SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NEW YORK 14853

TECHNICAL REPORT NO. 885

January 1990

THE WEIGHTED REGION PROBLEM: FINDING SHORTEST PATHS THROUGH A WEIGHTED PLANAR SUBDIVISION

By

Joseph S. B. Mitchell
Christos H. Papadimitriou

To appear: *Journal of the ACM*

The original version of this paper appeared as a Technical Report, Department of Operations Research, Stanford University, October, 1985. An extended abstract of the earlier version of this paper appears in the *Proceedings of the Third Annual ACM Symposium on Computational Geometry*, Waterloo, Ontario, June 8-10, 1987.

THE WEIGHTED REGION PROBLEM: Finding Shortest Paths Through a Weighted Planar Subdivision†

Joseph S. B. Mitchell[1]
School of Operations Research and Industrial Engineering
Cornell University
Ithaca, NY 14853

Christos H. Papadimitriou[2]
Computer Science Division
University of Califormia at San Diego
La Jolla, CA 92093

Abstract

We consider the problem of determining shortest paths through a weighted planar polygonal subdivision with n vertices. Distances are measured according to a weighted Euclidean metric: The length of a path is defined to be the weighted sum of (Euclidean) lengths of the subpaths within each region. We present an algorithm that constructs a (restricted) "shortest path map" with respect to a given source point. The output is a partitioning of each edge of the subdivision into intervals of ϵ-optimality, allowing an ϵ-optimal path to be traced from the source to any query point along any edge. The algorithm runs in worst-case time $O(ES)$ and requires $O(E)$ space, where E is the number of "events" in our algorithm and S is the time it takes to run a numerical search procedure. In the worst case, E is bounded above by $O(n^4)$ (and we give an $\Omega(n^4)$ lower bound), but it is likely that E will be much smaller in practice. We also show that S is bounded by $O(n^4L)$, where L is the precision of the problem instance (including the number of bits in the user-specified tolerance ϵ). Again, the value of S should be much smaller in practice. The algorithm applies the "continuous Dijkstra" paradigm and exploits the fact that shortest paths obey Snell's Law of Refraction at region boundaries, a local optimality property of shortest paths that is well-known from the analogous optics model. The algorithm generalizes to the multi-source case to compute Voronoi diagrams.

Key Words: shortest paths, terrain navigation, Voronoi diagrams, continuous Dijkstra algorithm, Snell's Law of Refraction, computational geometry

† A preliminary version of this paper appeared in the Proceedings of the Third Annual ACM Symposium on Computational Geometry, Waterloo, Ontario, June 8-10, 1987.

[1] Partially supported by NSF Grants IRI-8710858 and ECSE-8857642 and by a grant from Hughes Research Laboratories. Much of this research was conducted while the first author was with the Hughes Artificial Intelligence Center, Hughes Aircraft Company and supported by a Hughes Doctoral Fellowship at Stanford University.

[2] This research was conducted while the second author was at Stanford University.

1. Introduction

Algorithmic motion-planning for robotics involves several interesting geometric variants of the shortest path problem. An important such problem is that of determining the shortest path between two points in the plane in the presence of disjoint simple polygonal obstacles. This problem can be solved in $O(n^2 \log n)$ time ([Le, Mi1, SS]) by standard *visibility graph* techniques, where n is the total number of vertices describing the polygonal obstacles. Recent improvements in bounding the complexity of the shortest path problem include an $O(n^2)$ bound ([AAGHI, We]), an $O(nk + n \log n)$ bound [RS] (where k is the number of obstacles), various *output-sensitive* bounds ([GM, KaMa, Mi3]), and optimal ($\Theta(n \log n)$) bounds when the optimal paths possess certain monotonicity properties ([LP, Mi1]).

In this paper, we examine a new problem, a generalization of the two-dimensional shortest path problem with obstacles, in which we now assume that the plane is subdivided into polygonal regions each of which has an associated weight α specifying the "cost per unit distance" of traveling in that region. Our objective is then to find a path in the plane that minimizes total cost according to a *weighted* Euclidean metric. The shortest path problem with obstacles is easily seen to be a special case of the weighted region problem in which the weights are either 1 or $+\infty$ depending on whether a region is "free space" or obstacle, respectively. We refer to our generalized shortest path problem as the "weighted region problem".

Our main result in this paper is a polynomial-time solution to the general Euclidean weighted region problem. The input to our algorithm is a polygonal subdivision of size n, with integer weights ($\in [0, +\infty]$) assigned to each face and each edge. We are also given a source point s and an error tolerance ϵ. The output of our algorithm is a labeling of each vertex with the length of an ϵ-optimal path from s to it, and a partitioning of each edge of the subdivision according to "intervals of optimality", from which we can construct an ϵ-optimal path to any query point on any edge.

The running time of our algorithm in the worst case is $O(ES)$, where E is the number of "events" in our algorithm and S is the complexity of solving a certain numerical search problem. The size of E is related to the size of a "shortest path map". In the worst case, we show that $E = \Omega(n^4)$, but we conjecture that E will be much smaller in practice. We prove a worst-case upper bound of $E = O(n^4)$. (In an earlier version of this paper, [Mi1,MP], we erroneously claimed a bound of $O(n^3)$ on the size of E. Here, we correct our mistake and show that in fact E has a lower bound of $\Omega(n^4)$ in the worst case, indicating that our upper bound is best possible.) We give a numerical search procedure that shows that S can be bounded above by $O(n^4 L)$, where L is related to the number of bits necessary to encode the problem instance. In particular, $L = O(\log(nNW/\epsilon w))$, where N is the maximum integer coordinate of any vertex of the subdivision, W (resp., w) is the maximum finite (resp., minimum nonzero) integer weight assigned to faces of $\mathcal{S}$, and $\epsilon > 0$ is a user-specified error tolerance. Our upper bound on S is very generous; in practice, a fast and simple numerical algorithm may be used. Thus, the worst-case running time of our algorithm is $O(n^8 L)$, but since the algorithm is, in a sense, output-sensitive, it is likely to perform well in practice. Our algorithm generalizes immediately to the case of many source points to yield a Voronoi diagram structure within the same time bounds.

Our reasons for considering the weighted region problem are two-fold: First, we wish to generalize the theory of shortest paths to include this case. Second, we are motivated by an application in the field of terrain navigation for an autonomous vehicle. Imagine a mobile robot (considered to be a point) whose objective is to use a given terrain map to plan an optimal route through varied terrain from a source to a destination. The terrain map might look like the diagram in Figure 1.1. Here, "optimal" may be taken to mean minimal time. The robot can move at different speeds through different types of terrain (e.g., grassland, brushland, forest, blacktop, water, etc.). The map segments the terrain into regions according to the ground cover (or some traversability index). These regions may be modeled as polygonal patches. There may be roads given on the map as well. Roads can be modeled as very skinny regions or as linear features (with an assigned weight that is presumably much less than the surrounding regions). This path planning problem is solvable by our weighted region algorithm.

As a further motivation for the weighted region problem, note that the problem we are trying to solve is equivalent to a familiar problem in the calculus of variations. Define a piecewise-constant "cost" function $c(\cdot)$ from the plane into the real numbers. We wish to find a simple path, $\mathcal{P}$, such that the line integral

$$\int_{\mathcal{P}} c(u(t))\sqrt{1 + u'^2(t)}dt$$

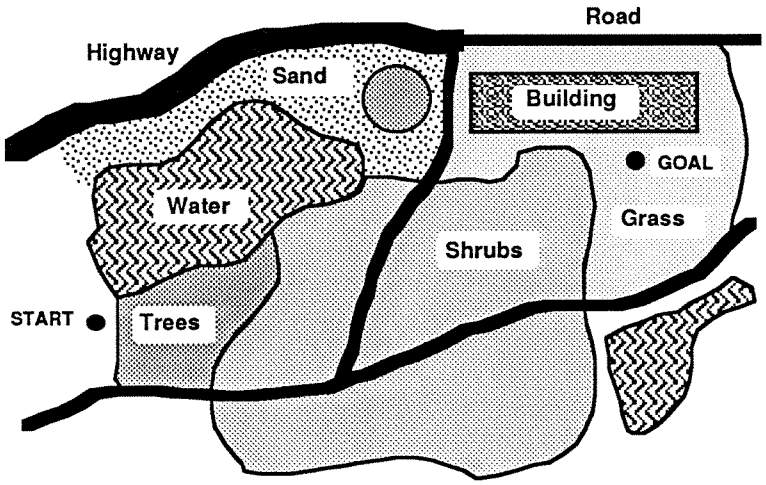

Figure 1.1. A map of varied terrain.

is minimal. So the weighted region problem is basically trying to solve a calculus of variations problem in which the cost function is specified in some discrete manner (here, piecewise constant; in the conclusion we discuss extensions to other cases), and we are measuring the performance of our algorithm as a function of the complexity of the discrete representation.

The approach to the terrain navigation problem taken by many other researchers has been to discretize the problem by laying down a grid on top of the surface ([Jo, KeMi, MPK, QFP], plus many others). This produces a grid graph (with either 4- or 8-connectivity depending on whether or not we count diagonal neighbors) of pixels each of which is a small square with sides equal in length to the fineness of the grid. Costs may be assigned in the natural way to arcs connecting adjacent pixels of the grid. Then, Dijkstra's algorithm [Di] may be used to compute minimum cost paths in the grid graph. The problem with this approach is that it may require extremely fine grids to capture the content of a simple map, and it creates a *digitization bias* because of the metrication error imposed by confining movements to 4 or 8 orientations. See [KeMi, MPK] for a more detailed discussion of digitization bias and the various possible remedies for it. See [QFP] for a technique that uses grid graphs of multiple resolutions to design a hierarchical algorithm.

Another approach to the weighted region problem is to build a region graph in which nodes correspond to regions and arcs correspond to edges. Without loss of generality, we can assume that regions are convex, since we can decompose simple polygons into convex subregions. (Note that the grid graph approach is really a case in which the regions are all squares of the same size.) Then we can think of placing a node at the "center" (e.g., center of mass) of a region. Two nodes are joined by an edge if the corresponding regions are adjacent. We then assign costs to arcs according to the weighted distance between adjacent nodes. Searching this graph for shortest paths yields a "region path" from the source to the destination, giving a sequence of regions through which a "good" path should pass. We could then do some post-processing (e.g., using the local optimality criterion of Snell's Law of Refraction from optics, which we will be discussing later) to make the path locally optimal at region boundaries. The problem with this approach is that it can produce paths that are not guaranteed to be optimal, or even close to optimal, for the optimal path need not have any relationship to the shortest region-path. [Mi4] contains a brief discussion of this method.

Instead of applying heuristics to solve the problem, our goal in this paper is to compute paths that are *guaranteed* to be optimal (within a user-specified error percentage ϵ). Our approach is to apply the *continuous Dijkstra* technique, which was originally developed in [Mi1,MMP] as a means of solving the Discrete Geodesic Problem (DGP): Find a shortest path between two points on a polyhedral surface, subject to the constraint that the path remain on the surface. While there are many similarities between the algorithm we give here and that of [MMP], there are many significant differences due to the complications involved in our problem. Our presentation will attempt to make clear where the details of our weighted region algorithm differ significantly from the algorithm of [MMP] for discrete geodesic paths on a surface.

2. Definition of the Problem

We are given a straight-line planar (polygonal) subdivision, $\mathcal{S}$, specified by a set of faces, edges, and vertices, with each edge occurring in two faces and two faces intersecting either at a common edge, a vertex, or not at all. We consider faces to be *closed* polygons (they include their boundaries) and edges to be *closed* line segments (they include their endpoints, which are vertices). The edge e shared by faces f and f' and *oriented* so that f is on the *right* will be denoted $e = \cap(f, f')$; when we wish to speak of an *undirected* edge e, we will write $e = f \cap f'$, without regard for the order of f and f'. We use $int(\cdot)$ to denote relative interior of an edge or face.

We are also given two special points s and t (the *source* and *destination*, respectively). We can assume without loss of generality that all faces are triangles and that s and t are vertices of the triangulation. We assume that $\mathcal{S}$ is stored in a data structure that allows the obvious simple operations (such as detecting the edges incident to a vertex, in order about the vertex, and detecting the faces incident on an edge). See [GS] for an example of such a data structure (the *quad-edge* data structure).

We concentrate here on the case of *bounded* subdivisions (a finite number of bounded regions); however, the generalization of the algorithm to subdivisions with a finite number of (possibly) unbounded regions is straightforward. We assume that $\mathcal{S}$ has n edges and, hence, $O(n)$ triangular faces and $O(n)$ vertices. Our complexity measures will be written in terms of n.

We define a notion of length according to a *weighted Euclidean metric*. Each face f has associated with it a *weight* $\alpha_f \in [0, +\infty]$ that specifies the *cost per unit distance* of traveling interior to face f. Similarly, each edge e has associated with it a weight $\alpha_e \in [0, +\infty]$. The weighted length of a line segment joining any two points x and y on edge e is simply the product $\alpha_e|xy|$, where $|xy|$ is the usual Euclidean distance between x and y. Likewise, the weighted length of a line segment joining any two points x and y on face f (but not both on the same edge of f) is the product $\alpha_f|xy|$. The weighted length of a path through the subdivision is then the sum of the weighted lengths of its subpaths through each face and along each edge.

If e is an edge shared by faces f and f', then we usually will have $\alpha_e = \min\{\alpha_f, \alpha_{f'}\}$. This indicates that when one travels on the boundary between f and f', it is as if one is traveling "just inside" the cheaper of the two regions. We may have $\alpha_e < \min\{\alpha_f, \alpha_{f'}\}$, in which case it is actually *cheaper* to travel along the boundary than interior to either of the two adjacent regions. This is a method of modeling a linear feature, such as a road running through grassland (which is passable, but not as easily traversed as the road). It never makes sense that $\min\{\alpha_f, \alpha_{f'}\} < \alpha_e < +\infty$, for then a path can be charged the weight $\min\{\alpha_f, \alpha_{f'}\}$ by traveling arbitrarily close to the edge e (but not actually on it).

The interpretation of an edge of weight $+\infty$ is that the edge should be a barrier. Hence, if $\alpha_e = +\infty$, then paths will not be allowed to *cross* edge e (this is a method of modeling a linear feature such as a fence that is not crossable); however, we do allow a path to travel *along* the edge e, "just inside" one of the (two) faces bounding e, at a cost per unit distance of α_f or $\alpha_{f'}$ (depending on which "side" of e the path lies). (One could think of a path crossing an edge e with $\alpha_e = \infty$ as having a cost made up of a length of 0 times a cost of $+\infty$, which we will resolve to mean a cost of $+\infty$.)

A vertex $v \in f \cap f'$ is *blocked between faces* f and f' if there exist infinite-weight edges e and e' incident on v such that the chain ee' "separates" f from f', meaning that the elements e, e', f, and f' appear in the order e, f, e', f' or e, f', e', f about vertex v. The interpretation is that paths cannot go from f to f' through the barrier (ee') at the vertex $v = e \cap e'$. We similarly define the notion of a vertex that is blocked between edges or that is blocked between an edge and a face.

We assume that all parameters of the problem are specified by integers. In particular, all vertices v have nonnegative integer coordinates. We let N be the maximum integer value for a coordinate. We also assume that all weights are nonnegative integers (or $+\infty$), and we let W be the maximum finite weight and w be the minimum non-zero weight (so $\alpha \in \{0, w, w+1, \ldots, W, +\infty\}$). Note that making these assumptions provides no loss of generality from the case in which coordinates and weights are allowed to be any rational numbers, for we could always rescale and shift coordinates and rescale the weights to be integers. The importance of these assumptions will be seen in Section 8, where we discuss the numerical issues involved in solving a certain subproblem.

We are asked to find the minimal-length (in the weighted sense) path from the source to the destination. We will refer to such a path as a *shortest path*, with the understanding that "shortest" is taken according to the weighted Euclidean metric. Our problem can now be stated formally.

(WRP) WEIGHTED REGION PROBLEM

Instance: *Two points s and t in the plane, a finite triangulation $\mathcal{S}$ in the plane, an assignment of weights α_e and α_f to edges and faces, and an error tolerance $\epsilon > 0$.*

Question: *Find a path within $\mathcal{S}$ from s to t whose weighted Euclidean length is ϵ-optimal among all paths from s to t.*

The problem we actually solve is the query form of the WRP, and from this solution we can solve the WRP within the same time bound.

(SSWRP) SINGLE-SOURCE WEIGHTED REGION PROBLEM

Instance: *One source point s in the plane, a finite triangulation $\mathcal{S}$ in the plane, an assignment of weights α_e and α_f to edges and faces, and an error tolerance $\epsilon > 0$.*

Question: *Build a structure (a "Shortest Path Map") that allows one to compute an ϵ-optimal path (in the weighted Euclidean metric) from s to any query point t.*

Two faces f and f' are said to be *edge-adjacent* if they share a common edge e. A *sequence of edge-adjacent faces* is a list, $\mathcal{F} = (f_1, f_2, \ldots, f_{k+1})$, of two or more faces such that face f_i is edge-adjacent to face f_{i+1} (sharing common edge e_i). We then refer to the list, $\mathcal{E} = (e_1, e_2, \ldots, e_k)$, of edges $e_i = f_i \cap f_{i+1}$ as an *edge sequence* and to the vertex of face f_1 that does not belong to e_1 as the *root*, $r(\mathcal{E})$, of $\mathcal{E}$. If no face (resp., edge) appears more than once in $\mathcal{F}$ (resp., $\mathcal{E}$), then we call the sequence *simple*.

A *path* is a continuous image of $[0, 1]$ in the plane. We will be interested in *piecewise-linear* paths that are *simple* (not self-intersecting). We can specify a piecewise-linear path by giving a list of points, $p = (x_1, \ldots, x_k)$, such that the path consists of line segments $\overline{x_i x_{i+1}}$, $1 \leq i \leq k-1$. A *geodesic path* is a path that is locally optimal and cannot, therefore, be shortened by slight perturbations. An *optimal* path is a geodesic path that is *globally* optimal. We will usually write $p(x)$ to indicate an optimal path from the source (s) to x, and we will write $d(x)$ to indicate the length of $p(x)$ (also called the *depth* of point x).

We say that path p *passes through the interior of edge* $e = f \cap f'$ at point y if there exist points $x \in int(f)$ and $x' \in int(f')$ such that $\overline{xy} \subset p$ and $\overline{yx'} \subset p$. We say that path p *connects* edge sequence $\mathcal{E} = (e_1, e_2, \ldots, e_k)$ if p consists of segments that join interior points of $e_1, e_2, \ldots, e_k$ (that is, $p = (x_1, x_2, \ldots, x_k)$, where point x_i is interior to edge e_i and $\overline{x_i x_{i+1}} \subset f_{i+1}$). (Note that such a path cannot pass through the endpoint of any e_i, for otherwise two distinct consecutive edges of $\mathcal{E}$ would have to be colinear.) Path p *goes through* edge sequence $\mathcal{E}$ if it has a subpath that connects $\mathcal{E}$ (note that it may have many such subpaths, and we do not require that $\mathcal{E}$ be simple).

3. Characterization of Geodesics and Optimal Paths

With our assumption that each region has a *uniform* weight, we are able to conclude that geodesic paths will always be piecewise linear. (The case of *non*uniformly weighted regions will be mentioned briefly in the conclusion.)

Lemma 3.1 *Geodesic paths are piecewise linear, except within regions of weight zero (where subpaths may be arbitrary). The intersection of a geodesic path with the interior of a face with $\alpha_f > 0$ is a (possibly empty) set of line segments.*

Proof: Simply note that if a subpath is contained in the interior of a face, then it must be a straight line segment if the weight of the face is positive. ∎

While geodesic paths are not necessarily simple paths, optimal paths must be simple, with the following exception: within the interior of a face with *zero* weight, an optimal subpath can do anything whatsoever, since travel is "free". It is , however, always possible to replace any nonsimple subpaths in zero-weighted regions with subpaths that are both piecewise-linear and simple. (For example, a path along the boundary

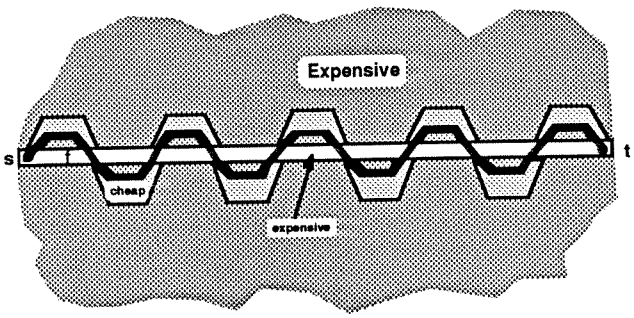

Figure 3.1. A shortest path crossing a face many times.

of the zero-weighted region will suffice.) We can, therefore, limit our attention to geodesic paths that are simple and piecewise-linear.

In the DGP, optimal paths also have the property that they do not go through any face more than once (Lemma 3.2 of [MMP]). This, however, is not necessarily true in the WRP, for an optimal path may, in fact, intersect a face in $O(n)$ line segments. See Figure 3.1. Also, in the DGP, if v is a vertex, then an optimal path from s to v can pass through the interior of at most one of the faces containing v (Lemma 4.1 of [MMP]). However, in the WRP, this no longer holds. See Figure 3.2. The basic problem is that in the WRP the shortest path between two points x and x' interior to a face f is not necessarily the line segment connecting them. See Figure 3.3. Note that geodesic paths (in both the WRP and the DGP) can intersect a single face in many line segments.

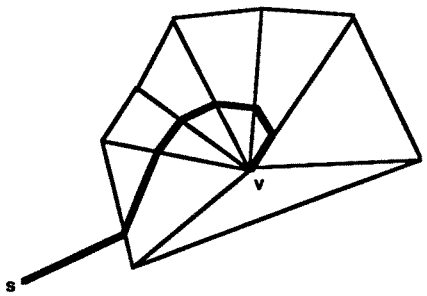

Figure 3.2. An optimal path to v passing through many faces adjacent to v.

Assume for the moment that $\alpha_e = \min\{\alpha_f, \alpha_{f'}\}$ for edge $e = f \cap f'$. Then, there is a simple property of geodesics (observed by [Ly], as well as by many others) that says that a geodesic path that passes through the interior of e must "bend" at the edge according to Snell's Law of Refraction for light. This is the local optimality criterion for the weighted region problem and is analogous to the fact that geodesic paths in three dimensions unfold into straight lines.

Snell's Law of Refraction. *The path of a light ray passing through a boundary e between regions f and f' with indices of refraction α_f and $\alpha_{f'}$ obeys the relationship that $\alpha_f \sin\theta = \alpha_{f'} \sin\theta'$, where θ and θ' are the angles of incidence and refraction (respectively).*

The *angle of incidence*, θ, is defined to be the counterclockwise angle between the incoming ray and the normal to the region boundary. (We consider the normal to be a vector perpendicular to the boundary, and it is oriented from the incoming region toward the outgoing region.) The *angle of refraction*, θ', is defined as the counterclockwise angle between the outgoing ray and the normal. Refer to Figure 3.4.

Remark: There is another physical analogy that can be made to illustrate the local optimality criterion. Consider the system of weights, pulleys, and ropes shown in Figure 3.5.

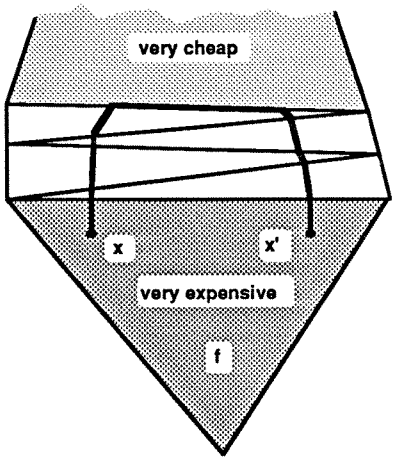

Figure 3.3. Shortest path from $x \in f$ to $x' \in f$ is not given by segment $\overline{xx'}$.

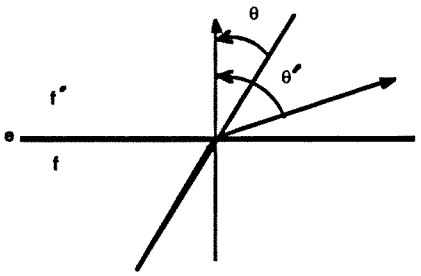

Figure 3.4. A light ray crossing a boundary.

> The ropes are tied to a ring that slides (without friction) on the wire along edge e. The tensions in the ropes are α_f and $\alpha_{f'}$, the weights of the masses being suspended on either rope. Then the shortest path from the center of one pulley to the center of the other pulley is given by the path of the ropes when in equilibrium. (We assume that the radius of the pulleys is negligible.)

The fact that light obeys Snell's Law comes from the fact that light seeks the path of minimum time (this is *Fermat's Principle*). The index of refraction for a region is proportional to the speed with which light can travel through that region. Hence, the shortest paths in our weighted region problem must also obey Snell's Law. This can be shown formally.

Lemma 3.2 *Assume that* $\alpha_e = \min\{\alpha_f, \alpha_{f'}\}$ *and that* $\alpha_f, \alpha_{f'} < +\infty$. *If* p *is a geodesic path that passes through the interior of edge* e, *then* p *obeys Snell's Law at edge* e.

Proof: The proof is a straightforward single-variable calculus problem. Refer to Figure 3.6. The function we wish to minimize is

$$F(x) = \alpha_f \sqrt{x^2 + y_0^2} + \alpha_{f'} \sqrt{(x_1 - x)^2 + y_1^2}.$$

Clearly, F attains its minimum on $[0, x_1]$, and it is easy to check that F is a strictly convex function on $[0, x_1]$. The condition that $F'(x^*) = 0$ becomes simply

$$\alpha_f \frac{x^*}{\sqrt{x^{*2} + y_0^2}} = \alpha_{f'} \frac{(x_1 - x^*)}{\sqrt{(x_1 - x^*)^2 + y_1^2}},$$

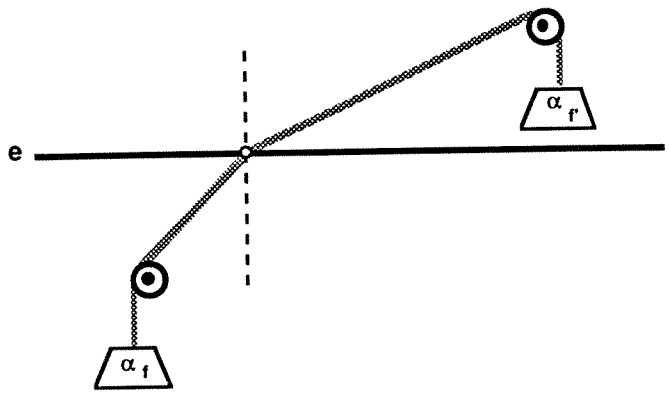

Figure 3.5. Physical model of local optimality criterion.

which implies that

$$\alpha_f \sin\theta = \alpha_{f'} \sin\theta'.$$

It is easy to check the boundary points to see that $x^* = 0$ (resp., $x^* = x_1$) if and only if $\alpha_{f'} = 0$ (resp., $\alpha_f = 0$), showing that the above relationship still holds in these special cases. ∎

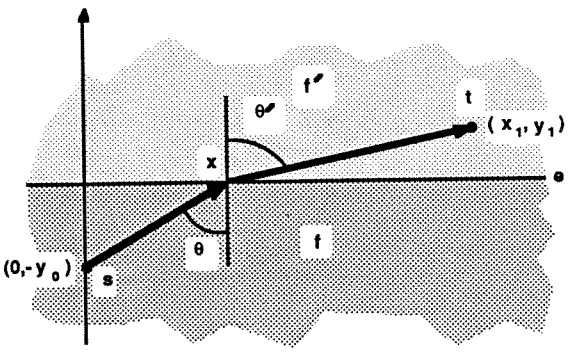

Figure 3.6. Snell's Law of Refraction: Local optimality criterion.

When applying the relationship specified in Snell's Law, we must be careful that it is well-defined. Without loss of generality, assume that $\alpha_f \geq \alpha_{f'} > 0$. Then, we must assure that

$$-1 \leq \sin\theta' = \frac{\alpha_f}{\alpha_{f'}} \sin\theta \leq 1.$$

Let $e = \cap(f, f')$ be the oriented boundary between regions f and f'. The angle, $\theta_c(e) = \theta_c(f, f')$, at which

$$\frac{\alpha_f}{\alpha_{f'}} \sin[\theta_c(f, f')] = 1$$

is called the *critical angle* defined by e. Critical angles come into play whenever we are considering geodesic paths going *from* one region (f) *to* a less expensive region (f'). If $\alpha_f > 0$ and $\alpha_{f'} \geq 0$, then the critical angle is defined and is given by $\theta_c(f, f') = \sin^{-1}(\alpha_{f'}/\alpha_f)$, *provided* that $\alpha_f \geq \alpha_{f'}$. If $\alpha_f = \alpha_{f'} = 0$, then a geodesic path can "bend" at e in any way whatsoever, since motion in regions f and f' is "free". If

$\alpha_{f'} = +\infty$ or if $\alpha_f = 0$, then there is no need to define $\theta_c(f, f')$ (we could define it to be $\pi/2$). (If $\alpha_f = +\infty$, then our path p had no business being in region f in the first place!)

A ray of light that travels through f to strike e at the critical angle will (theoretically) travel along e rather than enter into the interior of f'. We can think of it as if the light ray wishes to be "just inside" the "cheap" region on the other side of e. A light ray that hits e at an angle of incidence *greater* (in absolute value) than θ_c will be totally *reflected* from the point at which it hit the boundary. The light ray would leave the boundary at an angle of reflection equal to the angle of incidence.

In our problem, there can be no such reflections for geodesic paths. A geodesic path will never be incident to an edge at an angle greater (in absolute value) than the critical angle. The only form of "reflection" of a geodesic path is that which takes place at the critical angle in the following way: A path is incident (from face f) on edge e at the critical angle $\theta = \theta_c$ at point $y \in int(e)$, it then travels along edge e for some positive distance (in the direction of its orientation), and then exits edge e *back into face* f at point $y' \in int(e)$, leaving the edge at an angle $\theta' = \theta_c$. See Figure 3.7. We will then say that the path is *critically reflected* by edge e and that segment $\overline{yy'}$ is a *critical segment* of p along e. The phenomenon of critical reflection is not just an unlikely event of degeneracy. It is commonly the case that the shortest path between two points consists of a single critical reflection along an edge (for example, the shortest path from s to t in Figure 3.7 is given by the path of critical reflection).

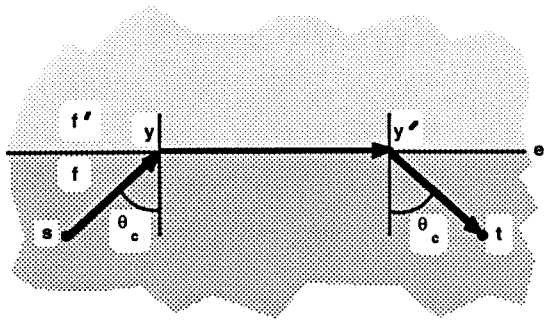

Figure 3.7. A geodesic path that is critically reflected.

Summarizing the behavior of geodesics with respect to critical angles, we get the following lemma:

Lemma 3.3 *Assume that* $e = \cap(f, f')$ *and* $\alpha_e = \min\{\alpha_f, \alpha_{f'}\} = \alpha_{f'}$. *If a geodesic path* p *passes through the interior of edge* e, *then the angle of incidence is less (in absolute value) than the critical angle* $\theta_c = \theta_c(f, f')$. *If* p *contains the subpath* (x, y, x'), *where* $x \in int(f)$, $y \in int(e)$, *and* $x' \in f \cap f'$ *(or* $x' \in int(f)$, $y \in int(e)$, *and* $x \in f \cap f'$*), then the angle (measured counterclockwise) from* $\overrightarrow{yx}$ *to* $\overrightarrow{yx'}$ *is* $\angle xyx' = \theta_c + \pi/2$. *Thus, if* p *travels along some part,* $\overline{yy'}$, *of* e *(where* $y, y' \in int(e)$*), then the angle of incidence at point* y *equals the critical angle* θ_c, *and the angle of exit at* y' *must also equal* θ_c.

Proof: The proof follows immediately from the proof of Lemma 3.2. Simply put $y_1 = 0$, so that the probem is to minimize the distance from point $(0, -y_0)$ to point $(x_1, 0)$ on the boundary e. Then Snell's Law gives $\theta = \theta_c(f, f')$. This also implies that a path will never hit a boundary at an angle greater (in absolute value) than θ_c. ∎

If $\alpha_e > \min\{\alpha_f, \alpha_{f'}\}$, then a geodesic path will never travel along edge e, since it would be better to travel "just inside" either region f or region f' (whichever is cheaper). A geodesic path that crosses through the interior of edge e behaves just as if $\alpha_e = \min\{\alpha_f, \alpha_{f'}\}$. In case $\alpha_e = +\infty$, we have assumed that e behaves like an obstacles, so that a geodesic path cannot cross through edge e; thus, $p \cap e$ must be either e, an endpoint of e, or the empty set.

In case $\alpha_e < \min\{\alpha_f, \alpha_{f'}\}$, the local optimality condition takes on a slightly different form. How does a geodesic path cross a boundary whose cost is *less* than that of the regions on either side of it?

It may now be possible for the geodesic path to hit edge e at the *incoming* critical angle $\theta_c(f,e) = \sin^{-1}(\alpha_e/\alpha_f)$ at point $x \in int(e)$, then travel along edge e for some distance, then leave edge e into f' at the *outgoing* critical angle $\theta_c(f',e) = \sin^{-1}(\alpha_e/\alpha_{f'})$ at a point $x' \in int(e)$. See Figure 3.8. We will then say that the path has *critically used* edge e along the critical segment $\overline{xx'}$. If one thinks of the edge e as a "road" that is relatively "fast moving" in comparison with the regions on either side of it, then one can say that the path "hitched a ride" along the road before crossing it. It is also possible, of course, for the path to be critically reflected by e from either region f or region f'.

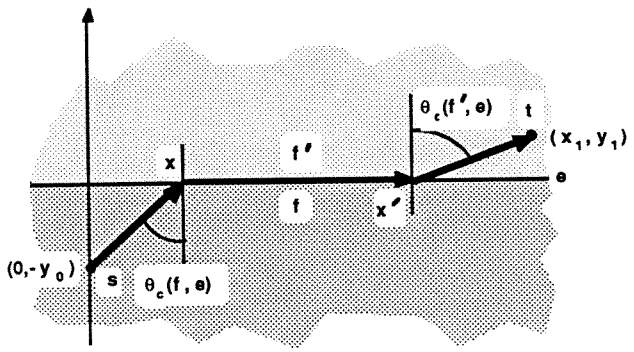

Figure 3.8. Shortest path "hitching a ride" on a road.

More formally, we can solve a simple two-variable minimization problem:

Lemma 3.4 *Assume that $\alpha_e < \min\{\alpha_f, \alpha_{f'}\}$. A geodesic path that crosses edge e will do so in one of two ways: It will either intersect edge e at one point of crossing and obey Snell's Law at that point, or it will hit edge e at the incoming critical angle $\theta_c(f,e) = \sin^{-1}(\alpha_e/\alpha_f)$, travel along the edge for some distance, and then exit the edge (to the other side) at an outgoing critical angle $\theta_c(f',e) = \sin^{-1}(\alpha_e/\alpha_{f'})$.*

Proof: The proof is a simple two-variable calculus problem. Refer to Figure 3.8. We wish to minimize the function

$$F(x,x') = \alpha_f\sqrt{x^2+y_0^2} + \alpha_e(x'-x) + \alpha_{f'}\sqrt{(x_1-x')^2+y_1^2}$$

subject to

$$x \le x'.$$

Clearly, F attains its minimum for values of x and x' that are nonnegative. Also, one can easily check that F is strictly convex. The Kuhn-Tucker conditions at point (x,x') give

$$\alpha_f \frac{x}{\sqrt{x^2+y_0^2}} = \alpha_e = \alpha_{f'}\frac{x_1-x'}{\sqrt{(x_1-x')^2+y_1^2}},$$

or, in terms of angles,

$$\alpha_f \sin\theta = \alpha_e = \alpha_{f'}\sin\theta',$$

if the Lagrange multiplier is zero. This simply says that the incoming angle must be the critical angle $\theta_c(f,e)$, and the outgoing angle must be the critical angle $\theta_c(f',e)$. The Lagrange multiplier will be zero if and only if

$$y_0 \tan\theta_c(f,e) + y_1 \tan\theta_c(f',e) > x_1,$$

that is, if and only if

$$y_0\frac{(\alpha_e/\alpha_f)}{\sqrt{1-(\alpha_e/\alpha_f)^2}} + y_1\frac{(\alpha_e/\alpha_{f'})}{\sqrt{1-(\alpha_e/\alpha_{f'})^2}} > x_1.$$

If the above inequality does not hold, then the Lagrange multiplier is *not* zero, and we get that $x = x'$ and

$$\alpha_f \sin\theta = \alpha_{f'}\sin\theta'.$$

In this case, then, the path crosses the edge at the single crossing point $z = x = x'$, and obeys Snell's Law while doing so. ∎

Let p be a geodesic path. In general, the set $p \cap e$ will be either empty, the whole edge $e = [v, v']$, or consist of a set of points and segments $\{z_1,\ldots,z_k\} \cup \{\overline{y_1y_1'},\ldots,\overline{y_jy_j'}\}$, where $v \leq z_1 < \ldots < z_k \leq v'$ and $v \leq y_1 < y_1' < \ldots < y_j < y_j' \leq v'$ (the order used here is the linear order along the line through the oriented edge e). The points z_i are called *crossing points* of edge e for path p, and the segments $\overline{y_iy_i'}$ are called *shared segments* for e and p. In summary, we can now fully characterize the behavior of geodesic paths at edges.

Proposition 3.5 (Local Optimality Criterion) *Let p be a geodesic path through a weighted finite triangulation $\mathcal{S}$. Consider any edge e of $\mathcal{S}$.*

(a). (Edge obstacles) *If $\alpha_e = +\infty$, then $p \cap e$ is either the entire edge e (meaning that p travels arbitrarily close to e), an endpoint of e, or the empty set;*

(b). (Crossing points) *If p passes through the interior of edge e at a crossing point $z \in int(e)$, then p obeys Snell's Law at z, namely, $\alpha_f \sin\theta = \alpha_{f'} \sin\theta'$, where θ is the angle of incidence from region f and θ' is the angle of refraction into region f'. If $\min\{\alpha_f, \alpha_{f'}\} \leq \alpha_e < +\infty$, then, without loss of generality, assume that $\alpha_{f'} \leq \alpha_f$. Then $|\theta| \leq \theta_c(f, f')$. If $0 \leq \alpha_e < \min\{\alpha_f, \alpha_{f'}\}$, then $|\theta| \leq \theta_c(f, e)$ and $|\theta'| \leq \theta_c(f', e)$.*

(c). (Shared segments) *Assume that p shares a segment $\overline{yy'}$ with edge e. If y is not a vertex, then the angle of incidence at y is critical: if p strikes y from the side of f (wlog), then $\alpha_f > \min\{\alpha_e, \alpha_{f'}\}$ and the angle of incidence is either $\theta_c(f, e)$ (if $\alpha_e \leq \alpha_{f'}$) or $\theta_c(f, f')$ (otherwise). Similarly, if y' is not a vertex, then the angle of exit at y' is critical.*

Proof: Apply Lemmas 3.2, 3.3, and 3.4. ∎

The local optimality criterion is sufficiently strong that it uniquely specifies a geodesic path that connects an edge sequence:

Lemma 3.6 *If p is a geodesic path through regions with $\alpha_i > 0$ from a point s to a point x that connects edge sequence $\mathcal{E} = (e_1, \ldots, e_k)$ with $e_i \neq e_{i+1}$ (so that there are no shared segments), then p is the unique geodesic path, and it obeys Snell's Law at each crossing point of each edge.*

Proof: The proof uses the following fact: *The function that gives the weighted length of the path that goes from point s to point t while connecting the edge sequence $\mathcal{E}$ is a strictly convex function of the coordinates of the crossing points at each edge.* (Assume that the weights are positive in the sequence of faces, and that s and t do not lie on the first and last edges of $\mathcal{E}$; otherwise, the function will still be convex, but possibly not strictly so.)

We begin with a proof of the above fact. The weighted length of the path from s to t that connects edge sequence $\mathcal{E} = (e_1, \ldots, e_k)$ is given by

$$g(\mathbf{u}) = g(u_1, \ldots, u_k) = \alpha_1|sU_1| + \sum_{1 \leq i \leq k-1} \alpha_{i+1}|U_iU_{i+1}| + \alpha_{k+1}|U_kt|,$$

where U_i is the point on line i at coordinate $u_i \in \Re^1$ on that line. We wish to show that g is strictly convex. It is easy to check that $|sU_1|$ is a convex function of u_1 (and, likewise, $|U_kt|$ is convex in u_k). One can also check that $|U_iU_{i+1}|$ is a convex function of the two (scalar) variables u_i and u_{i+1}. Also, these functions are strictly convex as long as $s \notin e_1$, $t \notin e_k$, and edges e_i and e_{i+1} are not parallel (which is indeed the case for edge sequences of the triangulation). Thus, since $\alpha_i > 0$, g is strictly convex.

Since function $g(\mathbf{u})$ is strictly convex, it possesses a unique global minimum, and any local minimum must be global. Since p is a local minimum, it is also the global minimum connecting the edge sequence, and we already know (Proposition 3.5) that it obeys Snell's Law at each crossing point. ∎

This lemma tells us that we can use Snell's Law to do *ray tracing* as follows. We begin with an angle ϕ and a point r. Follow a ray eminating from r at angle ϕ (where ϕ is measured counterclockwise from the positive x-axis) until it hits an edge e_1. Pass through edge e_1 while obeying Snell's Law, thereby obtaining a ray on the other side of e_1 that continues at some known angle. Continue tracing rays and passing through edges according to Snell's Law until either (1) the angle of incidence to an edge is greater than or equal (in absolute value) to the relevant critical angle, or (2) the ray hits a vertex. We refer to the (unique) path obtained by doing this ray-tracing as the *refraction path* from r at angle ϕ. A refraction path is specified uniquely either by giving r and ϕ, or by giving r, a point x through which the path is to pass, and the edge sequence $\mathcal{E}$ through which to go to hit x. In particular, we could specify r and a point $x \in e_1$, and

do ray tracing from r through x. This can be done in a straightforward manner using only the arithmetic operations ($+$, $-$, $*$, and $\div$) and the square root function, avoiding calculations of sines and inverse sines. We will describe later the function *Find-Point*, whose objective is to find the refraction path for a given r, x, and $\mathcal{E}$.

> *Remark:* It is tempting to think that we could just do a one-dimensional search in ϕ to determine a shortest path from s to t: simply find the values of ϕ that yield refraction paths from s that go through point t, and pick the one that gives the shortest path. The problem with this is that it ignores the effects of critical angles and paths through vertices. Basically, once a refraction path hits an edge at the critical angle, or hits a vertex, we no longer have complete information about where it goes next. It can travel along part of an edge and then get off at the critical angle, or it can pass through a vertex in many possible ways. The analogy with ray optics (which gave us our local optimality criterion) breaks down here, for in a real physical system we cannot apply ray optics to determine uniquely what happens to light rays at vertices and critical angles.

In the DGP, geodesic paths went through an alternating sequence of vertices and edge sequences (Lemma 3.5 of [MMP]). In the weighted problem, we have an additional complication with which to deal: critical points. For a geodesic path p, a point $y \in p \cap int(f \cap f')$ is called a *critical point of entry of path p from face f* if y is the closer (to the source, s) of the two endpoints of a shared segment $\overline{yy'}$, and p hits y from the side of f; that is, y is interior to the edge $e = f \cap f'$ and is such that the angle that path p makes at y is critical: $\angle xyx' = \theta_c + \pi/2$, where $\overline{xy}$ is the segment of p leading into y, $\overline{yy'} \subset e$ is the shared segment of p leading out of y along e, and θ_c is either $\theta_c(f, e)$ (if $\alpha_e \leq \alpha_{f'}$) or $\theta_c(f, f')$ (otherwise). Similarly, a point $y' \in p \cap int(f \cap f')$ is called a *critical point of exit of path p into face f* if y' is the further (from the source, s) of the two endpoints of a shared segment $\overline{yy'}$ and p goes from y' into face f.

Let v and v' be consecutive vertices encountered in the list of points describing a geodesic path p. Based on the characterization of geodesic paths we have so far (Proposition 3.5), we can make a simple observation about the structure of a geodesic path between v and v'. The subpath of p between vertices v and v' will, in general, be a (possibly empty) list of crossing points, followed by a critical point of entry to an edge and a critical point of exit from that edge, then another (possibly empty) list of crossing points, followed by another critical point of entry, etc. For example, the subpath between v and v' might be (v, $z_1 \in int(e_1)$, $z_2 \in int(e_2)$, $y_1 \in int(e_3)$, $y_1' \in int(e_3)$, $z_3 \in int(e_4)$, $y_2 \in int(e_5)$, $y_2' \in int(e_5)$, v'), where the z_i's are crossing points and the y_i's (y_i''s) are endpoints of shared segments. Refer to Figure 3.9. Alternatively, we could specify the subpath between v and v' by giving the list (v, $\mathcal{E}_1$, y_1, y_1', $\mathcal{E}_2$, y_2, y_2', $\mathcal{E}_3$, v'), where $\mathcal{E}_1 = (e_1, e_2)$, $\mathcal{E}_2 = (e_4, e_5)$, and $\mathcal{E}_3 = \emptyset$. (This follows from Lemma 3.6, since the edge sequence $\mathcal{E}_1$ and the points v and y_1 uniquely specify the geodesic path that connects $\mathcal{E}_1$.) Now note that the subpath between v and v' is, in fact, uniquely specified just by giving the list (v, f_1, f_2, f_3, e_3, f_4, e_5, v'). More generally, a geodesic path p from s to x may be specified uniquely by giving a list of vertices, edges, and faces whose relative interiors contain a portion of p. (Edges and faces *may* be repeated in this list.)

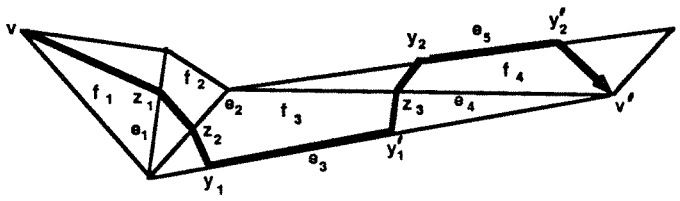

Figure 3.9. Specifying a path between v and v'.

We will see in Lemma 3.7 that the situation depicted in Figure 3.9 actually cannot occur, since we can show that between two consecutive vertices on a geodesic path there can be at most one shared segment. First, however, we need some more notation and facts.

Marching back from x along p, let r be the first vertex or critical point of entry we encounter. We call r the *root* of path p. (In the example of Figure 3.9, y_2 is the root of the path p.) We call the (possibly empty)

edge sequence crossed by p between r and x the *last* edge sequence of path p. If p is a geodesic path to point x that has last edge sequence $\mathcal{E}$ and root r, then we let $d_{r,\mathcal{E}}(x)$ be the weighted distance from r to x along p. Note that by the local optimality criterion, this distance is uniquely determined by r, x, and $\mathcal{E}$.

We will show that the alternating lists of crossing points, shared segments, and vertices have some special structure. We first establish the notation in Figure 3.10. Consider the refraction path from s through the edges $e_1 \in l_1$, $e_2 \in l_2$, ..., $e_7 \in l_7$. We consider each line to be oriented from left to right in the sense seen by an observer walking along the path. The coordinate, u_i, is the (unweighted) distance from the left endpoint of the edge e_i to the crossing point. The region between edge e_{i-1} and edge e_i has weight $\alpha_i \in (0, +\infty)$. We assume for simplicity that the weights on the edges themselves equal the minimum of the weights on either side; we need not assume this, but the other cases are messy and not especially enlightening. Angle ϕ_i is the counterclockwise angle from l_{i-1} to l_i. Angle θ_i is the angle of incidence of the path at edge e_i, while angle $\overline{\theta}_i$ is the angle of refraction at edge e_i. Of course, we must have $|\theta_i| \leq \theta_c(f_i, f_{i+1})$ if $\alpha_i > \alpha_{i+1}$. The length of edge e_i is called λ_i. Finally, L_i is the (unweighted) Euclidean length of the path between edge e_{i-1} and edge e_i.

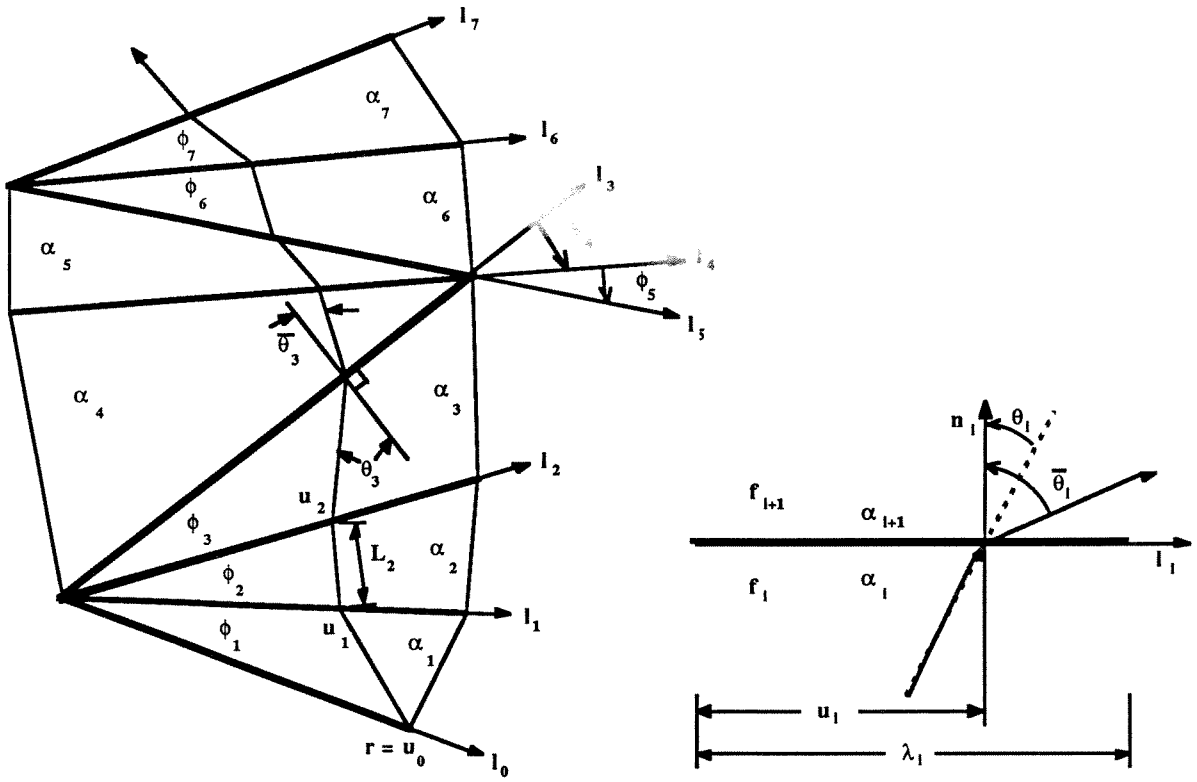

Figure 3.10. A refraction path passing through the triangulation.

Now we can make a few simple observations that will be of use later. We assume in the following claims that $0 < \alpha_i < +\infty$.

Claim 1. *Each coordinate u_i is a linear function of u_0 (we hold all other parameters fixed).*

Proof: First, u_0 is trivially a linear function of u_0. Now assume that u_k is a linear function of u_0. We

consider two cases. First, if $\phi_{k+1} > 0$, then we get, by the Law of Sines,

$$\begin{aligned} u_{k+1} &= \cos\overline{\theta}_k \frac{u_k}{\cos\theta_{k+1}} \\ &= \frac{\cos(\phi_{k+1} - \theta_{k+1})}{\cos\theta_{k+1}} u_k, \end{aligned}$$

which shows that u_{k+1} is also a linear function of u_0. If $\phi_{k+1} < 0$, then we have instead that

$$\begin{aligned} \lambda_{k+1} - u_{k+1} &= \cos\overline{\theta}_k \frac{\lambda_k - u_k}{\cos\theta_{k+1}} \\ &= \frac{\cos(\phi_{k+1} - \theta_{k+1})}{\cos\theta_{k+1}} (\lambda_k - u_k), \end{aligned}$$

showing that u_{k+1} is again a linear function of u_0. ∎

Claim 2. *Let $L = \sum_{i=1}^{K} \alpha_i L_i$ be the weighted path length from point r to point u_K. Then, L is a linear function of u_0.*

Proof: This follows simply from Claim 1 and the fact that, for each $i \geq 1$,

$$L_i = \sin\phi_i \frac{u_i}{\cos\overline{\theta}_{i-1}},$$

if $\phi_i > 0$, and

$$L_i = \sin\phi_i \frac{u_i - \lambda_i}{\cos\overline{\theta}_{i-1}},$$

if $\phi_i < 0$. ∎

The following two claims show that the refraction paths, parameterized by θ_1 (the first angle of incidence) or by u_1 (the first crossing point), sweep out a "wedge": as θ_1 is increased, the refraction path moves continuously to the "right".

Claim 3. *The angle of incidence on edge i, θ_i, is a continuous monotone increasing function of θ_1 (and hence of u_1), the angle of incidence on the first edge in the sequence, over any interval of θ_1 for which the refraction path exists through the given sequence of edges.*

Proof: Snell's Law at edge i yields

$$\alpha_i \sin\theta_i = \alpha_{i+1} \sin\overline{\theta}_i.$$

But we also have

$$\begin{aligned} \theta_{i+1} &= \phi_{i+1} + \overline{\theta}_i \\ &= \phi_{i+1} + \sin^{-1}\left[\frac{\alpha_i}{\alpha_{i+1}} \sin\theta_i\right], \end{aligned}$$

which shows that if θ_i is monotone increasing and continuous in θ_1, then so is θ_{i+1}. This follows since $\sin\theta$ (resp., $\sin^{-1} x$) is monotone increasing on the interval $\theta \in [-\pi/2, \pi/2]$ (resp., $x \in [-1, 1]$). By induction and the fact that θ_1 is trivially monotone in θ_1, we are done. ∎

Claim 4. *The crossing point on edge i, u_i, is a continuous monotone increasing function of θ_1 (and hence of u_1), over any interval of θ_1 for which the refraction path exists through the given sequence of edges.*

Proof: Clearly, u_1 is a monotone increasing function of θ_1. Now assume that u_i is increasing in θ_i. We consider two cases. First, if $\phi_{i+1} > 0$, then

$$\begin{aligned} u_{i+1} &= \frac{\cos(\theta_{i+1} - \phi_{i+1})}{\cos\theta_{i+1}} u_i \\ &= g(\theta_{i+1}) u_i, \end{aligned}$$

where $g(\cdot)$ is an increasing function, since

$$g'(\theta_{i+1}) = \frac{\sin\phi_{i+1}}{\cos^2\theta_{i+1}}.$$

But we already know that θ_{i+1} is increasing in θ_1, so u_{i+1} must also be increasing in θ_1. Now, if $\phi_{i+1} \leq 0$, then

$$\begin{aligned} u_{i+1} &= \lambda_{i+1} + \frac{\cos(\theta_{i+1} - \phi_{i+1})}{\cos\theta_{i+1}}(u_i - \lambda_i) \\ &= \lambda_{i+1} + g(\theta_{i+1})(u_i - \lambda_i), \end{aligned}$$

where, again $g(\cdot)$ is the increasing function defined above. Again, we have that u_{i+1} is increasing in θ_1. ∎

The next lemma is an important fact about critical points along geodesic paths. It will be needed later to prove facts about the complexity of our algorithm.

Lemma 3.7 *Let p be a geodesic path. Then either (1) between any two consecutive vertices on p, there is at most one critical point of entry to an edge e, and at most one critical point of exit from an edge e' (possibly equal to e); or (2) the path can be modified in such a way that case (1) holds without altering the length of the path.*

Proof: Assume the claim is false. Then, there is a geodesic path that has two shared segments (on edges e_1 and e_2) between which there are no vertices. Refer to Figure 3.11. The length of the subpath between critical point of entry y_1 and critical point of exit y_2' can be written as a linear function of the point y_1' (for a fixed sequence of edges through which the subpath passes and a fixed angle of exit at y_1'). The length is simply given by $\alpha_{e_1}|y_1y_1'|$ plus the path length from y_1' to y_2 plus $\alpha_{e_2}|y_2y_2'|$; by Claim 2, this is a linear function of the coordinate of y_1' on edge e_1. We know that the subpath is uniquely specified by giving point y_1' and the sequence of edges through which the path passes on its way from y_1' to y_2, since we know that the subpath exits edge e_1 at the critical angle.

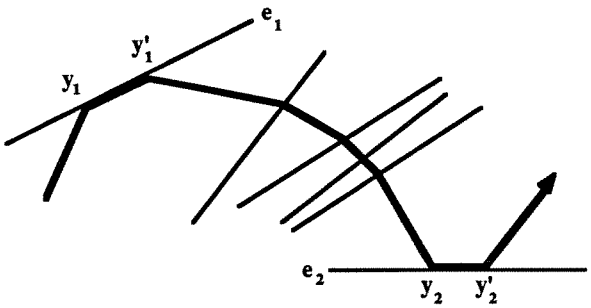

Figure 3.11. Proof of Lemma 3.7.

A linear function defined on an interval attains its minimum at an endpoint of the interval. (Of course, if it is flat, the function may attain its minimum everywhere in the interval.) Thus, we could slide the point y_1' along e_1 in some direction, without increasing the length of the subpath as we move it, and continue this motion until either y_1' coincides with y_1, or until the subpath hits another vertex. In either case, we either keep the length of the path the same or we get a local improvement of the original path. (Note that if this local improving operation slides y_1' until it coincides with y_1, then the path should not have shared a segment with edge e_1 in the first place.) ∎

Note that an immediate consequence of this lemma is that the path shown in Figure 3.9 between v and v' cannot be geodesic, since it has two shared segments between the (consecutive) vertices v and v'.

Remark: Note also that in order for a path to have several shared segments between two consecutive vertices the edges of $\mathcal{S}$ and the region weights have to be arranged in a very

special way. Indeed, when a path leaves one edge at a critical angle, the chances are small that it will strike another edge at *exactly* its critical angle. Thus, the case addressed by the above lemma is in some sense degenerate.

In summary, we have the following characterization of geodesics and optimal paths through weighted regions:

Proposition 3.8 *In the absence of zero weights, the general form of a geodesic path or an optimal path is a piecewise linear path that goes through an alternating sequence of vertices, (possibly empty) edge sequences, and shared segments such that the path obeys Snell's Law at each edge along any edge sequence and it obeys the local optimality criterion at the endpoint of each shared segment. Furthermore, between any critical point of exit and the next critical point of entry along the path, there must be a vertex. In the presence of zero weights the same characterization holds, except that optimal paths may be arbitrary within zero-weighted regions.*

4. Locally f-Free Paths and Intervals of Optimality

In the solution to the DGP, we made use of the concept of an *f-free path* to points on the boundary of a face f. In the weighted region problem, however, we will need the notion of a *locally f-free path*: Given a face f and e, one of its three edges (recall that (e, f) is an *edge-face pair*), a *locally f-free path* to $x \in e$ is defined to be a geodesic path p from s (the source point) to x such that there exists a $\delta > 0$ so that the path does not pass through the intersection of the interior of f with the ball of radius δ centered at x. Intuitively, a locally f-free path to a point $x \in e$ "hits x from the exterior of face f" and is locally optimal. The reason we require the local optimality of path p in the definition is that we need to rule out a path that is locally f-free only because it crossed e from f into the interior of face f' opposite f and then immediately "turned around" to hit x from the side of f' (such a path could be locally shortened into a locally f'-free path to x). A *shortest* locally f-free path to x, denoted by $p_f(x)$, is a locally f-free path to x that has minimal length among all locally f-free paths to x.

Because an optimal path to a point x on the interior of $e = f \cap f'$ must be either locally f-free or locally f'-free, it may be obtained from a shortest locally f-free and a shortest locally f'-free path to x:

Lemma 4.1 *Let $p_f(x)$ (resp., $p_{f'}(x)$) be a shortest locally f-free (resp., f'-free) path from s to $x \in int(f \cap f')$. An optimal path to x is given by the shorter of $p_f(x)$ and $p_{f'}(x)$.*

Another important property of shortest f-free paths for the DGP is that they are not allowed to cross (Lemma 4.3 of [MMP]). This is not true for locally f-free paths in the weighted region problem, as illustrated in Figure 4.1. Rather, we now have that shortest locally f-free paths cannot cross while "going in the same direction". More precisely, we get the following lemma:

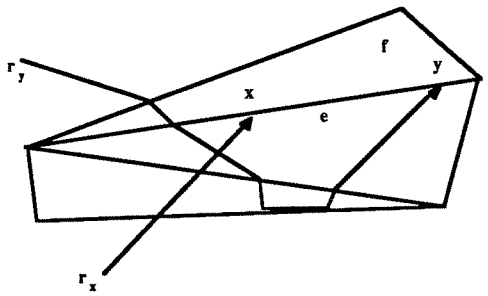

Figure 4.1. Crossing locally f-free paths.

Lemma 4.2

(1) *Optimal paths and shortest locally f-free paths are simple.*

(2) *If $p(x)$ and $p(y)$ are optimal paths from s to points x and y, then they can intersect only at vertices of $\mathcal{S}$, and if they do intersect at v, then that subpath of $p(x)$ from s to v has the same length as that subpath of $p(y)$ from s to v.*

(3) *Let $p_f(x)$ and $p_f(y)$ be two shortest locally f-free paths to points x and y on edge $e = f \cap f'$. Consider the segments of each path to be oriented in the direction of the path from the source. Then if an (oriented) segment of $p_f(x)$ and an (oriented) segment of $p_f(y)$ are both incident on some edge e', and both segments belong to the same face f_c, then those segments cannot cross.*

Proof:

The proof of the first claim is easy: if a path is not simple, then it can be locally improved (and kept f-free). The proof of the second claim is also easy and follows the proof given in [MMP] for the DGP (Lemma 4.3 of [MMP]). For the third claim, note that if the segments crossed at some interior point, γ, of f_c, then the two subpaths from s to γ must have equal lengths (otherwise, one path could be locally improved while remaining locally f-free). Then, each path could be improved by "shortcutting" around γ (e.g., from a point $w \in int(f_c)$ on the subpath of $p_f(x)$ before γ to a point $z \in int(f_c)$ on the subpath of $p_f(y)$ after γ), and because the two paths were both "going the same direction" when they passed through γ, the resulting improved path could be locally adjusted so that it will again be geodesic and still be locally f-free. (The reason this does not work in showing that two shortest locally f-free paths cannot cross at all is that if the paths were going different "directions" at the crossing point, so the segments are incident on different edges of f_c then the improved path could become non-locally f-free once it is perturbed into local optimality. Such would be the case if $\gamma \in f' = f_c$ and the segment of the path that reaches γ first were not incident on e, while the other segment were incident on e. Refer again to Figure 4.1.) A similar argument applies to the case in which the crossing point γ lies interior to an edge, since we can simply shortcut around γ with a refraction path instead of a direct path. ∎

Analogous to Lemma 4.4 of [MMP], we get the following result:

Lemma 4.3 *Let (e, f) be any edge-face pair, and let $p_f(x)$ be any shortest locally f-free path from s to $x \in int(e)$. Let r be the root of $p_f(x)$ and let $\mathcal{E}$ be the last edge sequence of $p_f(x)$. Then, the set $\mathcal{I}$ of points on e for which there exists a shortest locally f-free path to x with root r and last edge sequence $\mathcal{E}$ is connected (and, therefore, a subsegment of e).*

Proof: Let y and y' be points of e such that there exist shortest locally f-free paths $p_f(y)$ and $p_f(y')$ with root r and last edge sequence $\mathcal{E}$. Then we wish to show that if z is a point of e between y and y', then there is a shortest locally f-free path $p_f(z)$ that also has root r and last edge sequence $\mathcal{E}$ (in fact, we show that $p_f(z)$ is unique). Let $p_f(z)$ be *any* shortest locally f-free path to z. Now, march back from z along the path $p_f(z)$. Refer to Figure 4.2. (Although the figure shows the case in which r is a vertex, the argument we are about to give works equally well for the case of r being a critical point of entry of some edge.) The first line segment of $p_f(z)$ we encounter cannot take us into face f (by definition of locally f-free paths). Thus, the segment must take us through face f_{K+1}. Also, the segment cannot cross paths $p_f(y)$ or $p_f(y')$ (Lemma 4.2), so it must cross edge e_K at a point in between the points y_K and y'_K. Continuing this march back through edges e_{K-1}, ..., e_1 yields the fact that path $p_f(z)$ must stay "in between" paths $p_f(y)$ and $p_f(y')$ until the root r is reached (or, if r is a critical point of entry, until the corresponding critical point of exit of $p_f(z)$ is reached). Thus, $p_f(z)$ must pass through the same set of edges as the paths $p_f(y)$ and $p_f(y')$ do along their last edge sequences. Also, $p_f(z)$ must have root r. Thus, $z \in \mathcal{I}$, and we are done. ∎

The interval $\mathcal{I}$ of points can be open, closed, or half open and half closed. Let the subsegment $I = [a, b]$ be the closure of $\mathcal{I}$. (Normally, we would write $I = \overline{ab}$ to indicate the segment with endpoints a and b; however, we write $[a, b]$ to emphasize the fact that coordinate values can be attached to points of e, and I can be thought of as an interval of these coordinates.) The endpoint a (resp., b) is the one on the *left* (resp., *right*) when viewed looking into face f.

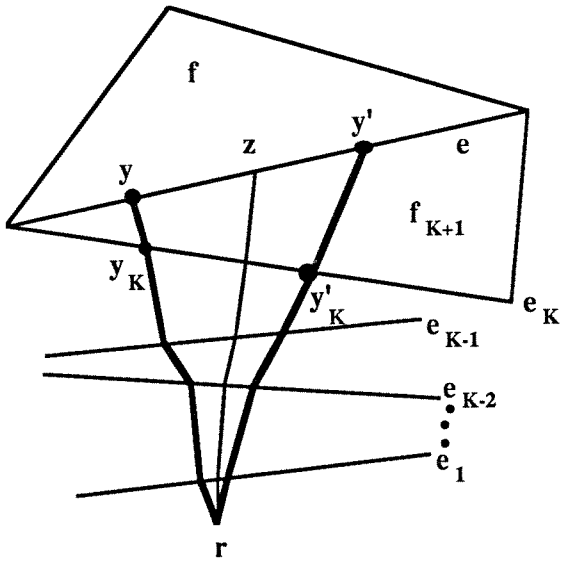

Figure 4.2. Proof of Lemma 4.3.

We call I the *interval of optimality* for $(r, \mathcal{E})$ *with respect to* (e, f). It describes the subset of an edge for which the shortest locally f-free paths to the edge have the same discrete structure (going through the same sequence of edges, after coming through the same root). Our algorithm will find the subdivision of each edge into its intervals of optimality with respect to both edge-face pairs defined on the edge. The set of intervals of optimality can be thought of as the restriction of the "shortest path map" (with respect to s) to the edges of the subdivision $\mathcal{S}$. (Actually, our subdivision into intervals will be according to ϵ-optimality, for a user-specified error tolerance ϵ. See section 8.)

Note that r may be an endpoint of e (and hence of I) in the degenerate case. Let us establish the convention that $\mathcal{E}$ does not include the edge e. If $\mathcal{E} = \emptyset$, then r will be one of the vertices of f'. A point $x \in e$ is an element of the interval of optimality for $(r, \mathcal{E})$ with respect to (e, f) if there exists a shortest locally f-free path to x whose root is r and whose last edge sequence is $\mathcal{E}$.

If r is a vertex, then the *angular extent* of I is the interval $[\theta_a, \theta_b]$ of angles, ϕ, about r such that if a refraction path is traced from r starting at angle ϕ, then a point of I will be crossed by the path. ϕ is measured as the counterclockwise angle from the positive x-axis; to insure that $[\theta_a, \theta_b]$ is an interval, we require that $\theta_a \in [0, 2\pi)$ and that $\theta_b \in (\theta_a, \theta_a + 2\pi)$. The fact that the angular extent is well-defined and is indeed an interval follows from Claim 3 of Section 3.

If $r \in int(e_r)$ is a critical root of I, then the "angular extent" of I will be defined to be the interval $[\tau_{r''}, \tau_{r'}]$ of real numbers that corresponds to the range of shared segments, from the minimal segment $\overline{rr''}$ to the maximal segment $\overline{rr'}$, where $\tau_{r''} = |rr''|$ and $\tau_{r'} = |rr'|$. As $\tau \in [\tau_{r''}, \tau_{r'}]$ varies from $\tau_{r''}$ to $\tau_{r'}$, the path rooted at r varies from having shared segment $\overline{rr''}$ (and passing through one endpoint of I) to having shared segment $\overline{rr'}$ (and passing through the other endpoint of I). See Figure 4.3. Each point in I is in one-to-one correspondence with a point in $[\tau_{r''}, \tau_{r'}]$. The path corresponding to $\tau \in [\tau_{r''}, \tau_{r'}]$ is the one that is rooted at r (which is a critical point of entry), travels along edge e_r a distance τ, then gets off of edge e_r (at the critical angle of exit) to continue through the edge sequence $\mathcal{E}$ to a point in I. For simplicity of notation, we will write the angular extent of I as $[\theta_a, \theta_b]$ even in the case that r is non-vertex. The understanding is that θ_a (resp., θ_b) equals $\tau_{r''}$ or $\tau_{r'}$, depending on which endpoint of $[\tau_{r''}, \tau_{r'}]$ corresponds to a path through a (resp., b). Note that no angle is varied to generate paths through I in this case; the angle of exit is always the critical angle, and the only parameter varied is the critical point of exit from e_r. Also note that the direction in which the path leaves edge e_r is uniquely specified by the last edge sequence $\mathcal{E}$. That is, whether the path is critically reflected at e_r or whether it hitched a ride on e_r before crossing it is determined by the first edge in $\mathcal{E}$.

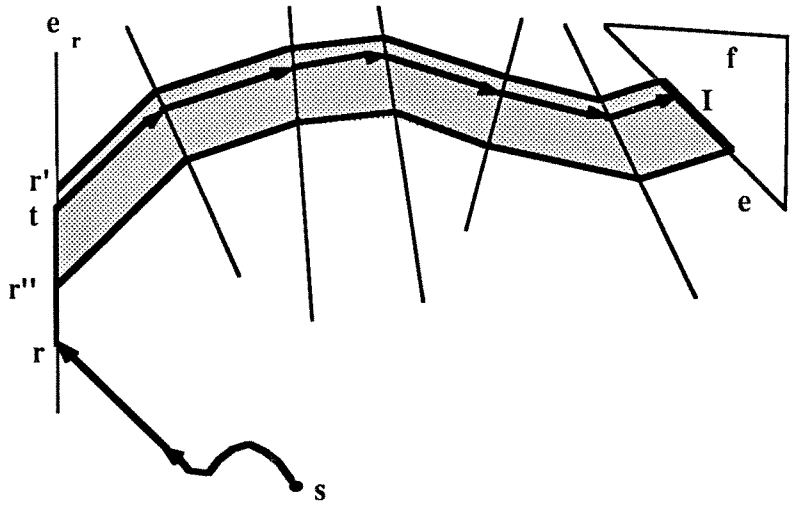

Figure 4.3. The "angular extent" of a critical root.

Lemma 4.4 *Intervals of optimality with respect to an edge-face pair (e, f) form a covering of e and have mutually disjoint interiors.*

Proof: First note that every point $x \in e$ must lie in *some* interval of optimality with respect to (e, f): if $p_f(x)$ is any shortest locally f-free path to x, then we can determine the root, r, and the last edge sequence, $\mathcal{E}$, of $p_f(x)$, and x will lie in the interval of optimality for r and $\mathcal{E}$ with respect to (e, f). Now, for a point $x \in e$ to be in the intervals of optimality for $(r, \mathcal{E})$ and for $(r', \mathcal{E}')$ with respect to (e, f), it must satisfy the equation $d(r) + d_{r,\mathcal{E}}(x) = d(r') + d_{r',\mathcal{E}'}(x)$. We claim that there can be at most one such "tie point" x satisfying this equation. If there were two points, x and x', then we would get a crossing of the shortest locally f-free path to x (through one of the edge sequences) with the shortest locally f-free path to x' (through the other edge sequence). Thus, there can be no open subinterval of e all of whose points lie in two or more intervals of optimality. This implies that the intervals must have disjoint interiors. ∎

The point, c, in I that is closest to r is called the *frontier point* of the interval. The fact that such a point exists and is unique for the case of positive weights along the path comes from the following lemma.

Lemma 4.5 *If r is a vertex that is the root of an edge sequence $\mathcal{E}$ and $I \subseteq e = f \cap f'$ is an interval of optimality with root r and last edge sequence $\mathcal{E}$, then there exists a point c in I that is closest to r among paths that go from r to I through $\mathcal{E}$. Further, c is either an endpoint of I or is such that the refraction path from r to c through $\mathcal{E}$ hits c at an angle of incidence of zero. If the shortest locally f-free paths to points of I do not go through zero-cost regions, then the point c is unique.*

Proof: The fact that there exists a closest point follows from the continuity of the length function and the closed and boundedness of I. To show uniqueness in the case of positive weights, note that the angles of incidence at points of I form an interval (from Claim 3 of Section 3). The point of I that has an angle of incidence closest in absolute value to 0 will be the frontier point of I; otherwise, a strictly shorter path to I could be found by sliding c along e by a slight amount, keeping all other crossing points the same along $\mathcal{E}$ (strictness requires that $\alpha_e, \alpha_{f'} > 0$). But for any closed interval of real numbers, there exists a unique number that is closest to 0. Hence, c is unique and c will either be an endpoint of I or will be an interior point if there exists an interior point with a zero angle of incidence. ∎

To see that there are cases in which the frontier point is not unique, consider what happens when $\alpha_{f'} = 0$. Then, all points of e are at equal distance from s (along paths with last edge sequence $\mathcal{E}$ and root r), since we are able to travel through f' at zero cost. In this case, then, the entire edge will serve as a frontier. We will say a little more about this case in the next section when we discuss how the algorithm handles the case of zero-cost regions.

For any interval I we can compute the frontier point c by calling the function *Frontier* (I) as follows:

Function *Frontier* (I)

(0). *Assume that* $\mathcal{E} = (e_1, \ldots, e_k)$ *is the last edge sequence of* I *(the case in which* $\mathcal{E} = \emptyset$ *can be handled trivially). Assume that* $I = [a, b]$ *is an interval with respect to* (e, f) *with root* r. *If* r *is a vertex, go to (2); otherwise, go to (1).*

(1). **(r is a critical root)** *Compute the distances from* $r \in int(r_c)$ *to* a *and to* b. *(This can be calculated easily since the paths from* r *to* a *and* b *are uniquely determined by the fact that they leave edge* e_r *at the critical angle.) Let* c *be the closer of the two points* a *and* b, *compute the length* λ *of the path from* r *to* c, *and let* u *(resp.,* β*) be the first (resp., last) bend point on the path from* r *to* c. *Return the list* (c, λ, u, β).

(2). **(r is a vertex)** *Trace backwards from edge* e *through the edge sequence* $\mathcal{E}$, *starting with an angle of incidence of* 0 *at* e. *At each iteration, compute the angle of incidence at edge* e_{i-1}, *using the known interior angles of the face and the known angle of incidence at edge* e_i. *This eventually gives us the angle,* ϕ, *at which a ray must leave point* r *so that it will refract along* $\mathcal{E}$ *and ultimately hit edge* e *perpendicularly. Now do ray tracing from* r *starting at angle* ϕ. *If the resulting refraction path stays within the sequence* $\mathcal{E}$ *and it hits* e *inside of the subsegment* $[a, b]$, *then let* c *be the hit point on* e. *Otherwise, let* c *be the endpoint of* I *that is closer to point* r. *Compute the length* λ *of the path from* r *to* c, *and let* u *(resp.,* β*) be the first (resp., last) bend point along the path. Return the list* (c, λ, u, β).

The intersection of the f-free path to c with the face f' (which shares edge e with face f) is a line segment, $\overline{\beta c}$, on face f'. We call point β the *access point* of interval I. We note that the point β computed by the function above will be the access point of interval I. If we draw the line segments $\overline{\beta c}$ for every interval of optimality with respect to (e, f), then we get a partitioning of face f' into *access channels* (since no two such segments can intersect). Note that an access channel is either a triangle, a quadrilateral, or a pentagon. We will refer to an access channel by giving a pair, (I_1, I_2), where I_1 and I_2 are the intervals of optimality that give rise to the bordering segments, $\overline{\beta_1 c_1}$ and $\overline{\beta_2 c_2}$. If $I_1 =$ NIL, the channel is simply that part of f' to the left of $\overline{\beta_2 c_2}$; and if $I_2 =$ NIL, the channel is that to the right of $\overline{\beta_1 c_1}$.

5. The Algorithm

As with the discrete geodesic algorithm of [MMP], our algorithm for the weighted region problem employs the "continuous Dijkstra" technique. The basic idea behind this methodology is to simulate the effect of a "wavefront" that propagates from the source point s. (The *wavefront* at distance d is the set of points such that the length of the shortest path from s to these points is d.) As d increases, the wavefront sweeps over points in the plane. At certain special values of d the wavefront collides with vertices or edges. We refer to these collisions as "events". Continuous Dijkstra simulates the continuous motion of the wavefront by keeping track only of the critical changes that occur at events. The technique closely resembles the discrete algorithm that Dijkstra [Di] suggested to find shortest paths in a graph by visiting the nodes in the order of increasing distance from the source. In the DGP, [MMP] applied the continuous Dijkstra technique and showed that the number of events is $O(n^2)$ in the worst case. For the application to the weighted region problem, we will give a version of a continuous Dijkstra algorithm in which the number of events is $O(n^4)$.

The algorithm uses a few simple data structures. We keep a list, ILIST, of *candidate* intervals of optimality. A *candidate interval* (or "interval", for short) is a subsegment of an edge that is a supersegment of some (possibly empty) interval of optimality. A candidate interval, I, has the following information associated with it: its extent, $[a, b]$; its *first bend points*, (u_a, u_b) (which are the first points where the paths from r to a and b "bend"; these are precisely the points that define the angular extent, $[\theta_a, \theta_b]$, of the interval (even when r is critical)); the *last bend points*, (w_a, w_b) (which are the points where the paths to a and b cross the last edge in the last edge sequence; these points give us the angles of incidence of paths at a and b, ϕ_a and ϕ_b); its edge-face pair, (e, f); its root, r (along with a pointer to the edge containing r, should r be a critical point of entry); its depth, $d = d(r)$; its frontier point, c; its access point, β; and its *predecessor*, $\overline{I}$ (which is the candidate interval (or vertex) whose "propagation" originally gave rise to I). We then will write $I = ([a, b], (u_a, u_b), (w_a, w_b), (e, f), r, d, c, \beta, \overline{I})$.

An edge-face pair, (e, f), has associated with it a sorted list of candidate intervals with respect to (e, f), with the intervals in the order that they appear along the edge (following the edge in the direction that keeps the face f on the *left*).

Each vertex v has associated with it a pointer to each candidate interval of which it is a root. Additionally, there is a *distance label*, $d(v)$, which indicates the length of the best-known path to vertex v. Initially, $d(v) = +\infty$ for every vertex v. At some stage of the algorithm, each vertex (and certain critical points of entry) will become "permanently labeled", meaning that the value $d(v)$ correctly represents the length of an optimal path from s to v. Before such time as a label becomes permanent, we will refer to it as being "temporarily labeled". (The hope is that our terminology parallels that of Dijkstra's algorithm.)

To be able to handle zero-cost regions, we define the following notation. First, we build a graph in which each zero-weight edge and each zero-weight face is associated with a node (either an *e-node* or an *f-node*, respectively). Two f-nodes are adjacent if the corresponding faces either share an edge whose weight is finite or share a vertex that is not *blocked* between the faces (so that a point in the interior of one face can get to a point in the interior of the other face along a zero-cost path). Two e-nodes are adjacent if the corresponding edges share a vertex that is not blocked between the edges. An e-node is adjacent to an f-node if the corresponding edge is an edge of the corresponding face, or if the edge and face share a vertex that is not blocked between the edge and face. Each connected component of this graph is called a *free region*, denoted by $\mathcal{Z}$, and we establish pointers from each zero-weight face and each zero-weight edge to its corresponding free region. The interpretation is that all of the faces and edges within a free region "communicate", in that zero-cost paths exist between all points of all of these faces and edges.

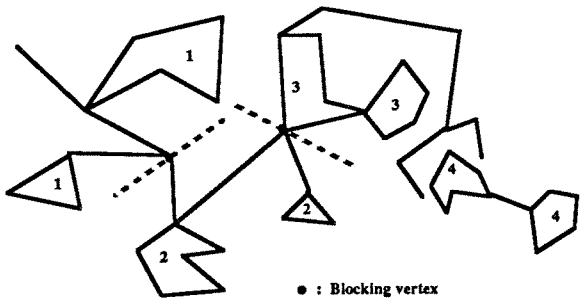

Figure 5.1. Free regions.

Distinct free regions can touch at vertices that are blocking or can share infinite-weight edges, but are otherwise disjoint. See Figure 5.1 for an example of a set of free regions. (Note that a free region is a "generalized polygon", in the sense defined in [RS].) The boundary of a free region is made up of a set of edges. Each edge on the boundary has one or both of its adjacent faces outside the free region. We say that an edge-face pair (e, f) is on the boundary of a free region $\mathcal{Z}$ if e is on the boundary of $\mathcal{Z}$ and f is not contained in $\mathcal{Z}$. We preprocess the free regions so that boundary information (the vertices and the edge-face pairs that form the boundary) is stored with $\mathcal{Z}$. Also, each free region $\mathcal{Z}$ has a distance label, $d(\mathcal{Z})$, (initially $+\infty$) which indicates its distance from the source. Only one label is needed for all points of $\mathcal{Z}$, since they are are all within zero distance of each other.

The algorithm also maintains a priority queue (called the *event queue*) whose entries are points of some candidate interval (either an endpoint or the frontier point), with labels that are the best-known distances back to the source.

The algorithm proceeds, at least in basic structure, just as it did for the discrete geodesic problem of [MMP]: we select the next event from the priority queue, propagate the wavefront through the corresponding interval by projecting the interval through the appropriate face, and then update the list of candidate intervals after performing a "trimming" step. The major differences are in the function *Project* (due to the new local optimality criterion) and in the way critical angles are handled (which results in the new procedure *Insert-Critical-Root*). We also need the special procedure *Propagate-Through-Free-Region* to handle the case of a "wavefront" encountering a free region.

When we propagate an interval through a face f, we need to determine the lengths of locally optimal paths that go through the interval and strike the boundary of f opposite to the interval. In the unweighted problem [MMP], we handled this case by keeping associated with an interval the "unfolded image" of its root. Then, in constant time, we can find the length of the locally optimal path to any point $x \in f$ (that is reachable through the interval) by drawing the segment from the unfolded root to x. In contrast, for the weighted region problem we know of no method to do these distance calculations except that of ray tracing from the root through the last edge sequence. (Each such ray tracing will take time proportional to the length of the edge sequence.) In order to find the refraction path that hits a particular point, we can do ray tracings to search for the point (using binary search), or we can run a simple numerical routine. For this purpose, we have the special functions *Find-Point* and *Find-Tie-Point*, which will be described in Section 8.

We now begin the formal specification of each of the major procedures and functions.

Main Algorithm

(0). **(Initialize)** *Assign $d(s) = 0$ (s is now "permanently labeled"). Create a degenerate candidate interval corresponding to the source: $I_s = ([s,s], (s,s), (s,s), (NIL, NIL), s, 0, s, s, s)$, with s assigned as the root, first bend points, last bend points, access point, predecessor, and frontier point. (The edge-face pair, (NIL, NIL), is left undefined for I_s.) For each edge-face pair, initialize its interval list to be empty. Initialize the event queue and ILIST to consist of the single element I_s.*

(1). **(Main Loop)** *While there is an entry in the event queue, remove the one with the smallest distance label and make its label permanent. If it is being labeled as the frontier point of some candidate interval, I, then do* Propagate*(I).*

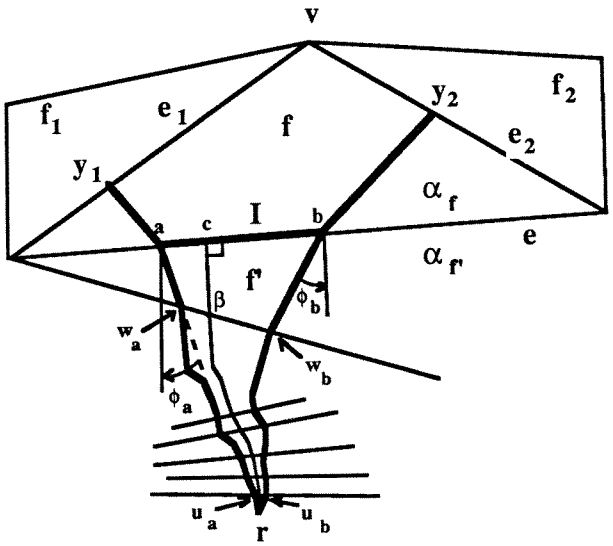

Figure 5.2. Propagation of an interval.

Procedure *Propagate* (I)

(0). *Assume that I is an interval with respect to (e, f), where $e = f \cap f'$. Let $c \in e$ be the frontier point of I. Let $e_1 = f \cap f_1$ and $e_2 = f \cap f_2$ be the edges of f opposite edge e. Refer to Figure 5.2.*

(1). **(Check for special cases)** *If $\alpha_e = 0$ or if $\alpha_f = 0$ and $\alpha_e < +\infty$, then let $\mathcal{Z}$ be the free region containing e (or f), do* Propagate-Through-Free-Region *$(c, \mathcal{Z})$, and stop. If $\alpha_e = +\infty$ or if $\alpha_f = +\infty$, then stop (no propagation is possible across face f or edge e). Otherwise, go to (2) or (3), depending on whether or not c is a vertex.*

(2). **(c is *not* a vertex)** *Let $I_i =$* Project *(I, e_i) and, if $I_i \neq NIL$, do* Insert-Interval *(I_i, c), for $i = 1, 2$.*

(3). **(c is a vertex)** *For each finite-weight face f_0 ($\neq f', f$) containing c such that c is not blocked between f' and f_0, create candidate intervals I_i (for $i = 1, 2, 3$) on the three edges (e_1,e_2, and e_3) of f_0, where the extent of interval I_i is the entire edge e_i and the root (and predecessor) of I_i is c. Do* Insert-Interval *(I_i, c) for each $i = 1, 2, 3$.*

Step (1) above checks to see if the the interval propagates into a free region, and, if so, it calls the procedure *Propagate-Through-Free-Region*. We also check for the case in which passage is blocked for paths going through interval I from face f' into face f. Basically, propagation stops if we hit an obstacle (either an infinite-cost edge or infinite-cost face).

Note that the first time *Propagate* is called, it will be applied to the degenerate interval I_s, whose edge-face pair is undefined; in this case, we go directly to step (3), where we propagate out from the vertex s.

Special treatment is necessary when we encounter a free region, for the wavefront can instantaneously traverse the region to all of its boundary. The result is that we turn the entire free region into one big "source", out of which wavefronts propagate in every direction.

Procedure *Propagate-Through-Free-Region* $(c, \mathcal{Z})$

(0). *If $d(c) \geq d(\mathcal{Z})$, then stop; otherwise, label $\mathcal{Z}$ with $d(\mathcal{Z}) = d(c)$, and continue to (1).*

(1). **(Propagate through boundary edges)** *For every edge-face pair (e, f), on the boundary of $\mathcal{Z}$ such that $\alpha_e \neq +\infty$, create a candidate interval, I, of the following special form. The extent of I is the entire edge. The angles of refraction of all paths eminating from the interval are zero, since paths must leave perpendicular to the edge. The root of I is considered to be $\mathcal{Z}$, and the depth is $d(\mathcal{Z})$. Any point in I can serve as the frontier point for the interval (since all points can be reached in the same distance from c). The access point is considered to be NIL, and the predecessor is just $\mathcal{Z}$. Add each of these candidate intervals to the ILIST. Now call* Propagate *(I) for each such interval.*

(2). **(Propagate through boundary vertices)** *For every vertex, v, on the boundary of $\mathcal{Z}$, permanently label v with $d(v) = d(c) = d(\mathcal{Z})$. For each finite-weight face $f_0 \notin \mathcal{Z}$ containing v such that v is not blocked between f_0 and a face of $\mathcal{Z}$, create candidate intervals I_i (for $i = 1, 2, 3$) on the three edges (e_1,e_2, and e_3) of f_0, where the extent of interval I_i is the entire edge e_i and the root (and predecessor) of I_i is v. Do* Insert-Interval *(I_i, v) for each $i = 1, 2, 3$.*

In step (1) above we have instantiated a type of candidate interval that does not technically fit within the framework of the definition given earlier. We could modify our definition to include this special case; instead, for simplicity, we omit the details here and note that all operations that we require are easily done on these candidate intervals. In particular, it is easy to project them, propagate them, and to find how to hit a vertex or critical point from them. (Hitting a vertex is accomplished by simply using the requirement that the angle of refraction out of the free region must be zero.) We immediately propagate each of the newly created intervals since the distance to the frontier point of each interval is the same as $d(c)$; motion is free from c to the boundary of $\mathcal{Z}$.

The next function, *Project* (I, e_1), finds the subset of e_1 that is hit by paths through I. We apply the local optimality criterion (Snell's Law) to determine how the "wedge of light" is extended through the next face. The function returns a candidate interval on e_1 whose points are accessible through I. If no part of e_1 is accessible, we return NIL.

Function *Project* (I, e_1)

(0). *Assume $I = [a, b]$ is an interval with respect to (e, f). Let $v \in f$ be the vertex opposite e and let (u_a, u_b) (resp., (w_a, w_b)) be the first (resp., last) bend points of I. We assume that e_1 is one of the two edges containing v. Refer to Figure 5.2.*

(1). **(Check for blockage)** *If $\alpha_e = +\infty$ or if $\alpha_f = +\infty$, then stop and return NIL (no propagation is possible across face f or edge e).*

(2). **(Projecting)** *Consider the "cone of light rays" obtained by doing ray-tracing from r through points in I, using initial directions determined by $[u_a, u_b]$. (If $r \in int(e_r)$ is a critical root, then we get instead a "belt" of light rays, as in Figure 4.3, that use points of $[u_a, u_b]$ as critical points of exit from e_r. Since all the paths exit e_r at the same angle, and they pass through the same last edge sequence, they are all incident on edge e at the same angle, $\phi_a = \phi_b$.) Let $A = [a_A, b_A]$ be the subsegment of e_1 that lies within this cone and let $[u_{a_A}, u_{b_A}]$ be the corresponding range of first bend points. We can determine points a_A and b_A as follows. First extend the paths through a and b until they exit face f. Do this by applying Snell's Law at the edge e, using the angles of incidence, ϕ_a and ϕ_b (determined by the points w_a and w_b), at e and extending the resulting rays of refraction through face f, leaving through points y_a and y_b. (If a (resp., b) is a vertex, then $y_a = a$ (resp., $y_b = b$).) If $y_a \in e_1$, then set $a_A = y_a$, $u_{a_A} = u_a$, and $w_{a_A} = a$; otherwise, $A = \emptyset$, and we return NIL. If $y_b \in e_1$, then set $b_A = y_b$, $u_{b_A} = u_b$, and $w_{b_A} = b$; otherwise, set $b_A = v$ and let $(\lambda, a_1, b_1, w) =$ Find-Point (I, v). Set $u_{b_A} = b_1$ and $w_{b_A} = w$.*

(3). **(Trimming)** *Calculate the angles of incidence, ϕ_{a_A} and ϕ_{b_A}, at a_A and b_A. Let $\theta_c = \theta_c(f, e_1)$ be the critical angle at edge e_1 coming from face f. Note that $\phi_{a_A} < \phi_{b_A}$ by Claim 3 of Section 3 if r is a vertex, and $\phi_{a_A} = \phi_{b_A}$ if r is a critical root. There are now three cases to consider.*

- (i). *($[\phi_{a_A}, \phi_{b_A}] \subset [-\theta_c, \theta_c]$) Return the candidate interval $I_A = [a_A, b_A]$ that has root r, edge-face pair (e_1, f_1), first bend points (u_{a_A}, u_{b_A}), last bend points (w_{a_A}, w_{b_A}), predecessor I, and a frontier point and an access point that are easily calculated by calling* Frontier (I_A). *All of the angles of incidence to this interval are less (in absolute value) than the critical angle.*
- (ii). *($[\phi_{a_A}, \phi_{b_A}] \cap [-\theta_c, \theta_c] = \emptyset$) Return NIL.*
- (iii). **(Otherwise;** *$[\phi_{a_A}, \phi_{b_A}] \cap [-\theta_c, \theta_c] \neq \emptyset$) If $\theta_c \in [\phi_{a_A}, \phi_{b_A}]$, then let $(b_A, u_{b_A}, w_{b_A}) =$* Find-Hit-At-Angle *$(\theta_c, I)$, and mark b_A as a critical point of entry (we have trimmed $[a_A, b_A]$ on the right). If $-\theta_c \in [\phi_{a_A}, \phi_{b_A}]$, then let $(a_A, u_{a_A}, w_{a_A}) =$* Find-Hit-At-Angle *$(-\theta_c, I)$, and mark a_A as a critical point of entry (we have trimmed $[a_A, b_A]$ on the left). Return the candidate interval $I_A = [a_A, b_A]$ that has root r, edge-face pair (e_1, f_1), first bend points (u_{a_A}, u_{b_A}), last bend points (w_{a_A}, w_{b_A}), predecessor I, and a frontier point and an access point that are easily calculated by calling* Frontier (I_A).

Step (2) above makes a call to *Find-Point* (I, v), which is not formally defined until Section 8. All we need to know at this point is that the function returns a list (λ, a_1, b_1, w), where λ is the length (actually, an approximate length) of the path from the root r through I that strikes v, and w is the last bend point on this path. The points a_1 and b_1 lie on the first edge in the last edge sequence $\mathcal{E}$ (corresponding to I), and they define an interval in which the first bend point of the desired path must lie. As we will see in Section 8, *Find-Point* is a binary search procedure that results in a small interval $[a_1, b_1]$ that traps the "true" value of the first bend point on the exact refraction path to v. In step (2) above, we would like to assign u_{b_A} to be the first bend point on the exact refraction path to v, but, instead, we have assigned $u_{b_A} = b_1$, leading to a slight overestimate of the interval $[u_{a_A}, u_{b_A}]$ of first bend points that correspond to the interval of optimality $[a_A, b_A]$.

Step (3) above needs to compute angles of incidence at a_A and b_A. If a_A or b_A is a vertex, then we must be careful how we define the angle of incidence at the edge e_1. We essentially take the limit of the angle of incidence as the path gets closer and closer to the vertex from within f. This can be calculated exactly by finding the angle of incidence to a line that is parallel to e_1 but slightly outside f for a path that passes through the vertex.

The next procedure figures out how to place a new candidate interval I among the existing intervals on edge-face pair (e, f). It checks for dominance by and of other intervals, and performs the necessary

"trimming". When done, it creates an instance of the interval in the ILIST, and inserts the appropriate elements in the event queue.

Procedure *Insert-Interval* $(I, \overline{c})$

(0). *Assume $I = [a, b]$ is an interval with respect to (e, f) and that it has root r, frontier point c, last bend point β, and depth d. Point $\overline{c}$ is the frontier point of $\overline{I}$, the predecessor of I.*

(1). **(Locate point $\overline{c}$ in access channel)** *Locate point $\overline{c}$ in an access channel, say (I_1, I_2), of (e, f). The wavefront can only get to I through this channel. Let $I_i = [a_i, b_i]$ (provided that $I_i \neq NIL$), for $i = 1, 2$. Let r_i be the root, c_i be the frontier point, β_i be the last bend point, and d_i be the depth of I_i.*

(2). **(Delete dominated candidate intervals)** *If $I_1 \neq NIL$ and $a_1 \in I$, then let $(\lambda_1, a', b', w') =$* Find-Point *$(\overline{I}, a_1)$. If $d(r) + \lambda_1$ is less than the current label on point a_1 (meaning that a_1 is better reached through r than through its current root), then do* Delete *(I_1), set I_1 to be the predecessor of I_1 in the interval list of e_1 (set I_1 to NIL if no predecessor exists), and return to step (2). Similarly, if $I_2 \neq NIL$ and $b_2 \in I$, then check for dominance and, if necessary, delete I_2 and return to step (2).*

If $I_1 \neq NIL$, a_1 lies to the right of I, and $\overline{\beta c}$ intersects $\overline{\beta_1 c_1}$, then compute the point of intersection $\gamma = \overline{\beta c} \cap \overline{\beta_1 c_1}$ and compare the distance to γ via r (through $\overline{I}$) with the distance to γ via r_1 and the last edge sequence of I_1. If γ is reached via r before it is reached via r_1, then do Delete *(I_1), set I_1 to be the predecessor of I_1 in the interval list of e_1 (set I_1 to NIL if no predecessor exists), and return to step (2). If γ is reached first from r_1, then stop. (There is no need to insert I, since it is dominated.) Similarly, check for dominance in the case that $I_2 \neq NIL$, b_2 lies to the left of I, and $\overline{\beta c}$ intersects $\overline{\beta_2 c_2}$.*

(3). **(Trim at tie points)** *If $I_1 \neq NIL$ and $b_1 \in I$, then let $(a, \lambda_a, u'_{b_1}, u_{b_1}, u_a, u'_a, w_{b_1}, w_a) =$* Find-Tie-Point *$(I_1, I)$. Then a will be the point in $I \cap I_1$ that can be reached equally well through I and through I_1. (We are guaranteed that such a "tie point" exists by the continuity of lengths of locally f-free paths.) Remove b_1 from the event queue, set $b_1 = a$, and update c_1 if necessary. If $I_2 \neq NIL$ and $a_2 \in I$, let $(b, \lambda_b, u'_b, u_b, u_{a_2}, u'_{a_2}, w_b, w_{a_2}) =$* Find-Tie-Point *$(I, I_2)$. Then b will be the point in $I \cap I_2$ that can be reached equally well through I and through I_2. Remove a_2 from the event queue, set $a_2 = b$, and update c_2 if necessary. We now have an interval $[a, b]$, which is a trimmed version of the original interval that was to be inserted. We have also appropriately trimmed the neighboring intervals I_1 and I_2 and updated their first and last bend points. If $a \neq b$, go to (4); otherwise, stop (no interval needs to be inserted).*

(4). **(Insert necessary critical roots)** *If a (resp., b) is marked as a critical point of entry to edge e (which would have been done in step (3) of* Project*), then do* Insert-Critical-Root *(a, I) (resp.,* Insert-Critical-Root *(b, I)).*

(5). **(Final updates)** *Let $(c_I, \lambda_c, u, \beta) =$* Frontier *$(I)$, so that c_I is the frontier point of $I = [a, b]$ with respect to r and last edge sequence $\mathcal{E}$, and β is the access point of I. Push the interval $I = ([a, b], (u_a, u_b), (w_a, w_b), (e, f), r, d, c_I, \beta, \overline{I})$ onto the list ILIST and insert it into the interval list of edge e (between intervals I_1 and I_2). Put the points a, b, and c_I (if $c_I \neq a, b$) into the event queue, with labels $d + \lambda_a$, $d + \lambda_b$, and $d + \lambda_c$, respectively.*

Note that step (2) above may be performed many times, as we discover several neighboring intervals that are dominated by the new candidate. At the conclusion of step (2), I_1 and I_2 represent the intervals that will neighbor I on the left and right.

Step (3) calls the function *Find-Tie-Point* in order to find the "tie points" that will be the new boundaries between I and its neighbors. *Find-Tie-Point* (I_1, I_2) determines the tie point, $x \in I_1 \cap I_2$, by doing binary search in the range of values of $(u_{a_1}, u_{b_1}) \times (u_{a_2}, u_{b_2})$ (doing ray tracing at each evaluation). It returns a list $(x, \lambda, u_1, u'_1, u_2, u'_2, w_1, w_2)$, where x is the required tie point, λ is the distance from s to the point x, $[u_1, u'_1]$ is the (small) range of first bend points of paths through I_1 to x, $[u_2, u'_2]$ is the (small) range of first bend points of paths through I_2 to x, and w_1 and w_2 are the corresponding last bend points for paths through I_1 and I_2, respectively. Its implementation is similar to that of *Find-Point*, and will be described more in Section 8.

The procedure *Delete* (I) simply deletes I from the ILIST, deletes I from the interval list of the corresponding edge-face pair, and deletes any entries of points of I in the event queue.

The procedure *Insert-Critical-Root* (r_c, I) establishes the point $r_c \in I \subset e$ as a critical entry point to edge e. From the point r_c, paths can follow the edge e for a while and then possibly get off (at the critical angle) into the faces on one or both sides of e. This means that we must instantiate new candidate intervals on the edges opposite e.

Procedure *Insert-Critical-Root* (r_c, I)

(0). *If $\alpha_e = +\infty$, then stop. (There is no need to use e critically.) Assume that I is an interval with respect to (e, f). We assume that $r_c \in int(e)$. Let f' be the face sharing edge e with face f. Let e_1 and e_2 be the edges opposite e on face f, and let e'_1 and e'_2 be the edges opposite e on face f'. Refer to Figure 5.3.*

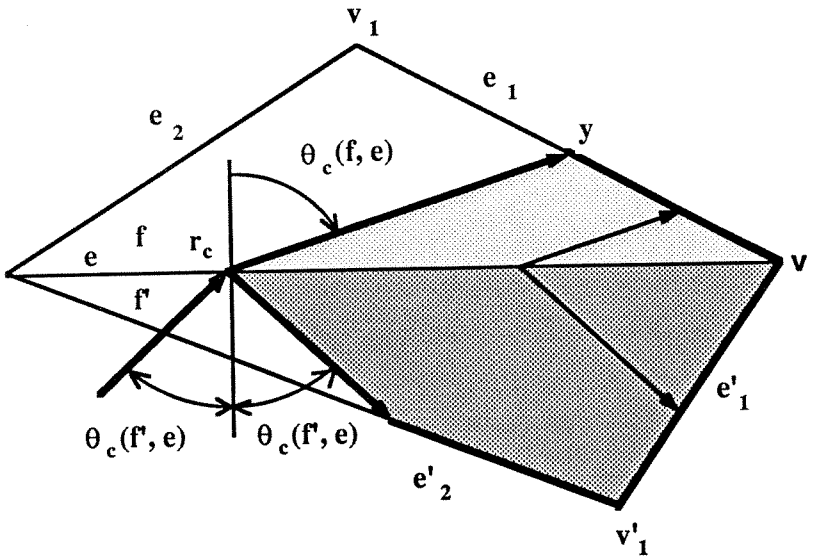

Figure 5.3. Result of inserting a root at a critical point of entry.

(1). **(Label v)** *If v is not in the event queue, then insert it with label $d(r_c) + \alpha_e|r_c v|$; otherwise, update the label on v by setting it to the minimum of its current label and $d(r_c) + \alpha_e|r_c v|$.*

(2). **(Instantiate new intervals)** *If $\alpha_e < \alpha_f$, then the critical angle $\theta_c(f, e)$ is well defined, and we proceed as follows. Draw a ray from r_c through f that leaves at the critical angle $\theta_c(f, e)$, and let y be the point of intersection with e_1 or e_2. If $y \in e_1$, then instantiate a candidate interval of optimality on e_1 with extent $[y, v]$. Otherwise, $y \in e_2$, and we instantiate two candidate intervals, one on e_1 with the entire edge $[v_1, v]$ as its extent, and one on e_2 with extent $[y, v_1]$. Similarly, if $\alpha_e < \alpha_{f'}$, then the critical angle $\theta_c(f', e)$ is well defined, and we proceed as above, instantiating intervals on one or both of the edges e'_1 and e'_2. Compute the necessary quantities, such as frontier points and first bend points. The intervals will all have r_c as both their root and predecessor. We add to ILIST each interval that we create and update the event queue accordingly.*

The function *Find-Hit-At-Angle* (θ, I) determines the point in I (if there is one) such that the refraction path from the root of I through the last edge sequence will hit the point at the angle of incidence θ. If the function succeeds in finding the desired path, it returns a list (x, u, w) of the point hit (x), the corresponding first bend point (u), and the corresponding last bend point (w); otherwise, it returns NIL.

Function *Find-Hit-At-Angle* (θ, I)

(0). *Assume I is an interval with respect to (e, f) and has root r.*

(1). **(Trace angles back)** *Starting with angle of incidence θ at edge e, determine the angles of incidence at each edge in the last edge sequence. This can be done by following back pointers to predecessors of intervals and using the fact that the angle of incidence at e_i uniquely specifies the angle of refraction from e_{i-1}, which in turn uniquely specifies the angle of incidence at e_{i-1}. It may be impossible at some stage to achieve the desired angle of refraction due to the constraint that the angle of incidence be less (in absolute value) than the critical angle; in this case, we stop and return NIL. This process will eventually give the angle ϕ such that a refraction path that starts from r at angle ϕ will refract through the last edge sequence and strike e at angle of incidence θ.*

(2). **(Ray trace forward)** *Now do ray tracing from r at angle ϕ, following the last edge sequence. If the refraction path leaves the sequence, then stop and return NIL. Otherwise, the refraction path will eventually cross edge e at some point x (and we know by construction that it will strike x at the desired angle of incidence θ). Let u be the first bend point and let w be the last bend point in the path from r to x. If $x \in I$, then return the list (x, u, w); otherwise, return NIL.*

6. Correctness Proof

We begin now a proof that the algorithm works. The proof largely follows the similar proof from [MMP], but we will point out the places where it differs. Our first objective is to prove the following proposition:

Proposition 6.1 *After each iteration of the Main Loop of the Algorithm, the current list of candidate intervals (ILIST) correctly gives the best locally f-free path so far from s to points of these intervals, where "best so far" has the following meaning: If $x \in e = f \cap f'$ lies in interval I for $(r, \mathcal{E})$ with respect to (e, f), then a locally f-free path to x that has root r, last edge sequence $\mathcal{E}$, and length $d(r) + d_{r,\mathcal{E}}(x)$ is optimal among those locally f-free paths to x that enter face f' through an interval whose frontier point has already been an event point (i.e., through an interval that has already been propagated).*

Proof: The proof proceeds by induction on the iteration count in the Main Loop. At the first iteration, the claim is true because the only candidate interval is the trivial one at the source s. Assume now that the claim holds for the first k iterations (this is the Induction Hypothesis). The inductive step requires us to prove that if, at iteration $k+1$, we introduce interval I for root r with respect to (e, f) or we modify an existing such interval, then the best locally f-free path so far to points of I is through root r and the corresponding last edge sequence. This will be shown in the following two lemmas. ∎

We now give two lemmas to complete the proof of Proposition 6.1. Assume now that the Induction Hypothesis holds (i.e., that the claim of Proposition 6.1 holds for the first k iterations).

A candidate interval will have one of two fates: Either its frontier point gets permanently labeled (by its becoming the top element in the priority queue), thereby assuring that some part of it will "survive" and become part of a final interval of optimality with the same attributes as the candidate; or, the candidate interval gets deleted in step (2) of *Insert-Interval*, in which case we can guarantee that the interval of optimality corresponding to the candidate will be empty.

Lemma 6.2 *If we delete interval I_1 in step (2) of procedure Insert-Interval, then the interval of optimality for $(r_1, \mathcal{E}_1)$ with respect to (e, f) has an empty interior. A similar statement holds if we delete interval I_2.*

Proof: We consider the case in which step (2) deletes I_1. A similar argument holds for the case of deleting I_2.

Assume that $a_1 \in I = [a, b]$ and $d_1 + d_{r_1,\mathcal{E}_1}(a_1) \geq d + d_{r,\mathcal{E}}(a_1)$. Then, the best locally f-free path so far to point a_1 is through root r and last edge sequence $\mathcal{E}$. Let w be the intersection of the refraction path from r through $\mathcal{E}$ to a_1 with edge e'. Since $a_1 \in I$, we know that $w \in \overline{I}$, by the method used to construct I from its predecessor, $\overline{I}$. Let us assume that there exists a point y in the interior of the candidate interval for $(r_1, \mathcal{E}_1)$ with respect to (e, f) (so the path $p_f(y)$ has root r_1 and last edge sequence $\mathcal{E}_1$). We will then arrive at a contradiction.

First, note that y is interior to $[a_1, b_1]$ (otherwise, y is either inaccessible from r_1 or can be reached from some other root of (e, f) along a shorter locally f-free path). Next, note that interval I_1 was established when its predecessor, $\overline{I}_1$, was propagated. Since point $\overline{c} \in \overline{I}$ was located in channel (I_1, I_2), we know that $\overline{c}$ lies to the *right* of the segment $\overline{\beta_1 c_1}$, and therefore to the right of the refraction path from r_1 to b_1 (otherwise, $\overline{c} \in \overline{I}_1$). Since $\overline{I}$ is connected, then, *all* points of $\overline{I}$ lie to the right of the refraction path from r_1 to z for *any* $z \in [a_1, b_1]$, and lie *strictly* to the right of the refraction path from r_1 to z for z *interior* to $[a_1, b_1]$. In particular, w lies strictly to the right of segment $\overline{w_1 y}$, the last segment in the refraction path from r_1 through $\mathcal{E}_1$ to y. But, a_1 lies strictly to the *left* of segment $\overline{w_1 y}$, so segments $\overline{w a_1}$ and $\overline{w_1 y}$ must intersect in some point γ that is interior to both segments (see Figure 5.3). Since γ lies on the path $p_f(y)$, the subpath of $p_f(y)$ from r_1 to γ is no longer than the subpath of $p_f(a_1)$ from r to γ. But this means that we can get a strict improvement to the path to a_1 by going through root r_1 (and $\mathcal{E}_1$) to a_1 (since $d_{r_1,\mathcal{E}_1}(\gamma) + \alpha_{f'}|\gamma a_1| > d_{r_1,\mathcal{E}_1}(w_1) + \alpha_{f'}|w_1 a_1|$). This contadicts our assumption that a_1 is reached better through r than through r_1. Thus, no such y interior to the candidate interval for $(r_1, \mathcal{E}_1)$ with respect to (e, f) can exist, and we were justified in deleting this candidate interval. ∎

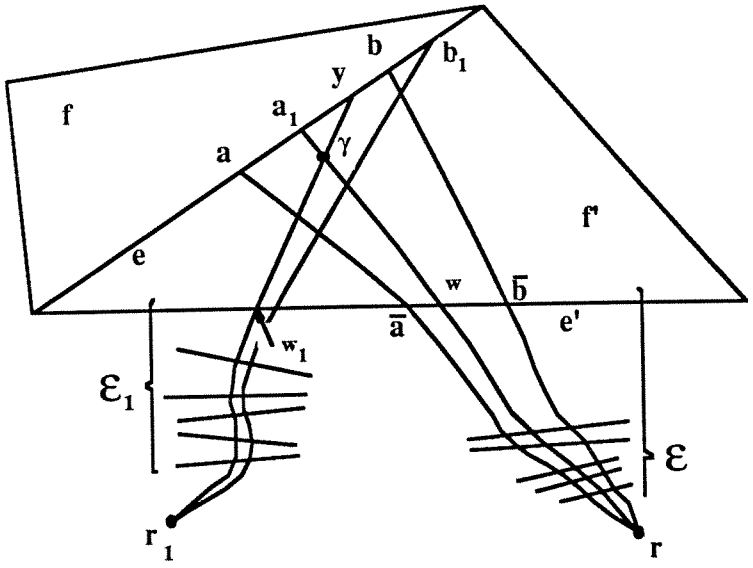

Figure 6.1. Illustration of proof of Lemma 6.2.

Next, we must prove that the interval that is inserted by procedure *Insert-Interval* is "correct" in the following sense:

Lemma 6.3 *When procedure Insert-Interval inserts an interval I between intervals I_1 and I_2, then the resulting interval list of edge e correctly subdivides points of e according to the best locally f-free paths so far.*

Proof: The proof remains largely the same as that of Lemma 6.3 from [MMP]. One change is that we must check that the trimming is done properly. We now have the added complication of trimming according to the constraints imposed by the critical angles (that an angle of incidence cannot exceed in absolute value the relevant critical angle). This trimming is done in steps (3).(i)-(iii) of the function *Project*, before the candidate interval is ever passed to *Insert-Interval*. In the event that trimming was done at the critical angle, and the corresponding critical point of entry survives the trimming at tie points (step (3) of *Insert-Interval*), we must call for an insertion of the critical point as a root in step (4) of *Insert-Interval*.

Another change is that measurements of lengths (from a given root through a given edge sequence) to points, such as a_1 in step (2) or tie points in step (3), and determinations of paths to these points (including

information like the first and last bend points) must be done by the functions *Find-Point* and *Find-Tie-Point*. As we discuss in Section 8, there are numerical issues involved with these functions that require that the numbers they return are precise only to a given tolerance.

The remainder of the proof follows closely that of Lemma 6.3 of [MMP], so we omit them here. ∎

This completes the proof of Proposition 6.1.

One can show that during an execution of Dijkstra's algorithm, when a node is "permanently labeled" (i.e., put on the "CLOSED" list), it is labeled with the length of the shortest path from the source to the node. Analogously, we have:

Lemma 6.4 *When a point interior to an edge is "permanently labeled" as part of a candidate interval with respect to (e, f), it is labeled with the shortest locally f-free path length to it. When a vertex is permanently labeled, it is labeled with the shortest path length to it.*

Proof: The proof is very similar to that of Lemma 6.4 of [MMP], so we omit it here. ∎

In our continuous Dijkstra framework, we think of a "wavefront" moving out from s and imagine that, as the front encounters new edges or boundaries of candidate intervals, events occur. When an event occurs at a certain distance from s, then all points closer to s than that distance have already encountered the "wavefront":

Lemma 6.5 *If the algorithm has just assigned a permanent label of δ to some point, then for any point x on an edge e such that $d(x) < \delta$, there is an interval I (for $(r, \mathcal{E})$ with respect to (e, f)) in the current ILIST that contains x such that an optimal path to x has root r, last edge sequence $\mathcal{E}$, and length $d(r) + d_{r,\mathcal{E}}(x)$.*

Proof: Again, the proof essentially follows that of the similar lemma (Lemma 6.5) of [MMP]. ∎

At the conclusion of our algorithm, the wavefront has encountered *all* points of $\mathcal{S}$:

Lemma 6.6 *At the conclusion of the algorithm, the interval list of (e, f) forms a covering of the edge e.*

Proof: Assume that there is some point $x \in e = f \cap f'$ that is *not* covered by a candidate interval with respect to (e, f). Let $p_f(x)$ be a shortest f-free path to x. Let $I_l = [a_l, b_l]$ be the *last* candidate interval intersected by $p_f(x)$ (there must be at least one), and assume that I_l is an interval with respect to (e_l, f_l). Since x is not covered, $e_l \neq e$. Let $\overline{\gamma x'} = f_l \cap p_f(x)$. Then, point x' lies on an edge $e_0 = f_l \cap f_0$. Let $p_{f_0}(x')$ be the subpath of $p_f(x)$ from s to x'. Clearly, $p_{f_0}(x')$ is a shortest f_0-free path to x'. The interval I_l *must* have been propagated (otherwise, its frontier point would still be in the event queue, and the algorithm would not have concluded). But then, by Lemma 6.3, there would have been a candidate interval created on edge e_0 that includes point x'. This contradicts the fact that I_l is the *last* candidate interval intersected by $p_f(x)$. ∎

Proposition 6.7 *The following are equivalent for a point x on edge e of face f:*

(1) *At the conclusion of the algorithm, x lies in the candidate interval for $(r, \mathcal{E})$ with respect to (e, f).*

(2) *There is a shortest locally f-free path p from s to x with root r and length $d(r) + d_{r,\mathcal{E}}(x)$; that is, x lies in the interval of optimality for $(r, \mathcal{E})$ with respect to (e, f).*

Proof: The proof follows immediately from Proposition 6.1 and Lemma 6.6. ∎

7. Complexity Analysis

We will write the complexity of our algorithm in terms of E, the number of events (candidate intervals of optimality). We turn now to the issue of proving a polynomial worst-case upper bound on E. It would be nice if we could prove that E is bounded by $O(n^2)$, as was the case in the DGP, but this is not true. An example is given in Figure 7.1 of a case in which there are $\Omega(n^3)$ intervals of optimality. We originally thought ([Mil,MP]) we had a proof that $O(n^3)$ is also an upper bound on the number of intervals of optimality, but, as shown in Figure 7.2, one can use the gadget of Figure 7.1 together with a gadget like Figure 3.1 to get a lower bound example in which there are $\Omega(n^4)$ intervals of optimality. In this section we will prove that the total number of intervals of optimality is at most $O(n^4)$ (which, by the example of Figure 7.2, is a tight

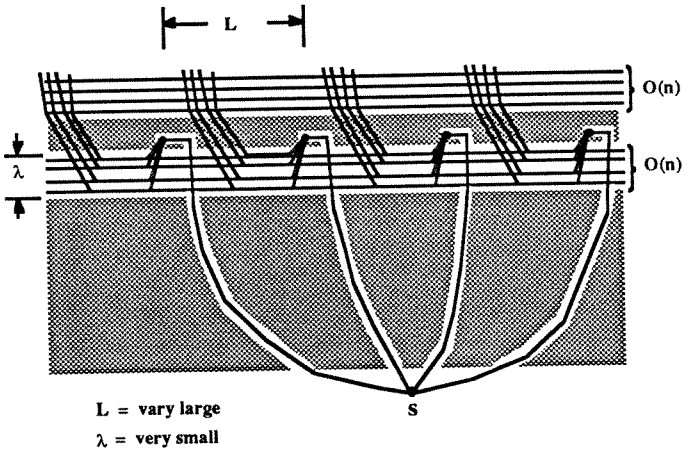

Figure 7.1. There may be $\Omega(n^3)$ intervals of optimality.

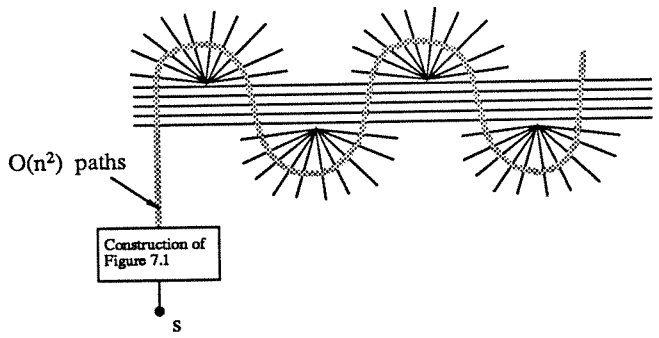

Figure 7.2. There may be $\Omega(n^4)$ intervals of optimality.

bound in the worst case), which also implies that the maximum number of candidate intervals (and hence events) is $O(n^4)$.

The lower bound of Figures 7.1-7.2 is based on the fact that there can be a quadratic number of *critical* roots. This implies that the argument given in [MMP] that vertices will be "trapped" in between pairs of representative paths to intervals of optimality must be modified for a couple of reasons. First, paths can be critically reflected and can critically use an edge, meaning that two paths to distinct adjacent intervals may not trap a vertex (as is the case for the example in Figure 7.1). Also, because shortest locally f-free paths *are* allowed to cross (although, by Lemma 4.2 they can do so only in a limited fashion), we are not able to proceed as in [MMP] by drawing a set of locally f-free paths to an edge (with one path per interval) and looking at the imposed planar subdivision. Since two paths may cross, the region "between" them is undefined.

A further complication in our case is that a shortest locally f-free path may cross an edge many times, implying that the length of an edge sequence crossed by a path may be very long (since there can be many occurences of each edge). Our first lemma shows a linear bound on the number of times any one edge can be crossed by a subpath, p, of a shortest locally f-free path, where p has no vertices or critical points (meaning that it crosses an edge sequence at crossing points internal to the edges of the sequence). Since it is possible for an edge to be crossed a linear number of times by an optimal path (as in the example of Figure 7.2), our bound is as tight as possible.

Lemma 7.1 *Let $\overline{p}$ be a shortest locally $\overline{f}$-free path. Let p be a subpath of $\overline{p}$ such that p goes through*

no vertices or critical points. Then, p can cross an edge e at most $O(n)$ times. Thus, in particular, the last edge sequence of $\overline{p}$ contains each edge $O(n)$ times, implying a bound of $O(n^2)$ on its length.

Proof: Let $e = f \cap f'$. Call a crossing point x_i between p and e an "up-crossing" if p is directed from f' to f when crossing at x_i. We will show that there are at most $O(n)$ up-crossing points. A similar argument shows that there are only $O(n)$ down-crossing points.

Let the up-crossings be labelled $x_1, \ldots, x_k$ along e from left to right when facing f. We group the points into triples: $(x_1, x_2, x_3), (x_4, x_5, x_6), \ldots$. Clearly, if we can show a linear bound on the number of triples, we are done.

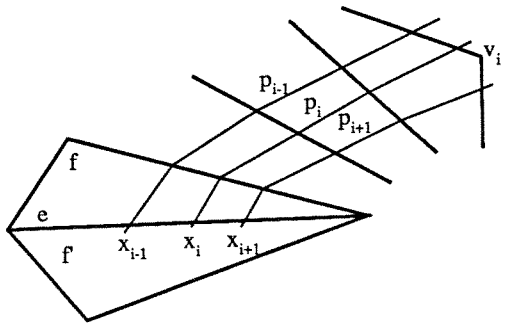

Figure 7.3. Vertex v splits paths p_{i-1}, p_i, p_{i+1}.

We now describe a "charging" scheme to assign triples of up-crossings to vertices of the subdivision. Consider a triple (x_{i-1}, x_i, x_{i+1}). We place three "markers" at points x_{i-1}, x_i, x_{i+1} and move the markers along the subpaths of p that start at points x_{i-1}, x_i, x_{i+1}. We advance the markers across edges in synchronization, noting the first time that the three markers fail to cross the same edge (which must eventually happen). When they fail to cross the same edge, there must be a vertex v_i that "splits" the three subpaths (two on one side of v_i, and one on the other side of v_i). Refer to Figure 7.3. (We have not attempted to draw Figures 7.3-7.8 to be accurate with respect to obeying the local optimality conditions at points where paths cross edges. The figures are meant only to give a qualitative understanding of the situation.)

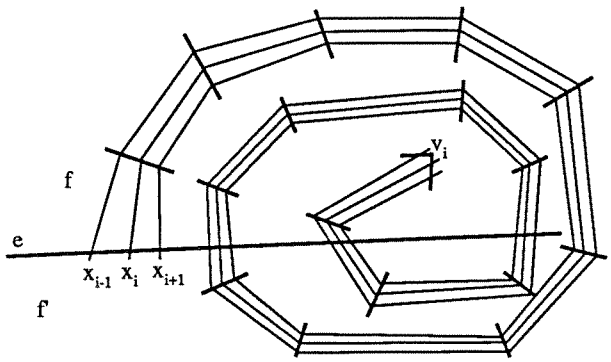

Figure 7.4. Vertex v_i is potentially charged many times.

We would like to charge the triple (x_{i-1}, x_i, x_{i+1}) to the vertex v_i, but we note that the vertex v_i may be charged by many triples, as in Figure 7.4. The problem is that during the advancement of the three markers, we may have crossed e several times before being split by vertex v_i. We must devise a method of "off-loading" such charges to other vertices of the subdivision so that no vertex is charged more than a constant number of times.

First, if the three markers get split before they are ever advanced across e, then we can go ahead and charge (x_{i-1}, x_i, x_{i+1}) to the vertex v_i. We say that v_i is charged BLUE in this case. It is easy to see that a vertex v_i can be charged BLUE at most once.

If, on the other hand, the markers cross e one or more times before being split, then we must do some more work. First, we need some more notation. Let p_{i-1}, p_i, p_{i+1} be the subpaths of p that start at x_{i-1}, x_i, x_{i+1}, respectively, and continue until the first splitting vertex v_i. Notice that, since the paths p_{i-1}, p_i, and p_{i+1} are disjoint subpaths of p, they have an ordering according to their positions as subpaths of p. There are thus $3! = 6$ cases associated with the triple (x_{i-1}, x_i, x_{i+1}), according to all possible orderings.

Refer to the triple of paths p_{i-1}, p_i, p_{i+1} as a *strip*. It may be that this strip weaves itself back and forth, around and around, across e many times. However, there is some limit to the amount of craziness, since, after all, these paths must all be optimal (shortest locally $\overline{f}$-free).

We refer to the subset of a strip that lies between two consecutive crossings of e as a *substrip*. For a substrip σ, the portion of path p_i that lies along it will be denoted $p_i(\sigma)$. Note that $p_i(\sigma)$ naturally defines a simple polygon, denoted $P_i(\sigma)$, whose boundary consists of $p_i(\sigma)$ together with the subsegment of e that joins the endpoints of $p_i(\sigma)$. We say that path $p_i(\sigma)$ is *outside* of path $p_{i-1}(\sigma)$ along a substrip σ if $P_{i-1}(\sigma)$ is nested inside of $P_i(\sigma)$.

Fact 1 *Assume that p_{i-1} precedes p_i in the ordering along p. Let σ and σ' be two adjacent substrips, with σ preceding σ'. Then it is not possible for $p_i(\sigma)$ to be outside of $p_{i-1}(\sigma)$ and for $p_{i-1}(\sigma')$ to be outside of $p_i(\sigma')$.*

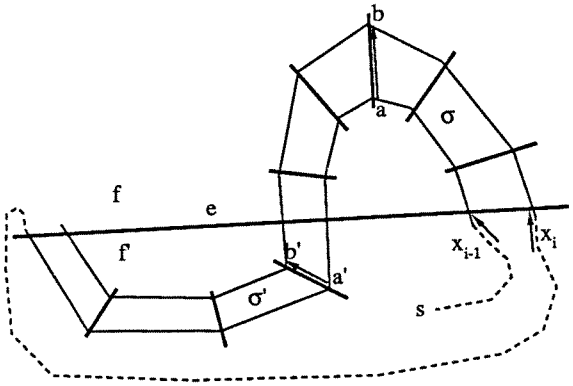

Figure 7.5. Proof of Fact 1.

Proof: Assume that it is possible to have the situation that $p_i(\sigma)$ is outside of $p_{i-1}(\sigma)$ and $p_{i-1}(\sigma')$ is outside of $p_i(\sigma')$. Refer to Figure 7.5. Along substrip σ (resp., σ'), let α (resp., α') be the weight of the cheapest region crossed by the substrip, and let $\overline{ab}$ (resp., $\overline{a'b'}$) be a chord in the cheapest region that joins path p_{i-1} to path p_i. Let $\ell_i(\sigma)$ (resp., $\ell_{i-1}(\sigma')$) be the weighted length of the path $p_i(\sigma)$ (resp., $p_{i-1}(\sigma')$). Note that $\alpha|\overline{ab}| \leq \ell_i(\sigma)$ (resp., $\alpha'|\overline{a'b'}| \leq \ell_{i-1}(\sigma')$), since the (unweighted) length of $\overline{ab}$ (resp., $\overline{a'b'}$) is certainly less than the (unweighted) length of $p_i(\sigma)$ (resp., $p_{i-1}(\sigma')$), and the weight α (resp., α') was chosen to be the *cheapest* weight in the substrip σ (resp., σ').

This means that if we "shortcut" p by going from a to b, we "save" $\ell_{i-1}(\sigma') \geq \alpha'|\overline{a'b'}|$ on the (weighted) length of p, while if we "shortcut" p by going from a' to b', we "save" $\ell_i(\sigma) \geq \alpha|\overline{ab}|$. Thus, by taking the better of the two shortcuts, we can improve the path p, a contradiction. ∎

The above fact implies that a strip cannot weave "back and forth" many times across e, since this would imply for some pair of subpaths that the earlier path (the one that occurs earlier in the ordering along p) would go from being inside to being outside of the other subpath. (The reader can check each of the 6 cases of the relative orderings of the three subpaths.) Thus, the only real worry we have is that of spirals, as in Figure 7.4.

We handle spirals by devising a method of charging them off to other vertices, as follows. Assume that p_{i-1} precedes p_i and p_i precedes p_{i+1} in the order along p; the other 5 cases are handled similarly. Then, Fact 1 tells us that the only situation we have to worry about is that the subpaths may spiral clockwise, crossing e many times, and then spiral counterclockwise, crossing e many times. (A switch from going counterclockwise to clockwise would violate Fact 1.) A clockwise spiral means, more precisely, that there is a sequence of substrips, $\sigma_1, \sigma_2, \ldots, \sigma_m$, such that $p_{i-1}(\sigma_j)$ is outside of $p_i(\sigma_j)$ for all $j = 1, 2, \ldots, m$.

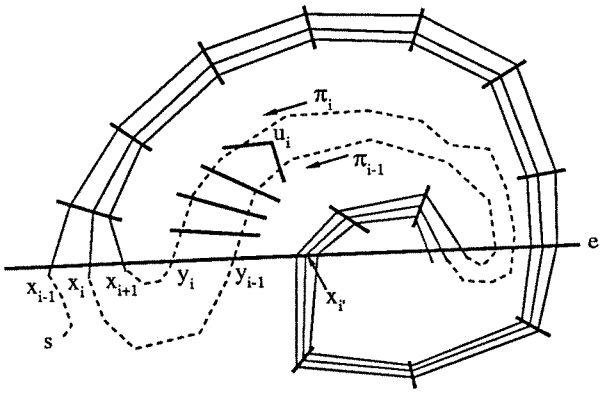

Figure 7.6. Handling spirals.

We must charge each crossing of e by the strip to some vertex, so that the vertex v_i that splits the strip is not charged over and over. Let us charge RED that vertex u_i obtained as follows. (Refer to Figure 7.6.) Since p_{i-1} and p_i precede p_{i+1} in their order along p, there must be subpaths of p that join the end of p_{i-1} to the beginning of p_i (at point x_i) and that join the end of p_i to the beginning of p_{i+1} (at point x_{i+1}); call these paths π_{i-1} and π_i, respectively. By the simplicity of p (Lemma 4.2), we know that paths π_{i-1} and π_i are "trapped" between two of the substrips of the spiral: that substrip originating at (x_{i-1}, x_i, x_{i+1}), and that substrip originating at $(x_{i'-1}, x_{i'}, x_{i'+1})$, in the figure. Let y_{i-1} and y_i be the points of e where π_{i-1} and π_i (respectively) cross the subsegment of e between x_{i+1} and $x_{i'-1}$. We place two markers on y_{i-1} and y_i and advance them *backwards* (in reverse order along the paths) along π_{i-1} and π_i until the first time that they are "split" by some vertex u_i. We charge u_i with a RED charge. The following facts assure that we do not charge a vertex RED more than once.

Fact 2 *The vertex u_i will be encountered before the advancement along π_{i-1} and π_i takes the two markers across e.*

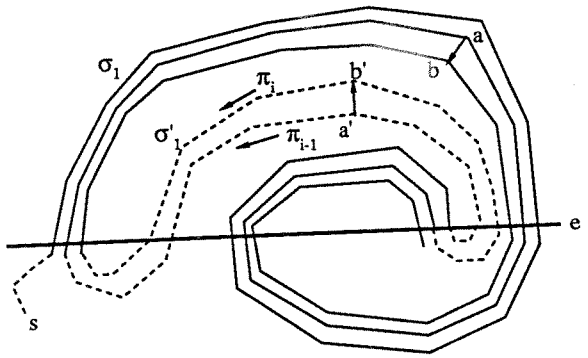

Figure 7.7. Proof of Fact 2.

Proof: If the claim is false, then we have the situation depicted in Figure 7.7. Let $\overline{ab}$ be a chord through the cheapest region crossed by substrip σ_1, joining path $p_i(\sigma_1)$ to path $p_{i-1}(\sigma_1)$¿ Let $\overline{a'b'}$ be a chord through the cheapest region crossed by the substrip σ_1' (defined by the pair of paths π_i and π_{i-1}, from points y_i and y_{i-1} to the next crossing of e tracing backwards), joining path π_{i-1} to path π_i. Then, using shortcut $\overline{ab}$ saves the weighted length of π_i along substrip σ_1' (which is greater than the weighted length of $\overline{a'b'}$), while using shortcut $\overline{a'b'}$ saves the weighted length of $p_i(\sigma_1)$ (which is greater than the weighted length of $\overline{ab}$). Thus, by using the shorter of the two shortcuts, we can shorten the path p. ∎

Fact 3 *The vertex u_i will be charged RED at most once.*

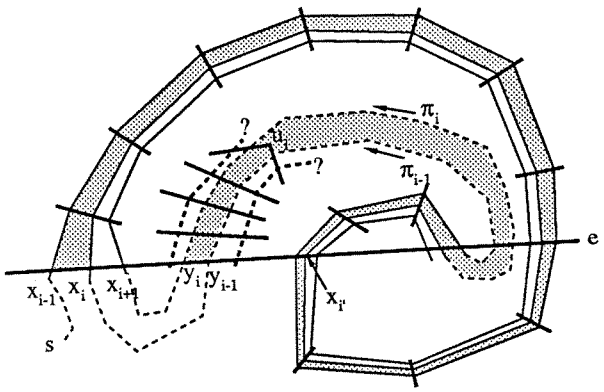

Figure 7.8. Proof of Fact 3.

Proof: Refer to Figure 7.8. If u_i is charged also by the triple (x_{j-1}, x_j, x_{j+1}), then the corresponding return paths π_j and π_{j-1} (shown as bold dashed paths) must either be outside of the substrip defined by π_i and π_{i-1} (as shown in the figure), or inside the substrip, or have one inside and one outside. In any case, since we know that p is simple, we will discover that at least one of the pairs of upcrossing points (x_i, x_{i+1}) or (x_j, x_{j+1}) are not consecutive, in violation of their definition. In the figure, we have shaded the simple polygon defined by the segments $\overline{x_i x_{i+1}}$, $\overline{y_i y_{i-1}}$, the subpath of p from x_i to y_{i-1}, and the subpath of p from x_{i+1} to y_i. ∎

In conclusion, we have given a means of charging each triple (x_{i-1}, x_i, x_{i+1}) to some vertex of the subdivision, either as a BLUE (splitting) vertex or as a RED vertex, and each vertex of the subdivision is charged BLUE (resp., RED) at most once. This implies that there are only $O(n)$ triples, so we are done. ∎

We suspect that, by more strongly using metric properties of shortest locally f-free paths, one could considerably simplify the proof of Lemma 7.1. We leave this as an open problem.

We define now the notion of a *fork point* between two paths, p_x and p_y, incident on points x and y interior to edge e. The fork point is defined as follows. We place "markers" at x and y and march backwards along the paths, advancing the markers through one face at a time on each path, and we stop when the markers first fail to cross the interior of the same edge. There are three ways in which this can happen: (1). the vertex root of one of the paths is encountered, (2). the paths get "split" by a vertex v (meaning that they cross the interiors of two different edges adjacent to v), or (3). a critical point of exit of one of the paths is encountered. In case (1) we define the fork point to be the root encountered. If the paths get split by a vertex (case (2)), then the vertex, v, that separates them is the *fork point separating* p_x and p_y, and we say that v is a *fork vertex*. Otherwise, our backward march stopped because of a critical point of exit for one of the paths, in which case we take the fork point to be the root corresponding to the critical point of exit. (The notion of fork point was used in the charging scheme of the proof of Lemma 7.1, where we had the additional information that there were no critical points or vertices along the paths.)

Our strategy for proving a bound of $O(n^4)$ on the number of events is to show that there can be at most $O(n^3)$ intervals of optimality per edge-face pair (e, f). This is done by showing that there can be only $O(n^2)$

intervals of optimality on edge $e = \cap(f', f)$ that correspond to *first* incidences of shortest locally f-free paths on e. Then, we appeal to Lemma 7.1 to conclude that there can be only $O(n^3)$ intervals on e, since each of the $O(n)$ shortest locally f-free paths that encounter e can intersect e at most $O(n)$ times.

Let (e, f) be an edge-face pair, where $e = \cap(f', f)$. Consider the set of all intervals of optimality with respect to (e, f). Some of these intervals correspond to shortest locally f-free paths whose first encounter with edge e is at the points of that interval (i.e., the edge e was not crossed by the paths to points of the interval). Let $x_1, x_2, \ldots, x_K$ be points interior to each of these special intervals, in sorted order along e. Our goal is to show that $K = O(n)$. This is done in the next two lemmas.

Lemma 7.2 *If the shortest locally f-free paths, $p_f(x_i)$ and $p_f(x_{i+1})$, share a common root r, then there must be a fork vertex v_i separating them, and v_i is not the fork vertex corresponding to any other pair $(p_f(x_j)$ and $p_f(x_{j+1})$, $j \neq i)$ of shortest locally f-free paths that share a common root. (That is, $v_i \neq v_j$ for $i \neq j$.)*

Proof: If there were no such fork vertex, then since the paths $p_f(x_i)$ and $p_f(x_{i+1})$ share the same root, they must have the same last edge sequences. (By the definition of the root of a path, the subpaths from r to x_i and to x_{i+1} do not pass through any vertices or critical points. Thus, they can traverse an edge only by going through its interior at a crossing point. If we had defined the root to be the last vertex along the path (regardless of critical points), then the claim would no longer be true, for there can be many paths with the same last vertex while there is no fork vertex trapped between the paths (refer to Figure 7.1).) But if $p_f(x_i)$ and $p_f(x_{i+1})$ have the same roots and the same last edges sequences, then x_i and x_{i+1} must belong to the same interval of optimality, a contradiction.

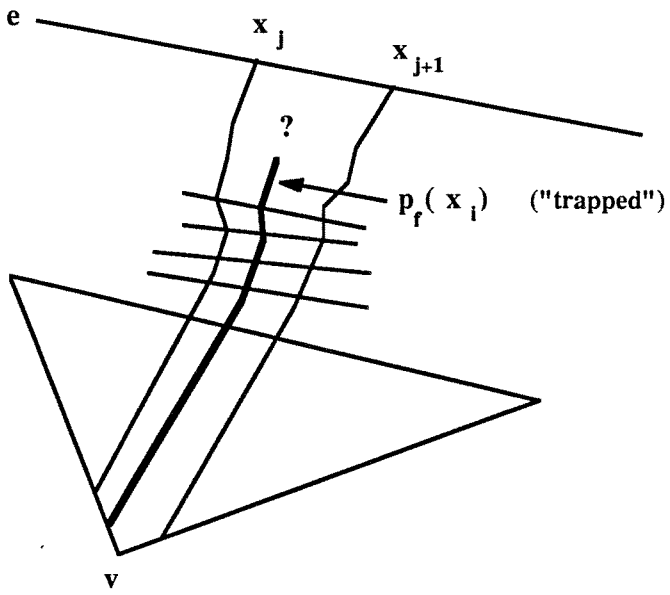

Figure 7.9. Uniqueness of fork vertex.

To show uniqueness, assume to the contrary that v is a fork vertex between $p_f(x_i)$ and $p_f(x_{i+1})$ and also between $p_f(x_j)$ and $p_f(x_{j+1})$. Refer to Figure 7.9. Then at least one of the four paths (say, $p_f(x_i)$) is "trapped" between the other pair of paths ($p_f(x_j)$ and $p_f(x_{j+1})$) in the following sense: from the segment of $p_f(x_i)$ in the face containing v where both pairs of paths diverged until the segment of $p_f(x_i)$ that is incident to e at point x_i, $p_f(x_i)$ is not allowed to cross either of the paths $p_f(x_j)$ and $p_f(x_{j+1})$, since to do so would violate Lemma 4.2. But this means that x_i lies between x_j and x_{j+1} on edge e (since x_i is the *first* point along $p_f(x_i)$ where the path intersects e), violating the ordering of the points $x_1, x_2, \ldots, x_K$ along the edge e. ∎

Similarly, we get the following claim about the uniqueness of a fork point that is a root:

Lemma 7.3 *If r is the root of $p_f(x_i)$ and $p_f(x_j)$ and is a fork point between $p_f(x_i)$ and $p_f(x_{i+1})$ (implying that $j \neq i+1$), then r is* not *a fork point between $p_f(x_j)$ and $p_f(x_{j+1})$.*

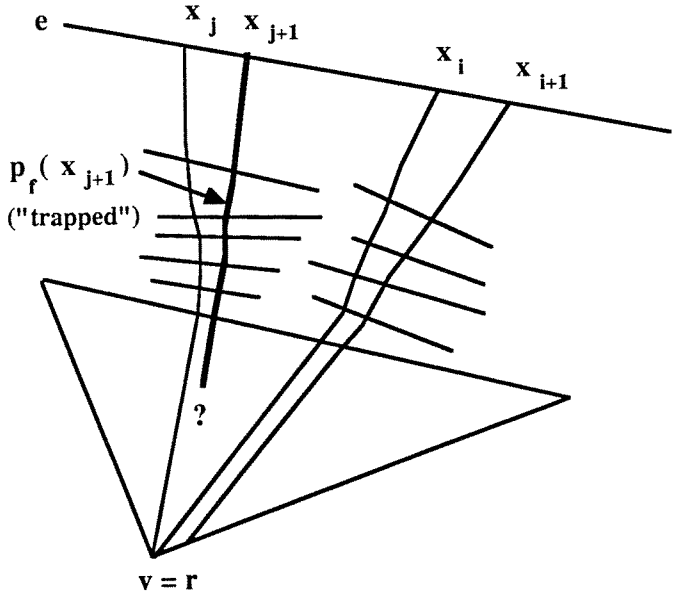

Figure 7.10. Illustration of Lemma 7.3.

Proof: Assume that r is a vertex (the case in which r is a critical root is similar). Since r is a fork point between $p_f(x_i)$ and $p_f(x_{i+1})$, it must be the case that $j < i-1$, so that x_j precedes x_i along the oriented edge $e = \cap(f', f)$. Refer to Figure 7.10.

If r were also a fork point between $p_f(x_j)$ and $p_f(x_{j+1})$, then the path $p_f(x_{j+1})$ would not be able to connect to x_{j+1} without intersecting $p_f(x_i)$: $p_f(x_{j+1})$ cannot have root r (otherwise, the fork point between $p_f(x_j)$ and $p_f(x_{j+1})$ would be a vertex, by Lemma 7.2), and it cannot have a root between x_{j+1} and the segment on the face where it gets separated from $p_f(x_j)$ (since this would violate the definition of fork point). ∎

We can now show that the number of critical points of entry (that is, the number of critical roots) is bounded.

Lemma 7.4 *There are at most $O(n)$ critical points of entry on any given edge e.*

Proof: Let $x_1, \ldots, x_K$ be the critical points of entry to edge e from the side of face f' ($e = f \cap f'$, and we must have $\alpha_e < \alpha_{f'}$ for critical points of entry to exist). Consider shortest locally f-free paths from s to each x_i ($1 \leq i \leq K$). For each pair of paths $p_f(x_i)$ and $p_f(x_{i+1})$, we "charge" the fork point between them. By reasoning similar to the proof of Lemma 7.2 above, we can see that each fork vertex can be charged at most once. Also, as in Lemma 7.3 above, we can see that a root can be charged as a fork point at most twice: once for being the root of the left path of some pair (x_i, x_{i+1}), and once for being the root of the right path of some pair (x_j, x_{j+1}). We make critical use of Lemma 3.7 here, for it allows us to claim that the root of $p_f(x_i)$ is a vertex for each i. This implies that there are $O(n)$ roots and $O(n)$ vertices, and hence that $K = O(n)$. An identical argument holds for critical points of entry to e from the side of face f. Thus, there are $O(n)$ entry points in all. ∎

Thus, the *total* number of roots cannot be more than $O(n^2)$, since there are a linear number of vertices, plus there are a linear number of critical points of entry per edge. We can now prove our overall bound on the number of intervals of optimality.

Lemma 7.5 *There are at most $O(n^4)$ candidate intervals created by the algorithm.*

Proof: For edge-face pair (e, f), let $I_1, I_2, \ldots, I_K$ be the intervals that correspond to paths whose *first incidence with e after the root* takes place in the interval; in other words, the intervals $I_1, I_2, \ldots, I_K$ are the subset of all intervals with respect to (e, f) such that e occurs only *once* in the last edge sequence of each I_i. Let x_i be a point interior to interval I_i. Consider shortest locally f-free paths from s to each x_i $(1 \leq i \leq K)$. For each pair of paths $p_f(x_i)$ and $p_f(x_{i+1})$, we "charge" the fork point between them. We know from Lemma 7.2 that each fork vertex can be charged at most once. Also, from Lemma 7.3 we know that a root can be charged as a fork point at most twice. (once for being the root of the left path of some pair (x_i, x_{i+1}), and once for being the root of the right path of some pair (x_j, x_{j+1})). Since there are $O(n^2)$ roots and $O(n)$ vertices, this shows that $K = O(n^2)$. Now, each of these $O(n^2)$ paths can go on to intersect e in many other places (at intervals of optimality that do not correspond to first incidences), but we know from Lemma 7.1 that each such subpath can be incident at most $O(n)$ times with e, since these subpaths can contain no vertices or critical points (since they cross last edge sequences). This implies that e can have at most $O(n^3)$ intervals of optimality in all, implying that the total number of intervals of optimality for all $O(n)$ edge-face pairs is $O(n^4)$. Note that the example in Figure 7.2 shows that this upper bound can be achieved in the worst case. ∎

The above lemma allows us to guarantee that there are at most $O(n^4)$ event points. It also allows us to bound the total number of calls to procedure *Delete* and to functions *Find-Point* and *Find-Tie-Point*:

Lemma 7.6 *There will be at most $O(n^4)$ calls to procedure Delete and to functions Find-Point and Find-Tie-Point.*

Thus, the size of the data structures needed is easily seen to be at most $O(n^4)$:

Lemma 7.7 *The data structures require at most $O(n^4)$ space.*

We are finally ready to assert the validity of our algorithm and its claimed efficiency:

Theorem 7.8 *The algorithm of Section 5 correctly computes the subdivision of each edge into intervals of optimality in $O(E)$ "steps", where E is the number of events and is bounded by $O(n^4)$ in the worst case. Each step involves an $O(\log n)$ binary search (to locate point $\overline{c}$ in an access channel in step (1) of* Insert-Interval*) and (possibly) a call to function* Find-Point *and function* Find-Tie-Point. *The data structures require $O(E) = O(n^4)$ space.*

Thus, the total complexity of our algorithm will be $O(E(\log n + F(n) + F'(n)))$, where $F(n)$ (resp., $F'(n)$) is the complexity of calling function Find-Point (resp., Find-Tie-Point). Note that the algorithm is output-sensitive, in that its running time depends on E (the number of events), rather than always requiring $\Omega(n^4)$ steps. In practice, we suspect that E will be much less than its worst-case upper bound of $O(n^4)$.

8. Numerical Issues

We have written the complexity of our algorithm in terms of $F(n)$ and $F'(n)$, the complexity of calling functions *Find-Point* and *Find-Tie-Point*. We now analyze these complexities.

The function *Find-Point* (I, x) determines a path from r, the root of I, to x that connects the last edge sequence of I. It would be great if we could calculate exactly the refraction path from r to x, but this problem may be intractible. We can perform ray tracing from r starting in a known direction, but we cannot solve exactly the inverse problem of finding the direction at which to start a ray tracing in order to hit a given point x. The problem is that when we write down the equations that represent the local optimality criterion (Snell's Law) at each edge, and then try to solve for the hit points along the path, we get k quartic equations in k unknowns (where k is the length of the edge sequence through which the refraction path is known to pass). Elimination among these equations yields a very high degree polynomial (of degree doubly exponential in k). Alternatively, we could apply the cylindrical decomposition technique of Collins [Co] to arrive at a doubly exponential time procedure to determine whether or not a given rational path length is achievable. This would also provide us with a technique of comparing the lengths of paths through two different edge sequences, and thus would solve the entire problem precisely in doubly exponential time (giving as output the sequence of edges and vertices along the optimal path). The same technique led [SS] to a doubly exponential time solution to the three-dimensional combinatorial shortest path problem.

> *Remark:* In the discrete geodesic problem of [MMP], the procedures *Find-Point* and *Find-Tie-Point* were not necessary, since we could solve the problems in constant time by keeping the "unfolded" image of the root of each interval of optimality. More specifically, we associate with each interval of optimality a point $\overline{r}$ (the "unfolded root"), which is the image of the root r obtained by "flattening" the polyhedron along the last edge sequence. This allows one to compute (geodesic) distances from the root r to points x on the interval of optimality without having to "unfold" each of the paths from r to various points x; instead, we simply compute (in constant time) the distance from the unfolded root $\overline{r}$ to point x. We have attempted a similar approach in the WRP, namely that of keeping an image of a curve whose tangent lines yield the path rays that propagate through a given interval, but have had no success. The problem is that we are not able to show that the degree of the resulting curves is polynomially bounded.

Let us emphasize that, in practice, a very straightforward numerical approach to solving *Find-Point* would be used. For example, one technique would be to begin with a path from r to v that connects the edges in $\mathcal{E}$ along their midpoints. We then iteratively shorten the path by applying the local optimality criterion to adjacent segments of the path. We simply pick an edge of $\mathcal{E}$ at which Snell's Law is violated, and then let the hit point "slide" along the edge until Snell's Law is obeyed. Each iteration can be computed in constant time and results in a strict decrease in the weighted path length. (This algorithm is just a coordinate descent method.) Hence, this procedure will converge to the locally optimal path from r to v. (This type of numerical procedure is also applicable to the three-dimensional shortest path problem, in which the local optimality criterion is that paths unfold to be straight. See [BM].) The problem with this approach is that we do not have good bounds on the rate of convergence or on the number of iterations necessary to guarantee that the solution is close (say, within $\epsilon\%$) to optimal. In practice, however, the convergence rate has been observed to be extremely fast. In fact, in some recent tests, Karel Zikan [Zi] has found that the floating point precision of a Symbolics 3640 Lisp Machine is exhausted after approximately $10k$ iterations (that is, an average of about 10 descents per coordinate). He has observed even faster rates of convergence after implementing certain "overrelaxation" rules.

In order to get a *theoretical* bound on the complexity of calling *Find-Point*, we write a formal function below. Rather than producing an exact answer, however, we give a path whose length is at most $(1 + \epsilon)$ times the length of the optimal path, where $\epsilon \in (0, 1)$ is a user-specified tolerance. (We can allow $\epsilon > 1$ in our algorithm, but it makes some of the analysis below messier in order to avoid having the term $\log \frac{1}{\epsilon}$ be negative.) Our function does a binary search to try to determine $u_x \in [u_a, u_b]$ such that the refraction path from r through u_x will pass through point x. It stops when it gets close to finding u_x (trapping it in a very small interval of size at most δ, a parameter that will be described later), pretends that u_x is exactly one of the endpoints of the interval, and then proceeds to search from u_x through the remaining edges of $\mathcal{E}$. Thus, we advance from edge to edge along $\mathcal{E}$, finding an approximate refraction path to x.

The function returns a list, (λ, u, u', w), where λ is the length of the path from the root, r, of I to x, $[u, u']$ is a range of first bend points (for the path that it finds from r to x) that is very small (small enough to guarantee that λ is within tolerance ϵ of optimality), and w is the corresponding last bend point of the path to x.

Function *Find-Point* (I, x)

(0). *Assume that $I = [a, b]$ is a candidate interval with respect to (e, f). Let $\mathcal{E} = (e_1, \ldots, e_{k-1}, e_k = e)$ be the last edge sequence of I (which can be computed in time $O(k)$ by backtracing), appended with the edge e that contains I, and let r be the root of I. We assume that $x \in f$. Furthermore, let $[u_a, u_b]$ be the subsegment of e_1 defined by the first bend points of I. (Refer to Figure 8.1.) Set $j = 1$, $a_1 = u_a$, and $b_1 = u_b$.*

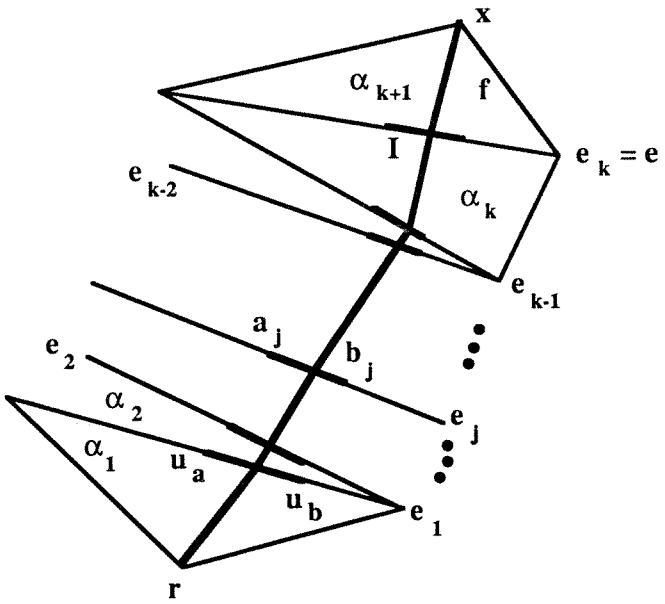

Figure 8.1. Illustration of *Find-Point*.

(1). **(Check interval size)** *If $|a_j b_j| < \delta$, then go to (3). Otherwise, go to (2).*

(2). **(Ray tracing)** *Let $u = (a_j + b_j)/2$ be the midpoint of segment $[a_j, b_j]$. Perform ray tracing from r starting with the initial ray through u. Stop the ray tracing when either the refraction path, p, leaves the edge sequence $(e_{j+1}, \ldots, e_k)$ (say, by missing edge e_i when it exits the face that contains e_{i-1} and e_i), or p emerges through edge e.*

(i). **(p emerges through e)** *If x lies to the left of the ray that emerges from $e_k = e$, then set $b_j = u$; otherwise, set $a_j = u$. Go to (1).*

(ii). **(p misses edge e_i)** *If p passes to the right of e_i, then set $b_j = u$; otherwise, set $a_j = u$. Go to (1).*

(3). **(Advance along $\mathcal{E}$)** *If $j = k$, then go to (4). Otherwise, let r_j be the midpoint of $[a_j, b_j]$. Let a_{j+1} (resp., b_{j+1}) be the point on edge e_{j+1} that is hit by the ray emerging from e_j when ray tracing is done from r through a_j (resp., b_j). Set $r = r_j$ and $j = j + 1$. Go to (1).*

(4). **(Output path)** *Calculate the weighted Euclidean length, λ, of the path $(r, r_1, \ldots, r_k, x)$. Return the list (λ, a_1, b_1, r_k).*

Before proving the correctness of the above function, we need some additional notation. For $1 \leq j \leq k+1$, let $p(j)$ be the (weighted) length of the path from r_{j-1} through $(e_j, e_{j+1}, \ldots, e_k)$ to x produced by *Find-Point*, namely $p(j)$ is the length of $(r_{j-1}, \ldots, r_k, x)$. (We let $r_0 = r$ and $p(k+1) = \alpha_{k+1}|r_k x|$.) Let $p^*(j,t)$ be the length of the shortest path from $t \in int(e_{j-1})$ to x through $(e_j, e_{j+1}, \ldots, e_k)$, and let $\overline{tz^*(t)}$ be the first segment in the shortest path. (Thus, $p^*(k+1,t)$ is simply the length of the path (t,x), namely $\alpha_{k+1}|tx|$.)

The following lemma shows that if we take the parameter δ (which we used in step (1)) to be sufficiently small, then the "almost" refraction path produced by *Find-Point* has a length of at most $(1+\epsilon)$ times the length of the "true" refraction path from r to x.

Lemma 8.1 *Let $\delta = \frac{h\epsilon}{6k(W/w)}$, where h is the minimum height of a triangluar face of $\mathcal{S}$, W (resp., w) is the maximum finite (resp., minimum nonzero) weight assigned to faces of $\mathcal{S}$, and $\epsilon \in (0,1)$ is a user-specified error tolerance. Then the path $(r, r_1, \ldots, r_k, x)$ found in* Find-Point *has a weighted length of at most $(1+\epsilon)$ times that of the shortest path from r to x through $\mathcal{E}$.*

Proof: We must show that $p(1) \leq (1+\epsilon)p^*(1,r)$. We proceed by induction to prove that for $i = 1, 2, \ldots, k+1$,

$$p(i) \leq (1+\frac{\epsilon}{2})p^*(i, r_{i-1}) + 3(k-i+1)\delta W. \qquad (*)$$

First, note that, trivially, $p(k+1) = \alpha_{k+1}|r_k x| = p^*(k+1, r_k) \leq (1+\frac{\epsilon}{2})p^*(k+1, r_k)$, so that $(*)$ holds for $i = k+1$. Now assume that $(*)$ holds for $i = j+1$, namely, assume that $p(j+1) \leq (1+\frac{\epsilon}{2})p^*(j+1, r_j) + 3(k-j)\delta W$. We wish to show that the inequality holds for $i = j$. By the specification of the function *Find-Point*, we have that

$$\begin{aligned} p(j) &= \alpha_j|r_{j-1}r_j| + p(j+1) \\ &\leq \alpha_j|r_{j-1}r_j| + (1+\frac{\epsilon}{2})p^*(j+1, r_j) + 3(k-j)\delta W \\ &\leq \alpha_j|r_{j-1}r_j| + (1+\frac{\epsilon}{2})[p^*(j+1, z^*(r_{j-1})) + \alpha_j|r_j z^*(r_{j-1})|] + 3(k-j)\delta W, \end{aligned}$$

where the first inequality comes from the induction hypothesis, and the second is just the triangle inequality. Now, using the stopping criterion of step (2), we know that $|r_j z^*(r_{j-1})| \leq \delta$. This, plus another application of the triangle inequality, gives us

$$\begin{aligned} &\alpha_j|r_{j-1}r_j| + \alpha_j(1+\frac{\epsilon}{2})|r_j z^*(r_{j-1})| + 3(k-j)\delta W \\ &\qquad \leq \alpha_j|r_{j-1}z^*(r_{j-1})| + \alpha_j(2+\frac{\epsilon}{2})|r_j z^*(r_{j-1})| + 3(k-j)\delta W \\ &\qquad \leq \alpha_j(1+\frac{\epsilon}{2})|r_{j-1}z^*(r_{j-1})| + 3\delta W + 3(k-j)\delta W, \end{aligned}$$

which implies that

$$\begin{aligned} p(j) &\leq (1+\frac{\epsilon}{2})[p^*(j+1), z^*(r_{j-1})) + \alpha_j|r_{j-1}z^*(r_{j-1})|] + 3(k-j+1)\delta W \\ &= (1+\frac{\epsilon}{2})p^*(j, r_{j-1}) + 3(k-j+1)\delta W. \end{aligned}$$

By induction, we are done proving $(*)$. In particular, we have shown that $p(1) \leq (1+\frac{\epsilon}{2})p^*(1,r) + 3k\delta W$. Now, by our particular choice of δ,

$$\delta = \frac{h\epsilon}{6k(W/w)},$$

we get that

$$\begin{aligned} p(1) &\leq (1+\frac{\epsilon}{2})p^*(1,r) + 3k\delta W \\ &= (1+\frac{\epsilon}{2})p^*(1,r) + wh\frac{\epsilon}{2} \\ &\leq (1+\epsilon)p^*(1,r), \end{aligned}$$

where the last inequality comes from the fact that wh must certainly be a lower bound on $p^*(1,r)$. ∎

We need to bound the minimum height, h, of a triangular face. This can be written as a function of N, the maximum integer used in expressing the coordinates of vertices of the triangulation. We get

Lemma 8.2 *Let $h(N)$ be the minimum height of any face for a triangulation whose vertices have nonnegative integer coordinates no greater than N. Then, $h(N) \geq 1/N\sqrt{2}$.*

Proof: The minimum height is given by

$$h(N) = \min_{\mathbf{u},\mathbf{v}} \frac{\mathbf{u} \times \mathbf{v}}{\|\mathbf{v}\|},$$

where $\mathbf{u}$ and $\mathbf{v}$ are nonzero vectors with nonnegative integer coordinates no greater than N, and $\mathbf{u}$ and $\mathbf{v}$ are not colinear. Thus,

$$\begin{aligned} h(N) &= \min_{\mathbf{u},\mathbf{v}}\left(\frac{u_x v_y - u_y v_x}{\sqrt{v_x^2 + v_y^2}}\right) \\ &\geq \frac{1}{N\sqrt{2}} \min_{\mathbf{u},\mathbf{v}}(u_x v_y - u_y v_x) \\ &\geq \frac{1}{N\sqrt{2}}. \end{aligned}$$

∎

Our next claim bounds the running time of *Find-Point* by showing that we are doing k binary searches, each of which has function evaluations requiring $O(k)$ per evaluation.

Lemma 8.3 *The running time of* Find-Point *is $O(k^2 \log(kNW/\epsilon w))$, where N is the maximum integer coordinate of any vertex, k is the length of the last edge sequence $\mathcal{E}$, and W (resp., w) is the maximum finite (resp., minimum nonzero) weight assigned to faces of $\mathcal{S}$.*

Proof: For each $j = 1, \ldots, k$, we do a binary search. How many iterations will each search take? At the beginning of a search, the interval $[a_j, b_j]$ is of length at most $\sqrt{2}N$ (the maximum length of any edge of $\mathcal{S}$), and we iterate until we reduce its size to $\delta = h\epsilon/6k(W/w)$. Thus, the number of iterations is

$$\log \frac{\sqrt{2}N}{\delta} = \log \frac{6\sqrt{2}Nk(W/w)}{h\epsilon},$$

with each iteration requiring a ray tracing costing $O(k)$. Thus, applying the preceding lemma, we find that the complexity of doing the k binary searches is

$$\begin{aligned} O(k^2 \log \frac{6\sqrt{2}Nk(W/w)}{h\epsilon}) &= O(k^2 \log \frac{N^2 k(W/w)}{\epsilon}) \\ &= O(k^2 \log \frac{kNW}{\epsilon w}). \end{aligned}$$

∎

Lemma 7.1 guarantees that the length of a last edge sequence is bounded above by $O(n^2)$. (In practice, we expect that last edge sequences are usually much shorter than the worst-case bound.) Applying this fact and Lemmas 8.3 we get that

$$\begin{aligned} F(n) &= O(k^2 \log \frac{kNW}{\epsilon w}) \\ &= O(n^4 \log \frac{nNW}{\epsilon w}) \\ &= O(n^4 L), \end{aligned}$$

where $L = \log N + \log \frac{W}{w} + \log n + \log \frac{1}{\epsilon}$ is the precision of the problem instance (the number of bits needed to specify the largest integer, the maximum weight, the number of faces, and the tolerance).

The function *Find-Tie-Point* (I, I') works in much the same way as *Find-Point*, except that now we are searching for the tie point between two roots. It is assumed that $I \cap I' \neq \emptyset$. (Note that I and I' are *candidate* intervals that are in the process of being trimmed; this allows them to overlap.) The search proceeds by "throwing out" at least one quarter of the set $[a_j, b_j] \times [a'_{j'}, b'_{j'}]$ at each iteration, and then calling itself recursively.

Find-Tie-Point (I, I') returns a list $(x, \lambda, u_1, u_2, u'_1, u'_2, w, w')$, where x is the required tie point, λ is the distance from s to the point x, $[u_1, u_2]$ is the (small) range of first bend points of paths through I to x, $[u'_1, u'_2]$ is the (small) range of first bend points of paths through I' to x, and w and w' are the corresponding last bend points for paths through I and I', respectively. If there is no "almost" tie point (that is, a point whose distances from s through the two different roots are within a fraction ϵ of each other), then the function returns NIL.

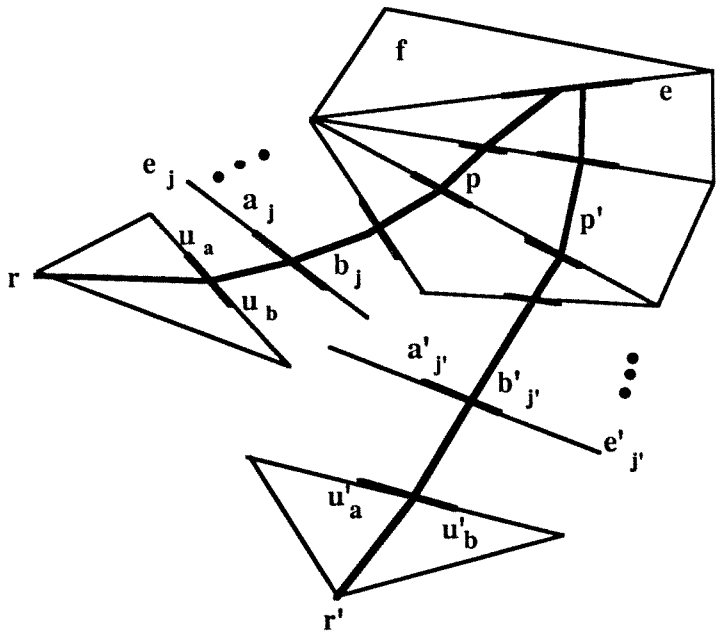

Figure 8.2. Illustration of *Find-Tie-Point*.

Function *Find-Tie-Point* (I, I')

(0). *Assume that* $I = [a, b]$ *and* $I' = [a', b']$ *are both candidate intervals with respect to* (e, f). *Let* $\mathcal{E} = (e_1, \ldots, e_k = e)$ *(resp.,* $\mathcal{E}' = (e'_1, \ldots, e'_{k'} = e)$*) be the last edge sequence (appended with* e*), let* (u_a, u_b) *(resp.,* (u'_a, u'_b)*) be the first bend points, and let* r *(resp.,* r'*) be the root of* I *(resp.,* I'*). (Refer to Figure 8.2.) We further assume that* $I \cap I' \neq \emptyset$. *Set* $j, j' = 1$, $a_1 = u_a$ $(a'_1 = u'_a)$, *and* $b_1 = u_b$ $(b'_1 = u'_b)$. *We assume that points on* e *are ordered from left to right as one looks at face* f *(e.g.,* $a \leq b$*), and that* I' *is right of* I *in the sense that* $a \leq b'$.

(1). **(Check interval sizes)** *If* $|a_j b_j| < \delta$ *and* $j < k$, *then go to (3). If* $|a'_{j'} b'_{j'}| < \delta$ *and* $j' < k'$, *then go to (4). Otherwise, go to (2).*

(2). **(Ray tracing)** *Let* $u = (a_j + b_j)/2$ *be the midpoint of segment* $[a_j, b_j]$, *and let* $u' = (a'_{j'} + b'_{j'})/2$ *be the midpoint of segment* $[a'_{j'}, b'_{j'}]$. *Perform ray tracing from* r *starting with the initial ray through* u. *Stop the ray tracing when either the refraction path,* p, *leaves the edge sequence* $(e_{j+1}, \ldots, e_k = e)$ *(say, by missing edge* e_i *when it exits the face that contains* e_{i-1} *and* e_i*), or* p *emerges through edge* e. *Similarly, perform ray tracing from* r' *starting with the initial ray through* u', *stopping when* p' *either leaves edge sequence* $(e'_{j'+1}, \ldots, e'_{k'} = e)$ *(say, by missing edge* $e'_{i'}$ *when it exits the face that contains* $e'_{i'-1}$ *and* $e'_{i'}$*), or* p' *emerges through edge* e.

(i). (p **and** p' **emerge through** e) *Let c (resp., c') be the point of e where p (resp., p') emerges. We get the following four cases. (Refer to Figure 8.3.)*

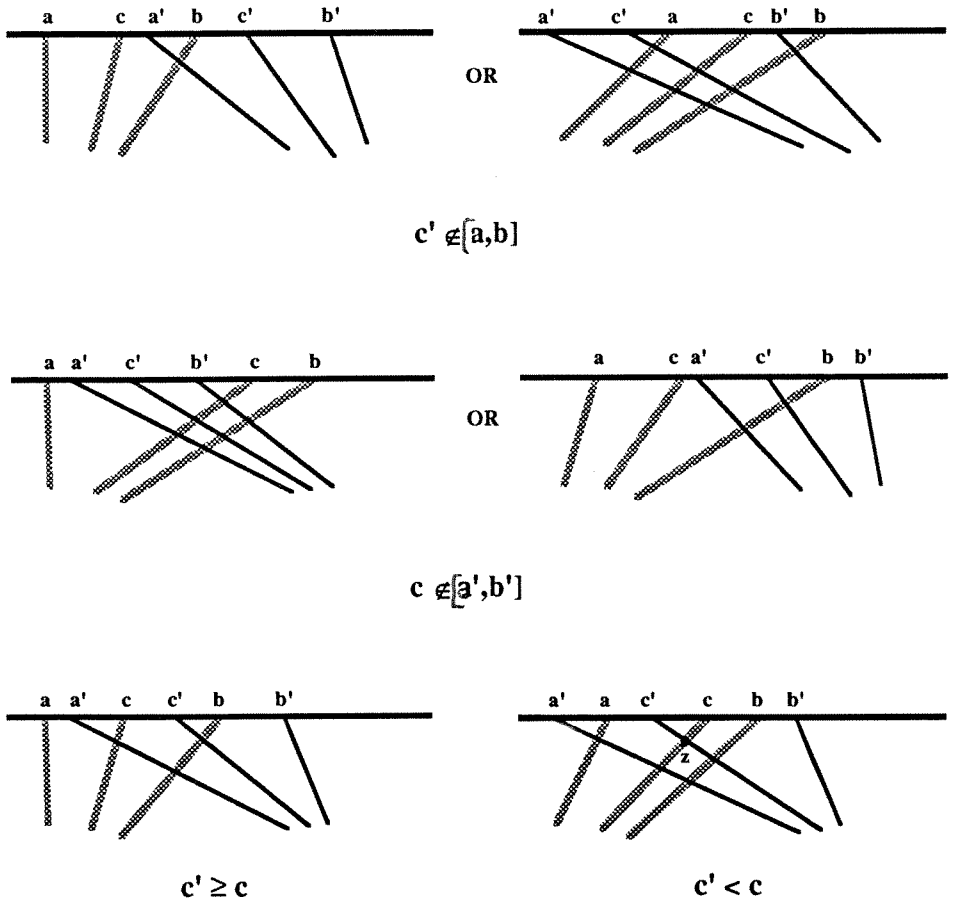

Figure 8.3. Four cases in *Find-Tie-Point*.

(a). ($c' \notin [a, b]$) *If $c' \geq b$, then set $b'_{j'} = u'$ (that is, throw out the right half of the interval $[a'_{j'}, b'_{j'}]$). Otherwise, set $a'_{j'} = u'$ (that is, throw out the left half of the interval $[a'_{j'}, b'_{j'}]$). Go to (1).*

(b). ($c \notin [a', b']$) *If $c \geq b'$, then set $b_j = u$ (that is, throw out the right half of the interval $[a_j, b_j]$). Otherwise, set $a_j = u$ (that is, throw out the left half of the interval $[a_j, b_j]$). Go to (1).*

(c). ($c' \geq c$) *Throw out the right half of the interval $[a'_j, b'_j]$ crossed with the left half of the interval $[a_j, b_j]$, and call* Find-Tie-Point *recursively. (This will be described fully below.) Go to (1).*

(d). (**Otherwise,** $c' < c$) *The path to c must cross the path to c'. Let z be the crossing point. If the path to c' gets to z before the path to c does, then throw out the right half of $[a'_j, b'_j]$ crossed with the right half of the interval $[a_j, b_j]$. Otherwise, throw out the left half of $[a'_j, b'_j]$ crossed with the left half of the interval $[a_j, b_j]$. Go to (1).*

(ii). (p **misses edge** e_i) *If p passes to the right of e_i, then set $b_j = u$; otherwise, set $a_j = u$. Go to (1).*

(iii). (p' **misses edge** $e'_{i'}$) *If p' passes to the right of $e'_{i'}$, then set $b'_{j'} = u'$; otherwise, set $a'_{j'} = u'$. Go to (1).*

(3). **(Advance along $\mathcal{E}$)** *If $j = k$ and $j' = k'$, then let $r_k = (a_k + b_k)/2$ and go to (5). Otherwise, let r_j be the midpoint of $[a_j, b_j]$. Let a_{j+1} (resp., b_{j+1}) be the point on edge e_{j+1} that is hit by the ray emerging from e_j when ray tracing is done from r_{j-1} through a_j (resp., b_j). Set $r = r_j$ and $j = j + 1$. Go to (1).*

(4). **(Advance along $\mathcal{E}'$)** *If $j' = k'$ and $j = k$, then let $r'_{k'} = (a'_{k'} + b'_{k'})/2$ and go to (5). Otherwise, let $r'_{j'}$ be the midpoint of $[a'_{j'}, b'_{j'}]$. Let $a'_{j'+1}$ (resp., $b'_{j'+1}$) be the point on edge $e'_{j'+1}$ that is hit by the ray emerging from $e'_{j'}$ when ray tracing is done from $r'_{j'-1}$ through $a'_{j'}$ (resp., $b'_{j'}$). Set $r' = r'_{j'}$ and $j' = j' + 1$. Go to (1).*

(5). **(Output path)** *Let λ be $d(r)$ plus the weighted Euclidean length of the path $(r, r_1, \ldots, r_k)$ and let λ' be $d(r')$ plus the weighted length of the path $(r', r'_1, \ldots, r'_{k'})$. If $\lambda' \notin ((1-\epsilon)\lambda, (1+\epsilon)\lambda)$, then return NIL (there is no tie point in $I \cap I'$). Otherwise, return the list $(r_k, \lambda, a_1, b_1, a'_1, b'_1, r_{k-1}, r'_{k'-1})$.*

We must define what we mean by the phrase "throw out the right half of the interval $[a'_j, b'_j]$ crossed with the left half of the interval $[a_j, b_j]$, and call *Find-Tie-Point* recursively", which was used above. We instantiate an interval $I'_L \subset e$ corresponding to paths through the left half of the interval $[a'_{j'}, b'_{j'}]$ and an interval $I_L \subset e$ corresponding to paths through the left half of $[a_j, b_j]$ and call recursively the function *Find-Tie-Point* (I_L, I'_L). (This solves the problem corresponding to the left half of the interval $[a'_{j'}, b'_{j'}]$ crossed with the left half of $[a_j, b_j]$.) Also, we instantiate an interval $I_R \subset e$ corresponding to paths through the right half of $[a_j, b_j]$ and an interval $I' \subset e$ corresponding to paths through $[a'_{j'}, b'_{j'}]$ and call *Find-Tie-Point* (I_R, I'). (This solves the problem corresponding to the interval $[a'_{j'}, b'_{j'}]$ crossed with the right half of $[a_j, b_j]$.) Similar meaning is assigned to the phrases in step 2(i)(d) above.

The fact that *Find-Tie-Point* finds to within ϵ a pair of paths whose lengths are equal follows by much the same analysis as in the case of *Find-Point*. It is also not hard to see that the complexity of *Find-Tie-Point* is the same as that of *Find-Point* (namely, $F'(n) = O(F(n))$).

Lemma 8.4 *The complexity of a call to* Find-Tie-Point *is $F'(n) = O(F(n)) = O(n^4 L)$, where L is the precision of the problem instance.*

Proof: This follows since at each iteration of each binary search, we throw away a constant fraction of the set $[a'_j, b'_j] \times [a'_{j'}, b'_{j'}]$ (at least one quarter). More precisely, for a fixed j and j', let $W(m, m')$ be the number of ray traces we must do before the next incrementation of j or j', where m (resp., m') is the number of intervals of size δ that fit in the original interval $[a_j, b_j]$ (resp., $[a'_{j'}, b'_{j'}]$). Then $W(m, m')$ must satisfy the following recursion:

$$W(m,m') = \begin{cases} W(m, \frac{m'}{2}) + O(1), & \text{in case (a),} \\ W(\frac{m}{2}, m') + O(1), & \text{in case (b), or} \\ W(m, \frac{m'}{2}) + W(\frac{m}{2}, \frac{m'}{2}) + O(1), & \text{in case (c) or (d).} \end{cases}$$

The boundary conditions on $W(m, m')$ are given by

$$W(m,1) = W(\frac{m}{2}, 1) + O(1)$$

$$W(1,m') = W(1, \frac{m'}{2}) + O(1)$$

$$W(1,1) = O(1).$$

From these recursions, we get that $W(m, m') = O(\log(m + m')) = O(\frac{nNW}{\epsilon w}) = O(L)$. Each time we do ray tracing it costs us $O(k + k') = O(n^2)$, and the number of times j or j' is incremented is $(k + k') = O(n^2)$, giving us a total complexity of $O(n^4 L)$. ∎

When we proved the correctness of our algorithm, we were making the assumption that *Find-Point* and *Find-Tie-Point* could be performed precisely. We must check that each of the steps of our algorithm and the proof of its correctness remains valid when we use the ϵ-approximation provided by our numerical

routine. All of these follow through with little change. We need only convince ourselves that the intervals of optimality become intervals of ϵ-optimality, meaning that a point in an interval can be reached along a locally f-free path whose length is at most $(1+\epsilon)$ times the length of the shortest locally f-free to the point.

> *Remark:* In step (3) of *Insert-Interval*, when we call *Find-Tie-Point* (I_1, I), we set the first bend point of the right endpoint of I_1, u_{b_1}, to be the rightmost point in the small interval of first bend points corresponding to the search through I_1, while we chose u_a to be the leftmost point in the small interval of first bend points corresponding to the search through I. (Similarly, when we call *Find-Tie-Point*(I, I_2), we set the first bend point of b, u_b, to be the rightmost point in the small interval of first bend points corresponding to the search through I, while we chose u_{a_2} to be the leftmost point in the small interval of first bend points corresponding to the search through I_2.) The reason for this is that it allows us always to overestimate slightly the size of an interval of optimality. That is, the range of first bend points, $[u_a, u_b]$, is always slightly larger than it needs to be (and will always contain the "exact" range), but it is very close in the sense that the lengths of paths through the bend points that we store are no longer than $(1+\epsilon)$ times the length of the shortest path.

Thus, we can summarize the complexity of our algorithm:

Theorem 8.5 *Our algorithm correctly computes the subdivision of each edge into intervals of ϵ-optimality in time $O(ES) = O(n^8 L)$ and space $O(E) = O(n^4)$, where E is the number of events (which is bounded above by $O(n^4)$), $S = O(F(n)) = O(n^4 L)$ is the time necessary to do the searches* Find-Point *and* Find-Tie-Point, *and $L = \log N + \log \frac{W}{w} + \log n + \log \frac{1}{\epsilon}$ is the precision of the problem instance.*

Although the exponent in the worst-case running time seems to be high, we believe that our algorithm *is* practical. We should emphasize that all of our bounds on the complexity of our search procedure are extremely crude, and the average running time of such a search procedure should be substantially less than the worst-case bounds we have produced. In particular, the coordinate descent method we described earlier seems to work very fast in practice. Also, it may be possible to improve our worst-case bounds.

9. Using the Subdivision

To recover the length, $d(x)$, of the shortest path from s to any point $x \in e = f \cap f'$, we do the following: If x is a vertex, then $d(x)$ is just the label that was given it when x was permanently labeled. Otherwise, locate x in interval of optimality I (resp., I') for $(r, \mathcal{E})$ (resp., $(r', \mathcal{E}')$) with respect to (e, f) (resp., (e, f')). (This point location is done simply in time $O(\log n)$.) Compute the sums $d(r) + d_{r,\mathcal{E}}(x)$ and $d(r') + d_{r',\mathcal{E}'}(x)$. By Lemma 4.1, the smaller of the two sums is the desired shortest path length. The calculation of $d_{r,\mathcal{E}}(x)$ or $d_{r',\mathcal{E}'}(x)$ requires either calling a numerical routine (such as coordinate descent), or calling the function *Find-Point* (which has worst-case running time $F(n) = O(n^4 L)$). Thus, after preprocessing, the length of the shortest path to vertices can be calculated in constant time, while the length of the shortest path to a point on an edge can be queried in time $O(n^4 L)$ (or just two calls to a "fast" numerical procedure).

> *Remark:* At this point, we should make a comment on why we used locally f-free paths to define intervals of optimality. If we we had instead defined intervals of optimality to be those subintervals of points on an edge that are reached optimally through the same root and edge sequence, without the requirement of being locally f-free, then the number of such intervals could potentially be huge (doubly exponential in n). The problem is that the bisector between two roots satisfies an equation that involves the sum of a polynomial number of square roots. When the square roots are eliminated, we get a polynomial of potentially doubly exponential degree. The number of intersections of the curve with a straight line is thus doubly exponential. (It is an interesting open problem to determine whether or not there could actually be such a large number of intersections. At this time, we do not see how to guarantee fewer than doubly exponential.) Of course, if our algorithm produced such a subdivision, then the last edge sequence and root of the optimal path to any particular point on an edge could be determined in constant time once the point is located in the subdivision (but this point location query could potentially take

exponential time). Instead, our approach gives us two possible candidates for the structure of the shortest path to a point on an edge, and we must then run a numerical routine or call *Find-Point* to determine to any desired degree of accuracy which of the two paths is better.

The problem with extending our algorithm to compute the subdivision of the interiors of faces is that the bisector curves that bound cells of the shortest path map subdivision will, in general, be curves of very high degree instead of simple hyperbolas as was the case for the DGP in [MMP]. Then, even if we had a representation of this subdivision, a point location query would involve testing on which side of such a curve the query point lies. This can be answered by calling a numerical routine or by calling *Find-Point*. If we build such a subdivision (which could be done by an extension of the technique used in [MMP], this time using a numerical routine similar to *Find-Tie-Point* to determine the distance to the next intersection of two consecutive boundary curves), then the complexity of processing one query is $O(F(n)\log n) = O(n^4 L \log n)$. Then, within this time bound, we could produce both the length of an ϵ-optimal path, as well as the path itself. Another option would be to approximate the bisector curves with something that is, say, piecewise-linear. It should be possible then to prove a bound on the size of the approximate subdivision in terms of the tolerance ϵ.

Alternatively, a brute-force approach is to compare all $O(n^3)$ intervals of optimality with respect to (e_i, f) (for edges e_1, e_2, and e_3 of face f), once we have located x in face f (which can be done by standard point location techniques in time $O(\log n)$). There must be a shortest path to x that passes through one of these intervals to get to x. This straightforward approach will, after preprocessing, yield the length of a shortest path (within the usual tolerance ϵ) to x in $O(n^3)$ "steps", with each step being a call to *Find-Point* (or a numerical routine), requiring time $O(n^4 L)$. An ϵ-optimal path can also be produced within the same time bound simply by tracing back through the predecessor pointers of intervals of optimality. Unfortunately, this query time is, in the worst case, only one factor of n less than the preprocessing time of $O(n^8 L)$. In practice, however, there are usually very few intervals of optimality surrounding any one face, and the numerical routines run much faster than the worst-case complexity bounds suggest.

Theorem 9.1 *After preprocessing, an ϵ-optimal shortest path from s to any point $x \in \mathcal{S}$ can be found in $O(n^3)$ "steps", each requiring time $O(n^4 L)$ in the worst case. The preprocessing can be done in time $O(n^8 L)$ (and space $O(n^4)$). Here, $L = \log N + \log \frac{W}{w} + \log n + \log \frac{1}{\epsilon}$ is the precision of the problem instance.*

10. Generalizations and Extensions

Several generalizations and extensions to our algorithm are of possible interest.

(1). First, we can generalize to the case of multiple source points s. The resulting structure is then a Voronoi diagram (on the edges of the subdivision) according to weighted Euclidean lengths. The only modification we need is to insert the appropriate intervals about each source point in the initialization step of the main algorithm.

(2). Although we described our algorithm in terms of a triangulation $\mathcal{S}$, we suspect that it will be more efficient in any implementation to write the algorithm in terms of an arbitrary polygonal subdivision, allowing regions to be many-sided polygons rather than breaking them into triangles. It is straightforward to write the details of our algorithm for this case.

(3). We may want to generalize the cost structure from being that of a *uniform* cost per unit distance in a region to that of allowing a cost function (such as a linear function) in each region. The function should be specified by some fixed number of parameters, and should allow computation of geodesics. In the uniform cost case, the geodesics within a given region are simply straight line segments. In more general cases, geodesics would have to be computed by techniques of calculus of variations. Our algorithm must then be modified to apply the local optimality criterion (Snell's Law) to curved paths at the boundaries of regions. If an optimal path passes through the point $y \in int(e)$, with $e = f \cap f'$, then the tangent lines at y to the geodesic curves in f and in f' must meet at y such that they obey Snell's Law. Using this local optimality criterion, then, the algorithm should proceed in a similar fashion to the uniform case, this time piecing together curved segments from region to region.

(4). We may want to generalize the problem to allow boundaries between regions to be curved. Then, Snell's Law at an interior boundary point y may be applied as if the boundary at y is the straight line tangent to the boundary at y. While it is not hard to describe the local optimality condition in this case, further work is necessary to figure out the details of the generalized algorithm and what restrictions should be placed on the curves.

(5). Our algorithm generalizes readily to allow the regions to reside on a polyhedral surface (e.g., we may want to plan paths on the surface of a polyhedron whose faces and edges are weighted). Then the local optimality criterion becomes Snell's Law applied to the "unfolded problem" at each boundary. That is, for an edge $e = f \cap f'$, we must "unfold" f' about e so that f and f' are made coplanar, and then we can apply Snell's Law to points interior to e in this rotated coordinate frame. With these minor modifications to our algorithm, one can get a solution to this more general problem within the same time bounds as the weighted planar case.

(6). We may want the cost per unit distance to depend on the direction of motion. For example, it may be more difficult to go up a hill than to go down. Then, for each point x of each region there would be a function $c_x(\theta)$ that is the cost per unit distance of traveling at orientation θ through point x. In case c_x depends only on the normal to the surface at x, we would have a "hill climbing" problem. We are currently investigating this extension. Some preliminary results have been obtained by [RoRo].

(7). Note that by allowing zero-cost regions, we have solved the problem of finding shortest paths from one *region* to another *region* (rather than just between two points). We simply make the source and destination regions of zero cost, and find a shortest path from any point of one region to any point of the other. The result will be a shortest path between the regions.

In conclusion, we should mention several related papers that have appeared since the original draft of our paper. Another group of reserachers, [Ri, RRZM, RoRi], has examined the general weighted region problem, applying the local optimality criteria of Snell's Law to obtain efficiently implemented algorithms. While their algorithms do not have worst-case polynomial time bounds or guaranteed performance (within a given ϵ), they show that they do well in practice. Also the recent PhD thesis of [Al] investigates the implementation of an algorithm that actually constructs a complete shortest path map in the weighted region context.

An algorithm that solves the WRP in the special case of $\alpha_f \in \{0, 1, +\infty\}$ is described in [Mi2,GMMN]. The algorithm constructs a special type of visibility graph (a *critical graph*) that exploits the fact that shortest paths to an edge of a free region must enter it at a normal. In this case, they solve the problem exactly in polynomial (quadratic) time. They generallize their results to include the case of arbitrary weights on edges of the subdivision. The edges can be thought of as "road" segments or as any other type of linear feature. Again, they use a type of critical graph, this time exploiting the local optimality property that a path that is incident on the interior of a road edge must be incident at the critical angle for that edge. Additionally, there may be a "fixed charge" $\xi_e \in [0, \infty]$ for crossing an edge e. This charge can model the cost of crossing a fence or a stream. (Note that the algorithm we present here in this paper can be modified to include such a charge.) [Ro] has also worked on the special case of the weighted region problem in the presence of linear features, and although his theoretical bounds are not as tight as those mentioned above, he gives favorable experimental results.

The recent papers of [PP1, PP2] discuss a generalization of the weighted region problem to the case in which boundaries between regions are curved, weights within regions may not be uniform, the speed of travel is direction-dependent, and the region boundaries and the weights may also be varying in time. The application motivating their research is to routing vessels among time-varying weather patterns in the ocean. They derive the appropriate local optimality criteria (a generalization of Snell's Law), and they discuss a ray-tracing procedure for searching for locally optimal paths. It is an interesting open problem to analyze the combinatorial complexity of searching for globally optimal paths in this general setting.

Another recent result on the weighted region problem has been obtained by [YCL]. They consider the problem in which the weighted regions are all rectilinear, and distances are measured according to the weighted L_1 metric (rather than the weighted Euclidean metric we considered here). They obtain an optimal $O(n \log n)$ time algorithm for the case in which the regions are all aligned rectangles, and they obtain running time $O(n \log^{\frac{3}{2}} n)$ when the regions are rectilinear polygons.

11. Acknowledgment

This research was partially supported by a grant from the Hughes Aircraft Company and by NSF Grants IRI-8710858 and ECSE-8857642. Much of this research was conducted while the first author (J. Mitchell) was supported by a Howard Hughes Doctoral Fellowship at Stanford University and was affiliated with the Hughes Artificial Intelligence Center, part of the Hughes Research Laboratories. We thank Dr. Rolf Klein and the referees for a careful reading, several suggestions on the exposition, and correcting some errors in our earlier draft.

12. References

[Al] Robert Alexander, "Construction of Optimal-Path Maps for Homogeneous-Cost-Region Path-Planning Problems", Ph.D. Thesis, Department of Computer Science, U.S. Naval Postgraduate School, September 1989.

[AAGHI] T. Asano, T. Asano, L. Guibas, J. Hershberger, and H. Imai, "Visibility of Disjoint Polygons", *Algorithmica*, Vol. 1, (1986), pp. 49-63.

[Ba] C. Bajaj, "Applying Galois Theoretic Algebraic Methods to Geometric Optimization Problems", *Proc. SIAM Conference on Geometric Modeling and Robotics*, 1985. (Available as technical report CSD-TR-523, Computer Science Department, Purdue University, West Lafayette, IN.)

[Co] G.E. Collins, "Quantifier Elimination for Real Closed Fields by Cylindric Algebraic Decomposition", *Proc. Second GI Conference on Automata Theory and Formal Languages*, Springer-Verlag LNCS 35, Berlin, 134-183, 1975.

[Di] Dijkstra "A Note On Two Problems in Connection With Graphs", *Numerische Mathematik*, 1 (1959), pp. 269-271.

[GMMN] L. Gewali, A. Meng, J.S.B. Mitchell, and S. Ntafos, "Path Planning in $0/1/\infty$ Weighted Regions With Applications", *Proc. Fourth Annual ACM Symposium on Computational Geometry*, Urbana-Champaign, June 6-8, 1988, pp. 266-278.

[GM] S.K. Ghosh and D.M. Mount, "An Output Sensitive Algorithm for Computing Visibility Graphs", Technical Report CS-TR-1874, Department of Computer Science, University of Maryland, July 1987. (Also appears in FOCS, 1987.)

[GS] L.J. Guibas and J. Stolfi, "Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams", *ACM Trans. Graphics* **4** (1985), 74-123.

[Jo] S.T. Jones, "Solving Problems Involving Variable Terrain. Part 1: A General Algorithm", *Byte*, Vol. 5, No. 2, February, 1980.

[KaMa] S. Kapoor and S.N. Maheshwari, "Efficient Algorithms for Euclidean Shortest Path and Visibility Problems with Polygonal Obstacles", *Proc. Fourth Annual ACM Symposium on Computational Geometry*, Urbana-Champaign, IL, June 6-8, 1988, pp. 172-182.

[KeMi] D.M. Keirsey and J.S.B. Mitchell, "Planning Strategic Paths Through Variable Terrain Data", *Proc. SPIE Applications of Artificial Intelligence*, 1984.

[Le] D.T. Lee, "Proximity and Reachability in the Plane", Ph.D. Thesis, Technical Report ACT-12, Coordinated Science Laboratory, University of Illinois, Nov. 1978.

[LP] D.T. Lee and F.P. Preparata, "Euclidean Shortest Paths in the Presence of Rectilinear Boundaries", *Networks*, 14 (1984), pp. 393-410.

[Ly] L.A. Lyusternik, *Shortest Paths: Variational Problems*, Macmillan Company, New York, 1964.

[Mi1] J.S.B. Mitchell, "Planning Shortest Paths", PhD Thesis, Department of Operations Research, Stanford University, August, 1986. (Available as Research Report 561, Artificial Intelligence Series, No. 1, Hughes Research Laboratories, Malibu, CA.)

[Mi2] J.S.B. Mitchell, "Shortest Paths Among Obstacles, Zero-Cost Regions, and Roads", Technical Report No. 764, School of Operations Research and Industrial Engineering, Cornell University, 1987.

[Mi3] J.S.B. Mitchell, "A New Algorithm for Shortest Paths Among Obstacles in the Plane", Technical Report No. 832, School of Operations Research and Industrial Engineering, Cornell University, 1988.

[MMP] J.S.B. Mitchell, D.M. Mount, and C.H. Papadimitriou, "The Discrete Geodesic Problem", *SIAM Journal on Computing*, **16**, No. 4, pp. 647-668, August, 1987.

[MP] J.S.B. Mitchell and C.H. Papadimitriou, "The Weighted Region Problem", Technical Report, Department of Operations Research, Stanford University, 1986. Extended abstract appears in *Proc. Third Annual ACM Conference on Computational Geometry*, Waterloo, Ontario, June 1987.

[MPK] J.S.B. Mitchell, D.W. Payton, and D.M. Keirsey, "Planning and Reasoning for Autonomous Vehicle Control", *International Journal of Intelligent Systems*, **II**, (1987), pp. 129-198.

[PP1] N.A. Papadakis and A.N. Perakis, "Deterministic Minimal Time Vessel Routing", Technical Report, Department of Naval Architecture and Marine Engineering, The University of Michigan, February 1987. To appear: *Operations Research*.

[PP2] N.A. Papadakis and A.N. Perakis, "Minimal Time Vessel Routing in a Time-Dependent Environment", Technical Report, Department of Naval Architecture and Marine Engineering, The University of Michigan, July 1987.

[QFP] F.K.H. Quek, R.F. Franklin, and F. Pont, "A Decision System for Autonomous Robot Navigation Over Rough Terrain", *Proc. SPIE Applications of Artificial Intelligence*, Boston, 1985.

[RS] J.H. Reif and J.A. Storer, "Shortest Paths in Euclidean Space with Polyhedral Obstacles", Technical Report CS-85-121, Computer Science Department, Brandeis University, April, 1985.

[Ri] R.F. Richbourg, "Solving a Class of Spatial Reasoning Problems: Minimal-Cost Path Planning in the Cartesian Plane", Ph.D. Thesis, Department of Computer Science, U.S. Naval Postgraduate School, Monterey, CA, June, 1987. Also issued as Technical Reports NPS52-87-021 and NPS52-87-022.

[RRZM] R.F. Richbourg, N.C. Rowe, M.J. Zyda, and R. McGhee, "Solving Global Two-Dimensional Routing Problems Using Snell's Law and A^* Search", *Proceedings of the IEEE International Conference on Robotics and Automation*, Raleigh, NC, February 1987, pp. 1631-1636. Also available as Technical Report NPS-86-021, Department of Computer Science, U.S. Naval Postgraduate School, October 1986.

[Ro] N.C. Rowe, "Roads, Rivers, and Rocks: Optimal Two-Dimensional Route Planning Around Linear Features for a Mobile Robot", to appear *International Journal of Robotics Research*. Also available as Technical Report NPS52-87-027, Department of Computer Science, U.S. Naval Postgraduate School, June 1987.

[RoRi] N.C. Rowe and R.F. Richbourg, "An Efficient Snell's-Law Method for Optimal-Path Planning Across Multiple Two-Dimensional Irregular Homogeneous-Cost Regions", to appear *International Journal of Robotics Research*. Also Technical Report NPS52-88-017, Department of Computer Science, U.S. Naval Postgraduate School, Monterey CA, July 1988.

[RoRo] N.C. Rowe and R.S. Ross, "Optimal grid-free path planning across arbitrarily-contoured terrain with anisotropic friction and gravity effects", Technical Report NPS52-89-003, Department of Computer Science, U.S. Naval Postgraduate School, Monterey CA, November 1988. Submitted to *IEEE Journal of Robotics and Automation*.

[SS] M. Sharir and A. Schorr, "On Shortest Paths in Polyhedral Spaces", *SIAM Journal of Computing* Vol. 15, No. 1, pp. 193-215, February 1986.

[We] E. Welzl, "Constructing the Visibility Graph for n Line Segments in $O(n^2)$ Time", *Information Processing Letters*, Vol. 20 (1985), pp. 167-171.

[YCL] C.D. Yang, T.H. Chen, and D.T. Lee, "Shortest Rectilinear Paths among Weighted Rectilinear Obstacles", Manuscript, Department of Computer Science, Northwestern University, 1989.

[Zi] K. Zikan, Personal Communication, Hughes Research Laboratories, Malibu, CA, 1986.