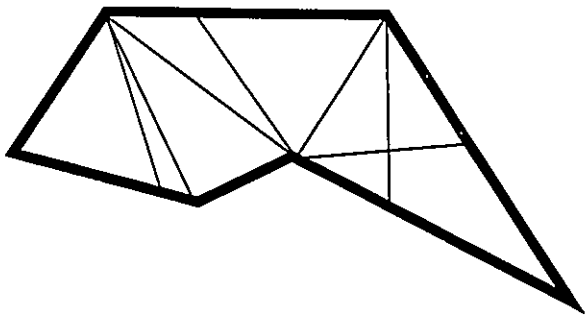

Figure 5.1: An α-visibility graph

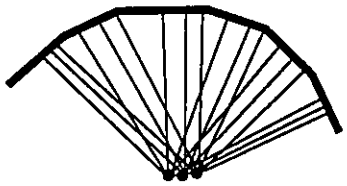

Figure 5.2: $O(n^2)$ shortcuts to edges

in the case where the cost of travelling inside the polygon is extremely high (ie. α is small).

Lemma 5 *We can compute the α-visibility graph of P in time proportional to the size of the visibility graph of P.*

Proof: We start by computing the visibility graph, using [Hers89], but in addition to vertex visibility, we store the sections of edges of P which are visible from the source v. Once we know what section of each edge is visible from a given vertex, we can compute the α-visibility graph in time proportional to the number of edges visible. Since we know the angle at which a shortcut must meet an edge of the polygon, it is easy to compute

where such a shortcut from vertex v would intersect a visible edge e. For each visible e, there can exist two such intersections. Then we must verify if the places of intersection are contained in the visible region of e from v. If so, the shortcuts are added to the α-visibility graph.

Note that this approach may not be optimal. While on average, the α-visibility graph may be $O(E)$, where E is the size of the visibility graph, in some cases, this is not true. An algorithm proportional to the number of α-visibility edges, E_α, would be preferred since it is always true that $E_\alpha \leq E$. So the question of whether $O(E)$ processing is optimal to compute the α-visibility graph remains open.

For orthogonal polygons, if we are only interested in edge-to-vertex shortcuts, we can build the α-visibility graph in $O(n \log n)$ time by simple ray shooting. This is because, as we will see in Chapter 8, the number of angles at which edge-to-vertex shortcuts can appear is a small constant. However, since we must also consider vertex-to-vertex shortcuts, we must use the algorithm described above.

The definitions and lemmas discussed in this chapter will be used in later chapters. They represent the basic properties that the weighted shortest path must follow.

Chapter 6

Shortest Path from a Single Source

In this chapter, we solve the problem of finding the shortest path from a single source v to any point in polygon P. We will prove that, using $O(E \log n)$ preprocessing, we can answer shortest path queries from v to any point $p \in P$ in $O(\log n)$ time, using $O(n \log n)$ space. We will also show a variation of this where queries are answered in $O(\log^2 n)$ time, but using only linear space.

For doing so, we use a popular tool called the *shortest path spanning tree.* As their name implies, spanning trees are trees which cover all vertices of a polygon. Euclidian shortest path spanning trees are spanning trees with root at a vertex v of P where all Euclidian shortest paths to other vertices of P follow a branch of the shortest path spanning tree. They can be built in linear time [Guib87], which allows $O(k + \log n)$ queries, where k is the size of the shortest path being reported. For an example of a shortest path

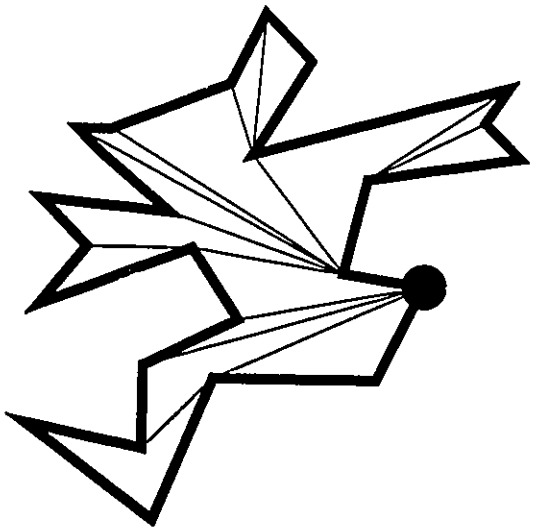

Figure 6.1: A Euclidian shortest path spanning tree

spanning tree, see Figure 6.1.

In order to use spanning trees for this thesis, we need to modify their construction slightly since we are not simply working with Euclidian distance. Like standard spanning trees, ours will be trees which represent the shortest path from a vertex v to other vertices of P. However, we need to follow the weighted shortest path to each vertex. Therefore, the tree can use new vertices on edges of P (see Figure 6.2). The formal definition follows:

Definition 3 *Let v be a vertex of a polygon P. The α-shortest path spanning tree rooted at v is a tree whose branches correspond to shortest paths from v to all other vertices of P.*

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

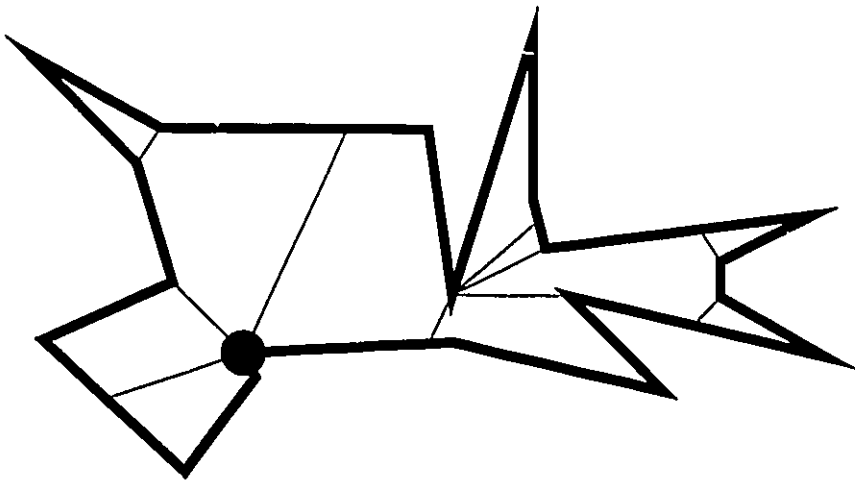

Figure 6.2: An α-weighted shortest path spanning tree

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

This is easily done by applying Dijkstra's algorithm to the α-visibility graph. We keep all shortest paths from v to all other vertices of P, yielding the tree.

Just like Euclidian shortest path spanning trees, α-shortest path spanning trees are planar. Because polygon P is simple, if two branches of the tree were to cross each other in P, it is obvious that the crossing would need to involve two shortcuts. It has already been shown in Chapter 4 that a shorter path would exist in that case.

We will call the α-shortest path spanning tree $T_\alpha(v)$. Because it is planar, is easy to see that $T_\alpha(v)$ induces a partition $\Pi_i'(P, v)$ into $O(n)$ regions

$R_1, \ldots, R_r$. (Notice that these regions are not necessarily convex.) 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.

Notice that a shortest path beginning in the interior of a region R_i will take a shortcut to $F(R_i)$ or to an edge of R_i which is also an edge of P (as opposed to an edge of R_i which is defined by a shortcut). Then the shortest path will travel on to $F(R_i)$ by following the boundary of R_i. What we need to know is which edge of R_i the shortest path first takes a shortcut to.

The regions can have arbitrarily large size but we want to find the distance from any point in R_i to $F(R_i)$ in constant time. In order to solve this we subdivide each R_i into $O(|R_i|)$ regions, $R_{i,1}, \ldots, R_{i,t}$, $1 \leq i \leq r$ in such a way that for every point $q \in R_{i,j}$, $1 \leq j \leq t$, we can calculate the distance from q to $F(R_i)$ in constant time. This is done by associating a sub-region to all edges of R_i which is also an edge of P.

The subdivisions are separated by *bisectors*. Travel cost to a point which lies on a bisector is equal whether we travel through either edges of R_i (and of P) which define that bisector.

For each edge e_j of P which belongs to R_i, we associate a region $R_{i,j}$ where the shortest path from any point p in $R_{i,j}$ to $F(R_i)$ travels on a shortcut directly from p to e_j (and then on to $F(R_i)$ along the boundary of R_i). Another edge e_k of P and R_i yields a region $R_{i,k}$ which is separated from $R_{i,j}$ by a bisector. For all points which lie on that bisector a path through e_j has the same cost as a path through e_k.

We now describe the aspects of bisectors. The bisector of two connected

edges is a straight line. This is easy to see by finding one point of the bisector, and scaling it, producing a straight line.

When the edges become disconnected, however, the bisector becomes multipart. Parts of it are straight lines, but other parts of it may be curved. Let us first analyze each section of the bisector separately. Later we will put these sections together to create a complete bisector.

Travel from an edge to $F(R_i)$ has a cost which is associated to the the closest vertex of the edge to $F(R_i)$. The shape of the bisector is influenced partly by this difference in wheights.

First, let us look at the bisector between two vertices (or the endpoints of two edges). We can imagine that both vertices *grow* at a set constant speed (although one may start growing before the other due to different distances to $F(R_i)$). See Figure 6.3 for an example. The growing represents the area which can be reached after a given time. The bisector is the curve where two growing circles of the same cost (but not necessarily of the same size) meet. This curve is identical to bisectors in weighted Voronoi diagrams: it is a hyperbola. If both vertices have the same weight, the hyperbola is actually a straight line. The bigger the difference in cost, the more the hyperbola is curved.

Now, we look at the bisector between a vertex and the interior of an edge (see Figure 6.4). It is easy to see that the bisector here is a parabola (from the definition of a parabola). Here we notice that a difference in cost can be simulated by a translation of the edge. The aspects of the curve therefore always remains a parabola. These bisectors are of the same type as bisectors

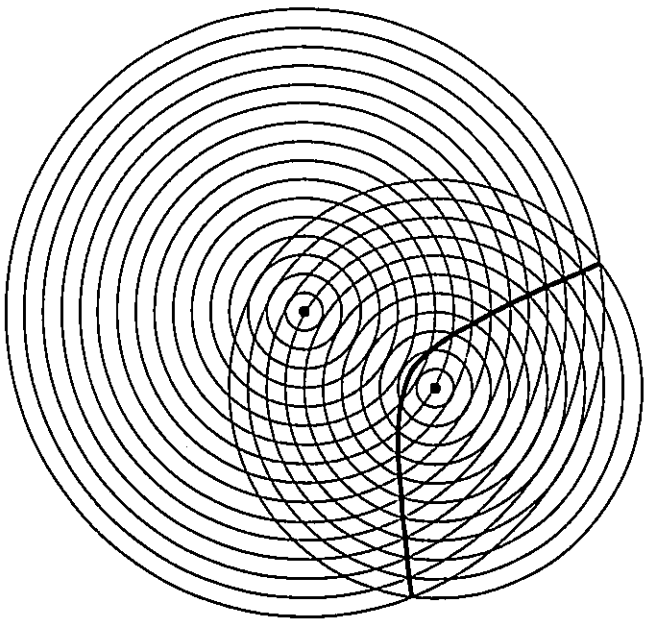

Figure 6.3: A hyperbolic bisector

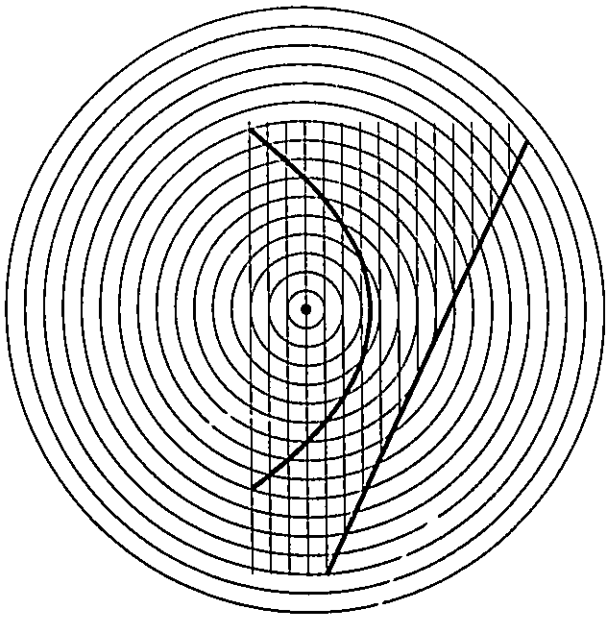

Figure 6.4: A parabolic bisector

in Voronoi diagrams of line segments.

The bisector between the interior of two edges is a straight line, which can be seen by *growing* both edges (see Figure 6.5). Notice that because travelling along the edge has a cost, the growing will be larger at the end of the edge at which we start. However, the boundary of the regions being grown will still be defined by straight lines. The intersection between two boundaries of the same cost is a straight line segment.

Of course, the types of bisectors described don't define (in most cases) the complete bisector between two edges. In order to describe an entire bisector, we must integrate all of these into one. If we consider two edges in space, in the worst case interaction between the two edges will be as follows: vertex-

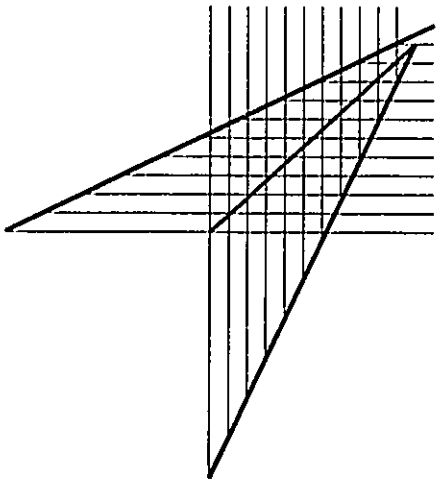

Figure 6.5: A straight line bisector

vertex; vertex-edge; edge-edge; edge-vertex; vertex-vertex. Of course, the relative position of the two will often restrict these interactions to a lesser number of components. So, in the worst case the bisector will be composed of a part of a hyperbola, a part of a parabola, a straight line segment, a part of another parabola, and a part of another hyperbola. If there is a lesser number of interactions, the number of components of the bisector will also be smaller. Each of the components of bisectors described above can be computed in constant time. The same holds for their intersection. It is therefore easy to compute a complete bisector in constant. So we have:

Lemma 7 *A bisector between two subpartitions $R_{i,j}$ and $R_{i,k}$ formed by two edges of P in R_i is composed of at most two hyperbolic arcs, two parabolic arcs, and one straight line segment and can be computed in constant time.*

The resulting sub-partitionings create $O(n)$ regions in total since there is one region created per edge of P in each R_i. This construction is very similar to computing a Voronoi diagram, which can be computed in $O(n \log n)$ time for a set of n points [Prep85] (provided we can compute each bisector in constant time, which is what we do here). Let $k_i = |R_i|$. Computing the partitioning of one region therefore takes $O(k_i \log k_i)$. Since $\sum_{i=1}^{t} k_i = n$, we have a final time complexity of $O(n \log n)$.

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 from p to $F(R_i)$

3. add to it the distance from $F(R_i)$ to v

The bisectors are not necessarily monotone, but since point location algorithms used in the query restrict us to using monotone regions, if a bisector is not single valued in one direction (say the x coordinate), we will split the bisector at its vertical tangents. We will also need to split the two neighbouring regions (with a line segment) starting at these vertical tangents. Notice that since the bisectors are, in the worst case, composed of a constant number of quadratic functions, there is a constant number of splits per bisector.

Step 1 above can be done in $O(\log n)$ time, using the *trapezoid* method for point location which can be used for curved bisectors [Prep85]. This requires $O(n \log n)$ space. Alternatively, we can compute Step 1 using the

chain method [Lee77, Prep85]. This has a query time of $O(\log^2 n)$, but only requires $O(n)$ space. Steps 2 and 3 are executed in constant time because of the preprocessing described earlier. So we have:

Lemma 8 *We can solve the single source shortest path problem in $O(\log n)$ query time, using $O(E \log n)$ preprocessing and $O(n \log n)$ space. Alternatively, we can reduce the space requirement to $O(n)$ by having $O(\log^2 n)$ query time.*

We will use α-spanning trees later in this thesis (see Section 7.3) for computing the distance between any point within P and the root of the α-spanning tree.

Chapter 7

The Simple Polygon Case

7.1 Introduction

In this section we give two algorithms to find shortest paths between two points p and q in a simple polygon P with n vertices. Note that this chapter does not stand on its own. Many lemmas, definitions, and observations from previous chapters are used here.

Our first algorithm uses $O(E)$ space, and $O(E \log n)$ time, where E is the number of edges in the α-visibility graph of P. Our second algorithm uses $O(nE \log n)$ preprocessing, and we show two variations for the query. The first variation uses $O(n^2 \log n)$ space, and answers queries in $O(n \log n)$ time. The second variation uses $O(n^2)$ space but the queries require $O(n \log^2 n)$ time.

From Lemma 2, we obtain the following result :

Lemma 9 *Given any two points p, q on the boundary of P, there is a shortest p-q path contained in the edges of the α-visibility graph of P.*

Now we move on to the more general case where p and q may be positioned anywhere within the interior 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.

Case 1 is very simple: it occurs when p and q are relatively close to each other, and going through the boundary of P would only increase the cost of the path. Case 2 can happen if both p and q are close to one edge e of P. In that case it would not be useful to travel to other parts of P. Because of their simplicity, Cases 1 and 2 can be dismissed easily. Finally Case 3, the general case, is much more interesting to study: the path can travel almost anywhere within the polygon.

Notice that if the shortest path from p to q travels through at least one vertex (which is included in the α-visibility graph) of P, the path must first travel from p to some edge or vertex of P, and then follow some path along the α-visibility graph (as in Lemma 9), to finally leave at an edge or vertex of P to reach q.

It now follows:

Lemma 10 *When $H(p,q)$ visits at least one vertex of P, 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 α-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 $\pi/2 - \alpha$.*

This lemma gives an important property to the shortest path.

7.2 First Algorithm

From Lemma 10 we have :

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

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. To start, we calculate the α-visibility graph of P. This can be done in $O(E)$ time. First we add all shortcuts from p (resp. q) to P. Those are all shortcuts from p (resp. q) to vertices of P which meet P at a valid angle, as well as all shortcuts from p (resp. q) to all edges of P where the angle that the shortcut hits the edges of P respects the law of refraction. This is done in linear time by computing visibility from p (resp. q). By Lemma 9 and Lemma 10 there is a shortest path from p to q contained in the graph obtained from the α-visibility graph by adding to it the shortcuts from p and q to the boundary

of P. This can be done in linear time since it is a matter of determining visibility between p (resp. q) and P. Our result now follows by applying Dijkstra's algorithm to the resulting graph.

This algorithm is easy to implement. Given an algorithm to compute the visibility graph and Dijkstra's algorithm, it would take very little programming to have a working algorithm.

7.3 Second Algorithm

We now show how to answer shortest path queries faster than with the first algorithm, at the expense of increasing the preprocessing time to $O(nE \log n)$. We describe two variations of the algorithm. The first variation runs in $O(n \log n)$ time and uses $O(n^2 \log n)$ space. The second variation is slower: $O(n \log^2 n)$ query time, but space is reduced to $O(n^2)$.

As in Theorem 1, 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.

From Lemma 8 we know that the shortest path between any point $p \in P$ and a vertex v of P can be computed in $O(\log n)$ time for the first variation of the algorithm, and in $O(\log^2 n)$ time for the second variation. The algorithm was shown in Chapter 6.

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

Preprocessing:

1. Calculate the α-visibility graph.

2. For every vertex v of P calculate $\Pi_1'(P, v)$ (see Chapter 6)

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 (or $O(\log^2 n)$ time for the second variation).

We now have the following theorem:

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

This algorithm is somewhat harder to implement than the previous one because of the partitioning of the polygon. On the other hand, query time is improved over the first algorithm by a factor of n in the worst case. Because this approach visits all vertices of P, regardless of any structure, we believe that this algorithm is not optimal. However, we have failed to find a better approach.

Chapter 8

The Orthogonal Case

8.1 Structure

Here we discuss the problem of finding the weighted shortest path in an orthogonal polygon. An orthogonal polygon is one whose edges are all horizontal or vertical. Note that orthogonal polygons are also often called rectilinear polygons. Here we study simple orthogonal polygons: we do not allow crossings or holes. Notice that all edges meet at angles of 90° or 270°. Orthogonal polygons are studied here and elsewhere in Computational Geometry because of the great amount of structure brought forth by orthogonality (for example, see [Sack84].

All ears of orthogonal polygons will be of angle $\pi/2$. It is interesting to consider the case where it is equally expensive to take a shortcut across such an ear than to travel around it on its boundary (see Lemma 3). One can see the geometry behind this by looking at Figure 8.3. We get $\pi/2 = 2\alpha$, and

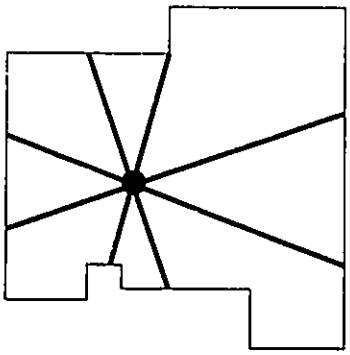

Figure 8.1: A vertex with 8 shortcuts to edges

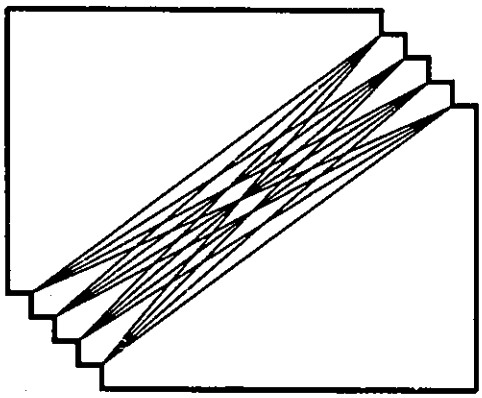

Figure 8.2: An orthogonal polygon with $O(n^2)$ shortcuts

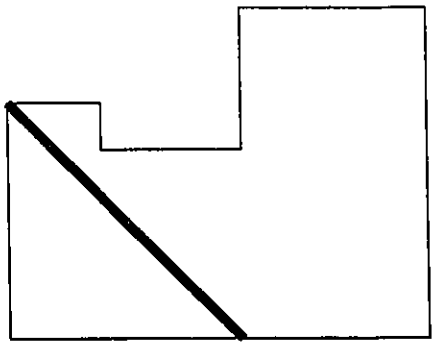

Figure 8.3: An ear where $2\alpha = \pi/2$

since $\sin\alpha = 1/s$, we have $\sin\pi/4 = 1/s$, so $s = \sqrt{2}$. So for any $s > \sqrt{2}$, shortcuts may be taken, but not across ears. And for $s < \sqrt{2}$, we will never travel on the entire length of an ear's boundary. We will analyze these two cases separately. We will show an algorithm for the case when $s \leq \sqrt{2}$. For $s > \sqrt{2}$, we will need to rely on the more general algorithm for simple polygons.

When P is an orthogonal polygon, we notice that any point in P has at most 8 vertex-to-edge shortcuts (see Figure 8.1), in contrast with $O(n)$ in the general case. Moreover a vertex v of P has at most 4 vertex-to-edge shortcuts (the other 4 shortcuts are impossible because of the edges adjacent to v). Unfortunately this doesn't help in providing a faster algorithm since there can be $O(n^2)$ shortcuts having both endpoints at vertices of P (see Figure 8.2).

We also notice that the case where $s = \sqrt{2}$ is simpler since there are only 4 possible directions (those being at $\pi/4$ from both axis) of travel from

a vertex when the other endpoint of the shortcut is an edge, instead of 8. Because this is the only noticeable difference, we were not able to find a way to improve the processing time. So we will not discuss this sub-case any further.

In the remainder of this chapter, we study sub-cases of the orthogonal problem. In the next section we will see an algorithm for when the value of s is small. Following that, we will note properties of the path for large s.

8.2 Algorithm for Small s

Here we study the case where it is preferable to cut across an ear (ie. when $s \leq \sqrt{2}$). We give an $O(E)$ time and space algorithm to solve the shortest path problem on an orthogonal polygon when $s \leq \sqrt{2}$.

In some cases where p and q are visible from each other, there is a quick solution. We call δx and δy the respective difference in the x and y coordinates of p and q. We assume w.l.o.g that $\delta y \geq \delta x$. We can state the following lemma:

Lemma 11 *It is always less expensive to go straight from p to q when we are travelling at an angle such that* $\tan\alpha > \delta y/\delta x$.

This can be easily verified by noticing than in such a case it would not be an advantage to travel horizontally or vertically (ie. along an edge of the orthogonal polygon). This could provide a heuristic approach to solve the shortest path problem.

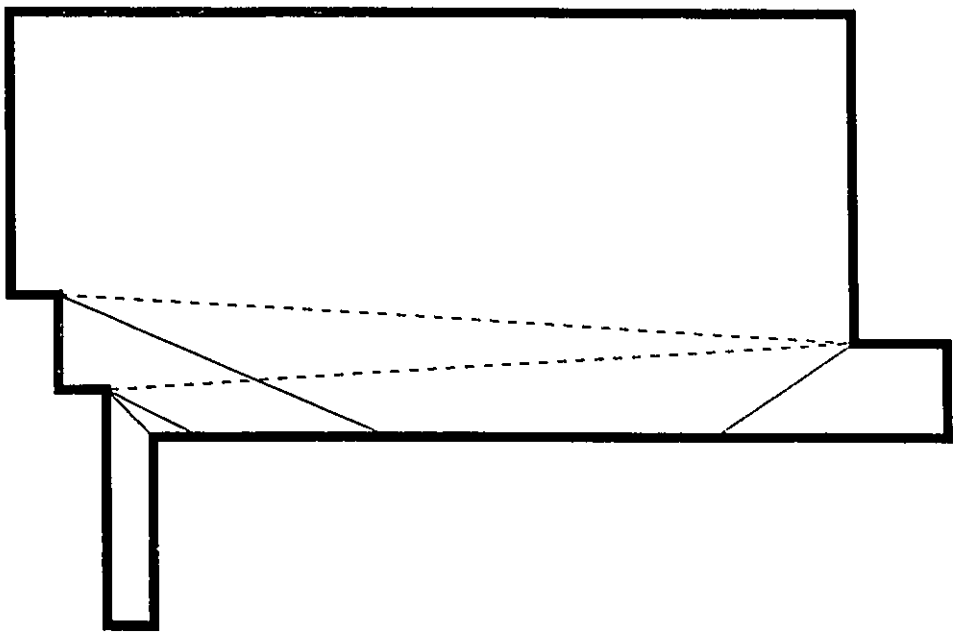

Figure 8.4: Two vertex-to-vertex shortcuts are not part of any shortest path

We will use Lemma 11 to classify shortcuts. Notice that for all edge-to-vertex shortcuts, $\tan\alpha > \delta y/\delta x$. However, tests will be needed when dealing with vertex-to-vertex shortcuts since they may actually not be part of any shortest path (ie. when the vertex-to-vertex shortcut is almost parallel to and not too far from an edge of P). For example, see Figure 8.4.

We will show the algorithm to prove the following theorem:

Theorem 3 *Let P be an orthogonal polygon. When $s \leq \sqrt{2}$, the shortest path in P can be calculated in $O(E)$ time and space.*

Proof: Since it is less expensive to cut across an ear than to travel around its boundary, an obvious approach is to trim unneeded ears. We go further

than that and trim polygons. Note that a special trimming technique is used, which is explained next.

First we start by computing the α-visibility graph of P, including the shortcuts which end at p and q. As shown in Lemma 5, this can be done in $O(E)$ time. Notice that any given shortcut of the α-visibility graph where $\tan\alpha > \delta y/\delta x$ will never be crossed by the shortest path unless p and q lie on different sides of it. (When p and q lie on different sides of the shortcut, the shortest path will cross the shortcut exactly once.) So when both p and q lie on one side of such a shortcut, we can discard the polygon on the other side of the shortcut.

So we start the recursive process of trimming parts of P. The approach taken is simple: we identify a sub-polygon of P, and an existing shortcut s across it. We test in constant time if p or q falls in it. If not, the ear is trimmed, making s a new special edge of P.

When trimming, if the trimming shortcut cuts another shortcut which is already trimmed (ie. the trimming shortcut has one end outside the trimmed polygon), we know that the shortest path will not travel on either of these two shortcuts (since it must travel through entire shortcuts at once).

We start the trimming process with all shortcuts where $\tan\alpha > \delta y/\delta x$ (ie. all edge-to-vertex shortcuts and some vertex-to-vertex shortcuts). Once no more trimming is possible for those shortcuts, we process the remaining shortcuts.

When trimming the remaining shortcuts (those where $\tan\alpha < \delta y/\delta x$), we must verify that the other boundary of the polygon being trimmed is not

in fact less expensive than the shortcut. If the cost of travel on the shortcut costs less, we trim the polygon. If not, we delete the vertex-to-vertex shortcut from the o-visibility graph and continue with other trimmings. Trimming these shortcuts last ensures that we can do the comparison of cost of travel with the *other side* of the region to be trimmed.

We notice that not all trimmed parts of the polygon are of the same shape. Ears are triangles, but we may also need to trim quadrilaterals, pentagons, ... (see Figure 8.5). Fortunately we have:

Lemma 12 *No trimmed region has more than eight sides.*

Proof: We know that there cannot be two consecutive shortcuts in a shortest path. Moreover, if the region to be trimmed contains two consecutive edges of P, it is easy to see that another shortcut would then exist, starting at the vertex between these two edges, such that it cuts the region into two parts. Finally, if the region is non-convex, there again exists another shortcut to divide the region in two. So the region's boundary must alternate between shortcuts and edges of P, and be convex. So there can be at most four edges of P involved, and four shortcuts, which yields a total of eight sides. In practice, this constant is smaller. Because the number of edges is constant, a test for inclusion of p and q can still be done in constant time.

When absolutely no more trimming is possible, we discard shortcuts which don't lie entirely within the new polygon's boundary. This is done because we know that since some parts of these unnecessary shortcuts are out of bounds for the shortest path, we are never going to travel along that

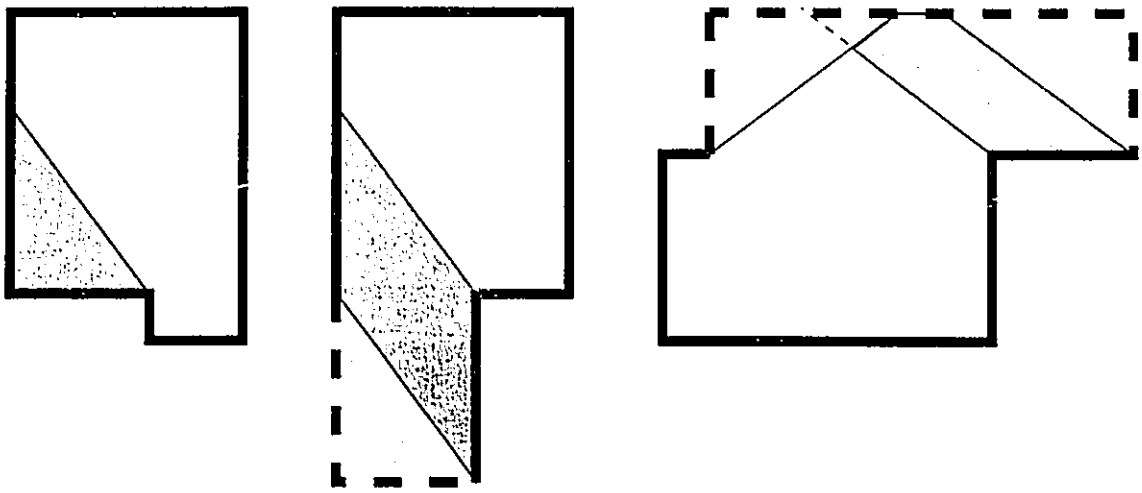

Figure 8.5: Polygons that can be trimmed

shortcut.

Notice that trimmed region will have boundaries that are not horizontal or vertical. However this doesn't transform the problem to the general polygon problem since we cannot travel on one of these shortcuts and then leave it before reaching its other end (shortcuts must be travelled on their entire length).

Notice that the trimming process always simplifies the polygon. We will need to look at all shortcuts of the α-visibility graph, so the trimming will run in amortized $O(E)$ time.

Once we are done trimming all possible ears, we are left with a chain of convex polygons which resembles a snake. Furthermore, these polygons are attached to each other with only one vertex, and no shortcut is possible within their boundary.

Lemma 13 *There is a linear number of edges in the resulting trimmed polygon.*

Proof: Edges of the trimmed polygons are a subset u of the edges of the α-visibility graph of P. Since they were created from a linear number of vertices from the α-visibility graph, and that the resulting polygons form a planar graph, there is a linear number of edges in total.

What remains to do is to travel through the chain of polygons, starting at p. Because the polygons are irreducible (ie. there is no more shortcuts inside them), the shortest path must follow one of the two sides of the polygon to reach its other end. It is therefore easy to sweep through the chain of polygons and determine which side of each polygon we should travel on. Because the chain is linear in size, the sweep takes $O(n)$ time.

This algorithm is relatively easy to implement since it only requires simple geometric test and a simple sweep once the α-visibility graph is built.

8.3 For Large s

We were unable to take advantage of the structure brought forth by orthogonality for when $s > \sqrt{2}$. For that reason, we do not show any algorithm. However, we look at possible approaches to solve this sub-case.

When s is large, it becomes less expensive to travel around the boundary of an ear than to cut across it. Here, we cannot trim ears of the polygon. So, the only possible shortcuts involve a reflex vertex.

A possible algorithm could therefore search for reflex vertices and then try to make shortcut connections in all four valid angles as well as shortcuts directly to other vertices if they fall within a valid range (ie. if the shortcut is not too close to be parallel to an existing edge). Information about visibility from these reflex vertices would be needed.

Once we have found possible shortcuts, trimming would be hard since we would need to determine if both p and q are on the same side of that shortcut to be able to trim an area. Because in this case the trimmed area is not of constant size, this could be determined in logarithmic time at best. But before trimming, we would still need to know if the shortcut actually costs less than travelling around the area. This brings the time complexity of each possible trim to $O(\log n + k)$, where k is the size of the area we are trying to trim. So trimming could take $O(E \log n + kE)$, which is less efficient than both algorithms shown in Chapter 7 for the more general simple polygon case.

After this trimming, we could be left with a polygon which is $O(n)$ in size, at which point an linear sweep similar to the algorithm given for the convex polygon (in Chapter 9) would solve the shortest path problem. We conjecture that solving this case cannot be done more efficiently than the regular polygon problem described earlier.

Chapter 9

The Convex Case

In this chapter, we study the case where P is convex. A linear time algorithm is shown here along with a proof of optimality. A convex polygon is one where all interior angles are convex (less than π). It is easily seen that in such a polygon, the entire polygon is visible from all vertices.

We now describe the algorithm. In linear amortized time, we can recursively trim parts of ears (as defined in Lemma 3) 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 parts of ears which are trimmed 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. Shortcuts can meet the polygon at vertices or on a polygon's edge, at angle $\pi/2 - \alpha$. Each edge will have at most four 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 now show an important property of the shortest path:

Lemma 14 *The line segment joining p to q is never crossed by the shortest path.*

Proof: The line segment joining p to q can only be crossed by the shortest path if the shortest path travels on one *side* of the polygon and then crosses over to the other side. But doing this is not to our advantage since it is not hard to see that crossing over to the other side would be less expensive if done later (or sooner). Because P is convex, the crossing will actually cost less towards one end, where P is narrower. So the shortcut which would cut the line segment joining p to q would be shorter if translated towards one end of P, until it hits p or q, in which case, travel on one of the sides will no longer be required.

Because of Lemma 14 and because P is irreducible, 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 this arbitrary intersection I_1, and travel counterclockwise, carrying the cost of travel from I_1 to the next intersection, I_2, and so on. If we meet another shortcut coming from p, we keep the minimum cost. We label each I_j with the minimum cost from p. 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 from p and q 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_i, in both direction of travel and to finally compare the minimum cost obtained to the straight path from p to q.

Thus we have the following theorem:

Theorem 4 *For P convex, the shortest path from p to q can be found in optimal $O(n)$ time.*

This algorithm is optimal since even if the length of the shortest path is small, we may need to compare it to a linear size path before determining that it is in fact the shortest possible path. This is obvious from looking at Figure 9.1.

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 $O(\log n)$.

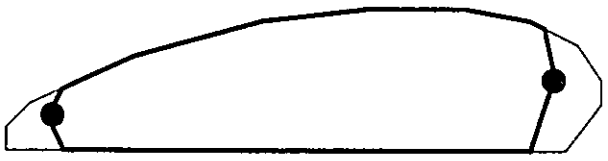

Figure 9.1: Verifying a path of length $O(n)$ is necessary

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.

Chapter 10

Conclusion and Open Problems

In this thesis we have seen a collection of algorithms to compute the weighted shortest path problem. The algorithm which solves the problem for a convex polygon runs in linear time, and is optimal. For orthogonal polygons, we give an $O(E)$ time and space algorithm (where E is the size of the visibility graph of P) when $s \leq \sqrt{2}$. If $s > \sqrt{2}$, we need to use an algorithm developed for simple polygons.

For simple polygons, we have shown two algorithms. The first is easier to implement. It uses $O(E)$ preprocessing and space and the query time is $O(E \log n)$. The second algorithm is also practical to implement (although somewhat harder than the first) and runs in $O(nE \log n)$ preprocessing, $O(n \log n)$ query time, using $O(n^2)$ space.

A common approach when calculating shortest paths is using a triangulation of the polygon to determine possible shortcuts. In the problem studied here, there is no obvious way an arbitrary triangulation can be used. For

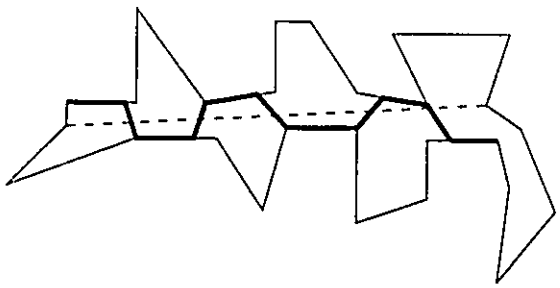

Figure 10.1: $O(n)$ crossings on a dividing line

example, a given triangulation edge could be crossed $O(n)$ times (see Figure 10.1). It is therefore hard to divide the polygon into simpler sub-cases, which makes this problem somewhat harder than what one may original anticipate.

Our algorithm for the simple polygon can be applied to non-simple polygons that have edges which cross each other (but without holes), 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 (by creating new vertices). We get a sub-problem which consists of finding a path from p to the first such vertex and so on until we enter the last region (where q lies).

It is always interesting to find an optimal algorithm. The algorithm for the convex case is therefore interesting for that reason. On the other hand, the other algorithms rely greatly on the structure of the problem. In that regards, they are also equally (if not more) interesting. This encourages further study in the area.

Open problems which arise from this thesis include:

1. Can we find the α-visibility graph optimally?
2. Can we compute the α-shortest path spanning tree using less than $O(E \log n)$ time?
3. Is it possible to compute vertex-to-edge shortcuts of the α-visibility graph of an orthogonal polygon in linear time?
4. What about randomized algorithms for all algorithms presented?
5. What if p is fixed in the orthogonal case?
6. What are the lower bounds for the simple and orthogonal cases?
7. What happens in higher dimensions?
8. If the cost of travel inside the polygon is a function instead of a constant, can the same algorithms be used? Can we use results from the Calculus of Variations field?
9. Can the same algorithms be used (with some modifications) if the edges of the polygon have different weights?

Bibliography

[Arki91] Arkin, Esther M., Joseph S. B. Mitchell, and Christine D. Piatko, Bicriteria shortest path problems in the plane, *Proceedings of the Third Canadian Conference on Computational Geometry*, Vancouver, 1991, 153–156.

[Asan86] Asano, T., T. Asano, L. Guibas, J. Hershberger, and H. Imai, Visibility of disjoint polygons, *Algorithmica*, Vol. 1, 1986, 49–63.

[Avis81] Avis, David, and Godfried T. Toussaint, An optimal algorithm for determining the visibility of a polygon from an edge, *IEEE Transactions on Computers*, Vol. c-30, 1981, 910–914.

[Berg91] de Berg, Mark, On rectilinear link distance, *Computational Geometry: Theory and Applications*, Vol. 1, 1991, 13–34.

[Chaz87] Chazelle, Bernard, Triangulating a simple polygon in linear time, *Discrete and Computational Geometry*, Vol. 6, 1991, 485–524.

[Chen90] Chen, Jindong and Yijie Han, Shortest paths on a polyhedron, *Proceedings of the 6th Annual ACM Symposium on Computational Geometry*, 1990, 360–369.

[Chen95] Chénier, Christian and Jorge Urrutia, Shortest paths in convex and simple weighted polygons, *Proceedings of the Seventh Canadian Conference on Computational Geometry*, Québec, 1995, 175–179.

[Dijk59] Dijkstra, E. W., A note on two problems in connection with graphs, *Numerische Mathematik*, Vol. 1, 1959, 269–271.

[ElGi81] El Gindy, Hossam A., and David, Avis, A linear algorithm for computing the visibility polygon from a point, *Journal of Algorithms*, Vol. 2, 1981, 186–197.

[Espi91] Espie, Marc, Computing efficiently shortest paths for degenerate metrics, *Proceedings of the Third Canadian Conference on Computational Geometry*, Vancouver, 1991, 149–152.

[Guib87] Guibas, L. J., J. Hershberger, 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.

[Hers87] Hershberger, John E., Efficient algorithms for shortest path and visibility problems, Ph.D. thesis, Stanford University, 1987.

[Hers89] Hershberger, John E., An optimal visibility graph algorithm for triangulated simple polygons, *Algorithmica*, Vol. 4, 1989, 141-155.

[Kirk83] Kirkpatrick, D. G., Optimal search in planar subdivisions, *SIAM Journal of Computing*, Vol. 12, 1983, 28-35.

[Lee83] Lee, D. T., Visibility of a simple polygon, *Computer Vision, Graphics and Image Processing*, Vol. 22, 1983, 207-221.

[Lee84] Lee, D. T., and F. P. Preparata, Euclidian shortest paths in the presence of rectilinear barriers, *Networks*, Vol. 14, 1984, 393-410.

[Lee90] Lee, D. T., T. H. Chen, and C. D. Yang, Shortest rectilinear paths among weighted obstacles, *Proceedings of the Sixth Annual ACM Symposium on Computational Geometry*, Berkeley, 1990, 301-310.

[Lee86] Lee, D. T. and A. K. Lin, Computing the visibility polygon from an edge, *Computer Vision, Graphics and Image Processing*, Vol. 34, 1986, 1-19.

[Lee77] Lee, D. T. and F. P. Preparata, Location of a point in a planar subdivision and its applications, *SIAM Journal on Computing*, Vol. 6, 1977, 594-606.

[McDo92] McDonald, Kenneth M. and Joseph G. Peters, Smallest paths in simple rectilinear polygons, *IEEE Transactions on Computer-Aided Design*, Vol. 11, 1992, 864-875.

[Meis75] Meister, G.H., Polygons have ears, *American Mathematical Monthly*, Vol. 82, 1975, 648–651.

[Mitc87a] Mitchell, Joseph S. B., David M. Mount, and Christos H. Papadimitriou, The discrete geodesic problem, *SIAM Journal of Computing*, Vol. 16, 1987, 647–668.

[Mitc87b] Mitchell, Joseph S. B., and Christos H. Papadimitriou, The weighted region problem (extended abstract), *Proceedings of the Third Annual 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, 1991, 18–73.

[ORou87] O'Rourke, Joseph, *Art Gallery Theorems and Algorithms*, Oxford University Press, New York, 1987.

[Prep85] Preparata, Franco P. and Michael I. Shamos, *Computational Geometry: an Introduction*, Springer-Verlag, New York, 1985.

[Sack84] Sack, Jörg R., Rectilinear computational geometry, Ph.D. thesis, McGill University, 1984.

[Sack90] Sack, Jörg R. and Subhash Suri, An optimal algorithm for detecting weak visibility of a polygon, *IEEE Transactions on Computers*, Vol. 39, 1990, 1213–1219.

[Schu93] Schuierer, Sven, Rectilinear path queries in a simple rectilinear polygon, *Tenth Annual Symposium on Theoretical Aspects of Computer Science*, 1993, 282–293.

[Stor94] Storer, J. A. and J. H. Reif, Shortest path in the plane with polygonal obstacles, *Journal of the ACM*, Vol. 41, 1994, 982–1012.

[Suri86] Suri, Subhash, A linear time algorithm for minimum link path inside a simple polygon, *Computer Vision, Graphics and Image Processing*, Vol. 35, 1986, 99–110.

[Welz85] Welzl, E. Constructing the visibility graph for n line segments in $O(n^2)$ time, *Information Processing Letters*, Vol. 20, 1985, 167–171.

[Wynt93] Wynters, Erik L. and Joseph S. B. Mitchell, Shortest paths for a two-robot rendez-vous, *Proceedings of the Fifth Canadian Conference on Computational Geometry*, Waterloo, 1993, 216–221.