All the Distant Horizon Edges of a Terrain

by

Daniel Archambault

B.Sc. (Hons.) in Computing Science, Queen's University (Kingston), 2001

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming
to the required standard

The University of British Columbia

August 2003

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
Vancouver, Canada

Date Aug 25 2003

Abstract

Terrains used in Geographic Information Systems (GIS) in some applications require a large number of points to express a fine level of detail over a large area. At distant viewpoints, this fine level of detail is often not visually important. It therefore makes sense to coarsen the level of detail in order to reduce rendering costs. There is some evidence that if models are constrained to preserve silhouettes, then coarser meshes can be used with little loss in visual quality.

The purpose of this thesis is to describe an algorithm that determines the set of edges of a polyhedral terrain that lie on some horizon for some distant view of the model. The algorithm computes this by maintaining the two-dimensional *visual hull* developed by Laurentini [30] of the intersection of the terrain with a horizontal plane sweeping from highest to lowest altitude. Changes in the structure of the visual hull delimit changes in the horizon. All such changes can be detected by using tools developed for kinetic visibility.

Contents

List of Figures

Acknowledgements

I would like to thank Dr. William Evans, Dr. David Kirkpatrick, and Dr. Michiel van de Panne for being an excellent committee. I've come a very long way. I would also like to thank Dr. Wolfgang Heidrich for being a critical but helpful second reader.

I would like to gratefully acknowledge the support of DARPA grant F30620-00-2-0560 which provided the funding for this project.

A very big thanks to all you guys in Imager and BETA for your support, especially: Barry Po, Matt Thorne, Hamish Carr, Lisa Streit, Roger Tam, Dr. Jason Harrison, Adrian Secord, Yushuang Liu, Dave Pritchard, and Viann Chan for your support during the hard times. I probably wouldn't have made it without all of you.

Thank you NGs! Snippets for everyone!

Big thanks to everyone at GC for all their support and encouragement!

A special thanks to the Jen. I'm so sorry that I lost you along the way.

And last but not least, thank you very much mom, dad, and Leah! My support network on the other side of the country, but always just a phone call away.

DANIEL ARCHAMBAULT

The University of British Columbia
August 2003

Chapter 1

Introduction

The silhouette is one of the strongest visual cues in understanding three dimensional shape. It has been shown that geometry can be coarsened without significantly impacting its appearance so long as the silhouette is preserved [44]. Many algorithms and data structures exist to compute the silhouette, and the silhouette has been used to reduce rendering costs of geometric models, while retaining the degree of realism in both photorealistic and non-photorealistic rendering of objects [44][34].

In some GIS applications, terrains require a high level of detail to be expressed over a large scale area. This results in large geometric models, which incur high rendering costs. When viewed from a large distance, this high level of detail is not required and therefore it makes sense to utilize silhouette information to reduce the total number of polygons in the model.

In this thesis, a terrain is a continuous piecewise linear two dimensional surface in three dimensions that intersects each vertical line at a single point. The height of the intersection is known as the *height value* or *altitude* of the point. A terrain is composed of vertices, edges, and faces. The distant horizon of a terrain from a horizontal viewing direction V is the set of all terrain edges that support a horizontal line parallel to V that intersects no other part of the terrain (figure 1.1). We develop a sweep plane algorithm to compute the set of all terrain edges that appear on some distant horizon.

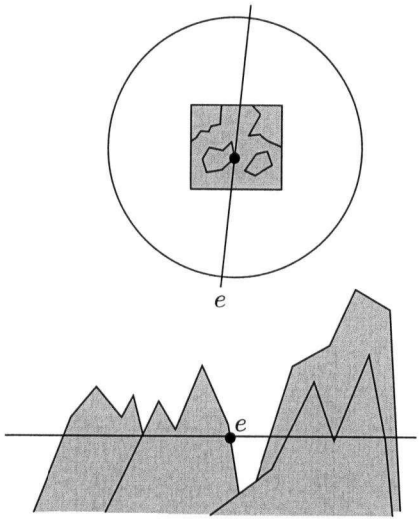


Figure 1.1: The line of sight e fits our definition of distant horizon.

The horizontal lines in our definition of distant horizon are the same as the *visual lines* in two dimensions described by Laurentini in his works on the visual hull [30] [31] [32] [33]. As the edges of the terrain are linear, their intersections with a horizontal sweep plane can be interpreted as points following linear trajectories when the plane is lowered continuously from highest to lowest height value. Thus, a way to determine the set of all edges that support a visual line is to maintain is to maintain the boundary of the set of visual lines as it moves continuously in the plane. This is a task handled frequently in the study of *kinetic data structures*, data structures for keeping track of geometric attributes in motion.

The structure of this thesis is as follows. Firstly, we review some of the previous and related work done on silhouettes, the visual hull, the aspect graph, visibility, and kinetic data structures. Next, we describe our algorithm and prove that it witnesses every change to the set of edges that support a visual line. Finally, we analyze the algorithm's runtime and discuss our results.

Chapter 2

Previous and Related Work

2.1 Silhouettes and Visibility

Silhouettes have been studied extensively in many areas of computer science. In the computer graphics community, they have been used to increase the realism of both the photorealistic and non-photorealistic rendering of three dimensional objects. They have been the primary tool for image-based reconstruction of geometry in the computer vision community for several years. They have been used for image-based registration in scientific visualization and medical computing. Finally, they have been studied in computational geometry through visibility research.

In this chapter, we will review some of the algorithms and data structures that have been developed to compute the orthographic and perspective silhouettes of objects from a particular viewpoint or multiple viewpoints. We then discuss the aspect graph–developed in the computer vision community for object recognition based on silhouettes. Finally, we will discuss the visibility skeleton and how silhouettes can be obtained from it.

2.1.1 Silhouettes and Triangular Meshes

We will define the *silhouette* of a polyhedral object as the set of edges shared by a front-facing face and a back-facing face with respect to a given viewing direction. Many data

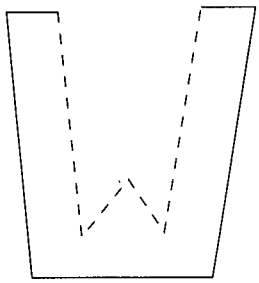


Figure 2.1: Silhouette edges of a cup. The dashed edges on the interior of the cup are silhouette edges that are occluded.

structures already exist to answer this query efficiently. In Sander et al. [44], for example, faces are clustered into a forest of hierarchical search trees where edges can be quickly culled if they do not lie on the silhouette. Hertzmann and Zorin [28] employ the concept of geometric duality. They intersect the dual plane of the viewpoint with the dual of the surface. The intersection points with the edges of the surface in dual correspond to planes that witness a particular edge of the geometry appearing on the silhouette. This is later reduced to a point location problem by Pop et al.[41] where the set of silhouette edges is maintained over a series of spatially coherent viewpoints. Markosian et al. [34] use randomization to discover silhouette edges probabilistically and trace them from a particular viewpoint. Gooch et al. [19] use a hierarchy of arcs between adjacent normals on the Gaussian sphere to quickly determine silhouettes from any orthographic viewpoint. This problem is reduced to line segment intersection in the plane by projecting the sphere onto a cube in Benichou and Elber [9], achieving an output sensitive query time.

These data structures and algorithms do return all edges that lie on a silhouette from a particular point of view, but some of those edges are occluded. For example, if we have a mesh of a cup with a bump at the bottom (figure 2.1) such algorithms will report edges in the interior of the cup. To recognize that these edges are occluded, we cannot perform a local test; we must consider other parts of the geometry present in the scene. Removing these edges would be beneficial in further reducing the geometric complexity of the model.

2.1.2 Horizons and Terrains

The *horizon* of a terrain is the boundary between the sky and the terrain from a particular point of view. More precisely, the horizon of a terrain from a point of view V is the set of terrain edges that support a line through V and intersect no other point of the terrain. From a particular viewpoint, the horizon of a terrain of n edges has near linear worst case complexity $\Theta(n\alpha(n))$ [10] where $\alpha(n)$ is the inverse Ackermann function, an extremely slow-growing function. Many algorithms exist for computing the horizon of a polyhedral terrain from a single or many viewpoints. We will briefly discuss some of these algorithms here.

From a single viewpoint V, the horizon around V is equivalent to projecting all edges of the terrain to the image plane and computing the upper envelope of a set of line segments. Many algorithms already exist to compute the upper envelope in optimal worst-case running times [5] [27]. De Floriani presents an incremental approach to computing the horizon from a single viewpoint which is non-optimal in worst case, but can be implemented in a randomized form to achieve an expected running time of $O(n\alpha(n)\lg n)$ [12].

From a set of $O(n)$ viewpoints, applying the algorithms to compute the horizon at each of them produces running times of $O(n^2 \lg n)$ in worst case. As the computation of the precise horizon from any viewpoint seems to be expensive, we can compute the *approximate horizon* from every viewpoint by dividing the view into s sectors. The horizon of each sector is assumed to be constant and equal to the point of maximum elevation. Max [37] describes an algorithm for computing the approximate horizon. This algorithm extends very well to the case of $O(n)$ viewpoints where it takes $O(sn^{1.5})$ time in worst case. This is further improved by Stewart to achieve a $O(n(\lg^2 n + s))$ worst case running time [47].

All of these algorithms produce the horizon, but the horizon computation is either from a single viewing direction or approximate. We would like to see if there exists an algorithm that computes the precise set of edges on the distant horizon from all horizontal viewing directions.

2.1.3 The Visual Hull

In [44], Sander et al. state: "Since the exterior silhouette of a shape is determined by its visual hull, silhouette extraction is unaffected by any simplification of the original mesh that preserves its visual hull ... Such simplification offers an opportunity for further reducing silhouette extraction cost." The *visual hull* (or more precisely the *external visual hull*) delimits the maximal object that can be reconstructed from a series of silhouettes. It has been extensively studied in computer vision for recovering three dimensional shape [30] [31] [32] [33], computer graphics for image based model acquisition [35] [36], and it has been formalized in computational geometry [42]. Algorithms for computing the visual hull directly from the geometry do exist, but their running times are quite high. For polyhedral objects in three dimensions with n vertices and m active segments (segments that define the boundary of the visual hull) these algorithm take $O((n^4 + m^3) \lg n)$ where m can be $\Omega(n^3)$ in worst case [42].

Our problem, however, does not have to consider all cases of the general visual hull. We deal only with distant, orthographic projections of terrain. Therefore, the computation of our part of the visual hull may be more reasonable.

The Visual Hull in Two Dimensions

The two dimensional version of the visual hull problem was studied extensively by Laurentini [30] and further formalized by Petitjean [42]. We have a set of simple polygons S in the plane which we call a *scene*. Let us define *free space* as the complement of the union of the object interiors. The visual hull in two dimensions ($V_H(S, R)$) is a partition of free space into cells that define the maximal object that can be recovered from a viewing region R somewhere in the free space of the plane. An example of a visual hull in two dimensions is presented in figure 2.2. The cells labelled zero in the diagram are the cells of the visual hull.

We now present the formal definition for a point that lies on the two dimensional external visual hull presented in [42] based on observations made in [30]. A point of the

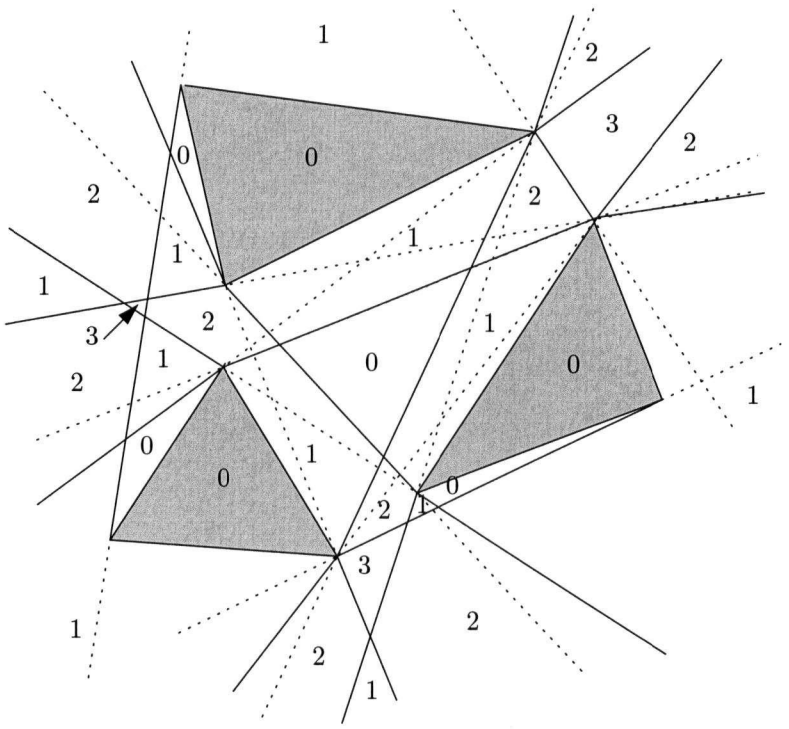


Figure 2.2: The visual hull in two dimensions (adapted from [42]). The visual hull is the cells of this partition with a visual number of zero. The *active segments* bounding the visual hull are the darkened line segments.

free space belongs to the external visual hull $V_H(S)$ if and only if every line passing through it contains a point of S. If the point does not lie on the external visual hull, it must support a *visual line*–a line that passes through the point and does not properly intersect any other object in the scene. Notice that we have dropped the viewing region R from our definition of the external visual hull [30]. This is due to the fact that the external visual hull ($V_H(S)$) of a scene S is the maximal set of points that have the same silhouette as S from all viewpoints outside the convex hull of S. For the rest of this thesis, we will refer to the external visual hull in two dimensions as the visual hull, as neither the internal visual hull nor visual hulls in higher dimensions are considered.

We can now sketch an algorithm for computing the visual hull based on the above definition by finding all regions of the plane that do not support a visual line. To do this, we note that visual lines can be grouped into *families* [30]–visual lines that can be translated and rotated into each other without properly intersecting an object in the scene. All we need to do is select a line that does not properly intersect any object and determine the extent of its family. By determining the extent of all families in the plane, we partition it into regions. The number of families present in a region corresponds to the *visual number* of that region. All regions that are part of the visual hull, by definition, have a visual number of zero because they do not support a visual line.

Interestingly enough, the boundary of each family is composed of parts of lines that are defined by the line segments of the *visibility graph* which has been studied extensively in computational geometry [4] [38] [20]. The visibility graph of a polygonal scene S is a graph whose nodes are the vertices of the polygonal obstacles and whose arcs are pairs of vertices (u, v) such that the line segment connecting u and v does not properly intersect any obstacle of S [20]. In fact, only the line segments of the *tangent visibility graph*, a subgraph of the visibility graph where (u, v) is required to be tangent to both obstacles at u and at v, need to be considered. The tangent visibility graph, as introduced in [40], works on n strictly convex obstacles of constant complexity and partitions the free space into cells where the set of objects seen from any viewpoint in the cell is the same. If polygonal objects

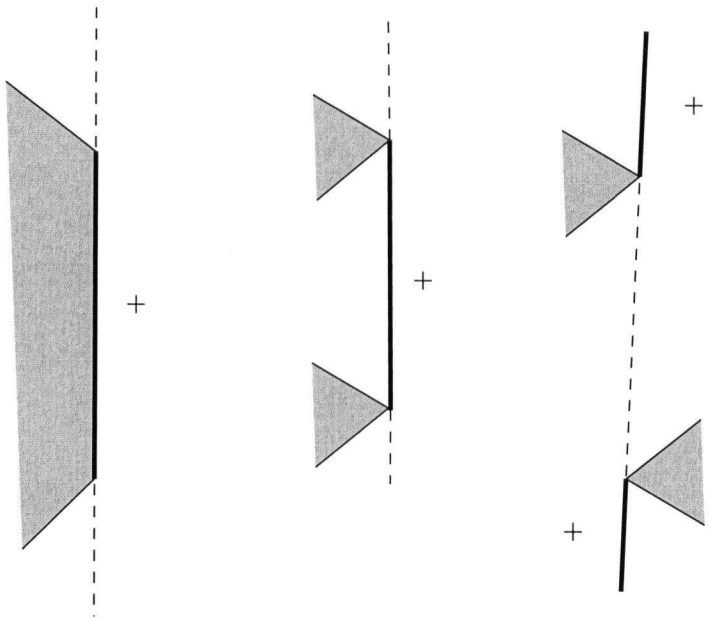


Figure 2.3: The active segments of the three possible active bitangents in a scene. The active segments are drawn as bold lines. The plus signs indicate regions of potentially greater visual number.

of non-constant complexity are of interest, we must also consider the extended edges of the polygons. The boundary of the visual hull is defined by parts of the extended line segments of the tangent visibility graph: lines that do not properly intersect any object in the scene. These lines are known as *active bitangents*. The parts of the lines that define the boundary of the visual hull are known as *active segments*. The three types of active segments are drawn in bold in figure 2.3.

Both papers describe algorithms to compute the visual hull in two dimensions. Laurentini [30] computes the visual hull directly from its definition. He computes the visibility graph of the objects in the plane, determines the active segments, computes the partition of free space in the plane by a standard sweep line technique, and determines the visual number of all cells. Any cell of visual number zero is part of the visual hull. The entire process takes $O(n^3 + m \lg m)$ time where n is the overall complexity of the objects in the scene and m is the number of cells in the plane. In worst case, m can be as large as $\Omega(n^4)$. The algorithm requires $\Theta(n^3)$ time as each of the potential $\Omega(n^2)$ extended arcs of the visibility

graph are intersected with every object in the scene.

Petitjean [42], only considers the arcs of the tangent visibility graph. He constructs the tangent visibility graph in a similar manner as presented in [40]: compute the two tangent lines to each object parallel to the x-axis and rotate them through an angle of π until they are parallel with the x-axis again. When two parallel tangents become collinear, we have a line that is tangent to two objects or a bitangent. Bitangents indicate places where tangents begin or cease intersecting an object. All occurrences of bitangents are placed on a queue, sorted by the angle at which they occur. Assuming the angles at which bitangents occur can be computed in constant time (the objects are of constant complexity), we can determine if the bitangent intersects an object in constant time. This algorithm is able to compute the visual hull in $O(n \lg n + k + m \lg m)$ time where k is the number of arcs of the visibility graph and n is the number of curved objects in the scene of constant complexity. Under most circumstances, their algorithm performs better than the one presented in [30] except when the $m \lg m$ term dominates.

2.1.4 The Aspect Graph

The aspect graph [43] was developed in computer vision research for the purpose of recognizing three dimensional objects from two dimensional images. Let us define the *topological structure* of a projection of a polyhedral object as the set of all edges, faces, vertices, and apparent intersections between these elements which are at least partially visible when projected onto a viewing plane from some fixed viewing direction. The aspect graph attempts to categorize the number of times the *aspect*–the topological structure of the projection–changes under an assumption of either orthographic or perspective projection. Assuming that changes in aspect are sufficient for recognition, a minimum number of viewpoints, one for each change in aspect, can be stored for an object. Algorithms are known for the construction of the aspect graph [22] [23]. In this section, we will primarily be concerned with understanding the structure of the orthographic aspect graph to illustrate another way to study our problem and to define some terminology that we use in future sections.

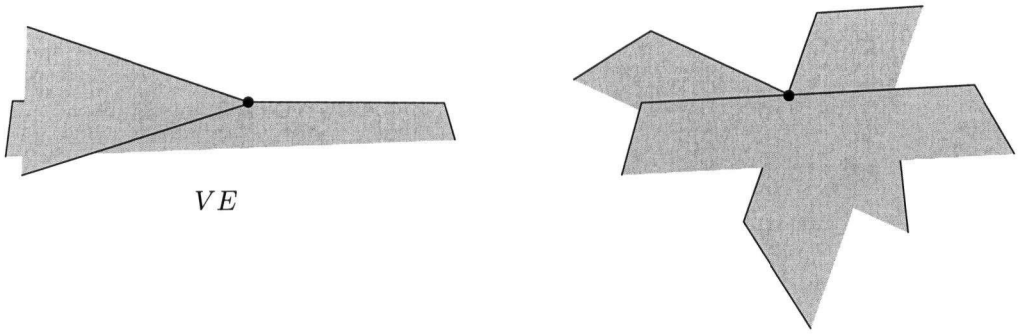


Figure 2.4: The apparent intersections corresponding to VE and an EEE events.

Under an assumption of orthographic projection, we have a sphere of viewing directions. We want to partition areas of the viewing sphere into maximal regions of constant aspect. In this setting, a node of the aspect graph is a region of the sphere of constant aspect and an arc connects two nodes if and only if their corresponding regions share a boundary.

To compute the partition of the viewing sphere, we must consider *events* where the aspect changes. These events occur when the viewing direction becomes coplanar with a face of the polyhedron or a configuration of edges of the polyhedron appears to intersect at a point in the orthographic projection. This apparent intersection occurs either when a vertex appears to become collinear with an edge, a VE event, or three edges appear to intersect at a point, an EEE event (figure 2.4). This apparent intersection will be visible from a range of viewing directions that trace a one dimensional curve on the viewing sphere. The set of lines that pierce the apparent intersection point in the viewing directions from which the intersection point is visible are known as *line swaths*.

The line swath for a VE event is a set of lines that lie in a plane containing a vertex and an edge of the polyhedral object. The set of lines in this swath form a planar surface. On the viewing sphere, the set of viewing directions correspond to an arc that is part of a great circle. For an EEE event, the set of lines in the swath form a quadratic surface; on the viewing sphere, the set of viewing directions corresponds to a curve. The lines forming the quadratic surface are the intersection of two planes for every point p on e_1 (figure 2.5):

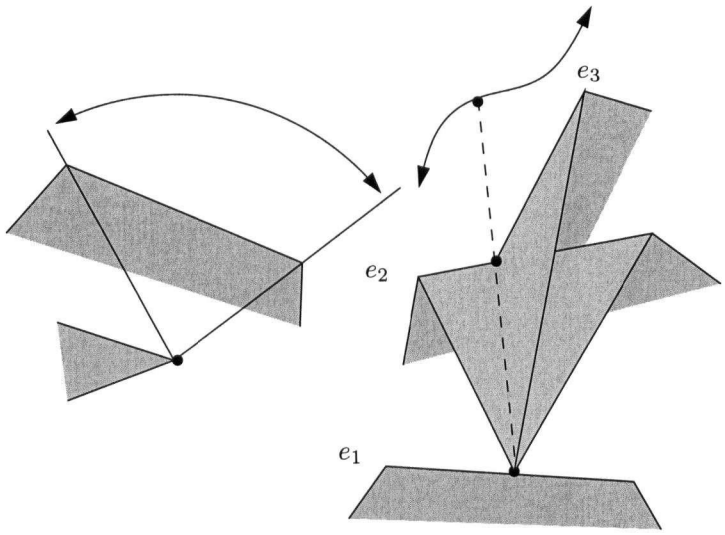


Figure 2.5: The line swaths corresponding to VE and EEE events.

the plane containing p and e_2 and the plane containing p and e_3.

However, in our problem, we do not need to consider all regions of the aspect graph. We only need to consider horizontal viewing directions and therefore regions of the aspect graph that lie on the equator of the sphere of viewing directions.

The Aspect Graph and Terrain

The aspect graph has been applied to bound the number of distinct orthographic views of a polyhedral terrain. Most of this work originated from the study of lower envelopes in three dimensions when Halperin and Sharir noted that the number of visible four tangents–lines that are tangent to four points –that lie above a polyhedral terrain could be bounded by the complexity of several lower envelopes in three dimensions [26] (The lower envelope in three dimensions is the point-wise minimum of several overlapping functions. The term will be formally discussed later in this thesis.). These visible four tangents correspond to the intersection of a constant number of line swaths in the aspect graph and therefore the number of nodes of the aspect graph could be bounded. De Berg et al. give an upper bound of $O(n^5 2^{c\sqrt{\log n}})$ for a positive constant c. They also give a lower bound of $\Omega(n^5\alpha(n))$ [11]. The value of $\alpha(n)$ is the value of the inverse Ackermann function. The upper bound was

improved by Agarwal and Sharir in [1] to $O(n^{5+\epsilon})$ for any $\epsilon > 0$ where $n^{\epsilon} < O(2^{c\alpha(n)})$ for a positive constant c as the bounds for the lower envelope in three dimensions were improved in [45]. The value of ϵ is very small and is related to the length of the longest *Davenport Schinzel Sequence* which will be defined and discussed later in this thesis.

2.1.5 The Visibility Skeleton

The *visibility skeleton* [14] can be viewed as a practical implementation of the visibility complex [15] [16] for polygonal meshes. In this discussion of the visibility skeleton, we will not describe how the visibility skeleton is computed, but rather, we would like to review the set of nodes and arcs that arise in its construction.

A node of the visibility skeleton is an *extremal line*, the intersection between many line swaths, defining a unique line in $\Re^3$. Our notation (the same notation as in [14]) is as follows: capital letters indicate independent mesh elements and small letters indicate mesh elements that are dependant on one and another (ie v is one of the defining vertices of face F or one of the defining vertices of edge E). The simplest node corresponds to a line that connects two vertices of the polygonal mesh, or a VV node. Next, a line that is tangent to a vertex and two edges yields a VEE or an EVE node. A line that is tangent to four edges of the mesh is an $EEEE$. Unique lines are also defined when a face of the mesh becomes tangent with two edges of the mesh in EFE or FEE nodes. A face and one of its defining vertices along with an edge of the mesh forms a FvE node. Two non-coplanar faces forms an FF node. An edge and the two vertices defining it form a Evv node. Finally, two vertices on the same face form a Fvv node.

There are four kinds arcs in the visibility skeleton. These can be viewed as the line swaths of the aspect graph. The four arcs are as follows. Firstly, a VE arc is the set of lines tangent to a vertex and an edge. An EEE arc is the set of lines tangent to three edges. These correspond to the line swaths of the aspect graph. Arcs of the visibility skeleton can also be defined by the line swath in the plane of a face and through one of its defining vertices, an Fv arc, and the line swath in the plane of a face and through a non-coplanar edge, an FE

arc. These arcs correspond to crossing a plane defined by a face of the polyhedron in the aspect graph.

For our problem, we consider a subgraph of the visibility skeleton. We constrain all of our lines of sight to lie within planes parallel to the xy-plane. We thus have three degrees of freedom in specifying a line of sight: choosing a plane π parallel to the xy-plane, and then a slope and y-intercept for the line in π. We assume that no two vertices in the terrain are at the exact same height value (this can be extended to any terrain using a *simulation of simplicity* described in [17]). Under this assumption, faces cannot lie in any π, thus, all nodes in the visibility skeleton involving faces need not be considered. Thus, the nodes of our subgraph are VE nodes, a line that is contained in a plane π that is tangent to an edge and a vertex; and EEE nodes, a line that is contained in a plane π and is tangent to three edges simultaneously. Our line swaths that transition between these nodes consist of EE swaths–lines that are tangent to two edges but have free selection of π and an F swath–a line tangent to a face of the mesh but has free selection of π. By sweeping the plane π through the terrain from highest to lowest altitude and keeping track of arcs and nodes in this subgraph, we can obtain our list of edges that support visual lines.

2.2 Kinetic Data Structures

Kinetic data structures, first introduced in [8], are designed to keep track of geometric attributes of objects in motion. The tools of kinetic data structures have been applied to maintaining geometric attributes of moving objects such as points, lines, or polyhedra to solve problems such as collision detection and visibility queries with respect to a moving observer. Kinetic data structures maintain a set of *certificates* defined by time varying functions that prove that no change in the attribute of interest has occurred. The attribute can change at a time t only if a certificate *fails* at time t. Sets of certificates are often obtained from static algorithms that compute the attribute. Deriving a set of certificates from an algorithm that computes the static attribute of interest is referred to as *kinetization*.

In this section, we review some of the previous work that pertains to our problem.

We review kinetic data structures that maintain the visibility of objects from a moving point of view and kinetic data structures to maintain the convex hull of a moving point set.

2.2.1 Kinetic Visibility

Kinetic Visibility, introduced in [21], deals with the kinetization of static visibility structures to allow for moving observers and objects in two and three dimensional scenes. To date, most of the research in kinetic visibility has dealt with a single moving observer and moving objects in a two dimensional scene. In [24] and [25], a topological line through the two dimensional visibility complex is maintained along with the series of visibility complex edges that it intersects. In [29], the visibility polygon of a set of objects, which are not necessarily stationary, is maintained with respect to a moving viewpoint. The common theme in both papers, however, is the maintenance of a radial decomposition of the scene. The *radial decomposition* of the scene is the partition of the plane induced by the arrangement of tangents to objects through the viewpoint; these lines are known as *visibility tangents*. The set of visibility tangents changes when two become collinear. Certificates that measure the angle between every pair of adjacent visibility tangents are maintained to prove the cyclical order of them around a viewpoint. These certificates fail when the angle between two successive visibility tangents degenerates to zero.

2.2.2 Kinetic Maintenance of the Convex Hull

In [8], an algorithm is described to maintain the convex hull of a point set where each point of the set follows an algebraic trajectory of bounded degree. The algorithm that is kinetized is a simple divide and conquer algorithm for the convex hull: divide the points into two roughly equal point sets, compute the convex hull of each of them, and merge the results. The merge is accomplished by walking around the two convex hulls with parallel tangents.

For simplicity (as [8] did), we will focus on the kinetic data structure for maintaining the upper hull. The merge step is described in the dual space where a moving point $(p(t), q(t))$ is described by the line $y = p(t)x + q(t)$. In dual space, we merge two upper

envelopes, consisting of doubly-linked lists of dual edges and vertices known as *chains*. At every internal node, we have two chains, a red chain and a blue chain (the upper hulls of the dual edges in the left and right subtrees), that we merge into a single purple chain. The problem is to determine a set of certificates between the red and blue chains that ensure that there are no changes to the purple chain as the edges move. The set of certificates consists of x certificates which prove the x-ordering of the dual vertices, yli and yri certificates which prove the points of intersection between the two chains, and certificates which prove that a particular edge is not on the upper envelope: either three tangent certificates slt, srt, and yt, or a certificate proving that there is no tangent sr or sl (figure 2.6).

This structure has been shown to *strongly certify* the upper envelope and therefore strongly certify the convex hull (page 44 lemma 4.2 of [8]). A certificate set L strongly certifies an attribute A if for any configuration π and any other configuration π' where the certificate set L is valid $A(\pi) = A(\pi')$ [8]. The certificate structure will therefore report any combinatorial change to the convex hull.

If we consider time as a third dimension, the lines of the upper envelope move in such a way that they define the surfaces of bivariate functions of bounded degree in three dimensions. The number of times the upper envelope in two dimensions can change is bounded by the number of faces that can appear on the three dimensional upper envelope. The number of faces on the upper envelope is known to be $O(n^{2+\epsilon})$ for any $\epsilon > 0$ [2] [45]. This bound is nearly tight. Even when the points follow linear trajectories, the convex hull of the point set can change $\Omega(n^2)$ times [3].

2.2.3 Lower Envelopes and Davenport-Schinzel Sequences

The problem of maintaining the convex hull of a set of moving points in $\Re^2$ can be reduced to determining the lower envelope of a set of bivariate functions in $\Re^3$ as described above, to realize non-trivial worst-case running times. In this section, we will review lower envelopes in $\Re^2$ and $\Re^3$ and demonstrate how to achieve tight bounds on their combinatorial complexity. Such analysis involves the use of *Davenport-Schinzel sequences* and so we also

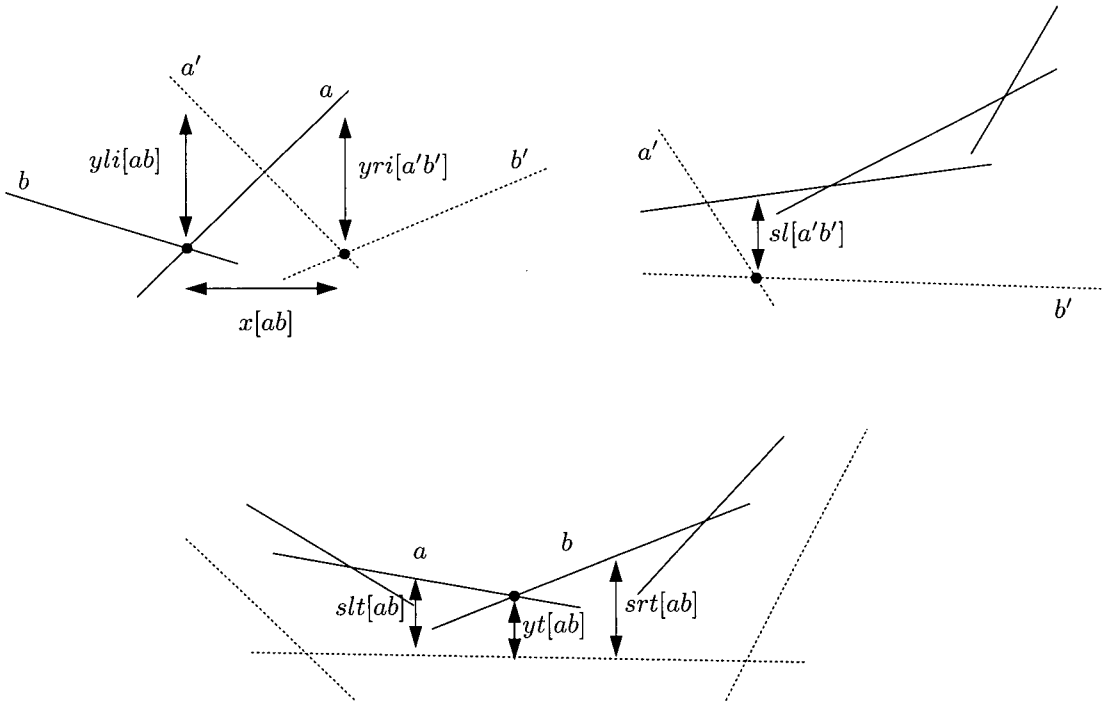


Figure 2.6: The dual space upper envelope with certificates proving how the two chains are merged. Adapted from [8] page 44.

discuss these sequences here.

Let us define a family F of n functions $y = f(x)$ of bounded degree s in $\Re^2$. The *lower envelope* (figure 2.7) of F is the set of functions that form the point-wise minimum of F. In $\Re^2$, the combinatorial complexity of the lower envelope is the sum of the number of times each function of F realizes the point-wise minimum.

In $\Re^3$, F is a set of surfaces $z = f(x, y)$ of bounded degree s. A function of F could be the point-wise minimum over an area of the plane (x, y). If the functions that realize the point-wise minimum are projected down into the (x, y) plane we have a planar map or the *minimization diagram* (figure 2.7) of F. The minimization diagram of F is composed of faces (where a single function of F realizes the point-wise minimum), edges (where two functions of F simultaneously realize the point-wise minimum), and vertices where three functions of F simultaneously realize the point-wise minimum assuming the functions are in general position. The combinatorial complexity of F is the number of faces, edges, and vertices of the minimization diagram. As the minimization diagram is a planar map, its size can be bounded its number of vertices.

Analyzing the combinatorial complexity of a lower envelope involves the use of Davenport-Schinzel sequences, so we define them here. A (n, s)-Davenport-Schinzel sequence is a sequence of m integers (σ_i) with the following properties [46]:

1. $1 \leq \sigma_i \leq n$ for all $i = 1, ..., m$
2. $\sigma_i \neq \sigma_{i+1}$ for all $i = 1, ..., m - 1$
3. There does not exist a subsequence of $s + 2$ indices $1 \leq i_1 < i_2 < i_3 < ... < i_{s+2} \leq m$ such that $\sigma_{i_1} = \sigma_{i_3} = \sigma_{i_5} = ... = \sigma_{i_{s+1}} = a$ and $\sigma_{i_2} = \sigma_{i_4} = \sigma_{i_6} = ... = \sigma_{i_{s+2}} = b$ and $a \neq b$.

We will denote the longest (n, s)-Davenport-Schinzel sequence by $\lambda_s(n)$ [8] [46].

From this definition, we can see for a family of n functions F in $\Re^2$ of degree at most s, the combinatorial complexity of the lower envelope in $\Re^2$ can be described as a Davenport-Schinzel sequence; each function can be labelled with an integer $1...n$ and any

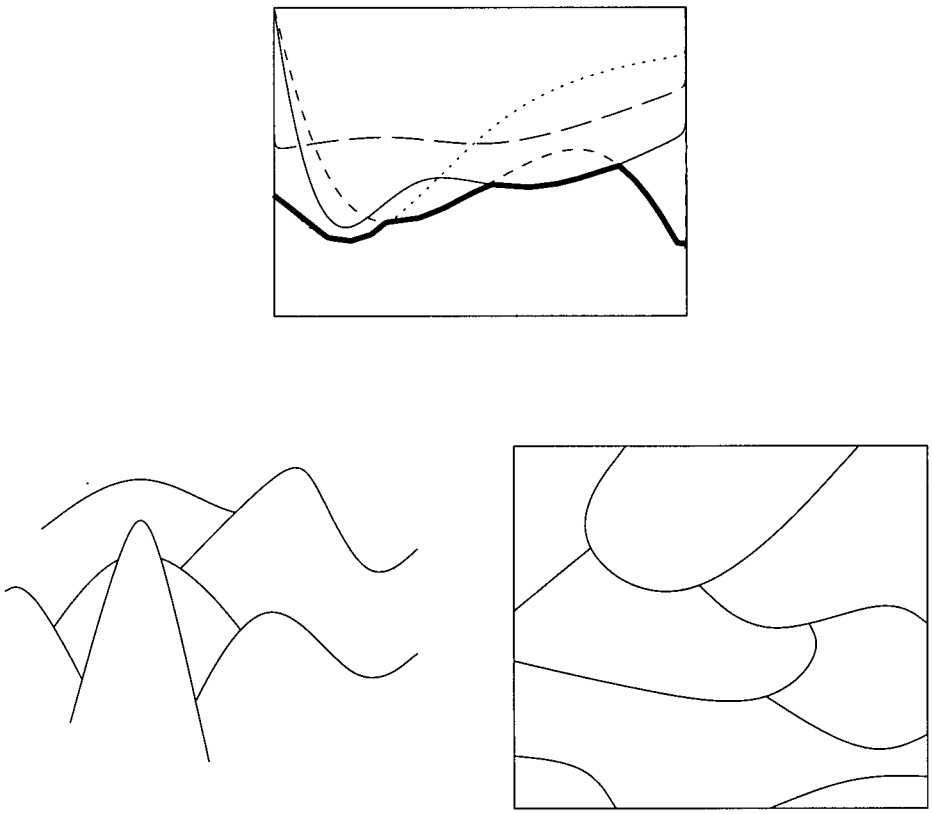


Figure 2.7: The lower envelope in $\Re^2$ (top) and $\Re^3$ (bottom). The lower envelope in $\Re^2$ is the bold line in the figure. It can be interpreted as a fixed vertical slice of $\Re^3$. A minimization diagram is shown to the bottom right.

two functions alternate (not necessarily consecutively) on the upper envelope at most $s+1$ times. Thus, the combinatorial complexity of the lower envelope of F is bounded by $\lambda_s(n)$.

Tight bounds for Davenport-Schinzel sequences are known. In worst-case, for any constant s the bound on $\lambda_s(n)$ is nearly linear. Known bounds on $\lambda_s(n)$ [46] are:

- $\lambda_1(n) = n$
- $\lambda_2(n) = 2n - 1$
- $\lambda_3(n) = \Theta(n\alpha(n))$
- $\lambda_s(n) \leq n2^{(1+o(1))\alpha(n)^{\frac{s-2}{s}}}$ if n is even
- $\lambda_s(n) \leq n2^{(1+o(1))\alpha(n)^{\frac{s-2}{s}\log\alpha(n)}}$ if n is odd

Near linear bounds in $\Re^2$ leads one to conjecture that near quadratic bounds on the combinatorial complexity of the lower envelope in $\Re^3$ are attainable. This near quadratic bound was proved by Sharir in [45] to be $O(n^{2+\epsilon})$ for any $\epsilon > 0$ with ϵ chosen such that $\lambda_s(n) < n^{1+\epsilon}$. The bound also holds for partially defined functions with domains defined by a constant number of inequalities.

Chapter 3

Algorithm

In this chapter, we will describe our algorithm to compute the horizon of a terrain from any distant point of view. The algorithm is a sweep plane algorithm very similar to that of [7]. We maintain the intersection of a horizontal plane with the terrain, or the terrain's *contours*, over the course of a sweep from highest to lowest altitude. We also maintain the set of active bitangents to determine which edges of the terrain support visual lines. As no point inside the convex hull of a contour can support a visual line, we maintain the convex hulls of the contours in the sweep plane so that we can easily determine the next edge to support an active bitangent in constant time.

We start by stating our assumptions and defining a few terms that will be used to describe our algorithm. Then, we describe the major components of it: processing events where vertices of the terrain appear in the sweep plane and processing events where the visual hull changes. Finally, we conclude by demonstrating that our algorithm detects all edges of the terrain that support a visual line.

3.1 Problem Definition and Assumptions

We assume, as mentioned the introduction, that the z-axis coincides with the up direction of the height field and the viewpoint is distant. We define a *terrain edge* and a *terrain vertex* as an edge and a vertex of the polyhedral terrain respectively. A terrain edge is *downhill* from

a terrain vertex if its other endpoint is of a lower altitude. Otherwise, the terrain edge is *uphill*. Since the viewer is distant, the viewer's lines of sight are all parallel to the xy-plane.

With the above assumptions, we reduce the problem to a set of two dimensional problems. We can simply consider *slices* of the terrain at planes parallel to the xy-plane of the height field. In this two dimensional setting, we have a series of simple polygons or *contours* in the plane [7]. These contours are polygonal chains that define the boundary of the simple polygons. These chains do not properly intersect in any slice as the terrain does not self-intersect. They can, however, be nested inside one and another. If they are nested, we will only consider the outer contour. We do not need to consider any inner contour as no tangent to a vertex on an inner contour can be drawn such that it does not intersect the outer contour; thus, the vertices of the inner contour cannot support visual lines. The vertices of the contours are *slice vertices*. They are the two dimensional points defined by the intersection of a terrain edge with a horizontal plane. The *lifetime* of a slice vertex is the range of height values for which a horizontal plane intersects the terrain edge that defines it. The edges of the contours are faces of the terrain cut by the plane at a particular height value.

We will now review and slightly change some of the definitions of terms used in the visual hull and visibility complex literature. We will define a *bitangent* as an extended convex hull edge or a line that is mutually tangent to any pair of contours. This definition of bitangent differs from the definitions found in visibility complex and visual hull literature in three ways: the points of tangency do not have to be mutually visible; a bitangent is a line rather than a line segment; and extended convex hull edges of the contours are considered bitangents. A bitangent is *active* if it does not properly intersect any other contour in the scene. A bitangent is *inactive* if it does. The *activity* of a bitangent is whether or not it is active or inactive. *Hull bitangents* will refer to only the extended convex hull edges, while *scene bitangents* will refer to all other bitangents in the slice.

If we are only concerned about slice vertices and not arbitrary points, supporting an active bitangent is equivalent to supporting a visual line, prompting the following theorem.

Theorem 3.1. *A slice vertex supports an active bitangent if and only if it supports a visual line, assuming the slice vertex is not the only slice vertex in the slice.*

Proof. We will first demonstrate that if a slice vertex supports an active bitangent, it must support a visual line. Consider a slice vertex that supports an active bitangent. Since the bitangent is active, the line does not properly intersect any other polygon in the slice. Therefore, there must be free space to one side of the active bitangent with respect to each support vertex. Rotate the line slightly in that direction about the slice vertex. Now the line touches one point in the slice and properly intersects no other polygon and is a visual line. We can do this for both slice vertices of this bitangent. Therefore, if a point supports an active bitangent, it will also support a visual line.

We now prove the implication in the other direction, so we assume a point supports a visual line. By definition, the visual line is tangent to that point and intersects no other part of the scene. Rotate the visual line around the tangent point until it runs into a polygon edge or another point in the scene. This is a bitangent and the bitangent is active since this is the first polygonal object the visual line has begun to intersect. □

By theorem 3.1, it suffices to determine if the slice vertex of a terrain edge supports an active bitangent. If it does not, it is not on the horizon.

3.2 Algorithm Description

As mentioned previously, we employ a sweep plane technique to determine if an edge of the terrain supports an active bitangent. The sweep plane sweeps through the range of height values in the terrain from highest to lowest value. Thus, we start by sorting all terrain vertices by altitude and inserting them into an *event queue*–a priority queue of events that could potentially change the set of active bitangents. Each edge of the terrain has a set B and a boolean value *keep* associated with it throughout its lifetime. The set B will contain the set of all bitangents to the slice vertex that occur at any time throughout the sweep. At the end of the slice vertex's lifetime, *keep* will be set to true if B held an active bitangent

some time in the past.

In our problem, we want to keep track of the set of all active bitangents in the sweep plane. This set can change in four ways (proved later in lemma 3.16).

1. *Peak vertex events*: altitudes where local maxima of the terrain appear in the sweep plane
2. *Non-peak vertex events*: all the other altitudes where terrain vertices appear in the sweep plane.
3. A bitangent becomes collinear with an active scene bitangent.
4. The convex hull of a contour changes.

To keep track of these four possible ways to change the set of active bitangents, we first keep track of all outer contours and their convex hulls. The set of outer contours can be maintained using chains of circular, doubly-linked lists of vertices similar to the data structures presented in [7]. The convex hulls of these chains can be maintained using the kinetic data structure described in [8]. When a vertex appears on the convex hull of an outer contour, the kinetic data structure tells us where the dual line appears on the upper or lower envelope, and therefore we can add it to the chain of vertices defining the convex hull of the contour efficiently. Between pairs of convex hulls, we maintain certificates to ensure that the set of active bitangents does not change, and by theorem 3.1 we have shown that maintaining the set of active bitangents is sufficient to maintain the set of visual lines in the plane. The only certificates needed (as we will show later) are certificates between every scene bitangent and its closest hull bitangent or active scene bitangent (see figure 3.1). The certificate set needs to be updated when a bitangent becomes collinear with an active scene bitangent or hull bitangent and when the hull bitangent changes due to a change in one of the convex hulls of the contours supporting the bitangent. The active scene bitangents and hull bitangents around each convex hull will be stored in a search tree that has been kinetized, permitting efficient searches and inserts. A sketch of the algorithm is presented in algorithm 3.1.

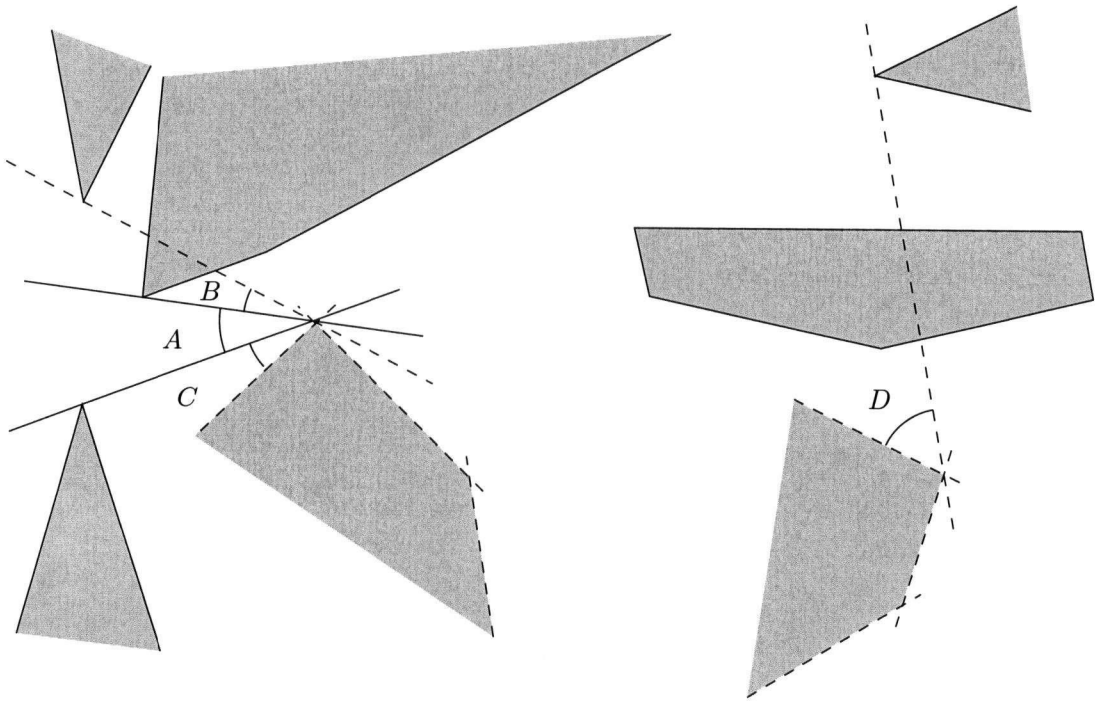


Figure 3.1: The certificate structure for bitangents. Every scene bitangent has a certificate to its closest hull bitangent or its closest active bitangent. This leads to four types of certificates: a certificate between two active scene bitangents (A), a certificate between an active scene bitangent and an inactive scene bitangent (B), a certificate between an active scene bitangent and a hull bitangent (C), and a certificate between a inactive scene bitangent and a hull bitangent (D).

Algorithm 3.1 Layout of the Algorithm

```
Let v_1...v_m be the m vertices of the terrain.
Let e_1...e_n be the n edges of the terrain.
Sort v_1...v_m by height value.
Insert all v_1...v_m as vertex slice events into the event queue.
for i = 1 to n do
  e_i.keep ← false
  e_i.B ← ∅
end for
while !eventQueue.isEmpty() do
  currEvent ← eventQueue.pop()
  if currEvent.type == vertex slice event then
    if currEvent.v is a peak vertex then
      processPeakVertex (currEvent.v)
    else
      processNonPeakVertex (currEvent.v)
    end if
  else
    if currEvent.type == Bitangent Certificate Failure then
      bitangentFailure (currEvent.b1, currEvent.b2)
      updateSearchTree (currEvent)
    else
      convexHullCertificateFailure (currEvent.slv1, currEvent.slv2)
      if convex hull of contour changed then
        updateSearchTree (currEvent)
      end if
    end if
  end if
end while
```

In the following subsections, we will provide details on how each of these parts of the algorithm function independently.

3.2.1 Peak Vertex Events

When a peak vertex event occurs, we introduce a new contour into the sweep plane, along with a new set of bitangents. We compute the convex hull of the new contour and distribute the new set of bitangents around it. To determine the activity of the bitangent being inserted, however, we need some more information about active scene bitangents and visual lines.

The cyclic list of tangents to a contour is divided into lists of tangents that are visual lines and lists of tangents that are not by active bitangents. For example, consider figure 3.2. We notice that if we were to continuously rotate bitangent a counterclockwise into b around polygon P, all intermediate lines would be visual lines or active bitangents. However, if we were to rotate bitangent b counterclockwise into c, all the intermediate lines are not visual lines. This idea is directly related to the concept of *families* of visual lines [30], *visual number* [30], and the rotational sweep used in [40] and [42]. We prove this here.

Lemma 3.2. *The cyclic list of tangents to a contour is divided into lists of tangents that are visual lines and lists of tangents that are not by active bitangents. If there are no scene bitangents, all tangents are visual lines.*

Proof. Let A and B be two consecutive lists of tangents where the tangents of A are visual lines and the tangents of B are not. By the definition of a visual line, there must exist a polygon in the plane to properly intersect every tangent of B. When the polygon ends, by the definition of bitangent, a bitangent must exist dividing the lists A and B. Since A consists of visual lines, there must be free space to one side of that bitangent and therefore it must be active.

If there are no scene bitangents to the point or convex hull there are no other contours in the scene. Thus, all points support visual lines. □

When a peak vertex event occurs, around each contour we have a kinetized search

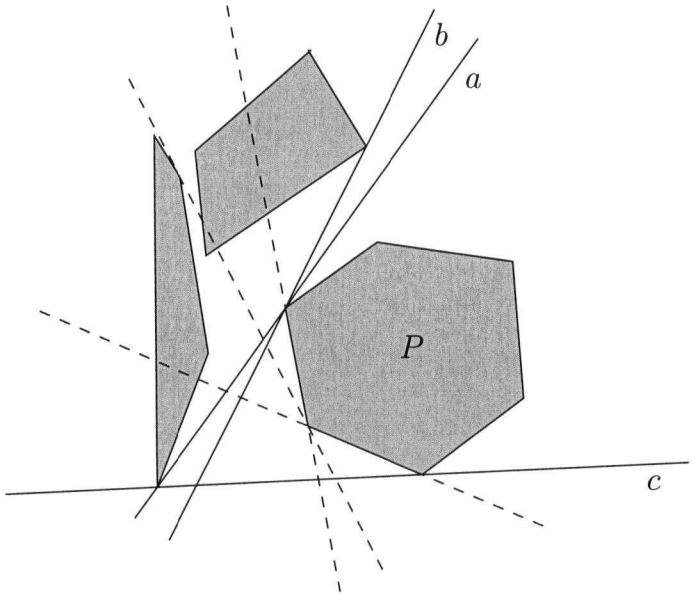


Figure 3.2: The following diagram demonstrates how active bitangents delimit transitions between areas of the slice where only active or inactive bitangents exist.

tree based on the bitangents to its convex hull. We can perform binary search to determine the set of new bitangents to the peak vertex (see figure 3.3). Let us define a bitangent b and a bitangent l as being *adjacent* to each other on a polygon P if their slopes are adjacent in the cyclically ordered list of slopes around P. The activity of a new bitangent l from convex hull P to peak vertex v can be determined in constant time by using the activity of its adjacent bitangents as we prove here.

Lemma 3.3. *A tangent l to a convex polygon P is a visual line if and only if the two adjacent bitangents (b_1 and b_2) to l on P are active and l is not on the same side as the other supporting hull of b_1 or b_2.*

Proof. The bitangents b_1 and b_2 could both be inactive, they could both be active, or one could be inactive and the other could be active. The cases are drawn in figure 3.3.

If either b_1 or b_2 is inactive, we can rotate the inactive bitangent into the other. As no active scene bitangent is passed, all the lines encountered are not visual lines (lemma 3.2). Therefore by theorem 3.1, l is not a visual line.

If both b_1 and b_2 are active, l is in a list of tangents that are visual lines or l is in a list of tangents that are not visual lines (lemma 3.2). If l is in a list of tangents that are visual lines, when b_1 is rotated toward b_2 all intermediate lines are visual lines. Therefore, b_1 must not intersect its other support polygon when rotated toward b_2 as it would not be a visual line. Therefore, l must not lie on the same side as the other supporting hull of b_1. Otherwise, l is in a list of tangents that are not visual lines as b_1 must intersect its other support polygon when rotated toward b_2. □

We perform binary search on the set of active scene bitangents and hull bitangents around each of the k convex hulls in order to find the new scene bitangents to the peak vertex P as they do in [6] (see figure 3.4): determine if the point P lies in the region defined by the two tangents and a ray c between them. The bitangents are selected from the leftmost a and rightmost b leaf of the left subtree at any internal node v. If the bitangent is a scene bitangent the vertex of the ray is the vertex of the scene bitangent on the contour. If it is a hull bitangent the vertex of the ray is the vertex at the tip of the vector that defines the directed segment. The search progresses by asking if the peak is in either region A (wedge between the first bitangent and the ray) or B (wedge between the ray and the second bitangent) (left of figure 3.4). If it is then the region defined by A and B is recursively searched. If not, the right subtree which defines region C, is subdivided using its left and right subtrees into $C.A$, $C.B$, and $C.C$, and those regions are recursively searched (right of figure 3.4). If it is none of the three, it is inside the convex hull or in a region D. If the peak lies in some region D then its points are added to the kinetic data structure of the convex hull that contains it. It is important to note that at the first step of the algorithm region D is not defined.

Once all new scene bitangents have been determined, we sort them by slope around the peak vertex. We then distribute the bitangents around the new contour that forms beneath the peak vertex. We lower the sweep plane a small distance δ below the peak vertex, and take the convex hull of the contour. Since taking the convex hull sorts the new contour's upper hull and lower hull edges by slope, we merge the hull bitangents of the upper and lower hulls with the sorted list of scene bitangents. We then place certificates by re-

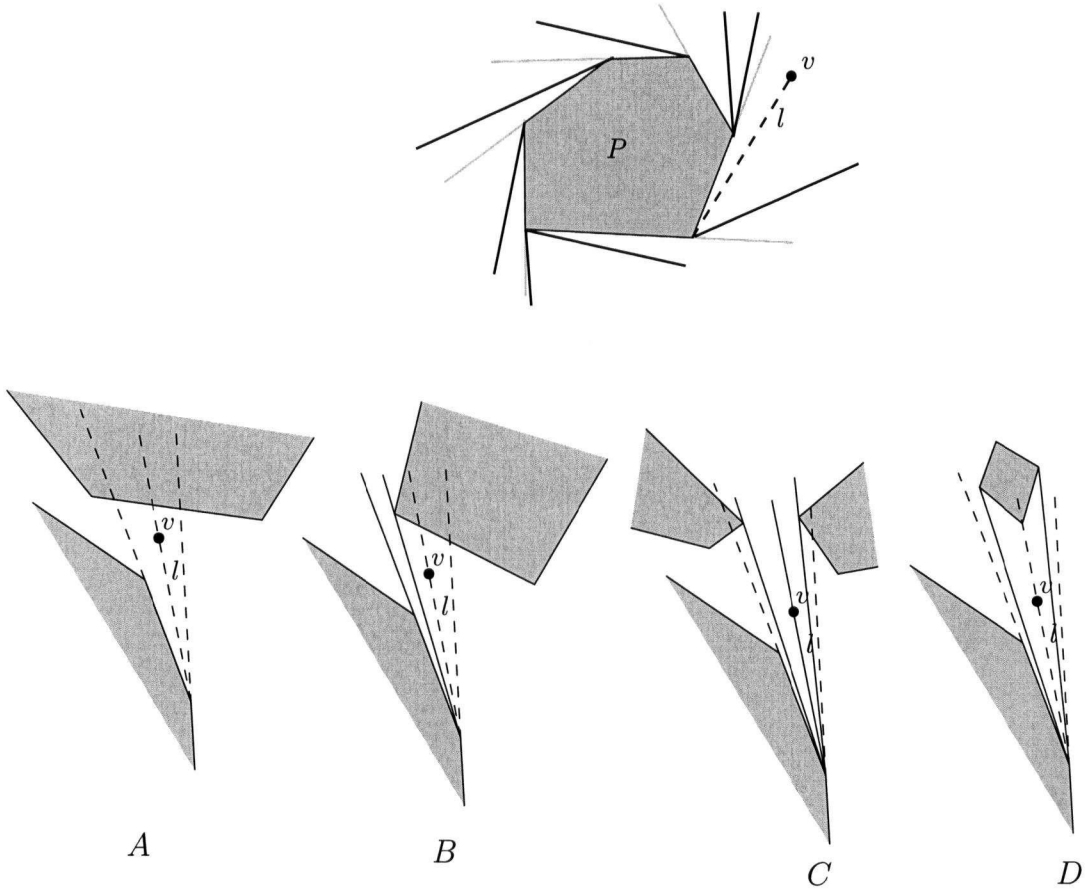


Figure 3.3: The following diagram shows how a counterclockwise binary search can be done (top) and the four cases for determining the activity of the new bitangent (bottom). The cases are: both adjacent bitangents are inactive (A), one adjacent bitangent is inactive and the other is active (B), both adjacent bitangents are active and l is in a list of tangents that are visual lines (C), and both both adjacent bitangents are active and l is in a list of tangents that are not visual lines (D).

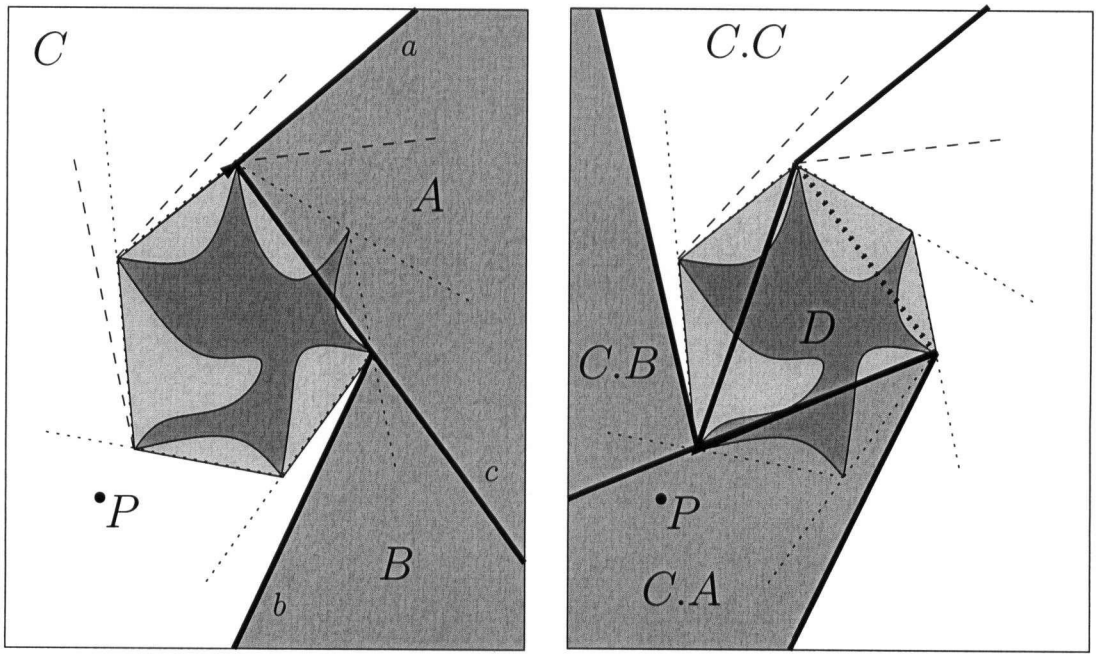


Figure 3.4: Searching the plane for a peak vertex P using the algorithm described in [6]. After the search, the point is either determined to lie inside the convex hull or the tangent line between the peak vertex and the convex hull has been determined. Left is the first step of the search at the root of the tree, while right is the second step of the search within the right subtree of the root.

membering the last active scene bitangent or hull bitangent passed and the list of scene bitangents placed since then. The entire process is shown in figure 3.5.

We would like to ensure that the algorithm as described satisfies the following property after a peak vertex event.

Lemma 3.4. *The new contour has all scene bitangents ordered around the convex hull after its peak vertex event has occurred. Each scene bitangent around the contour is active if it does not properly intersect another contour in the scene and it is inactive if it does.*

To ensure that this is the case, we prove three lemmas:

1. All scene bitangents to the peak vertex are found and are active if they do not properly intersect a contour and inactive if they do.

2. There exists a δ by which the plane can be lowered from a peak vertex so that the counterclockwise order of the vertices on the hull below the peak vertex do not change and the slopes of the convex hull edges do not change.

3. The scene bitangents are distributed around the convex hull of the new contour such that the list of bitangents is ordered around the convex hull.

Lemma 3.5. *All scene bitangents to the peak vertex are found and are active if they do not properly intersect a contour and inactive if they do.*

Proof. Since the peak vertex is only a point at this elevation, it suffices to determine where the peak vertex would lie on the convex hull of each contour in the slice to determine the new bitangents. Binary search can be used to add the peak to a convex hull and determine if the point lies inside the convex hull [6] (figure 3.4). By lemma 3.3, we can check the activity of the adjacent bitangents and the side on which the other support contour of one them lies to determine the activity of the new bitangent. Therefore, all new bitangents will be active if they do not properly intersect a contour of the slice and inactive if they do. □

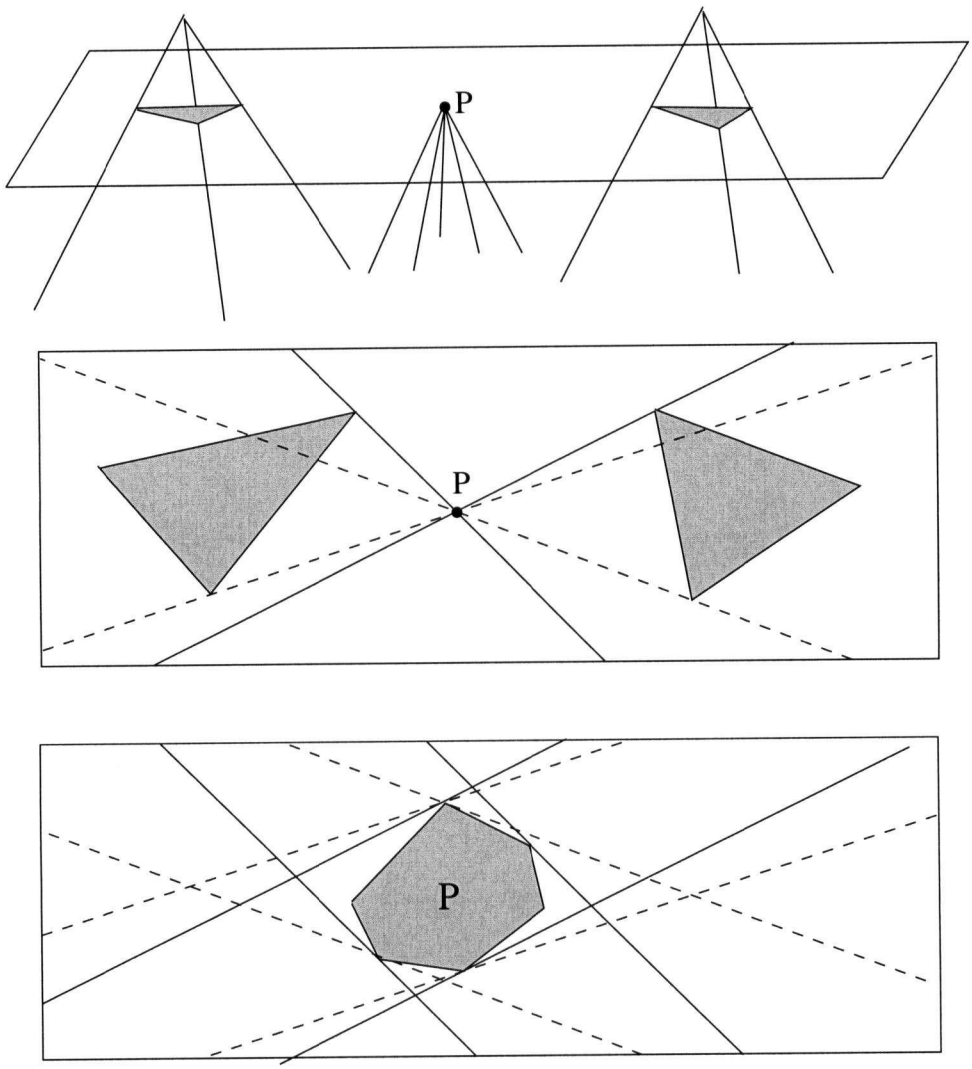


Figure 3.5: The following diagram shows the bitangents around a peak vertex in a slice of terrain and the distribution of those bitangents around the new contour.

Lemma 3.6. *There exists a δ by which the plane can be lowered from a peak vertex so that the counterclockwise order of the vertices on the hull below the peak vertex do not change and the slopes of the convex hull edges do not change.*

To prove lemma 3.6 we prove the two parts of it independently.

Lemma 3.7. *There exists a δ by which the plane can be lowered from a peak vertex so that the counterclockwise order of the vertices on the hull below the peak vertex do not change.*

Proof. We will demonstrate that any δ that is at an elevation above all downhill terrain vertices of the edges adjacent to the peak vertex is sufficient. We will do this by examining the more general case of three points moving with linear trajectories in the plane.

Let $v_1(t)$, $v_2(t)$, and $v_3(t)$ be the velocities of the vertices. The velocities will be further decomposed as follows: $v_1(t) = v_1 t + d_1$ where v_1 and d_1 are constant two dimensional vectors that do not depend on t representing the velocity and initial starting positions of point 1. The points $v_2(t)$ and $v_3(t)$ are decomposed in a similar fashion.

We will split the velocities of the vertices into components: $v_1^x x$ and v_1^y will be the x and y components of the velocity of vertex 1 (vertex 2 and 3 are defined in the same way). Similarly, d_1^x and d_1^y will be the x and y coordinates of the start position.

The slope $m(t)$ for the line between vertices 1 and 2 over time is:

$$m(t) = \frac{(v_1^y t + d_1^y - v_2^y t - d_2^y)}{(v_1^x t + d_1^x - v_2^x t - d_2^x)}$$

Therefore the y-intercept is $b(t) = v_1^y t + d_1^y - (v_1^x t + d_1^x) m(t)$ for a line $y = m(t)x + b(t)$.

If we substitute $v_3^x t + d_3^x$ for x and $v_3^y t + d_3^y$ for y, we get the following quadratic:

$$\begin{aligned}
&((v_1^y - v_2^y)(v_3^x - v_1^x) + (v_1^y - v_3^y)(v_1^x - v_2^x))t^2 \\
&+((v_1^y - v_2^y)(d_3^x - d_1^x) + (v_3^x - v_1^x)(d_1^y - d_2^y) \\
&+(v_1^y - v_3^y)(d_1^x - d_2^x) + (v_1^x - v_2^x)(d_1^y - d_3^y))t \\
&+(d_1^y - d_2^y)(d_3^x - d_1^x) + (d_1^y - d_3^y)(d_1^x - d_2^x) = 0
\end{aligned}$$

We notice that for $d_1^x = d_2^x = d_3^x$ and $d_1^y = d_2^y = d_3^y$ both roots of the quadratic are zero. Thus, since all slice vertices of the hull start at the same location, they will only be collinear with an edge of the convex hull at $t = 0$, and will remain in the same half-plane of the convex hull edge until the lifetime of one of the slice vertices adjacent to the peak ends. Thus, there exists a δ by which the plane can be lowered that preserves the order of the vertices on the convex hull. □

We will now demonstrate that the slopes of those convex hull edges do not change either.

Lemma 3.8. *There exists a δ by which the plane can be lowered from a peak vertex so that the slopes of the the convex hull edges below the peak vertex do not change.*

Proof. Recall that the slope of a line between two slice vertices 1 and 2 is given by:

$$m(t) = \frac{v_1^y t + d_1^y - v_2^y t - d_2^y}{v_1^x t + d_1^x - v_2^x t - d_2^x}$$

We notice once again that $d_1^x = d_2^x$ and $d_1^y = d_2^y$ so:

$$m(t) = \frac{(v_1^y - v_2^y)t}{(v_1^x - v_2^x)t}$$

$$m = \frac{v_1^y - v_2^y}{v_1^x - v_2^x}$$

This is constant as the velocities of vertices 1 and 2 are constant. □

Lemma 3.9. *The scene bitangents are distributed around the upper and lower hulls of the new contour such that the list of bitangents is ordered by slope.*

Proof. We have explicitly sorted the scene bitangents and we have sorted the hull bitangents by taking the convex hull of the new contour. We know that merging two sorted lists produces a combined list that is sorted. □

We have therefore satisfied all of the requirements of lemma 3.4.

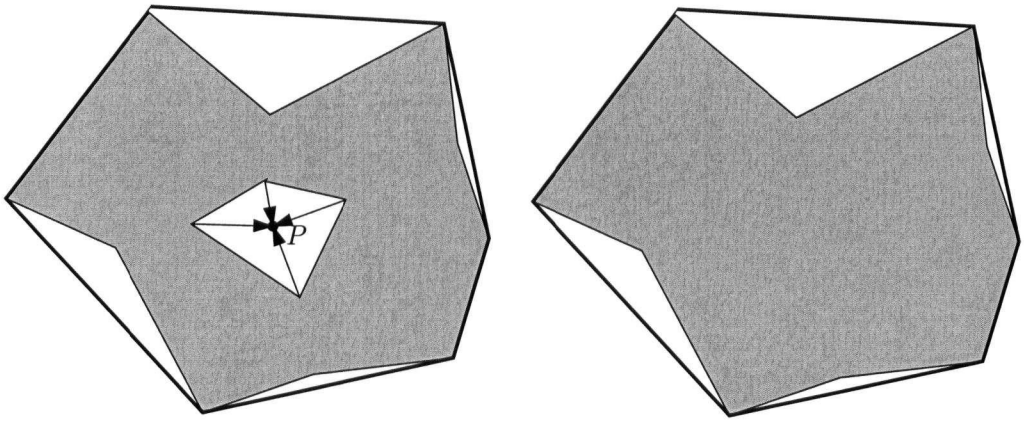


Figure 3.6: A local minimum in the terrain (P). As a local minimum is always surrounded by terrain, it is always to the interior of a convex hull of a contour.

3.2.2 Non-Peak Vertex Events

If the terrain vertex is not a peak vertex (in which all incident edges are downhill edges), it is either a local minima (the edges in counterclockwise order around the non-peak vertex are all uphill), a non-critical vertex (the edges in counterclockwise order around the non-peak vertex form a sequence of uphill edges followed by a sequence of downhill edges), or a saddle point (the edges in counterclockwise order around the non-peak vertex form several sequences (alternating) of uphill and downhill edges) [7]. In this section, we will discuss what our algorithm does to handle these three types of non-peak vertices.

At a local minima, contours in the sweep plane disappear (figure 3.6). As they are inner contours, they cannot be on a convex hull and there are no bitangents to any of their slice vertices. We only need to delete these vertices from the convex hull maintenance structure which is discussed in section 3.2.4.

At a non-critical vertex, several slice vertices corresponding to uphill edges are removed and several slice vertices corresponding to downhill edges are added (figure 3.7). The non-critical vertex could be inside the convex hull of the outer contour in which case the slice vertices are removed and added from the convex hull maintenance structure. If the non-critical vertex is on the convex hull, then its uphill slice vertices could support bitangents. If they do, then those bitangents are distributed just below the non-critical

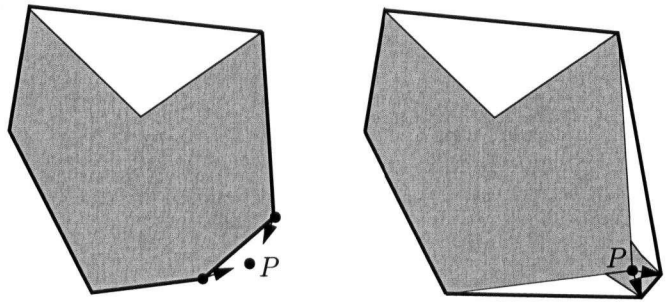


Figure 3.7: A non-critical vertex of the terrain (P). These add and delete slice vertices from a contour. Some of these vertices for the convex hull. If so, the affected bitangents are distributed as they were for a peak vertex.

vertex in the same way as they were for the peak vertex.

At a saddle point several contours merge or a single contour may become many contours (figure 3.8). Fortunately, in our algorithm, we only need to consider the convex hull of each contour and therefore we are only required to merge one or more convex hulls in the slice. These convex hulls will not separate as contours delimit the boundary between function values that are above and below the current altitude and cannot be passed twice in a sweep from highest to lowest altitude. We merge the convex hull maintenance structures by rebuilding them. We determine which bitangents to keep by using rotating parallel tangents on a pair of convex hulls and then on each subsequent convex hull with the result. We can decide in constant time if a point is inside the merged convex hull, on the boundary of the merged convex hull, or supports a bridge to the merged convex hull (figure 3.9). If the slice vertex is inside the merged convex hull, all of its bitangents are deleted. If the slice vertex is on the boundary of the merged convex hull all of its bitangents are kept. If the slice vertex supports a bridge on the merged convex hull all of the bitangents that penetrate the merged convex hull are deleted and all others are kept. For the bitangents that are kept, any certificates with hull bitangents not the merged convex hull are updated to become a certificate with the bridge.

We would like to ensure that the algorithm satisfies the following property after a non-peak vertex event.

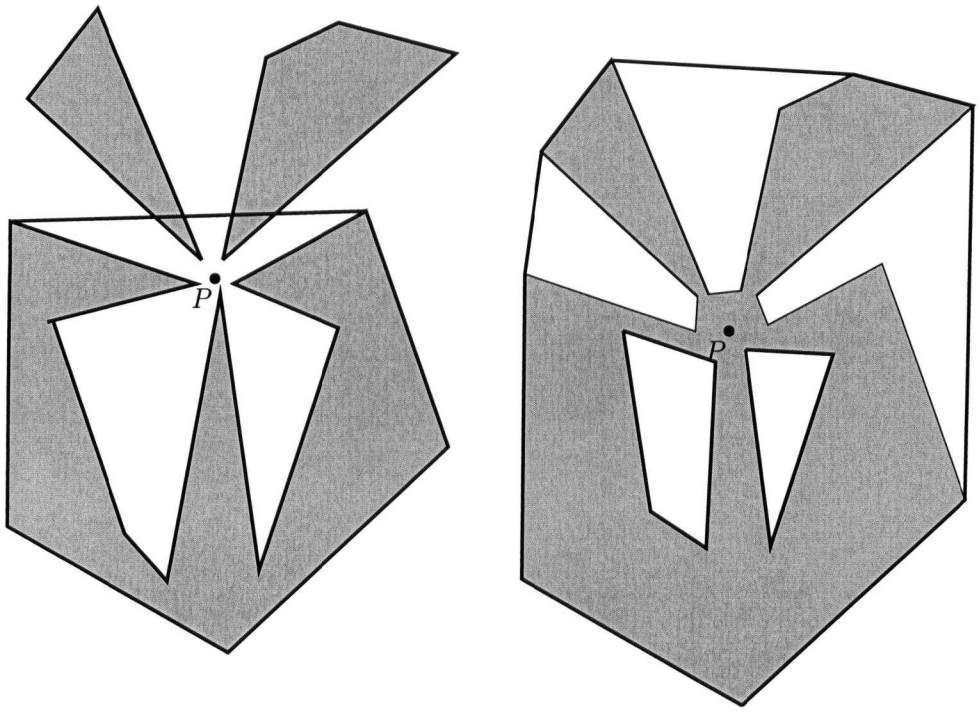


Figure 3.8: At a saddle point (P), several convex hulls of contours merge. We perform walking tangents to determine which bitangents are kept

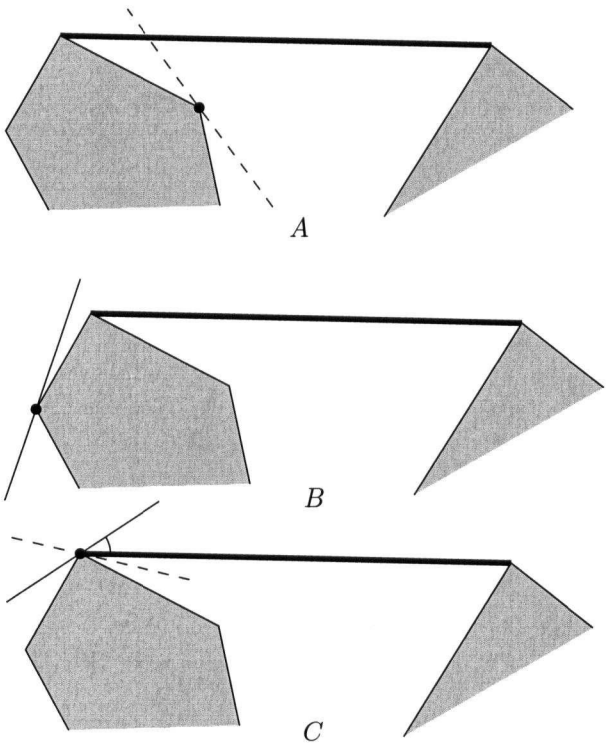


Figure 3.9: The three cases for a slice vertex on a merging convex hull: inside the boundary of the convex hull (A), on the boundary of the convex hull (B), and incident to a bridge on the convex hull (C). Dashed lines indicate the bitangent will be deleted. The new bitangent certificate to the bridge is indicated by the small arc in (C).

Lemma 3.10. *After a non-peak vertex event, the (possibly) merged contour has a valid set of bitangent certificates around its convex hull. Each scene bitangent around the contour is active if it does not properly intersect another contour in the scene and it is inactive if it does.*

To ensure that this is the case, we prove three lemmas:

1. The bitangents (if any) that are tangent to one of the merging contours and intersect the final merged convex hull are removed. Bitangents that do not intersect the final merged convex hull are retained and their certificates are valid.

2. There exists a δ by which the sweep plane can be lowered below the non-peak vertex where the counterclockwise order of the vertices on new convex chain do not change and the slopes of the edges of the new convex chain do not change.

3. The scene bitangents are distributed around the upper and lower hulls of the merged convex hull such that the list of scene bitangents and hull bitangents are ordered by slope.

The final two items can be shown to be true with simple modifications to lemmas 3.6 and 3.9. The first item requires some proof.

Lemma 3.11. *The bitangents (if any) that are tangent to one of the merging contours and intersect the final merged convex hull are removed. Bitangents that do not intersect the final merged convex hull are retained and their certificates are valid.*

Proof. At a local minima, slice vertices are not on the convex hull. Therefore they do not support bitangents.

At a non-critical vertex, slice vertices may be on the convex hull and can support bitangents. Those bitangents are distributed as they were for a peak vertex.

At a saddle, several convex hulls of contours can merge. A particular slice vertex can lie inside the boundary of the convex hull, lie on the boundary of the convex hull, or supports a bridge between two merging convex hulls.

If a slice vertex is inside the boundary of the convex hull of a contour, any line that passes through it intersects the convex hull. Therefore, any bitangent to that vertex is not a bitangent after the merge and the bitangent is removed.

For any slice vertex on the boundary of the convex hull, both adjacent convex hull edges were on the same merging contour's convex hull. Its hull edges are not affected by the merge and therefore all of its bitangents are kept and their certificates are valid.

For any slice vertex that supports a bridge between two merging convex hulls, the scene bitangents that intersect the merged convex hull need to be removed since they are no longer bitangents. A bitangent to a slice vertex that is adjacent to a bridge intersects the convex hull if the slope of the active or inactive bitangent does not lie between the slope of the bridge and the other convex hull edge. Therefore, finding the bridge between the two merging convex hulls is sufficient to find all bitangents that intersect the merged convex hull. All kept bitangent certificates are valid after, assuming they were valid before, and those bitangents with a certificate to a hull edge interior to the convex hull will have their certificate updated. □

We have therefore satisfied all of the properties of lemma 3.10.

3.2.3 Bitangent Certificate Failures

A bitangent certificate fails when a bitangent becomes collinear with an active scene bitangent or hull bitangent, causing either a change to the set of bitangents or a change to the set of active bitangents. After the failure, a new certificate to the next convex hull edge or active bitangent is computed in constant time per bitangent as the list of active scene bitangents and hull bitangents is known and stored at the leaves of the search tree. If the certificate failure causes two convex hulls to intersect, the inner bitangents between them are deleted (figure 3.10). The test to see if a bitangent certificate failure caused the two convex hulls to intersect is as follows: see if the vertex of the other polygon lies on the line segment (c, d) (top) or not (bottom). If the vertex lies on the line segment (c, d), the convex hulls intersect. The process is described in pseudo-code in algorithm 3.2. Bitangent $b1$ has greater slope

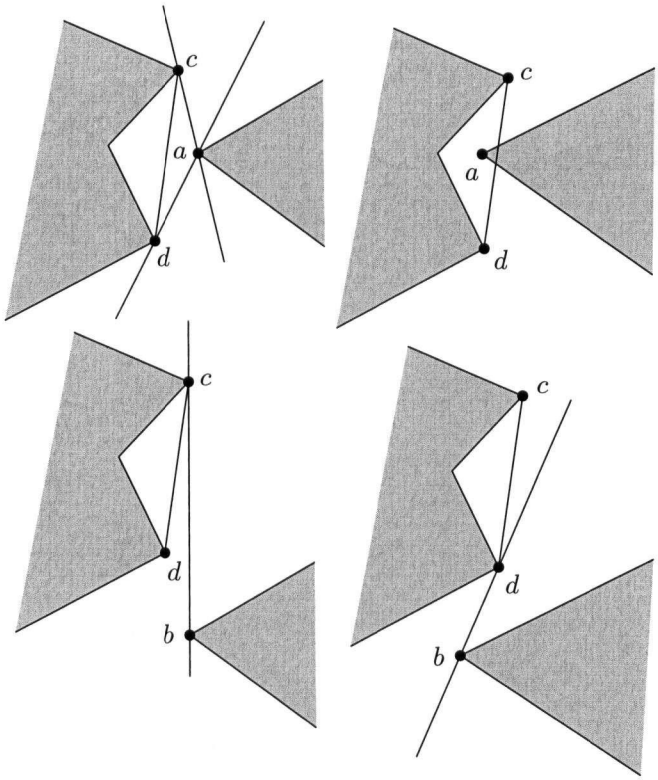


Figure 3.10: Test to see if the other end point of a bitangent lies inside the convex hull after a certificate failure with a hull bitangent. We check to see if the point on the other convex hull lies on the line segment (c, d) (top) or not on the line segment (bottom).

than $b2$ before they become collinear. For a bitangent b, the value of *prev* is the adjacent active scene bitangent or hull bitangent of slope less than b where *next* is the adjacent active bitangent or hull bitangent of slope greater than b.

We must demonstrate that bitangent certificates satisfy the following requirements.

1. After the certificate failure, the algorithm's set of bitangents and active bitangents matches the true set of bitangents and active bitangents.

2. For every bitangent there is a bitangent certificate to its adjacent hull bitangent or active scene bitangent after a certificate failure.

Lemma 3.12. *After the certificate failure, the algorithm's set of bitangents and active bitangents matches the true set of bitangents and active bitangents.*

Algorithm 3.2 Algorithm for handling bitangent certificate failures.

bitangentFailure (bitangent $b1$, bitangent $b2$)

```
if (b1 and b2 are scene bitangents) then
  if (b1.active and b2.active) then
    for every certificate to b1 recompute certificate to b2.prev
    for every certificate to b2 recompute certificate to b1.next
  else if (b1.active or b2.active) then
    Determine inactive bitangent i
    Determine active bitangent a
    every certificate to a becomes certificate to i
    Create a certificate between a and i
  else
    for every certificate to b1 recompute to b2.prev
    for every certificate to b2 recompute to b1.next
  end if
  b1.active ← !b1.active
  b2.active ← !b2.active
else
  Determine the scene bitangent s
  Determine the hull bitangent h
  if (other end point of s inside h's convex hull) then
    Delete s
  end if
  if (s.active) then
    h.active ← !h.active
  end if
  Update the bitangent certificates of the scene bitangent
end if
```

Proof. For the set of active bitangents to change, an active bitangent begins to or ceases to properly intersect part of the terrain. This means that a bitangent must become collinear with an active scene bitangent as the slice vertex begins or ceases to support a visual line (lemma 3.2 and theorem 3.1). This active scene bitangent a could become collinear with either a hull bitangent or a scene bitangent, which could be either inactive or active. Therefore, we have four cases. In all of these four cases, we label the other bitangent b. We will demonstrate that algorithm 3.2 handles these four cases in such a way that the true set of active bitangents is known by the algorithm. Three of these cases are shown in figure 3.11.

Let us first assume that b is a scene bitangent. The bitangents have one common support vertex just before the certificate failure, which we will call v. If the bitangent certificate fails a and b have two unique support vertices as well. Those support vertices will be a part of the convex hull of two contours P and Q respectively. The bitangent a is tangent to two slice vertices in the sweep plane and touches no other. When the certificate between a and b fails, a and b have become collinear and a begins to intersect Q. Thus, a becomes inactive. The bitangent b changes sides with respect to an active bitangent and therefore it inverts its activity (lemma 3.2 and theorem 3.1).

If b is a hull bitangent, b is on one of the support hulls of a. Let us call that hull P and the other hull is Q. When a changes order with b, a changes support vertex. Since a is an active scene bitangent, it divides the perimeter of the convex hull into lists of tangents that are visual lines and lists of tangents that are not (lemma 3.2 and theorem 3.1). Thus, the activity of b is inverted. The activity of a remains active as nothing begins to properly intersect it. □

Lemma 3.13. *For every bitangent there is a bitangent certificate to its adjacent hull bitangent or active scene bitangent after a certificate failure.*

Proof. The certificate structure changes when a hull bitangent and a scene bitangent become collinear or when a scene bitangent becomes collinear with an active scene bitangent. If a bitangent certificate between a scene bitangent and a hull bitangent fails, the certificate

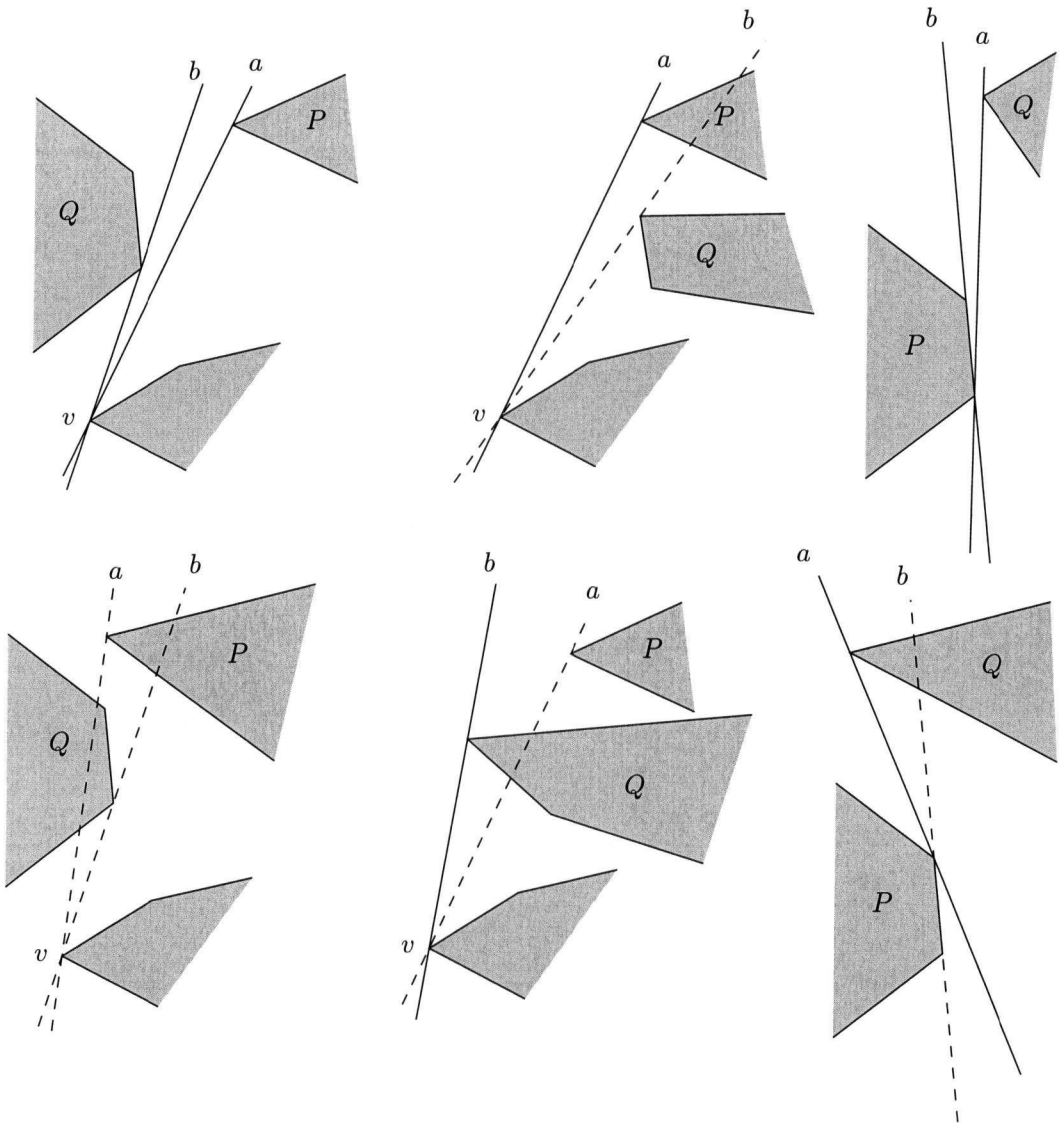


Figure 3.11: Shows how the activity of the bitangents is adjusted during certificate failures. Left shows a certificate failure between two active scene bitangents. Center shows a certificate failure between an inactive scene bitangent and an active scene bitangent. Right shows a failure between an active scene bitangent and a hull bitangent.

is passed onto the next hull bitangent or active scene bitangent. Thus, the certificate structure remains valid. We now concern ourselves with the situation when a scene bitangent becomes collinear with an active scene bitangent.

When a scene bitangent becomes collinear with another scene bitangent, three points in the sweep plane become collinear. Thus, $\binom{3}{2}$ or three bitangents become collinear. At any vertex we have either two active bitangents, an active and an inactive bitangent, or two inactive bitangents (see figure 3.12).

If we have two active bitangents become collinear, they will become inactive after they do, as each changes sides with respect to an active bitangent (lemma 3.2 and theorem 3.1). The certificates to the two active scene bitangents share with inactive scene bitangents are updated using the list of active scene bitangents and hull bitangents at the leaves of the search tree.

If an active scene bitangent becomes collinear with a inactive scene bitangent, the activity of both are inverted. The new active scene bitangent takes all the certificates that the active scene bitangent had previously.

Finally, if two inactive bitangents become collinear and become active (due to the third bitangent being active), certificates must be computed to the new active scene bitangents. The set of inactive bitangents that share the same support vertex as the splitting inactive bitangents are tested for sidedness and a bitangent certificate is placed to the appropriate active bitangent.

In all three cases, the certificate set is valid as every scene bitangent has a certificate to its adjacent hull bitangent or active scene bitangent to either side. □

3.2.4 Convex Hull Maintenance

We maintain the convex hull of the contours using the kinetic data structure described in [8]. Insertions will be done always to the right adding a path to the tree. Merges require rebuilding the entire structure. In this section, we start by describing how to build the kinetic data structure presented in [8]. We then describe how inserts, merges, and deletions

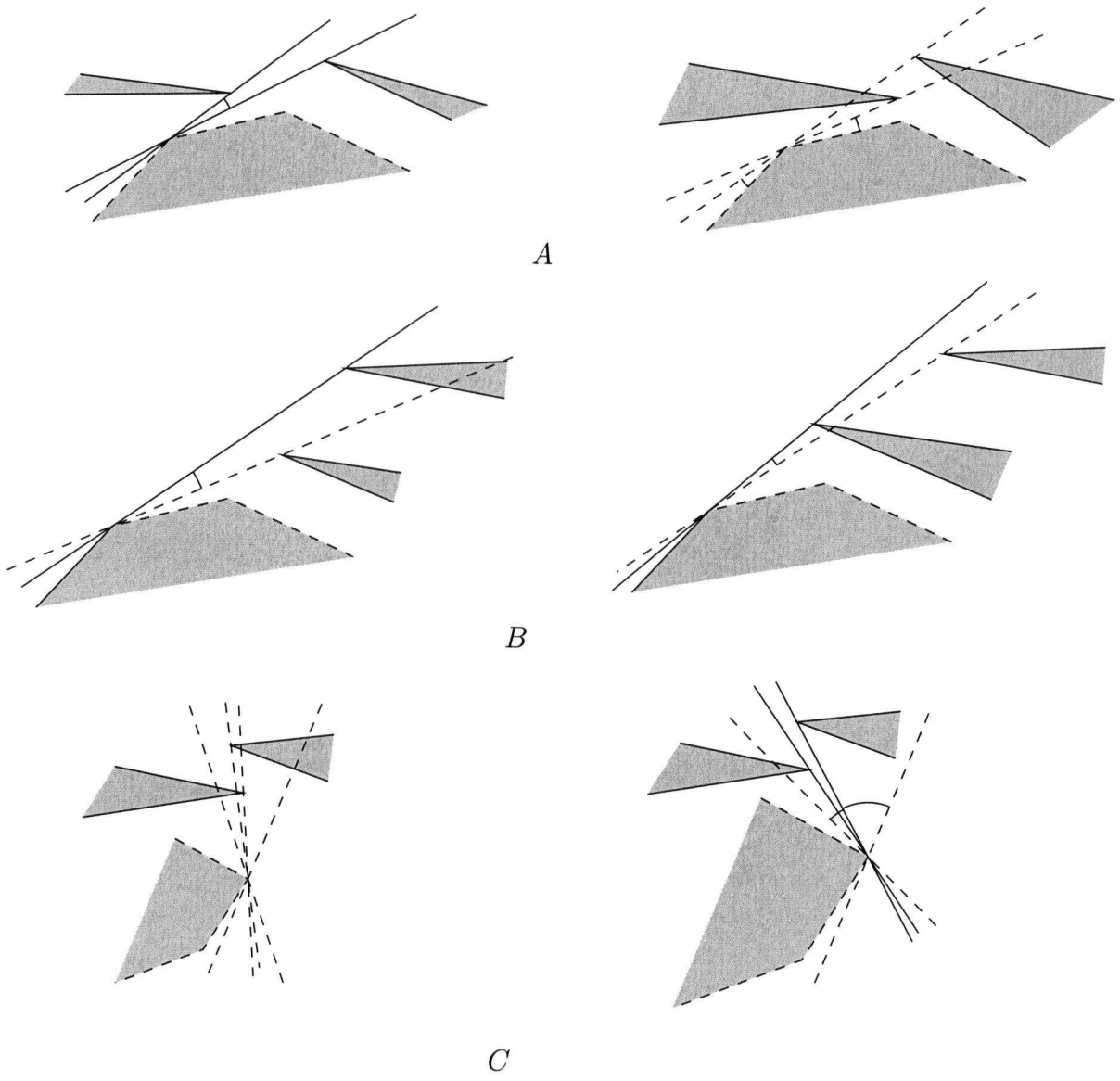


Figure 3.12: The three ways the certificate structure between scene bitangents can change: active scene bitangent and active scene bitangent (A), active scene bitangent and inactive scene bitangent (B), and inactive scene bitangent and inactive scene bitangent (C).

are done.

Building the Structure

Rebuilding the recursive divide and conquer tree is easy, but we must also describe how to generate the set of certificates for each internal node inside the tree. The set of certificates will require three passes to determine the x certificates, the yli yri certificates, and all remaining certificates. We will only describe this construction for the upper hull as the construction for the lower hull is symmetric.

We start by computing the set of x certificates. In dual space, the vertices of the upper envelope correspond to the edges of the convex hulls in primal. As we know the convex hulls of the two children of an internal node v, we can obtain a list of the dual vertices of their upper envelopes sorted by x-coordinate. The x certificates can now be computed by merging the two sorted lists of dual vertices and noting when two vertices from different children are placed side by side.

We can compute the set of bridges for the convex hull by walking around the two convex hulls with parallel tangents. Whenever a bridge is created, we simply create a pair of yli yri certificates in dual.

Finally, we show how to compute the yt, slt, srt, sl, and sr certificates. Since we know the convex hull of the left (red chain) and right (blue chain) of the children at an internal node, we can obtain a list of dual edges sorted by slope. For this section, $m(x)$ will denote a function that returns the slope of dual edge x.

If we ever place two edges a and b on the merged list of the same colour that are both on the upper envelope at this node, and there exists an edge c' of a different colour that is not on the upper envelope at this node and the slope order is $m(a) < m(c') < m(b)$, then we create yt, slt, and srt certificates (see figure 2.6).

If we ever place an edge a on the merged list that is on the upper envelope, and it was immediately preceded by an edge of the other chain b' which does not appear on the upper envelope and the previous (next) dual vertex is $a'b'$ and is below a then we create a

sl (sr) certificate.

This concludes the description of our algorithm. The algorithm correctly produces the set of certificates for the certificate tree because it produces the correct convex hull via the divide and conquer algorithm and it places certificates only under the conditions presented in [8]. The certificates can be computed as the recursive hull is computed if the parallel tangents for the merge were rotated from minimum x-coordinate to maximum x-coordinate in primal. The dual vertices would also be sorted by x-coordinate as the edges of the upper hull in primal are ordered from greatest to least slope.

Once the tree has been constructed the points are put into motion. We have already seen in chapter two that the tree strongly certifies the upper envelope.

Additions, Deletions, Merges, and Splits

Vertices of contours begin when their edges start intersecting the sweep plane and end when they stop intersecting the sweep plane. Contours can also merge and split, creating larger or smaller outer contour lines in the plane. We must demonstrate how to handle these types of events, using Julien Basch's structure [8].

We always add vertices to the right hand side of the tree. This keeps the tree depth to a reasonable size and the upper/lower envelopes inside each internal node the same. When a vertex is added, a path equal to the current depth of the tree is added. Any preexisting convex hull on that path is recomputed and the event queue is updated.

For vertex deletions, the leaf is deleted and the dual line is deleted from all nodes up the path of the tree. The convex hulls on each of those nodes is recomputed and the event queue is updated.

For merges, we simply reconstruct the tree entirely to determine the proper tree for the new outer contour. For splits, we do nothing and leave the vertex in the tree.

3.2.5 Kinetic Maintenance of the Search Trees

The search trees contain all active bitangents and hull bitangents around a contour in cyclical order. They are stored in a doubly-linked list at the leaves of a balanced binary tree. At each internal node of the binary tree, besides the normal parent and children pointers, we store a pointer to the leftmost leaf in the right subtree and the rightmost leaf in the left subtree. These leaves also store pointers to the internal nodes. Binary search through the tree is done as in [6] by asking if a query point lies in the range defined by the left subtree, or the range defined by the right subtree. We need to show how the tree is updated when a bitangent certificate fails.

If a bitangent certificate fails between two active scene bitangents both bitangents become inactive. This results in two deletions from the tree. We will show later that rebalancing can be done for deletions and thus the tree will remain balanced. When a bitangent certificate fails between an inactive scene bitangent and an active scene bitangent the activity of both bitangents is inverted. We only need to change the contents of a leaf and therefore no insertions are required. When two inactive bitangents become active due to a certificate failure with an active bitangent, this results in two deletions from the tree. When a bitangent certificate fails between an active scene bitangent and a hull bitangent the order of two leaves in the tree is swapped. We will demonstrate that this can be done efficiently. Finally, when the convex hull changes, two hull bitangents are inserted into the tree. As the tree only contains hull bitangents and active scene bitangents, bitangent certificate failures between an inactive bitangent and a hull bitangent do not change the tree.

The contents of two leaves in the doubly-linked list of leaves (figure 3.13) can be swapped efficiently. As there is one pointer for each link between two leaves to an internal node of the tree, there are two pointers per leaf to internal nodes in the tree. Thus, after a swap we have four pointers to update: The two pointers across the internal node that splits at this link and the two other internal nodes that define splits adjacent to the left and the right of this link. No other pointers to internal nodes in the tree need to be updated.

Assuming that we can do rotations to keep our tree balanced (we will show this in

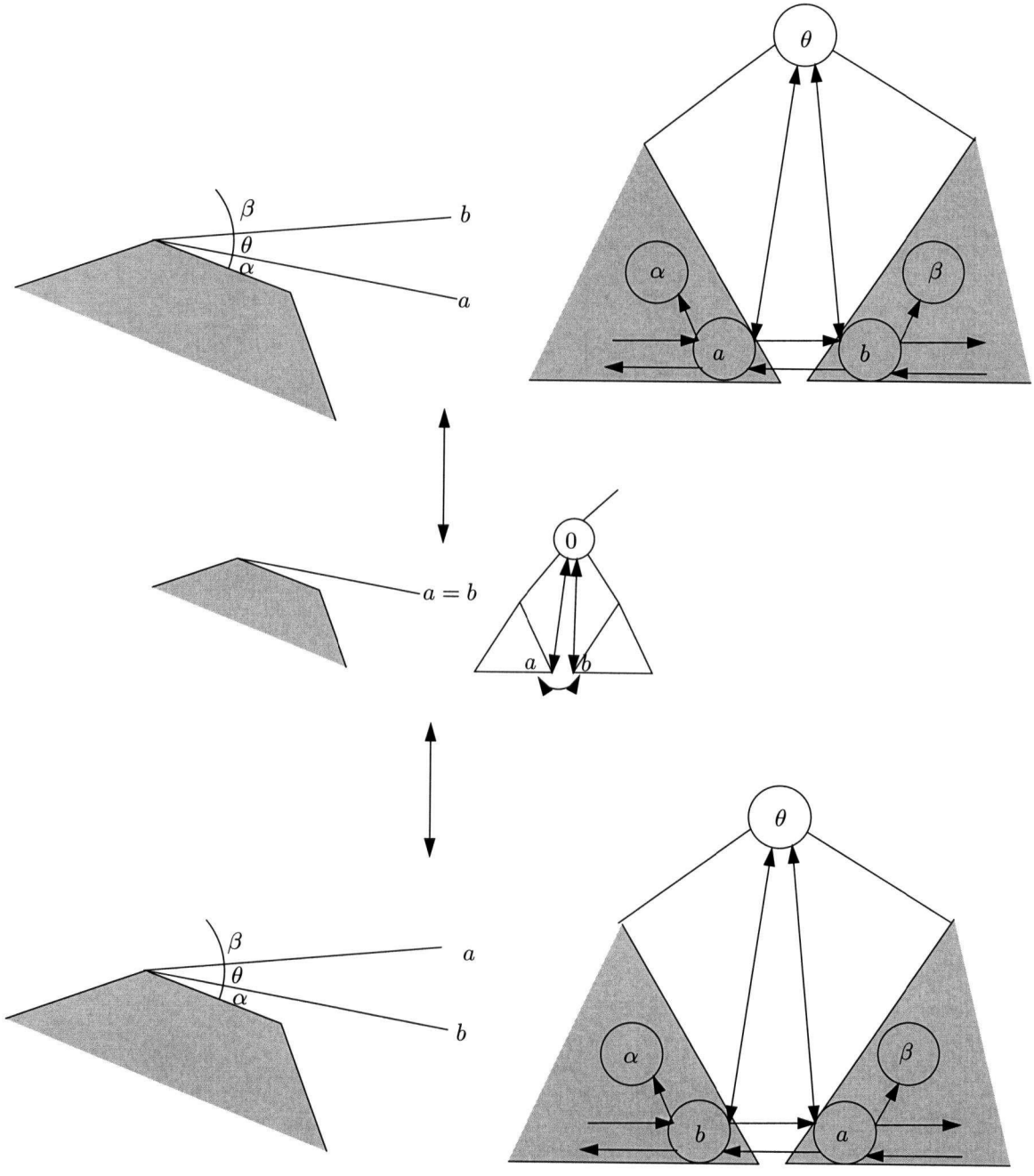


Figure 3.13: Updates to the search tree for bitangents on certificate failures.

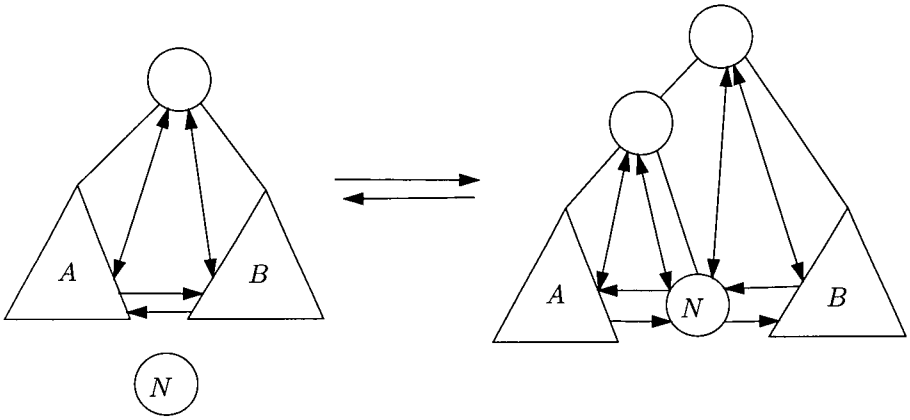


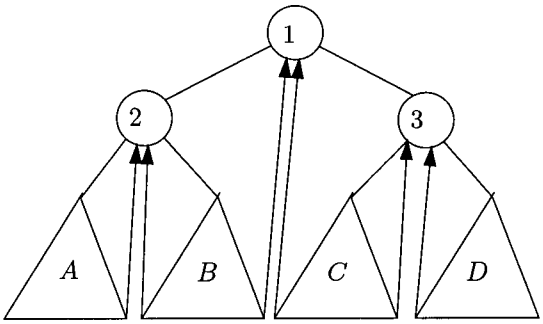
Figure 3.14: Update to the range pointers after an insertion or removal of a leaf from the tree.

the next paragraph), insertions and deletions can be done efficiently. We just need to show that the range pointers can be updated efficiently. For a node N being inserted or removed from the tree, we find where it belongs in the tree, and insertion or removal simply involves changing the range pointers at the adjacent leaves and inserting/deleting an internal node in the tree (figure 3.14). Rotations are then done to re-balance the tree.

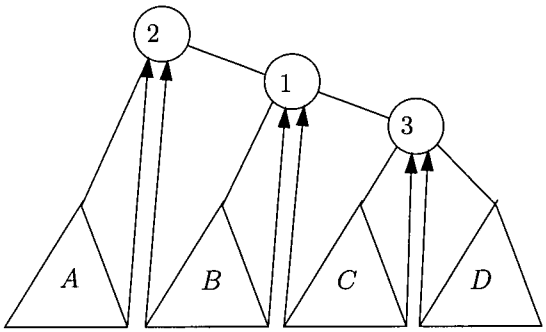
After insertions, the tree could become unbalanced. To ensure the tree remains balanced, we perform AVL rotations at each internal node. Range pointers at an internal node do not need to be updated under rotations as the internal node still divides the list of leaves at the same link (figure 3.15).

3.3 Proof of Overall Correctness

We now demonstrate that the algorithm described above produces all the edges of the terrain that support a visual line. As we have shown in theorem 3.1, this is equivalent to determining if a slice vertex supports an active bitangent in its lifetime. We prove this inductively by showing that the initial state of the algorithm is correct, and that the three cases behave correctly and are adequate for tracking all changes to the set of active bitangents as the sweep progresses.



Right Rotation



Left Rotation

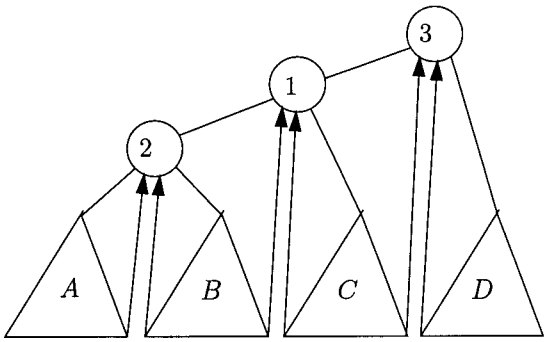


Figure 3.15: Figure illustrating that the range pointers at an internal node need not be updated under left and right AVL rotations. Notice that 1 always points to B and C, 2 always points to A and B, and 3 always points to C and D.

Theorem 3.14. *The algorithm described above produces all edges of the terrain that that support a visual line.*

Lemma 3.15. *Initially, there are no scene bitangents.*

Proof. Initially, there are no contours and therefore there are no active bitangents. Thus, all $e_i.keep \leftarrow$ false and $e_i.B \leftarrow \emptyset$. □

We now show that all changes to the set of active bitangents at any altitude z is known through the sweep through vertex events, bitangent certificate failures, and hull certificate failures. Let $S(z)$ be the set of pairs of terrain edges that support an active bitangent in a slice at a height value of z (ie the set of horizontal active bitangents at altitude z). Let $u(z)$ be the slice vertex obtained by intersecting the terrain edge u with a horizontal plane at altitude z. A slice vertex $u(z)$ is undefined if the terrain edge u does not intersect the horizontal plane at altitude z. A plane at altitude z^- is a plane that is slightly of lower altitude then z and therefore reached slightly after z in a sweep from highest to lowest altitude in the terrain. The operator $\oplus$ indicates the symmetric difference between two sets.

Lemma 3.16. *If* $(u(z), v(z)) \in S(z) \oplus S(z^-)$ *then there exists a vertex event, bitangent certificate failure, or convex hull certificate failure involving* u *or* v *to detect that change in the interval* (z, z^-).

Proof. If $(u(z), v(z)) \in S(z) \oplus S(z^-)$ then either $(u(z), v(z)) \in S(z)$ and $(u(z^-), v(z^-)) \notin S(z^-)$, or $(u(z), v(z)) \notin S(z)$ and $(u(z^-), v(z^-)) \in S(z^-)$. We show that either a vertex event occurs, or a bitangent or convex hull certificate fails when the set of active bitangents changes.

If $(u(z), v(z)) \in S(z)$ and $(u(z^-), v(z^-)) \notin S(z^-)$, then either $u(z^-)$ or $v(z^-)$ is undefined or $(u(z^-), v(z^-))$ properly intersects a contour. If either $u(z^-)$ or $v(z^-)$ is undefined, we have passed a terrain vertex (the lower endpoint of u or v) and it causes either a non-peak vertex event. If $(u(z^-), v(z^-))$ properly intersects a contour, then let $z^* \in (z, z^-)$ be the first time that $(u(z^*), v(z^*))$ intersects a contour and let $w(z^*)$ be the first intersection point (by continuity of motion we know that $w(z^*)$ exists). If $u(z^*)$,

$v(z^*)$ and $w(z^*)$ are on the same contour then there is a change to the convex hull and therefore a convex hull certificate failure. If $u(z^*)$ and $v(z^*)$ are on the same contour and $w(z^*)$ is not, then $w(z^*)$ is on the convex hull of some other contour and it supports a scene bitangent with either $u(z^*)$ or $v(z^*)$. Also, the bitangent $(u(z^*), v(z^*))$ is a hull bitangent. There is a bitangent certificate between every scene bitangent and its adjacent hull bitangent and this certificate fails. If $u(z^*)$ and $v(z^*)$ are on different contours then it is an active scene bitangent and either $(u(z^*), w(z^*))$ or $(w(z^*), v(z^*))$ is a hull bitangent. There is a bitangent certificate between every scene bitangent and its adjacent hull bitangent and this certificate fails. If $u(z^*)$, $v(z^*)$ and $w(z^*)$ are on three different contours, then $(u(z^*), v(z^*))$ is an active scene bitangent and $(u(z^*), w(z^*))$, $(w(z^*), v(z^*))$ are scene bitangents. There are certificates between any scene bitangent and its adjacent active scene bitangent and this certificate will fail.

If $(u(z), v(z)) \notin S(z)$ and $(u(z^-), v(z^-)) \in S(z^-)$ then either $u(z)$ or $v(z)$ is undefined or $(u(z), v(z))$ ceased to properly intersect a contour somewhere in the interval (z, z^-). If either $u(z)$ or $v(z)$ is undefined, we have passed a terrain vertex (the upper endpoint of u or v) and a peak or non-peak vertex event occurs. If $(u(z), v(z))$ ceased to properly intersect the contour in the interval (z, z^-), then let $z^* \in (z, z^-)$ be the first time that $(u(z^*), v(z^*))$ ceases to intersect a contour let $w(z^*)$ be the last intersection point (by continuity of motion we know that $w(z^*)$ exists). If $u(z^*)$, $v(z^*)$ and $w(z^*)$ are on the same contour then there is a change to the convex hull and therefore a convex hull certificate failure. If $u(z^*)$ and $v(z^*)$ are on the same contour and $w(z^*)$ is not, then $w(z^*)$ is on the convex hull of some other contour and it supports a scene bitangent with either $u(z^*)$ or $v(z^*)$. Also, the bitangent $(u(z^*), v(z^*))$ is a hull bitangent. There is a bitangent certificate between every scene bitangent and its adjacent hull bitangent and this certificate fails. If $u(z^*)$ and $v(z^*)$ are on different contours then it is a scene bitangent and either $(u(z^*), w(z^*))$ or $(w(z^*), v(z^*))$ is a hull bitangent. There is a bitangent certificate between every scene bitangent and its adjacent hull bitangent and this certificate fails. If $u(z^*)$, $v(z^*)$ and $w(z^*)$ are on three different contours, then $(u(z^*), v(z^*))$ is an a scene bitangent

and $(u(z^*), w(z^*))$, $(w(z^*), v(z^*))$ are scene bitangents. By lemma 3.2 and theorem 3.1, $(u(z^*), w(z^*))$ or $(w(z^*), v(z^*))$ is active as $(u(z^-), v(z^-))$ is active and must change sides with respect to an active bitangent. As there are certificates between any scene bitangent and its adjacent active scene bitangent, this certificate will fail. □

We can now prove theorem 3.14.

Proof. By lemma 3.15, we know that initially there are no bitangents and therefore $S(0)$ should be empty. By lemma 3.16, we know that if x is an event that effects the set of active bitangents $S(t)$, x is an event in our event queue.

Let X be the set of all x sorted by elevation or time t. Since $S(0)$ is empty and X reports all insertions and deletions into $S(t)$, $S(t)$ will contain all active bitangents encountered in the sweep. By lemma 3.1, this is sufficient to report all terrain edges that support a visual line. □

Chapter 4

Complexity Analysis

We will now analyze the time complexity of the algorithm to determine the set of edges that support at least one visual line on a terrain of n edges and k local maxima.

We first show some tight bounds on the number of changes to contours in the sweep plane and then we show some tight bounds on the number times an active bitangent can become collinear with another bitangent in the scene. We then analyze our algorithm in worst case, using the problem analysis to realize an $O(kn^{2+\epsilon} \lg n \lg(n \lg n + k^2)$ for any $\epsilon > 0$ algorithm.

4.1 Bounds on Geometric Combinatorial Complexity

4.1.1 Changes to the Convex Hull of a Single Contour

In this section, we prove tight bounds on the number of worst case changes to the convex hull of a contour that can happen through the sweep of the terrain.

Lemma 4.1. *There are $O(n^{2+\epsilon})$ for any $\epsilon > 0$ changes to the convex hull of one contour between merges as a sweep plane is lowered from highest to lowest altitude on a terrain. There exists an example with $\Omega(n^2)$ changes.*

This tight bound is shown through lemmas 4.2 and 4.4.

Lemma 4.2. *There are at most $O(n^{2+\epsilon})$ for any $\epsilon > 0$ combinatorial changes to the convex hulls of a contour between merges.*

Proof. Slice vertices in the sweep plane are points that follow linear trajectories. Consider them as line in dual space. When we consider time as the third dimension in this space, we notice that the moving lines become bivariate functions in $\Re^3$ of constant bounded degree (as the motion of the points in primal is of bounded degree). When a slice vertex appears on the convex hull, its bivariate function appears on the upper envelope in dual space. This can happen at most $O(n^{2+\epsilon})$ for any $\epsilon > 0$ times for bivariate functions with a domain partially defined by a constant number of inequalities and a boundary defined by algebraic functions of maximum constant degree [45]. Each terrain vertex delimits the beginning and ending of a set of these bivariate functions. All terrain vertices passed that do not involve merges only add to or delete from the set of vertices that defines the contour. The domains of our bivariate functions are defined by two inequalities (an edge begins and ends) and the complexity of their boundaries are linear (two dual space lines whose slope and y-intercept are determined by the height value at which t begins and ends). Thus, the bound presented in [45] applies. □

Now, we will show that for a polygonal chain that does not self intersect, the convex hull can change $\Omega(n^2)$ times.

Lemma 4.3. *There exists a deforming, non self-intersecting, polygonal chain whose vertices follow linear trajectories that has $\Omega(n^2)$ combinatorial changes to the boundary of its convex hull.*

Proof. Consider the example depicted in figure 4.1. Let t be fixed unit time intervals defined by the time it takes for the point $b_0(t)$ to move one unit along its trajectory. Let j be the index of the points on a convex chain A and let k be the index of the points on a convex chain B. Each vertex of the convex chain A travels in a vertical direction and supports a bitangent with all of the vertices of B in turn, causing $\Omega(n)$ changes to the convex hull.

Since there are n vertices on chain A, this leads to a bound of $\Omega(n^2)$. We will now derive the velocities for the vertices to show that this example can be constructed.

For a vertex $a_j(t)$ of A, we want it to intersect the line $(b_0(t), b_\infty)$ at two times: when $t = j$ and when $t = j + 1$. Consider the figure 4.2. By similar triangles:

$$\frac{h_j}{\frac{j}{\sqrt{2}}} = \frac{j(2d+j)}{d - \frac{j}{\sqrt{2}}}$$
$$h_j = \frac{j(2d+j)}{\sqrt{2}d - j}$$

and

$$\frac{h_{j+1}}{\frac{j+1}{\sqrt{2}}} = \frac{(2d+j)(j+1)}{d - \frac{j+1}{\sqrt{2}}}$$
$$h_{j+1} = \frac{(2d+j)(j+1)}{\sqrt{2}d - (j+1)}$$

The difference between these two values is the velocity of $a_j(t)$, which is:

$$v_j = \frac{\sqrt{2}d(2d+j)}{(\sqrt{2}d - (j+1))(\sqrt{2}d - j)}$$

Now, we solve for the initial position of each point $a_j(0)$, knowing that we must reach y-value h_j after j intervals of time. Therefore:

$$a_j(j) \equiv h_j = a_j(0) + jv_j$$

$$a_j(0) = h_j - jv_j$$

$$a_j(0) = -\frac{(j+1)j(2d+j)}{(\sqrt{2}d - j)(\sqrt{2}d - (j+1))}$$

Thus, the motion of each point $a_j(t)$ is defined by:

$$a_j(t) = \frac{\sqrt{2}d(2d+j)}{(\sqrt{2}d - (j+1))(\sqrt{2}d - j)}t - \frac{(2d+j)j(j+1)}{(\sqrt{2}d - (j+1))(\sqrt{2}d - j)}$$

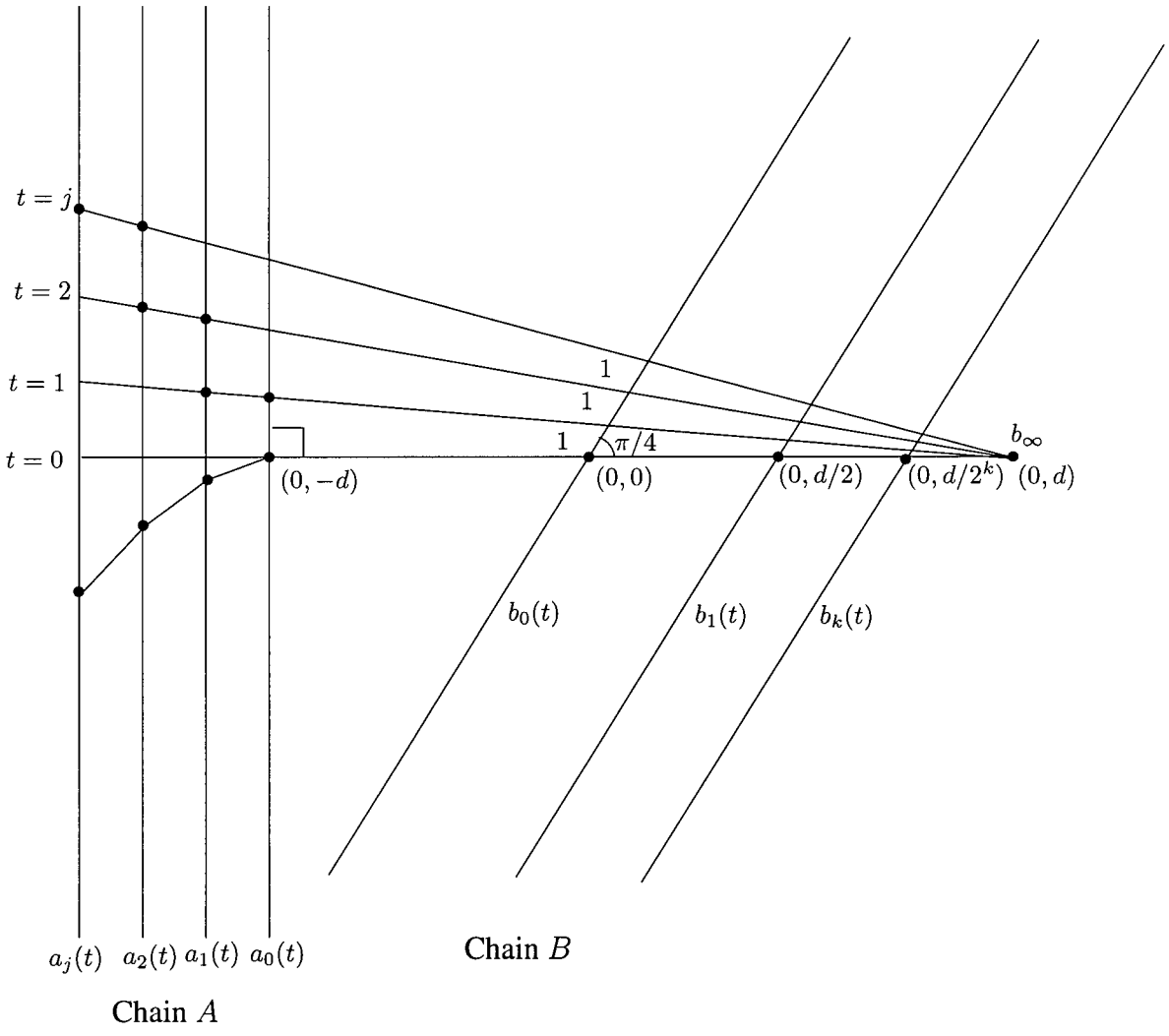


Figure 4.1: An example showing $\Omega(n^2)$ changes to the convex hull of one of our contours.

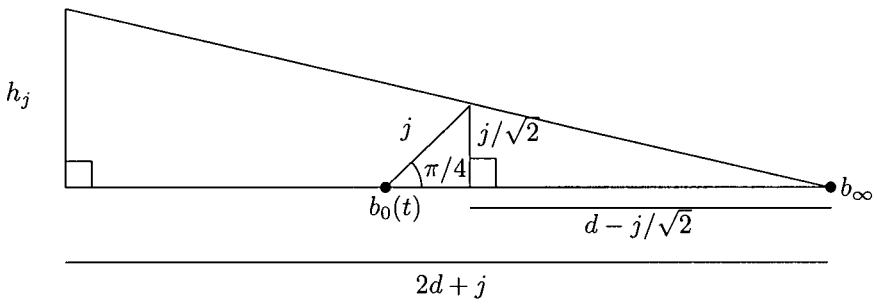


Figure 4.2: Geometry used to define the velocity of the points on chain A

We now prove that A is convex throughout the motion for all t with all vertices of A present on the chain. we do this by showing that the difference between adjacent slopes is always positive on chain A for sufficiently large d or:

$$a_{j+1}(t) - a_{j+2}(t) - (a_j(t) - a_{j+1}(t)) > 0$$

This expression simplifies to:

$$\frac{(8d^2 + (16 + 12\sqrt{2} + 12j + 8\sqrt{2}j - 8t - 12\sqrt{2}t)d - 2\sqrt{2}j(t+1))d}{(\sqrt{2}d - j)(\sqrt{2}d - (j+1))(\sqrt{2}d - (j+2))(\sqrt{2}d - (j+3))}$$

Since $j \leq n$ and $t \leq n$, this is positive for $d \geq 4n$. Since every vertex j on chain A obeys this relation, every vertex is also on chain A for sufficiently large d.

Thus, $a_0(t), a_1(t)...a_n(t)$ forms a convex chain for any $0 < t < n$ for sufficiently large d. We now need to show that each $a_j(t)$ is above $(b_0(t), b_\infty)$ in the interval $(j, j+1)$.

Let $t = j + \alpha(\alpha \in (0,1))$. The intersection of $(b_0(t), b_\infty)$ with $x = -d - j$ occurs at $y = \frac{(2d+j)(j+\alpha)}{\sqrt{2}d-(j+\alpha)}$.

We want to show:

$$a_j(j+\alpha) = \frac{(2d+j)(\sqrt{2}d(j+\alpha) - j(j+1))}{(\sqrt{2}d - j)(\sqrt{2}d - (j+1))} > \frac{(2d+j)(j+\alpha)}{\sqrt{2}d - (j+\alpha)} = y$$

If we set the denominators equal, we compare the numerators:

$$(\sqrt{2}d(j+\alpha) - j(j+1))(\sqrt{2}d - (j+\alpha)) > (j+\alpha)(\sqrt{2}d - j)(\sqrt{2}d - (j+1))$$

which simplifies to:

$$-(j+\alpha)^2 - j(j+1) > -(j+\alpha)(j+1) - (j+\alpha)j$$

$$\alpha^2 < \alpha$$

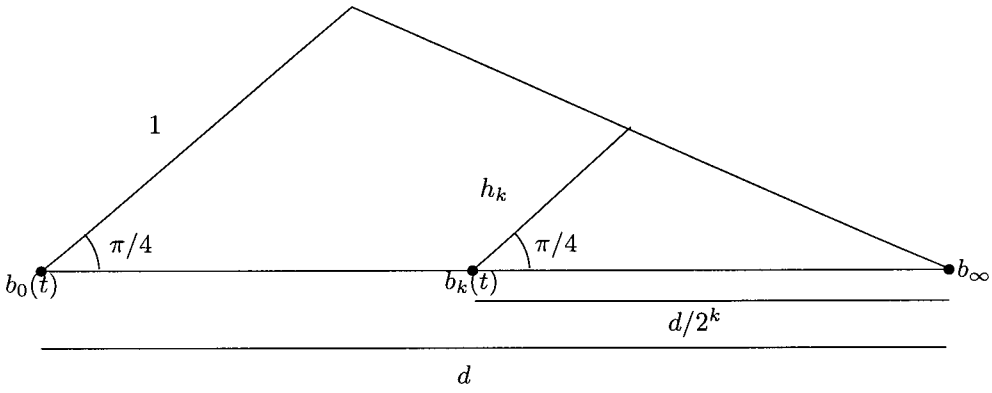


Figure 4.3: Geometry used to define the velocity of points on chain B.

This is true for $0 < \alpha < 1$ and therefore the slice vertex $a_j(t)$ is above $(b_0(t), b_\infty)$ during the interval $(j, j+1)$.

For the chain B, we use similar triangles again to derive the velocities of all the $b_k(t)$ points. Consider figure 4.3.

$$\frac{x}{1} = \frac{\frac{d}{2^k}}{d}$$
$$x = \frac{1}{2^k}$$

We therefore want $b_k(t)$ to cover a distance of $\frac{1}{2^k}$ for each unit time interval.

However, we want this chain to be convex. Therefore, each point will start at $k\delta$ below the x-axis along its path. As each point moves a fixed distance $\frac{1}{2^k}$ per unit time interval, the point will always be $k\delta$ units below $(b_0(t), b_\infty(t))$ for any t. Thus, the points $b_0(t), b_1(t)...b_n(t)$ form a convex chain.

If a sufficiently small δ is chosen, then during the interval $t \in [j, j+1]$ the bitangent between A and B will be $(a_j(t), b_k(t))$ for all $k = 0...n, n...0$ (each $b_k(t)$ appears twice). Therefore there are $\Omega(n^2)$ changes to the convex hull. □

Now, we will show that this simple polygonal chain can be turned into a terrain.

Lemma 4.4. *There exists a terrain for which there are $\Omega(n^2)$ changes to the convex hull of a single contour.*

Proof. We do this by showing that a terrain satisfies lemma 4.3, using $6n$ terrain vertices and thus $O(n)$ edges, and experiences $\Omega(n^2)$ changes to the convex hull of its contour. Top-down views of the terrains whose slices form the chains A and B are shown in figures 4.4 and 4.5 respectively. All vertices are labelled with their height. Sufficiently large h and p_h can be chosen so that all heights are positive. The height of x is chosen such that $h - n > x$. All edges that have a vertex at height x as an endpoint are concave edges and therefore cannot lie on the convex hull. For chain A, there are n parallel edges of the terrain forming the n vertices that remain on the convex chain A in lemma 4.1. For $j = 1$ to n, each edge j has length:

$$\frac{\sqrt{2}d(2d+j)n}{(\sqrt{2}-(j+1))(\sqrt{2}d-j)}$$

For chain B, there are n parallel edges, each at an angle of $\pi/4$, forming the n vertices that lie on the convex chain B in lemma 4.1. For $k = 1$ to n, each edge k has length:

$$\frac{n}{2^k}$$

We now join chains A and B as depicted in figure 4.6. The lines which have endpoints at height p_h and x are concave and thus cannot appear on the convex hull. All other edges involving x appear on the convex hull after the height $h - n$ has been passed and therefore do not effect the $\Omega(n^2)$ changes to the convex hull that take place.

From this construction we can apply lemma 4.3 and conclude there are $\Omega(n^2)$ changes to the convex hull.

How many vertices were used in this construction? The construction of chains A and B require $2n + 2n$ terrain vertices for the $2n$ edges. There are $2(n-1)+1$ vertices at a height value of x (one between each pair of adjacent parallel edges and one used in the final construction), and two vertices at height p_h. There are $6n$ terrain vertices used in this construction, thus $O(n)$ terrain edges.

Thus, there exists a terrain of $O(n)$ edges with a convex hull of an contour that changes $\Omega(n^2)$ times. □

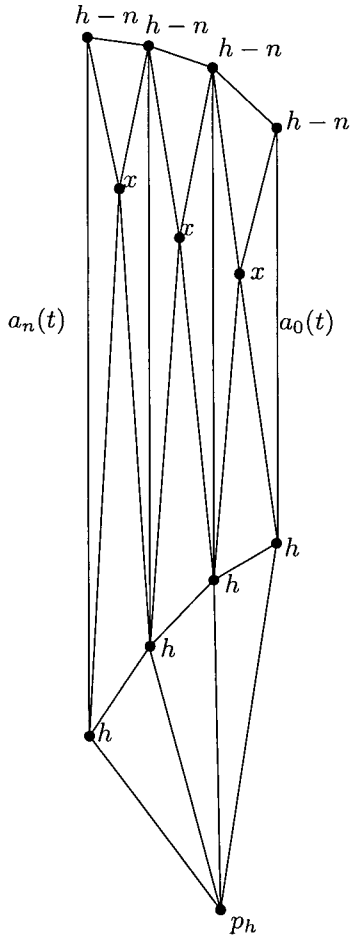


Figure 4.4: A terrain that represents chain A.

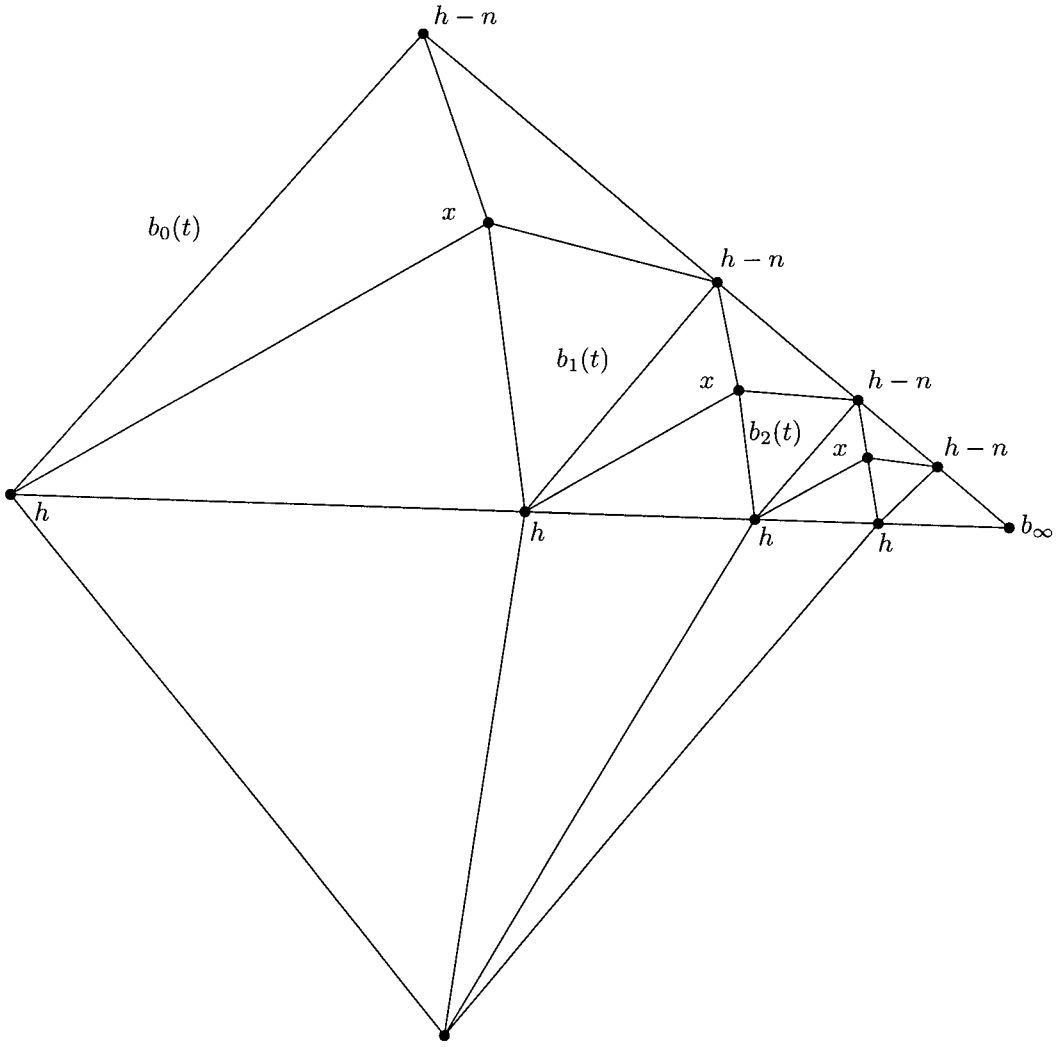


Figure 4.5: A terrain that represents chain B.

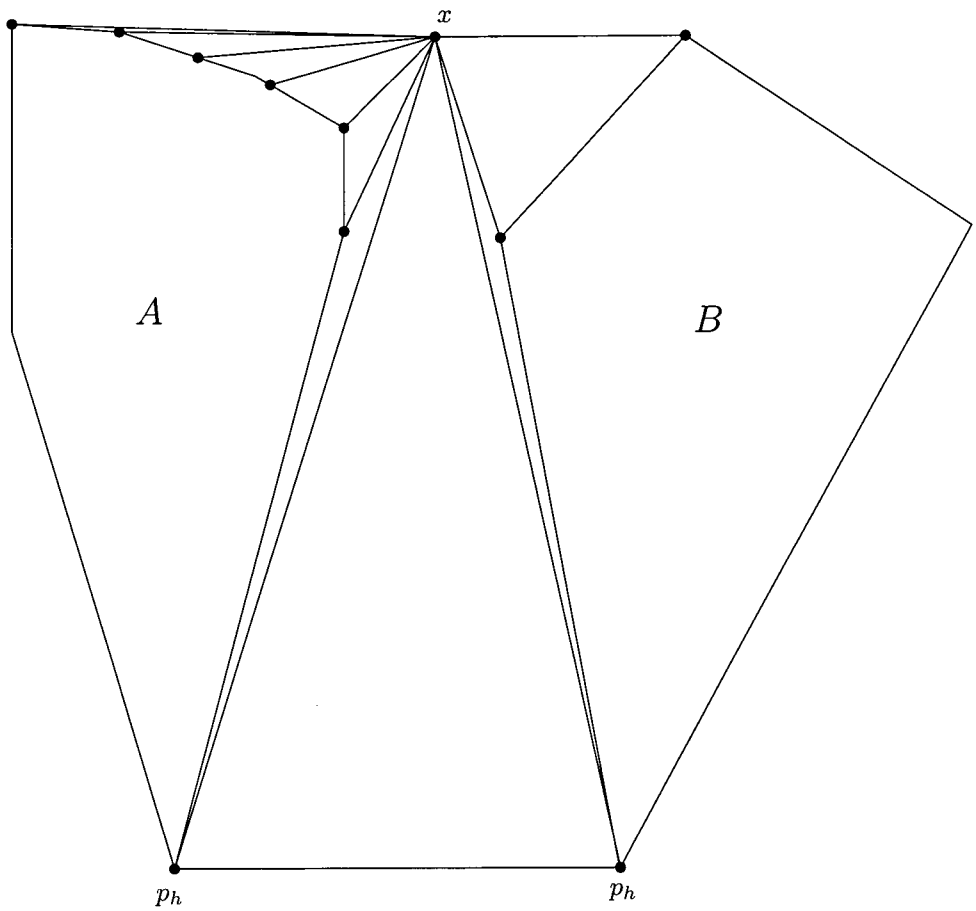


Figure 4.6: The overall terrain with $\Omega(n^2)$ changes to the convex hull of a contour.

Now that we have bounded the number of combinatorial changes to the convex hull of a single contour, we can bound the number of combinatorial changes to the convex hull of a contour when merges are permitted through the sweep.

Lemma 4.5. *The number of changes to the convex hull of a contour throughout the sweep is $O(kn^{2+\epsilon})$ for any $\epsilon > 0$ if contour lines merge.*

Proof. The number of combinatorially different convex hulls for a single contour is $O(n^{2+\epsilon})$ (lemma 4.2). There are only k outer contours that can enter the sweep plane and each of them can only merge once. Thus, there are only $O(kn^{2+\epsilon})$ changes to the convex hulls of the outer contours through the entire sweep. □

4.1.2 Number of Active Tritangents

The set of active bitangents changes when an active bitangent and an inactive bitangent become collinear or when two active bitangents become collinear. This is equivalent to situations where there exists an active tritangent: a horizontal line that touches three edges of the terrain and properly intersects no other part of it. In this section, we obtain tight bounds on the number of active tritangents that can exist on a terrain and therefore the number of times an active bitangent can become collinear with another bitangent.

We prove the following tight bound for the number of active tritangents above a terrain.

Lemma 4.6. *There are at most $O(n^{2+\epsilon})$ for any $\epsilon > 0$ horizontal active tritangents for a terrain of $O(n)$ edges. There exists a terrain with $\Omega(n^2)$ horizontal active tritangents.*

Lemma 4.7. *There are at most $O(n^{2+\epsilon})$ for any $\epsilon > 0$ horizontal active tritangents for a terrain of $O(n)$ edges.*

Proof. We reduce the problem of bounding the number of active tritangents to the computation of the number of vertices on a lower envelope in $\Re^3$. We do this in a similar manner as was done in [26]: transform the set of horizontal lines that lie above the terrain into a

series of bivariate functions in $\Re^3$ and show that the vertices of the lower envelope in this space correspond to active tritangents to the terrain. Since we know that the number of vertices on the lower envelope is bounded by $O(n^{2+\epsilon})$ for any $\epsilon > 0$ [45], the number of active tritangents is bounded by this amount as well.

Consider the terrain from a point of view on the z-axis that is above the maximum height of the terrain. In this setting, the terrain is a rectangle below us. Consider the set of all horizontal lines that intersect the terrain. We characterize them by slope m and y-intercept b. For a fixed (m, b), if we let z vary we form a vertical plane that cuts a number of terrain edges. We draw horizontal lines in this vertical plane tangent to each edge and notice that only the maximum horizontal line could possibly be active (figure 4.7).

Our dual transform is as follows: for every horizontal line, we create a point in $(m, b, -h)$ space where h is the height of the intersection point between the edge and the vertical plane. For a fixed (m, b) in this space, we have a set of height values for each edge cut by the vertical plane. The crucial observation here is that the maximum height value for a particular (m, b) point has the minimum $-h$ value and thus is on the lower envelope in $(m, b, -h)$ space. We notice that if we move m slightly, the function varies as a function of $tan(\theta)$, where θ is the angle between the horizontal line and the x-axis. When we move in the b direction slightly, the function is linear. Thus, the set of all horizontal lines that are tangent to an edge is a bivariate function of bounded degree in $(m, b, -h)$ space. The horizontal line is active if and only if it has the minimum value for all functions defined at (m, b), or lies on the lower envelope. When two functions lie on the lower envelope simultaneously, we have an edge on the lower envelope; the vertices on this edge consist of active bitangents: a horizontal line that is tangent to two edges of the terrain simultaneously and intersects no other part of the terrain. When three functions meet at a vertex on the lower envelope we have an active tritangent: a horizontal line that is tangent to three edges of the terrain simultaneously and intersects no other part of it. The number of tritangents is therefore bound by the number of vertices on the lower envelope in $(m, b, -h)$ space, which we know is bounded by $O(n^{2+\epsilon})$ for any $\epsilon > 0$ [45]. □

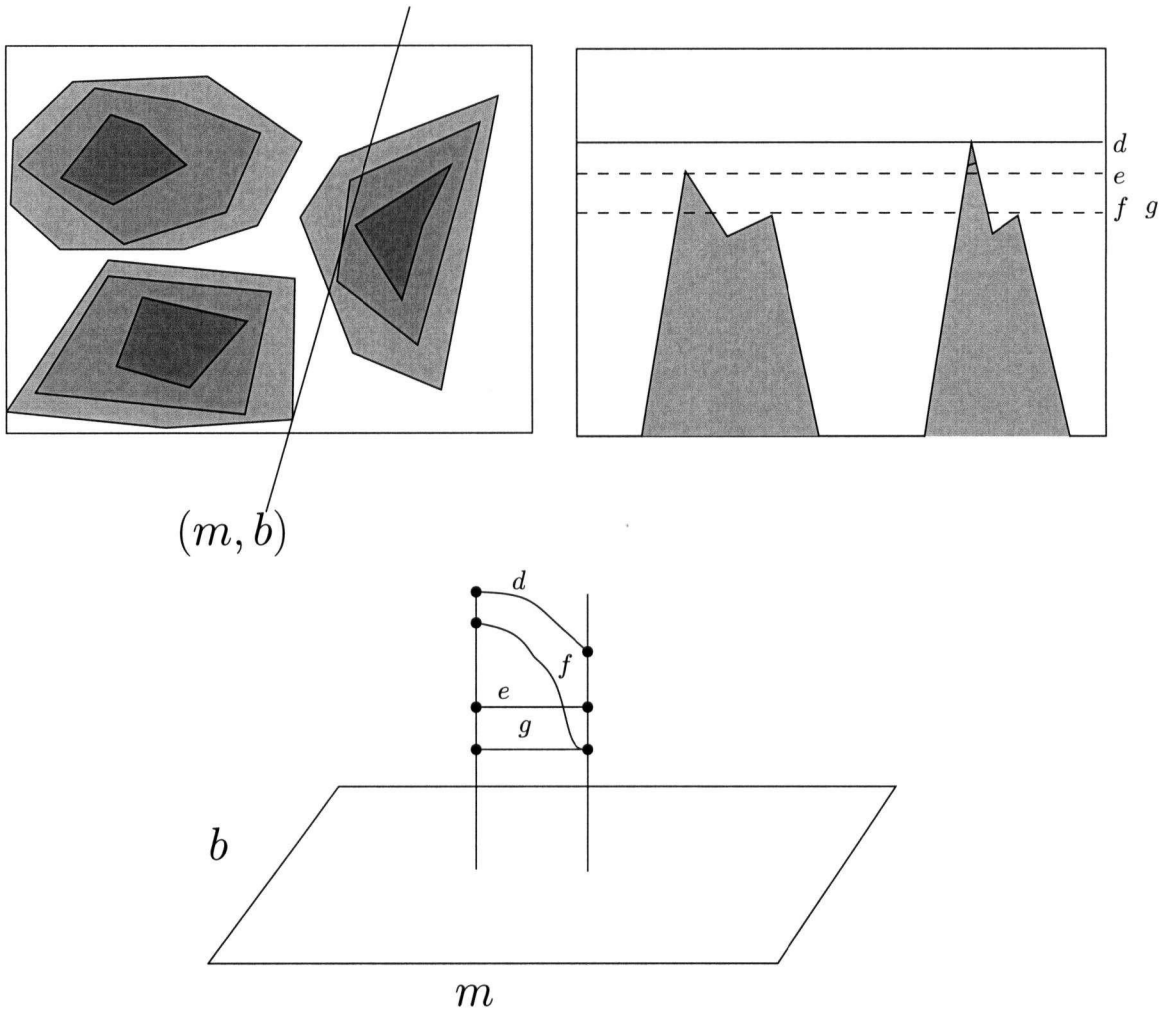


Figure 4.7: The view from an (m, b) slice of the terrain.

We will now show that there exists such a terrain.

Lemma 4.8. *There exists a terrain that has $\Omega(n^2)$ horizontal active tritangents.*

Proof. In lemma 4.4, we considered two chains A and B and demonstrated how they could be linked together to form a single contour line that changes its convex hull $\Omega(n^2)$ times. Now, let us consider chain A and chain B as two separate contours. The former bridge is now an outer bitangent which is supported by $\Omega(n^2)$ different pairs of vertices. □

4.1.3 Number of Inactive Bitangents/Hull Bitangent Certificate Failures

We now obtain a tight bound on the number of times an inactive bitangent can be added or deleted from the set of all bitangents. We prove the following lemma.

Lemma 4.9. *The number of additions and deletions of inactive scene bitangents to the set of all bitangents is $O(n^{2+2\epsilon})$ for any $\epsilon > 0$. There exists a terrain for which there are $\Omega(n^2)$ additions and deletions of inactive bitangents from the set of all bitangents.*

Lemma 4.10. *The number of additions and deletions of inactive bitangents to the set of all bitangents is $O(n^{2+2\epsilon})$ for any $\epsilon > 0$.*

Proof. For any pair of the k contours, there are four bitangents and all of them could be inactive. In this proof we will deal with the inner bitangents and the outer bitangents separately.

Consider the set of lines dual to the vertices of a red polygon with n_i vertices and a blue polygon with n_j vertices. Scene bitangents can only arise from the intersection of a red line and a blue line. These points move with rational quadratic degree. Suppose we align a linear separator between the convex hull of the contours with the y-axis then inner bitangents are lines of maximum/minimum slope between two convex polygons and outer bitangents are lines of maximum/minimum y-intercept. Thus, to know how many times the inner/outer bitangent changes, we need to know how many times a red/blue point realizes the maximum slope or y-intercept of the set of all $n_i n_j$ red/blue points. Since the maximum

of n partially defined univariate functions of bounded degree s changes at most $O(\lambda_{s+2}(n))$ times [46], the maximum and minimum change at most:

$$\sum_{i=1}^{k}\sum_{j=1,j\neq i}^{k} O(\lambda_4(n_i n_j)) = \sum_{i=1}^{k}\sum_{j=1,j\neq i}^{k} O((n_i n_j)^{1+\epsilon}) \leq \sum_{i=1}^{k}\sum_{j=1,j\neq i}^{k} O(n_i^{1+\epsilon} n_j^{1+\epsilon})$$
$$\leq O(n^{2+2\epsilon})$$

□

Lemma 4.11. *There exists a terrain for which there are $\Omega(n^2)$ additions and deletions of inactive bitangents from the set of all bitangents.*

Proof. We simply add a large "mountain" of constant complexity to figure 4.4 that blocks all $\Omega(n^2)$ bitangents. □

Finally, a certificate may need to be updated when a hull bitangent changes due to the change in the convex hull of a contour. This hull bitangent may be involved in many bitangent certificates and all of the failure times of those bitangent certificates need to be updated.

Lemma 4.12. *The number of times certificates need to be updated when a hull bitangent changes due to a change in the convex hull of a contour is $O(kn^{2+\epsilon})$ for any $\epsilon > 0$. There exists a terrain with $\Omega(kn^2)$ such updates.*

Lemma 4.13. *The number of times certificates need to be updated when a hull bitangent changes due to a change in the convex hull of a contour is $O(kn^{2+\epsilon})$ for any $\epsilon > 0$.*

Proof. Once again, consider a pair of convex hulls P and Q with n_p and n_q vertices respectively. The set of certificates to a hull bitangent of either P or Q needs to be updated when either the convex hull of P changes or the convex hull of Q changes. Therefore, there can be at most $O(n_p^{2+\epsilon} + n_q^{2+\epsilon})$ changes for any $\epsilon > 0$.

Thus, the number of times certificates need to be updated when a hull bitangent changes due to a change in the convex hull of a contour is:

$$\sum_{p=1}^{k} \sum_{q=1, q \neq p}^{k} O(n_p^{2+\epsilon} + n_q^{2+\epsilon}) \leq O(kn^{2+\epsilon} + kn^{2+\epsilon}) \leq O(kn^{2+\epsilon})$$

□

We now prove the lower bound.

Lemma 4.14. *There exists a terrain with $\Omega(kn^2)$ such updates.*

Proof. Consider the example in lemma 4.4 once more. This time, we place n vertices in chain A and $n-(k-1)$ vertices in chain B. Under this construction, one of the bridges between chain A and B changes $n(n-(k-1))$ or $\Omega(n^2)$ times. The last k vertices of chain B, will be $k-1$ contours of constant complexity. Each of these $k-1$ contours will have one slice vertex that follows the same path as if it were one of the $k-1$ remaining vertices of chain B. Each of these $k-1$ bitangents is inactive, except for the last one, and thus, they have certificates to the to the bridge that changes $\Omega(n^2)$ times, except for the outermost one. The construction of chain B is depicted in figure 4.8.

For $l = 1...(k-l)$, each vertex will be offset a distance of $l\delta_2$ above the line $(b_0(t), b_\infty(t))$. A value of δ_2 can be chosen small enough so that each of those $(k-1)$ vertices will support a bitangent, for each $j = 0...n$ vertices of chain A during the interval $[j, j+1]$ as they each stay $l\delta_2$ above the line $(b_0(t), b_\infty(t))$. While supported by $a_j(t)$, $\Omega(n^2)$ changes happen to the hull bitangent just beneath. Thus, $\Omega(k)$ certificates need to be updated each time.

Through the entire sweep, we therefore incur a cost of $\Omega(kn^2)$ certificate updates due to changes in hull bitangent. □

4.2 Algorithm Complexity Analysis

In this section, we will prove the runtime of our algorithm. Basically, we prove the following theorem.

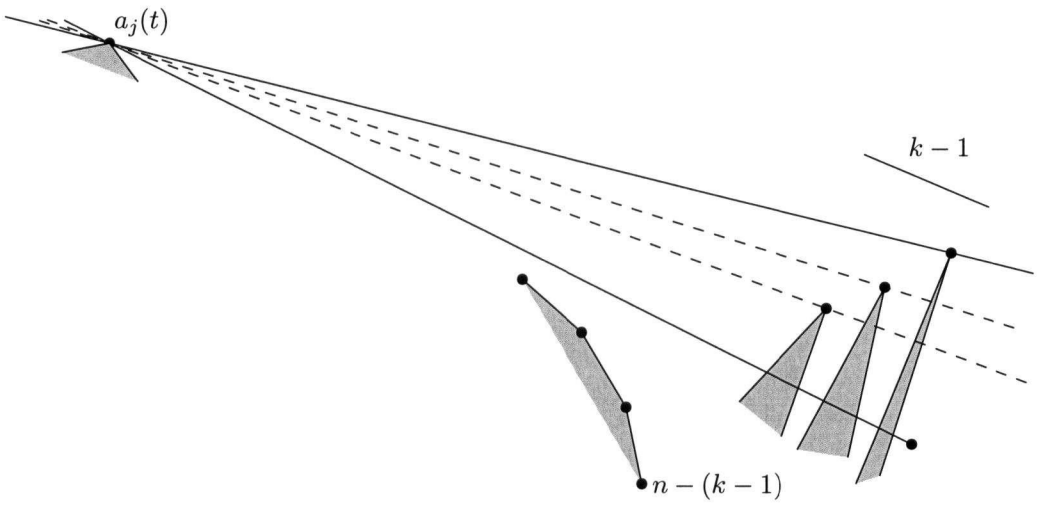


Figure 4.8: The example showing $\Omega(kn^2)$ changes in hull bitangent.

Theorem 4.15. *The algorithm as described takes* $O(kn^{2+\epsilon} \lg n \lg(n \lg n + k^2))$ *for any* $\epsilon > 0$ *time in worst case.*

4.2.1 Event Queue Size

Before we look at the number of times that certificates can fail and the time required to recover from certificate failures, we need to have bounds on the size of our event queue.

Lemma 4.16. *The size of the event queue is bounded by* $O(n \lg n + k^2)$ *at any time during the sweep.*

Proof. There are at most $O(n \lg n)$ certificates in the event queue due to Basch's hull maintenance structure [8] as any of the n convex hull vertices may appear in with a constant number of certificates in any of $O(\lg n)$ nodes of the balanced binary tree.

There are at most $O(n)$ peak and non-peak events in the event queue at any time as there is at most $O(n)$ terrain vertices.

Finally, there are at most $O(k^2)$ bitangent certificates in the event queue. An inactive bitangent has at most two certificates: one to the adjacent hull bitangent or active bitangent to either side. Therefore, on a single contour there are at most $O(k)$ certificates

involving inactive bitangents at any time. An active bitangent has several certificates to inactive bitangents, which we've already counted, and one to its adjacent active bitangent or hull bitangent to either side. Since there are at most $O(k)$ contours in the sweep plane at any time, there are at most $O(k^2)$ bitangent certificates in the event queue at any time. □

4.2.2 Peak and Non-Peak Events

We start by analyzing the amount of time required to process all the peak and non-peak vertices in the sweep plane.

Lemma 4.17. *The time to introduce all peak vertices over the entire sweep is* $O(k^2 \lg(n + k))$.

Proof. For each contour in the scene, we perform binary search on at most $O(n + k)$ bitangents. We then sort those bitangents around the peak vertex and distribute them to the slice vertices just below the peak. The convex hulls of all peak vertices just below the peak can be determined in $O(n \lg n)$ time as each terrain edge participates in at most one peak vertex. We have shown this is equivalent to merging the upper and lower hull of new slice vertices just below the peak ($O(n)$) with the sorted list of bitangents to the peak vertex ($O(k)$).

Thus, to compute the set of bitangent certificates takes $O(k(k \lg(n + k) + k \lg k + n + k))$ or $O(k^2 \lg(n + k))$ time.

To insert the new $O(k)$ certificate that arise from new scene bitangents into the the event queue then it requires $O(k \lg(n \lg n + k^2))$ time. □

Lemma 4.18. *The time to introduce all non-peak vertices over the entire sweep is* $O(n \lg n + nk)$.

Proof. When we merge the convex hulls and discard points and tangents interior to them, it takes at most $O(n + k)$ time as we may need to update bitangent certificate with bridges. Since convex hulls cannot separate when they merge, there can be only $O(k)$ merges. Thus for the entire sweep this takes $O(nk)$ time.

Each edge in the terrain is added at one non-peak vertex. Thus, all convex hull calculations take at most $n \lg n$ time for the entire sweep. At the merge point, at most two slice vertices need to be considered and therefore there are at most $O(k)$ bitangents. We do this merge $n - k$ times with at most $O(n)$ distinct edges. Therefore the total complexity is $O(nk + n \lg n + n + nk)$ or $O(n \lg n + nk)$ time. □

We now consider the total amount of time required to rebuild the kinetic maintenance trees for convex hulls and the search trees at merges.

Lemma 4.19. *The total cost for rebuilding the convex hull maintenance tree for all merges is* $O(kn \lg n \lg(n \lg n + k^2))$.

Proof. We have shown that to reconstruct the kinetic convex hull structure used in Julien Basch's thesis [8], requires $O(n \lg n)$ time. We reconstruct the trees at $O(k)$ merges and therefore $O(kn \lg n)$ time is required. The last logarithmic factor is due to the time required to update the event queue. □

Lemma 4.20. *The total cost for all insertions/deletions into the convex hull tree is* $O(n^2 \lg(n \lg n + k^2))$.

Proof. There are $O(n)$ insertions that effect one path in the tree. Each dual edge is in that tree is involved in at most $O(\lg n)$ re-computations. Deletions is the same since it involves removing a dual line and recomputing the contents of the node. Thus we can multiply the sum by two.

Thus, the amount of time spent on insertions and deletions is computed by the recurrence relation $T(n) = T(n/2) + O(n)$ which is $O(n)$.

Thus, we need to insert and remove $O(n)$ certificates from the event queue $n - k$ times. Thus the total time required for inserts and deletes of slice vertices is bounded by $O(n^2 \lg(n \lg n + k^2))$. □

Lemma 4.21. *The total cost for rebuilding the search trees at the merge of two contours is* $O(k(n + k) \lg(n + k))$ *time.*

Proof. There are only k merges in the sweep plane as two contour lines only merge once. Each merge requires $O((n+k)\lg(n+k)$ time to rebuild the search tree as the convex hull of a contour has at most n hull bitangents and at most k scene bitangents. □

4.2.3 Maintenance of Certificate Structures

In this section, we now bound the number of certificate failures in worst case.

Lemma 4.22. *At most $O(kn^{2+\epsilon}\lg(n\lg n + k^2))$ for any $\epsilon > 0$ time is spent maintaining the bitangent certificate structure.*

Proof. A certificate failure involving an active bitangent takes at most $O(k)$ time as there are at most $O(k)$ bitangents to a convex hull that may share a certificate with this active bitangent. Since there are at most $O(n^{2+\epsilon})$ active tritangent events (lemma 4.6), there can be at most the same number of bitangent certificate failures involving an active bitangent. For each failure, we must update at most $O(k)$ certificates on an event queue of size $O(n\lg n + k^2)$ by lemma 4.16. Thus, we require $O(kn^{2+\epsilon}\lg(n\lg n + k^2))$ time for failures of a bitangent certificate involving an active bitangent.

By lemma 4.9, we have shown that the number of times an inactive bitangent becomes collinear with a convex hull edge is $O(n^{2+\epsilon})$. Thus, the amount of time required for failures involving an inactive bitangent and a hull bitangent is $O(n^{2+\epsilon}\lg(n\lg n + k^2))$.

By lemma 4.13, we have shown the number of times certificates need to be updated when a hull bitangent changes due to a change in the convex hull of a contour is $O(kn^{2+\epsilon})$. Thus, the amount of time required for these certificate updates is at most $O(kn^{2+\epsilon}\lg(n\lg n + k^2))$. □

Lemma 4.23. *At most $O(kn^{2+\epsilon}\lg(n+k))$ time is spent kinetically maintaining the search trees.*

Proof. By lemma 4.6, there are at most $O(n^{2+\epsilon})$ for any $\epsilon > 0$ failures between an active bitangent and another bitangent. The worst case time to recover from a failure is an insertion or a deletion from the tree which takes $O(n+k)$ time as there are at most $O(n)$ convex

hull edges and $O(k)$ scene bitangents in the search tree. Therefore, these certificate failures require at most $O(n^{2+\epsilon} \lg(n+k))$ time.

By lemma 4.5, there are at most $O(kn^{2+\epsilon})$ for any $\epsilon > 0$ new hull bitangents encountered during the sweep. Each requires an insert into the structure that takes $O(\lg(n+k))$ time. Thus, at most $O(kn^{2+\epsilon} \lg(n+k))$ time is required due to changes in the set of hull bitangents. □

Lemma 4.24. *At most $O(kn^{2+\epsilon} \lg n \lg(n \lg n + k^2))$ time is spent maintaining the convex hulls throughout the sweep.*

Proof. We only reconstruct the tree at k merges. At any other time, at each internal node we have a tree the same as in [8] with a set of bivariate functions that are partially defined. From [45], we know the complexity of the lower envelope is $O(n^{2+\epsilon})$ and the number of certificate failures in [8] is proportional to the combinatorial complexity of the lower envelope. Since the trees are reconstructed k times, there are at most $O(kn^{2+\epsilon})$ certificate failures and a certificate failure could require $O(\lg n)$ certificates to be updated. □

4.2.4 Overall Worst Case Complexity

We now can bound the worst case running time stated in theorem 4.15.

Proof. The total time taken by the algorithm is defined by the counting arguments present in lemmas 4.17, 4.18, 4.21, 4.19, 4.20, 4.22, 4.23 and 4.24. The term that dominates is $O(kn^{2+\epsilon} \lg n \lg(n \lg n + k^2))$ for any $\epsilon > 0$ from lemma 4.24. □

Chapter 5

Conclusions and Future Work

We have presented and fully analyzed an algorithm that computes the set of terrain edges on the horizon from any distant point of view. We have analyzed the problem of determining if an edge lies on some distant horizon and we have determined that this question if proportional to the number of topologically distinct horizons from any distant point of view. The algorithm, thus, has a $O(k \lg^2 n)$ loss in efficiency compared to the near tight bounds on the number of active tritangents.

Part of this loss of efficiency is due to the fact that we are required to update as many as $O(k)$ certificates when there is a bitangent certificate failure involving an active bitangent. Much of this loss of efficiency could be alleviated by the use of a *kinetic tournament tree* for maximum maintenance. It is a simple balanced binary tree where each element is a leaf and each internal node takes the largest value of its children. The tree has a worst case efficiency of $O(\lambda_{\delta+2}(m) \lg^2 n)$ where δ is the degree of the motion, n is the maximum number of elements at any time t for which we would like to maintain the maximum, and m is the number of times we are allowed to change the set of elements. One of the factors of $\lg n$ is due to the cost to update an event queue of size $O(n)$. As we are maintaining the set of $O(k^2)$ bitangents and we change that set at most $O(n^{2+\epsilon})$ times through bitangent certificate failures one might hope to achieve a worst case running time of $O(\lambda_{\delta+2}(n^{2+\epsilon}) \lg k \lg(n \lg n + k^2))$ time or $O(n^{2+\epsilon_2} \lg k \lg(n \lg n + k^2)$ for any $\epsilon_2 > 0$

with $\epsilon_2 > \epsilon$. This would lead to only a $O(\lg k \lg n)$ loss in efficiency in worst case. It is, however, at this time unclear how to merge and split the tournament trees when two active bitangents become inactive or two inactive bitangents become active.

More efficiency is lost due to our convex hull maintenance structure. To ensure the $O(n^{2+\epsilon})$ for any $\epsilon > 0$ certificate failures, we must ensure that the lower envelopes at each node contain the same bivariate functions. Merges currently cause the convex hull maintenance trees to be reconstructed, incurring the extra factor of $O(k)$. It is not known if an example can be constructed to incur this extra factor, so a tighter analysis may lead to a better running time. Also, the current method could be further improved by maintaining search trees for each of the convex hulls at the internal nodes of the tree.

The primary contribution of this thesis was the development of an algorithm that produces the set of edges that lie on some distant horizon. We have made this algorithm sensitive to the number of times the set of active bitangents can change, resulting in an algorithm that is approximately $O(k \lg^2 n)$ from optimal. If we are able to solve the above two problems, the algorithm's worst case runtime could be further improved to $O(\lg^2 n)$ from optimal.

Bibliography

[1] Pankaj K. Agarwal and Micha Sharir *On the Number of Views of Polyhedral Terrains* in *Discrete and Computational Geometry*, 177-182, 1994.

[2] Pankaj K. Agarwal, Otfried Schwarzkopf, and Micha Sharir, *The Overlay of Lower Envelopes and its Applications* in *Discrete Computational Geometry*, Vol. 15 No. 1, 1-13, 1996.

[3] Pankaj K. Agarwal, Leonidas J. Guibas, John Hershberger, and Eric Veach, *Maintaining the Extent of a Moving Point Set* in *The 5th Workshop on Algorithms and Data Structures*, 31-34, 1997.

[4] T. Asano, T. Asano, L. Guibas, J. Hershberger, and H. Imai, *Visibility of Disjoint Polygons* in *Algorithmica*, Vol. 1 No. 1, 49-63, 1986.

[5] M. Atallah, *Dynamic Computational Geometry* in *Proceedings of the 24th IEEE Syposium on the Foundations of Computer Science*, 92-99, 1983.

[6] D. Avis, H. ElGindy, and R. Seidel, *Simple On-Line Algorithms for Convex Polygons* in *Computational Geometry* edited by G. T. Toussaint, North-Holland Publishing Company, Amsterdam, 1985.

[7] C. Bajaj, M. van Kreveld, R. van Oostrum, V. Pascucci, and D. Schikore, *Contour trees and small seed sets for isosurface traversal* in *Proceedings of the 13th Annual ACM Symposium on Computational Geometry*, 212-220, 1997.

[8] Julien Basch *Kinetic Data Structures*, Ph.D. Dissertation, Stanford University, 1999.

[9] F. Benichou and G. Elber *Output Sensitive Extraction of Silhouettes from Polygonal Geometry* in *Proceedings of the 7th Pacific Graphics Conference*, 60-69, 1999.

[10] R. Cole and M. Sharir, *Visiblility Problems for Polyhedral Terrains* in *Journal of Symbolic Computation 17*, 11-30, 1989.

[11] M. de Berg, D. Halperin, M. H. Overmars, and M. van Kreveld *Sparse Arrangements and the Number of Views of Polyhedral Scenes* in *International Journal of Computational Geometry*, Vol. 7 No. 3, 175-195, 1997.

[12] L. de Floriani and P. Magillo *Horizon Computation on a Hierarchical Terrain Model* in *The Visual Computer: An International Journal of Computer Graphics 11*, 134-149, 1995.

[13] L. De Floriani and P. Magillo *Intervisibility on Terrains* in *Geographic Information Systems: Principles, Techniques, Management, and Applications* 543-556, 1999.

[14] Frédo Durand, George Drettakis and Claude Puech, *The Visibility Skeleton: A Powerful And Efficient Multi-Purpose Global Visibility Tool.* in *ACM SIGGRAPH 1997 Conference Proceedings*, 89-100, 1997.

[15] Frédo Durand *3D Visibility: Analytical Study and Applications* Ph.D. Dissertation, Université Joseph Fourier, 1999.

[16] Frédo Durand, George Drettakis and Claude Puech, *The 3D Visibility Complex* in *ACM Transactions on Graphics*, Vol. 21 No. 2, 176-206, 2002.

[17] H. Edelsbrunner and E. P. Mucke *Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms* in *ACM Transactions on Graphics*, Vol. 9, No. 1, 66-104, 1990.

[18] Alon Efrat, Leonidas J. Guibas, Olaf A. Hall-Holt, Li Zhang *On Incremental Rendering of Silhouette Maps of a Polyhedral Scene* in *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, 910-917, 2000.

[19] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter Shirly and Richard Riesenfeld, *Interactive Technical Illustration* in *ACM Symposium on Interactive 3D Graphics*, 31-38, 1999.

[20] Subir Kumar Ghosh and David M. Mount, *An Output-Sensitive Algorithm for Computing Visibility Graphs* in *SIAM Journal of Computing*, Vol. 20 No. 5., 888-910, 1991.

[21] Olaf Hall-Holt *Kinetic Visibility* Ph.D Dissertation, Stanford University, 2002.

[22] Ziv Gigus and Jitendra Malik *Computing the Aspect Graph for Line Drawings of Polyhedral Objects* in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, No. 2, 113-122, 1990.

[23] Ziv Gigus, John Canny, and Raimund Seidel *Efficiently Computing and Representing Aspect Graphs of Polyhedral Objects* in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 13, No. 6, 542-551, 1991.

[24] Olaf Hall-Holt *Kinetic Visible Set Maintenance in the Plane, Submitted for Publication*, as of 2003.

[25] Olaf Hall-Holt and Szymon Rusinkiewicz *Visible Zone Maintenance for Real-Time Occlusion Culling, Submitted for Publication*, as of 2003.

[26] Dan Halperin and Micha Sharir *New Bounds for Lower Envelopes in Three Dimensions, with Applications to Visibility in Terrains* in *Proceedings of the 9th Annual Symposium on Computational Geometry*, 11-18, 1993.

[27] J. Hershelberg *Finding the Upper Envelope of n Line Segments in $O(n \lg n)$ Time.* in *Information Processing Letters 33*, 169-174, 1989.

[28] Aaron Hertzmann and Denis Zorin, *Illustrating Smooth Surfaces* in *ACM SIGGRAPH 2000 Conference Proceedings*, 517-526, 2000.

[29] Samuel Hornus and Claude Puech, *A Simple Kinetic Visibility Polygon* in *The proceedings of the 18th European Workshop on Computational Geometry*, 27-30, 2002.

[30] Aldo Laurentini, *The Visual Hull Concept for Silhouette-Based Image Understanding* in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 16 No. 2, 150-162, 1994.

[31] Aldo Laurentini, *How Far 3D Shapes Can Be Understood from 2D Silhouettes* in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 17 No. 2, 188-195, 1994.

[32] Aldo Laurentini, *The Visual Hull of Curved Objects* in *IEEE 7th International Conference on Computer Vision.* 356-361, 1999.

[33] Aldo Laurentini, *Computing the Visual Hull of Solids of Revolution* in *Pattern Recognition*, Vol. 32 No. 3, 377-388, 1999.

[34] Lee Markosian, Michael A. Kowalski, Samuel J. Trychin, Lubomir D. Bourdev, Daniel Goldstein and John F. Huges, *Real-Time Non-photorealistic Rendering* in *ACM SIGGRAPH 1997 Conference Proceedings*, 415-420, 1997.

[35] Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J. Gortler, and Leonard McMillan, *Image-Based Visual Hulls* in *Proceedings of ACM SIGGRAPH 2000*, 369-374, 2000.

[36] Wojciech Matusik, Chris Buehler, Leonard McMillan, and Steven J. Gortler, *An Efficient Visual Hull Computation Algorithm* MIT LCS Technical Memo 623, 2002.

[37] N. L. Max *Horizon Mapping: Shadows for Bump-Mapped Surfaces* in *The Visual Computer* Vol. 4 No. 2, 1988.

[38] M.H. Overmars and E. Welzl, *New Methods for Computing Visibility Graphs* in *Proceedings of the 4th ACM Symposium on Computational Geometry*, 164-171, 1988.

[39] Michel Pocchiola and Gert Vegter, *The Visibility Complex* in *Proceedings of the 9th Annual Symposium on Computational Geometry*, 328-337, 1993.

[40] M. Pocchiola and G. Vegter, *Topologically Sweeping Visibility Complexes via Pseudotriangualtions* in *Discrete Computational Geometry*, Vol. 16 No. 4, 419-453, 1996.

[41] Mihai Pop, Christian A. Duncan, Gill Barequet, Michael T. Goodrich, Wenjing Huang, and Subodh Kumar, *Efficient Perspective-Accurate Silhouette Computation and Applications* in *Proceedings of the 17th Annual Symposium on Computational Geometry*, 60-68, 2001.

[42] Sylvain Petitjean, *A Computational Geometric Approach to Visual Hulls* in *International Journal of Computational Geometry & Applications.* 407-436, 1997.

[43] Harry Plantinga and Charles R. Dyer, *Visibility, Occlusion, and the Aspect Graph* in *International Journal of Computer Vision* Vol. 5 No. 2, 137-160, 1990.

[44] Pedro V. Sander, Xianfeng Gu, Steven J. Gortler, Hugues Hoppe, John Snyder, *Silhouette Clipping* in *ACM SIGGRAPH 2000 Conference Proceedings*, 327-334, 2000.

[45] M. Sharir, *Almost Tight Upper Bounds for Lower Envelopes in Higher Dimensions* in *Discrete Computational Geometry*, Vol. 12 No. 1, 327-345, 1994.

[46] M. Sharir and P. K. Agarwal, *Davenport-Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, New York, 1995.

[47] A. J. Stewart, *Fast Horizon Computation at All Points of a Terrain with Visibility and Shading Applications* in *IEEE Transactions on Visualization and Computer Graphics* Vol. 4 No. 1, 1998.