Universitat de Girona

VISIBILITY AND PROXIMITY ON TRIANGULATED SURFACES

Marta FORT MASDEVALL

ISBN: 978-84-691-5959-0
Dipòsit legal: GI-1196-2008

VISIBILITY AND PROXIMITY ON TRIANGULATED SURFACES

PhD. Dissertation

by

Marta Fort

Advisor

Dr. Joan Antoni Sellarès

Programa de Doctorat de Software
Universitat de Girona

2008

Abstract

In this thesis, we solve visibility and proximity problems on triangulated surfaces concerning generalized elements. As generalized elements, we consider: points, segments, polygonal chains and polygonal regions. The proposed strategies use algorithms of Computational Geometry and Graphics Hardware.

We start by studying multi-visibility problems on triangulated terrain models concerning a set of generalized view elements. We present two methods to obtain approximate multi-visibility maps. A multi-visibility map is a subdivision of the terrain domain encoding visibility according to different criteria. The first method, of complex implementation, uses exactly computed visibility information to approximately reconstruct the unknown multi-visibility map. The second, from which implementation results are provided, uses approximate visibility information to compute and visualize discrete multi-visibility maps by exploiting graphics hardware capabilities. As applications, we compute multi-visibility maps, solve inter-region multi-visibility problems and approximately answer point and polygonal region multi-visibility queries.

Next, we tackle proximity problems on triangulated polyhedral surfaces, where generalized obstacles are allowed, considering generalized sources. We present two methods, with implementation results, to compute distances on polyhedral surfaces from a generalized source. The first method computes exact shortest path distances from generalized sources. The second provides approximate weighted shortest path distances from generalized sites on weighted polyhedral surfaces. Both methods are posteriorly extended to handle the multiple-site problem where the corresponding distance field is obtained. As applications, we compute discrete order-k Voronoi diagrams and approximately solve some facility location problems. We also provide a theoretical study on the order-k Voronoi diagram complexity of a set of generalized sources for the non-weighted case.

Resum

En aquesta tesi es solucionen problemes de visibilitat i proximitat sobre superfícies triangulades considerant elements generalitzats. Com a elements generalitzats considerem: punts, segments, poligonals i polígons. Les estratègies que proposem utilitzen algoritmes de geometria computacional i hardware gràfic.

Comencem tractant els problemes de visibilitat sobre models de terrenys triangulats considerant un conjunt d'elements de visió generalitzats. Es presenten dos mètodes per obtenir, de forma aproximada, mapes de multi-visibilitat. Un mapa de multi-visibilitat és la subdivisió del domini del terreny que codifica la visibilitat d'acord amb diferents criteris. El primer mètode, de difícil implementació, utilitza informació de visibilitat exacte per reconstruir de forma aproximada el mapa de multi-visibilitat. El segon, que va acompanyat de resultats d'implementació, obté informació de visibilitat aproximada per calcular i visualitzar mapes de multi-visibilitat discrets mitjançant hardware gràfic. Com a aplicacions es resolen problemes de multi-visibilitat entre regions i es responen preguntes sobre la multi-visibilitat d'un punt o d'una regió.

A continuació tractem els problemes de proximitat sobre superfícies polièdriques triangulades considerant seus generalitzades. Es presenten dos mètodes, amb resultats d'implementació, per calcular distàncies des de seus generalitzades sobre superfícies polièdriques on hi poden haver obstacles generalitzats. El primer mètode calcula, de forma exacte, les distàncies definides pels camins més curts des de les seus als punts del poliedre. El segon mètode calcula, de forma aproximada, distàncies considerant els camins més curts sobre superfícies polièdriques amb pesos. Com a aplicacions, es calculen diagrames de Voronoi d'ordre k, i es resolen, de forma aproximada, alguns problemes de localització de serveis. També es proporciona un estudi teòric sobre la complexitat dels diagrames de Voronoi d'ordre k d'un conjunt de seus generalitzades en un poliedre sense pesos.

Resumen

En esta tesis se solucionan problemas de visibilidad y proximidad en superficies trianguladas considerando elementos generalizados. Como elementos generalizados consideramos: puntos, segmentos, poligonales y polígonos. Las estrategias propuestas utilizan algoritmos de geometría computacional y de harware gráfico.

Se empieza con los problemas de visibilidad sobre terrenos triangulados considerando un conjunto de elementos de visión generalizados. Se presentan dos métodos para obtener de forma aproximada mapas de multi-visibilidad. Un mapa de multi-visibilidad es una subdivisión del dominio del terreno que codifica la visibilidad. El primer método, de difícil implementación, utiliza información de visibilidad exacta para reconstruir aproximadamente el mapa de multi-visibilidad. El segundo, que se presenta con resultados de implementación, obtiene información de visibilidad aproximada para calcular y visualizar mapas de multi-visibilidad discretos utilizando hardware gráfico. Como aplicaciones, se resuelven problemas de multi-visibilidad entre regiones y se responden preguntas sobre la multi-visibilidad en un punto o una región.

A continuación se tratan los problemas de proximidad sobre superficies poliédricas trianguladas considerando sedes generalizadas. Se presentan dos métodos, con resultados de implementación, para calcular distancias desde sedes generalizadas sobre superficies poliédricas dónde pueden haber obstáculos generalizados. El primer método calcula, de forma exacta, distancias definidas por los caminos más cortos desde las sedes hasta los puntos del poliedro. El segundo método da distancias aproximadas considerando los caminos más cortos sobre superficies poliédricas con pesos. Como aplicaciones, se calculan diagramas de Voronoi de orden k, y se resuelven, de forma aproximada, algunos problemas de localización de servicios. En el caso en el que no se consideran pesos, se realiza un estudio teórico sobre la complejidad de los diagramas de Voronoi de orden k de un conjunto de sedes generalizadas.

Published Work

Publications

Visibility:

- N. Coll, M. Fort and J.A. Sellarès, Approximate Multi-Visibility Map Computation, 21th European Workshop on Computational Geometry, pp 97-100, Eindhoven, March 2005.

- N. Coll, M. Fort and J.A. Sellarès, Multi-visibility in terrains, Actas XI Encuentros de Geometría Computacional, Santander, Spain, pp 71–78, Santander, June 2005.

- N. Coll, M. Fort, N. Madern and J.A. Sellarès, Computing Terrain Multi-visibility Maps for a Set of View Segments Using Graphics Hardware,ICCSA 2006, Lecture Notes on Computational Science 3980, pp. 81–90, Springer-Verlag, Glasgow, May 2006.

- N. Coll, M. Fort, N. Madern and J.A. Sellarès, Multi-visibility maps of triangulated terrains, International Journal of Geographical Information Science, Volume 21, Issue 10, pp 1115-1134, November 2007.

Proximity:

- M. Fort and J.A. Sellarès, Generalized Source Shortest Paths on Polyhedral Surfaces, 23th European Workshop on Computational Geometry, pp 186–189, March 2007.

- M. Fort and J.A. Sellarès, Generalized Voronoi Diagrams on Polyhedral Terrains Actas XII Encuentros de Geometría Computacional, Valladolid, Spain, pp 239-246, 2007.

- M. Fort and J.A. Sellarès, Generalized Higher-Order Voronoi Diagrams on Polyhedral Surfaces, The 4th International Symposium on Voronoi Diagrams in Science and Engineering, pp. 74-83, Wales, July 9th-11th, 2007.

- M. Fort and J.A. Sellarès, Computing Distance Functions from Generalized Sources on Weighted Polyhedral Surfaces, (accepted to The 2008 International Conference on Computational Science and Its Applications, Perugia (Italy), June 30th to July 3rd).

- M. Fort and J.A. Sellarès, Generalized Higher-Order Voronoi Diagrams on triangulated Surfaces (submitted to Applied Mathematics and Computation)

- M. Fort and J.A. Sellarès, Generalized Distance Functions on Weighted Triangulated Surfaces with Applications, (submitted to International Journal Computational Geometry and Applications).

Acknowledgements

First of all, I would like to thank my advisor Joan Antoni Sellarès for his ideas, support, advise, dedication, hard work, and for given me every facility during the development of the thesis that I have needed. Also warmest thanks must go to my parents, Josep Fort and Maria Lluïsa Masdevall, my sister Carme Fort, my partner Albert Sánchez and my grandparents, whose patience, support and impeccable understanding allowed me to write this thesis.

Thank also to the Departament d'Informàtica i Matemàtica Aplicada, especially to Xavier Pueyo and Toni Sellarès, wo welcomed me at the beginning of my PhD, as well as to my cousin Maria Fuentes and Narcís Coll who encouraged me to start a PhD on Computational Geometry in GGG.

I am also very grateful to Narcís Coll and Sergio Cabello for their contributions in some parts of the thesis. I also give thanks to Anna Casas, Teresa Paradines and Eduard Oliver for the implementations of the different algorithms proposed throughout: to Yago Díez for revising most of my papers and English documents and Narcís Madern and Pau Estalella for providing me GPU and programming advice.

Finally, my thanks to my friends and colleagues at the Universitat de Girona, with whom I shared lunch times, coffee-breaks and long talks, with special thanks to: Yago Diez, Pau Estalella, Anton Bardera, Narcís Mardern, Ferran Prados, Quim Chaves, David Hugas, as well as Imma Boada, Marité Guerrieri and Teresa Paradinas.

This thesis was partially supported by project TIN2004-08065-C02-02, TIN2007-67982-C02-02 and grant AP2003-4305.

Contents

Chapter 1

Introduction

In this thesis we solve visibility and proximity problems by using techniques of Computational Geometry and Computer Graphics.

Computational Geometry deals with the study of efficient algorithms to solve geometric problems. As a consequence, a fundamental task is identifying concepts, properties, and techniques which aim at efficient algorithmic development. This leads to the analysis of the combinatorial complexity of geometric structures, the study of geometric data structures, the complexity of algorithms, etc. The methods studied in this area allow the design and analysis of algorithms for the efficient solution of numerous geometric problems that arise in other application areas such as astronomy, geographic information systems, CAD/CAM, data mining, graph drawing, graphics, medical imaging, metrology, molecular modelling, robotics, signal processing, textile layout, typography, video games, vision, VLSI, and windowing systems. Recently computational geometers exploit Computer Graphics algorithms and techniques to make their algorithms more implementable and practical.

Computer Graphics is a flourishing field within computer science. Driven by the tremendous increase in speed and quality of hardware and software, it has rapidly gained popularity among a wide variety of users. With applications as far-reaching as special effects, synthetic content, interactive TV, graphical user interfaces, information visualization, interactive art, industrial design, education, computer games, virtual reality, and the Internet, Computer Graphics plays an increasingly important role in our lives, both practically and culturally. In many of its applications it uses geographical information systems in its applications.

There is a strong disparity between the nature of the contributions of papers in each of these two areas of knowledge. Computational Geometry papers most often focus on improvements for upper bounds and do not, in general, report on an implementation. The field of Computer Graphics, on the other hand, considers that an implementation is an integral part of the work. However, geographic information systems (GIS) are used in both Computational Geometry and Computer Graphics. GIS can be seen as a system of hardware, software and procedures designed to support the capture, management, analysis, modelling and displaying of spatially referenced data, pertaining to land, water, and air resources to solve complex planning and management problems. A substantial part of all activities performed by a GIS involves complex computations such as location, shape, proximity, and spatial distribution depending on the geometry of data. Since the amount of data stored in a GIS is usually very large, data structures and algorithms that achieve a good tradeoff between high computational efficiency and low storage space are required.

An important application of GIS is the representation of a terrain surface using a *triangular terrain model*. This representation plays a crucial role in a considerable number of simulations (traffic or flooding simulations) and in problems related to terrain analysis. Classical terrain analysis problems include visibility determination, proximity computations (nearest neighbors, optimal path), topographic feature extraction (degree and direction of slopes, contour lines) or computation of watershed and drainage networks. Such studies on terrains allow the resolution of facility location problems where the best placement of a facility (hospital, school, nuclear power plant, etc.) under various constraints has to be determined. Solving these problems implies working with algorithms that are difficult to implement, present high computational cost and demand complex data structures. Algorithms with similar drawbacks have been recently solved by taking advantage of a completely different strategy that uses graphic processing units (GPUs).

The increasing power, programmability, and precision of GPUs have motivated a great deal of research on General-Purpose Computation on Graphics Hardware (GPGPU) as an alternative to CPUs. GPGPU researchers and developers use the GPU as a computational coprocessor rather than as an image-synthesis device. Ideal applications have: large data sets, high parallelism and minimal dependencies between data elements. Some applications that use computational geometric research topics including visibility computations, distances, data structures for ray tracing, clipping and radiosity; hidden surface elimination algorithms; automatic simplification for distant objects; morphing; converting triangulated surfaces to strips of triangles. There also exist specialized applications

using GPU in which other geometric ideas as well as GIS models are needed including architecture, virtual reality, and video game programming.

In this thesis we are going to focus on the study of some terrain analysis problems by using Computational Geometry strategies together with the GPU power.

1.1 Contribution

The contributions presented in this thesis can be grouped in two main topics: 1) visibility problems on triangulated terrains, and 2) proximity problems on triangulated polyhedral surfaces. In both cases we consider a set of generalized elements containing points, segments, polygonal lines and polygonal regions.

Visibility problems

We present two ways to obtain approximate multi-visibility maps on triangulated terrains corresponding to a set of generalized view elements (points, segments, polygonal lines or a polygonal regions). The first strategy uses mainly Computational Geometry tools while the second strategy also Graphics Hardware.

Multi-visibility on terrains

- An algorithm to exactly compute weak or strong visibility on a segment of a triangulated terrain from a generalized view element.
- The structure of any specific multi-visibility map of a set of view elements is approximately obtained.
- An algorithm to approximately solve the inter-region multi-visibility problem on terrains.
- A way to solve different point and polygonal region multi-visibility queries.
- The study of the computational cost of the presented algorithms. Which is adapted for the special case or a realistic terrain.

Multi-visibility on terrains by using GPU

- An approximate algorithm for obtaining weak or strong visibility on a segment of a terrain from generalized view elements, with its implementation.

- An improvement on the previous algorithm by providing a faster and more robust visibility computation algorithm on a segment from a generalized view element.
- A practical and implemented algorithm for visualizing approximate multi-visibility maps, obtained by using the GPU, for a triangulated terrain and an heterogeneous set of generalized view elements.
- An algorithm to solve efficiently and approximately various point and polygonal region multi-visibility queries.
- The study of the computational cost of the provided algorithms. It is adapted to the special case of a realistic terrain.
- The implementation of the proposed methods and experimental results.

Proximity problems

We consider weighted and non-weighted triangulated polyhedral surfaces and provide a way to obtain shortest paths on each of the two kinds of surfaces when a generalized source is considered. The algorithms are extended to consider a set of generalized sources, in this case the algorithm outputs its distance field which gives for each point of the surface the shortest path distance from the point to the closest site in the set. As applications we obtain discrete high-order Voronoi diagrams and approximately solve some facility location problems. We also provide a theoretical study on the complexity of high-order Voronoi diagrams on non-weighted polyhedral surfaces.

Distance computation on polyhedral surfaces

- An algorithm to compute shortest paths from generalized sources and polyhedral surfaces with obstacles.
- A generalization of the algorithm to the case of several generalized sites providing their implicit distance field, which implicitly represents their closest Voronoi.
- A way to obtain the shortest path from arbitrary points on the polyhedral surface to the source and to answer punctual shortest path distance queries.
- The study of the computational cost of all the presented algorithms.
- The implementation of the proposed methods for the special case of triangulated terrains and experimental results.

- A theoretical study of the complexity of the Voronoi diagram of a set of generalized sites.
- A theoretical study of the complexity and properties of order-k Voronoi diagrams of generalized sites on triangulated polyhedral surfaces.
- The complexity analysis of the algorithms and order-k Voronoi diagrams for a special triangulated polyhedral surface: a realistic terrain.

Distance computation on weighted polyhedral surfaces

- A discretization scheme to build a discrete graph on a weighted polyhedral surface that provides $(1+\epsilon)$-approximate weighted shortest path distances from generalized sources.
- A way to compute approximate distance functions from a generalized site on a possibly non-convex weighted polyhedral surface $\mathcal{P}$ with obstacle (using Bushwack strategy).
- A generalization of the algorithm to the case of a set of generalized sites, giving their distance field.
- The process for obtaining the distance from and the shortest path from S to any point of $\mathcal{P}$.
- The study of the computational cost of the provided algorithms.
- The implementation of the proposed methods for the special case of a weighted triangulated terrain and experimental results.

Voronoi diagrams and facility location problems

- An algorithm that uses hardware graphics to discretize the obtained distance function or fields.
- An algorithm, based on distance functions and hardware graphics capabilities, specifically designed to compute discrete closest Voronoi diagrams of a set of r generalized sites on a weighted polyhedral surface $\mathcal{P}$ with obstacles.
- An algorithm, based on distance functions and hardware graphics capabilities, that allows the computation of all discrete order-k Voronoi diagrams, $k = 1 \ldots r-1$.
- An algorithm to compute an approximate 1-Center of a set of generalized sites.
- An algorithm to compute an approximate 1-Median of a set of generalized sites.

- The complexity analysis of the mentioned algorithms.
- An implementation of the previously mentioned algorithms for the special case of triangulated polyhedral terrains we also present experimental results.

1.2 Overview

The rest of the thesis can be subdivided into four main parts. The first one consists on Chapter 2, where preliminaries are presented. The second one contains Chapter 3 and Chapter 4 where visibility problems are considered. The third one is where proximity problems are tackled and consists of Chapter 5 to Chapter 7. Finally, the last one is Chapter 8 where conclusions and further comments are done.

In Chapter 2 **Basic concepts an previous work**, we review previous work upon which our research draws. We provide some basic notation, concepts and algorithms to solve visibility and proximity problems in $\mathbb{R}^2$ and $\mathbb{R}^3$. Next, we introduce the Graphics Hardware and provide some applications and basic algorithms. We give a brief overview on how terrains are represented, and finally, we present the existent algorithm for solving visibility and proximity problems from punctual sources on terrains.

In Chapter 3 **Multi-visibility on terrains**, we present some basic properties, next, we provide the algorithm to compute exact weak or strong visibility information from a view segment and we explain how multi-visibility information is computed and stored. Finally we explain how the multi-visibility map is obtained and specifying some applications. Complexity analysis is provided along the chapter.

In Chapter 4 **Multi-visibility on terrains by using graphics hardware**, an approximate algorithm for computing visibility information from a view segment by using Graphics Hardware is provided. Next, we explain how the multi-visibility map is obtained and visualized on the screen and mention some applications. The study of the computational cost of the algorithms is also given. We present some experimental results and images of our implementation.

In Chapter 5 **Distances on polyhedral surfaces**, we provide an algorithm for obtaining exact shortest paths distance functions on a non-convex polyhedral triangulated surface with generalized obstacles from generalized sources. The algorithm is generalized for the case of a set of generalized elements to provide their distance field. We also present a way to compute distances and shortest paths from a point of the polyhedral

surface to the closest generalized element. The Chapter ends with a theoretical study of the properties and complexity of the Voronoi diagram of a set of generalized sites on a non-convex polyhedral surface. We also study the complexity and properties of high-order Voronoi diagrams of such a set of sites on a triangulated polyhedral surface. The proposed algorithms have been implemented and experimental results are presented in Chapter 7.

In Chapter 6 **Weighted distances on polyhedral surfaces**, we present an algorithm for obtaining approximate shortest paths distance functions on a non-convex polyhedral triangulated surface with generalized obstacles from generalized sources. We start by providing a discretization scheme to obtain a discrete graph on the polyhedral surface that gives $(1 + \epsilon)$-approximate distances from generalized sites. The algorithm is generalized for the case when a set of generalized sources is considered and provides their distance field. Finally we give a way to compute distances and shortest paths from a point of the polyhedral surface to the closest generalized element. The proposed algorithms have been implemented and experimental results are presented in Chapter 7.

In Chapter 7 **Discrete distance function and applications**, a discrete representation of the distance functions and distance fields obtained in Chapter 5 and Chapter 6 are obtained by using Graphics Hardware. We provide an algorithm to obtain the Closest Voronoi diagram from the distance field. Next, general algorithms are used to compute the closest, furthest and any order-k Voronoi diagram. An algorithm to obtain an approximate 1-Center and an approximate 1-Median is also presented. Finally, some experimental results and images are given.

Finally, in Chapter 8 **Conclusions and further comments**, we provide the conclusions of this thesis and we end with some further comments.

Chapter 2

Basic Concepts and Previous Work

In this chapter some basic concepts and the previous work related to visibility on terrains and distances on polyhedral surfaces developed in Computational Geometry, Geographical Information Systems and Computer Graphics are presented.

First, some general definitions and notation are given (Section 2.1). Next, some basic data structure and algorithms that are used in the algorithms presented in this thesis are given (Section 2.2). Graphics hardware is introduced and some general algorithms base on the GPU are provided (Section 2.3). Next, we present some definitions related to terrain models (Section 2.4) together with previous work related to visibility on terrain models (Section 2.5). Finally, proximity problems on triangulated polyhedral surfaces are presented with the previous work related to these problems (Section 2.6).

2.1 General definitions and notation

In this section the definitions and notation that are used in this thesis are provided, starting with some general definitions in $\mathbb{R}^d$ for $d = 2, 3$ where $u = (u_1, \ldots, u_d)$ and $v = (v_1, \ldots, v_d)$ denote d-dimensional vectors, and $p = (p_1, \ldots, p_d)$ and $q(= q_1, \ldots, q_d)$ d-dimensional points.

- *Sum of vectors.* The vector $u + v = (u_1 + v_1, \ldots, u_d + v_d)$.

- *Product between a vector and a real number.* The vector $\lambda u = u\lambda = (\lambda u_1, \ldots, \lambda u_d)$.
- *Scalar product between vectors.* The real number $u \cdot v = u_1 v_1 + \cdots + u_d v_d$.
- *Norm of a vector.* The positive real number $||u|| = \sqrt{u_1^2 + \cdots + u_d^2}$.
- *Vector that joins two points.* The vector $\overrightarrow{pq} = (q_1 - p_1, \ldots, q_d - p_d)$.
- *Euclidean distance between two points.*

$$d(p, q) = ||\overline{pq}|| = \sqrt{(q_1 - p_1)^2 + \cdots + (q_d - p_d)^2}.$$

- *Convex set A in $\mathbb{R}^d$.* A set of points A in $\mathbb{R}^d$ such that for any two point p and q in A segment $\overline{pq}$ is entirely contained in A.
- *Generalized element.* A point, segment, polygonal chain or polygonal region in $\mathbb{R}^d$.
- *Convex hull* of a set of points A in $\mathbb{R}^d$ ($CH(A)$). The smallest convex set in $\mathbb{R}^d$ containing A.
- *Planar Subdivision $\mathcal{S}$.* A set of planar regions with disjoint interiors and whose union is the total plane (See Figure 2.1).
  - *Vertex of $\mathcal{P}$.* The intersecting point (if it exists) of two different edges of $\mathcal{P}$.
  - *Edge.* The common points of two adjacent regions.
  - *Face.* Each region of $\mathcal{P}$.
  - *Adjacent regions.* Two not disjoint regions of a planar subdivision.

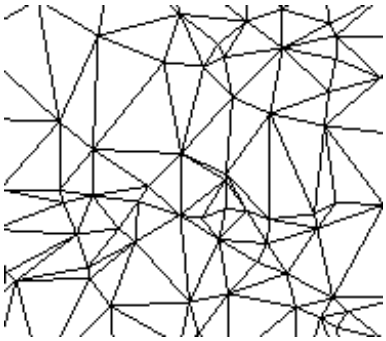


Figure 2.1: Planar subdivision (Delaunay triangulation).

- *Triangulation.* A planar subdivision whose bounded regions are triangles (Figure 2.1).

- *Triangulation of a set of points P.* A triangulation whose vertices are the elements of P.

- *Delaunay triangulation of a set of points P.* A triangulation of a set of points P such that no point in P is inside the circumcircle of any triangle of the triangulation.

- *Constrained Delaunay triangulation of P.* A triangulation of P where some edges are specified. It is such that if the circumcircle of a triangles contains a point $p \in P$, the line segments from p to the vertices of the triangles intersect a specified edge.

- *Polyhedron.* A polyhedron is a solid in $\mathbb{R}^3$ bounded by planar facets.

- *Polyhedral surface.* A polyhedral surface is the boundary of a polyhedron, which contains vertices, edges and faces.

- *Arrangement of r surfaces in $\mathbb{R}^d$.* The subdivision of $\mathbb{R}^d$ induced by the surfaces.

  - *Upper envelope.* The set of points with $r-1$ surfaces below them, and no surfaces above them.

  - *Lower envelope.* The set of points without surfaces below and strictly $r-1$ above them.

  - *k-th level.* The set of points with at most $k-1$ surfaces strictly below it, and at most $r-k$ surfaces strictly above. The lower and upper envelopes correspond to the 1-th and r-th levels, respectively.

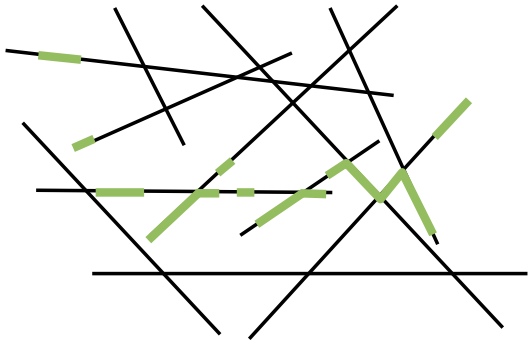


Figure 2.2: The third level in an arrangement of segments.

2.2 Basic data structures and algorithms

In this section some data structures and algorithms used in Computational Geometry are presented. Defining the doubly-connected edge list is the first step(Section 2.2.1). Next *sweep algorithms*, which are widely used in Computational Geometry for solving several basic problems, are presented (Section 2.2.2). We provide basic algorithms and the complexity of basic structures such as the convex hull (Section 2.2.3), an arrangement of surfaces (Section 2.2.4) or a Delaunay triangulation (Section 2.2.5). Next, we describe an algorithm to traverse a planar subdivision, which optimally reports the regions, edges and vertices of a planar subdivision (Section 2.2.6), together with another one to approximately reconstruct an unknown planar subdivision using line probes and a DCEL (Section 2.2.7). Next we start with specific concepts and problems related to visibility and proximity. We start with basic concepts and algorithms related to visibility on the plane and space (Section 2.2.8), we go on with basic algorithms to compute Voronoi diagrams on the plane and space, which are used to solve proximity problems (Section 2.2.9), and we finish with, facility location problems on the plane and space(Section 2.6.4).

2.2.1 Doubly-connected edge list

A *doubly-connected edge list* (DCEL) is a structure used to represent planar subdivisions (triangulations, Voronoi diagrams, etc.), polyhedral surfaces, polytopes in $\mathbb{R}^3$, etc. [35]. It supports operations for example: given a region, it obtains its adjacent regions, edges or vertices. A DCEL contains a record for each face, edge, and vertex of the subdivision (Figure 2.3). Besides the geometric and topological information each record may also store additional information. The geometric and topological information stored in the DCEL enables us to perform basic operations like: walking around a face in counterclockwise order, accessing one face from an adjacent one given the common edge, visiting all the edges around a given vertex. In the DCEL structure, edges in the subdivision are represented by two directed half-edges: one of the half-edges bounds one face incident to the half-edge of the other half-edge bounds the other face.

2.2.2 Sweep algorithm

Sweep algorithms are used to simplify several problems in $\mathbb{R}^d$ reducing them to problems in $\mathbb{R}^{d-1}$. Problems in the plane are reduced to problems in a line by using a *sweep line*

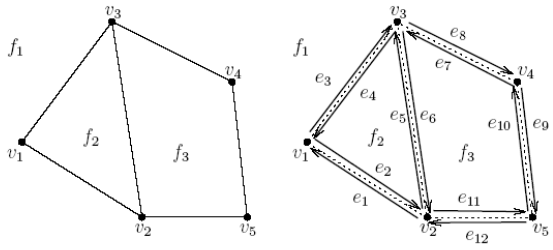


Figure 2.3: DCEL structure.

algorithm, in the same way, problems in space are reduced to problems in the plane by using a *sweep plane algorithm*. Sweep algorithms already exist to compute: the intersection points of a set of segments in the plane, the overlay of two planar subdivisions, the Voronoi diagram of a set of points in the plane, the visible parts of a scene in the plane, the closest pair of a set of points in space, the intersection of planar polygons as objects in space, the computation of tetrahedralizations, of convex hulls, etc [35, 39, 74].

A *sweep line/plane* algorithm splits the problem domain into two regions: an explored region and an unexplored one. The sweep line/plane applies an ordering to the problem. Thus, we get a result on the explored area based on what we have seen so far and ignore the unexplored area. In fact, only the elements that have already been reached or crossed can possibly affect the current computation. The entire problem area/volume is examined by "sweeping" the line/plane across the set of elements from one extreme to another. While we sweep, we keep track of all the elements intersecting the sweep line/plane. The *status* of the sweep line/plane is the set of elements intersecting it. The points where the status changes are called *event points*. In Figure 2.4 the status of the sweep line when a sweep line algorithm is used to compute the intersection points of a set of segments in the plane can be seen.

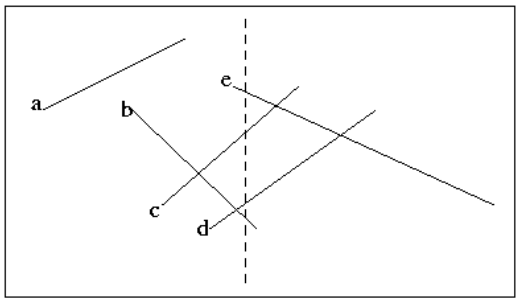


Figure 2.4: Sweep line algorithm for computing the intersection points of a set of line segments.

2.2.3 Convex hull

The convex hull $CH(A)$ of a set A of points of $\mathbb{R}^d$ is the smallest convex set in $\mathbb{R}^d$ containing A and has complexity $O(n)$ where n is the cardinality of A and $d = 2$ or 3.

Several algorithms to compute $CH(A)$ already exist, which can be obtained by using a divide and conquer technique [119] obtaining a $O(n \log n)$ time complexity algorithm. An incremental construction with expected $O(n \log n)$ time complexity also exists [35]. There also exist two optimal output-sensitive algorithms for the planar case with $O(n \log h)$ time complexity, where h is the number of points in the hull [24, 25, 85].

2.2.4 Arrangements

The arrangement induced in the plane by a set L of n lines in the plane has $O(n^2)$ vertices, edges and faces. A DCEL representing such an arrangement of lines in $O(n^2)$ time can be built [35]. The complexity of the k-th level is $O(n\sqrt{k+1})$, and the lower and upper envelope can be obtained in maximal $O(n \log n)$ time complexity [99]. It has also been proved that the complexity of any single level in an arrangement of n line segments in the plane is $O(n^{3/2})$ [3].

When a set of n planes in space is considered, the arrangement has complexity $\theta(n^3)$ and a k-th level $O(n^2k^{2/3})$ [3, 25]. The upper and lower envelope of an arrangement of r surfaces in space, under some general conditions, is $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$, and the complexity of all the $l \leq k$ levels is $O((k+1)^{1-\varepsilon}n^{2+\varepsilon})$ [128]. Sharir studied arrangements in higher dimensions and provided upper bonds on their complexity and applications of these arrangements [134].

2.2.5 Delaunay triangulation

There are several algorithms to compute Delaunay triangulations, as well as some incremental algorithms that repeatedly add one vertex at a time and retriangulate the triangulation until a Delaunay triangulation is obtained. With this technique, a naive incremental algorithm results in a running time of $O(n^2)$. If a sweep line to insert the vertices sorted is used the expected running time becomes $(n^{3/2})$ time, and if vertices are inserted at random the expected running time is $O(n \log n)$. An efficient $O(n \log n)$ time incremental algorithm that uses a tree structure is given [36].

There are in existence optimal algorithms which use a divide and conquer algorithm for triangulations in two dimensions due to Lee and Schachter [147]. The set of vertices is recursively split into two sets. The Delaunay triangulation is computed for each set, and then the two sets are merged in $O(n)$ time. The total running time is $O(n \log n)$.

A constrained Delaunay triangulation can be constructed in three steps: first the given points and the endpoints of the constrained edges are inserted, then, the specified edges are enforced and finally the edges are swapped if the Delaunay criterion is violated [61].

2.2.6 Subdivision traversal

A subdivision traversal algorithm enumerates all vertices, edges and faces of a planar subdivision, $\mathcal{S}$, stored in a data structure, for example in a $DCEL$ (Section 2.2.1). Some techniques traverse $\mathcal{S}$ by using extra storage such as mark bits on the edges, vertices, or faces of $\mathcal{S}$, a data structure, etc (for a survey [36]). However, there are some algorithms without extra storage to traverse $\mathcal{S}$ [36, 20]. De Berg et al. [36] traverse any connected subdivision with n vertices in $O(n^2)$ time and any convex subdivisions in $O(n)$ time. The algorithm can be adapted to solve region queries and enumerates only the connected subset $\mathcal{S}'$ of $\mathcal{S}$. In these cases the running time depends on the vertices in $\mathcal{S}'$. Bose and Morin [20] slightly modified the algorithm of de Berg et al. so that it runs in $O(n \log n)$ time.

2.2.7 Unknown planar subdivision approximation

In this Section the algorithm of Coll et al. [33] that reconstructs an approximation of an unknown planar subdivision with information gathered from linear probes of the subdivision is presented.

Given an unknown bounded planar subdivision $\mathcal{S}$, the algorithm approximately recovers the vertices, edges and faces of $\mathcal{S}$ from the information of linear probes (Figure 2.5). The algorithm maintains an approximation of the planar subdivision which is updated after processing the information of each line probe and converges to $\mathcal{S}$ as the number of line probes increases. The accuracy of the reconstruction and the convergence speed depends on how line probes are chosen. The best way to chose them is by generating lines uniformly distributed over a bounding box B of $\mathcal{S}$.

Subdivision $\mathcal{S}$ partitions a line probe L into a finite set of segments with endpoints on edges or vertices of $\mathcal{S}$. These segments have to be computable. Once obtained, each

segment has to be labelled with the face of $\mathcal{S}$ where it is contained. The reconstruction algorithm maintains a triangulation of the endpoints of the segments from where an approximation of the unknown subdivision can be extracted. The mean computational cost of the algorithm when k lines are processed is $O(k \log k)$.

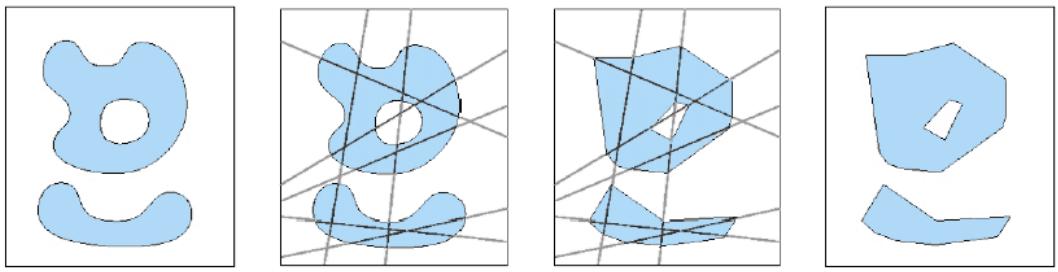


Figure 2.5: From left to right we have: the unknown planar subdivision, the line probes and the approximate reconstruction with the information gathered from the line probes.

2.2.8 Visibility in the plane

Visibility problems have been widely studied from different areas such as Computer Graphics, Computational Geometry and also specialists working on GIS when dealing with visibility on terrains.

A scene consisting of a set of opaque elements placed in the plane or in space (sometimes the elements are the faces of a terrain) and an view element are usually given. Generally, a *view element* is a point, V, or a segment, s, from where the visibility is studied. The aim is to determine the parts of the scene that are visible from the view element. By definition, a point q *is visible from* V if the interior of the line segment $\overline{pq}$ does not intersect with any opaque element conforming the scene; q is *strongly visible from* s if q is visible from every point of the view segment s; q is *weakly visible from* s if q is visible from some point of s.

Visibility studies in $\mathbb{R}^2$ are the basis for some of the important algorithms working in $\mathbb{R}^3$ and in terrains. A typical $\mathbb{R}^2$ scene contains a set of static segments and a view point or view segment (See Figure 2.6). Following this, the typical algorithms to compute visibility from different types of view elements and the structures used to compute and store visibility information are presented.

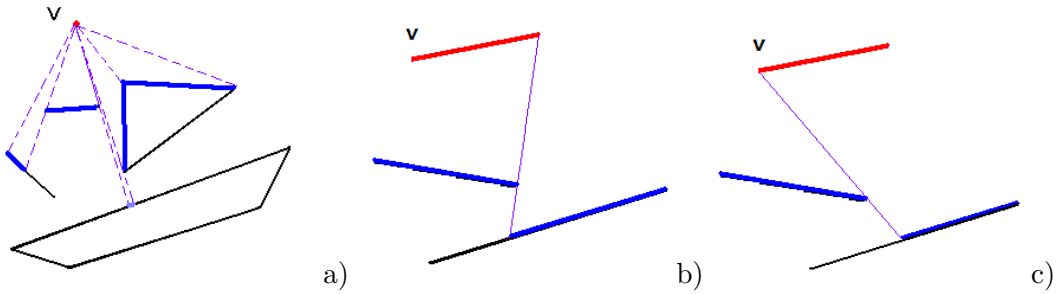


Figure 2.6: Typical scenes in $\mathbb{R}^2$ where the visible parts are marked in blue when considering a: a) view point; b) view segment and weak visibility; c) view segment and strong visibility.

2.2.8.1 View point

The visible parts of a scene from a view point can be computed as: a) the lower envelope of the set of segments [29, 72] by using a dynamic approach, a divide-and-conquer technique, a parallel algorithm, etc. b) as the illuminated parts of the scene when a point light source is placed on the view point, these algorithms deal with shadow points and visibility cycles [62] which are explained in the next section. c) using a ray tracing technique in the GPU, one of the most used rendering techniques in Computer Graphics, [122]. It checks whether a ray intersects the scene, and where the scene is intersected, consequently whether the intersection point is visible (Section 2.3.3).

2.2.8.2 View segment

When considering algorithms to compute visibility form a view segment we can distinguish between the exact and approximate algorithms. There are some exact algorithms that provide the visible parts of the scene from every point of s, and some others that compute the weakly or strongly visible parts of the scene. Among the approximate algorithms an algorithm that uses graphics hardware and the GPU to compute visibility of a segment of the scene from the view segment is highlighted (Section 2.3.3).

Exact algorithms that provide the visible parts of the scene from every single point of s are sweep algorithms [63, 64, 113]. A *moving point* V is placed on the view segment and it is moved along s keeping track of the *critical points*. Critical points are those points where the visible scene in both directions of any of its neighborhoods in s is different. In a critical point it is said that a *topological or visibility change* occur. This moving point technique can be generalized to handle free trajectories of the moving point. On the other

hand, the weakly or strongly visible parts of the scene from a view segment are obtained by placing a *linear light source* on the view segment [62]. The amount of light received by each part of the scene is computed after finding the critical points by using the moving point strategy. Depending on the amount of light, a part of the scene is weakly, strongly or not visible.

2.2.8.3 Visibility data structures

Several structures are used to compute or maintain the visible parts of the scene. The main ones are: the *visibility graph* [64, 63, 113, 148], the *topological map* [113, 148], the *visibility cycle* [62] and the *visibility complex* [118, 125].

In our algorithms visibility cycles which are ordered sequences of visible segments and segments endpoints conforming the scene that are from the view point V are used. Depending on whether the segments conforming the scene are considered to be transparent or opaque, the *transparent visibility cycle* or the *opaque visibility cycle*, is respectively obtained. Transparent visibility cycles usually store for each vertex, w, the first edge after w intersected by the ray going from V to w [62].

2.2.9 Voronoi diagrams in the plane and in space

A Voronoi diagram encodes proximity information for a set of given objects which are named sites. It is a fundamental concept of Computational Geometry and has been widely studied [12, 13, 32, 41, 115, 120]. Algorithms and studies related to Voronoi diagrams on the plane, space and also on non-weighted polyhedral surfaces can be found. Voronoi diagrams on the plane or space have been extended to work with different types of sites and distances. In the next sections some definitions and results for Voronoi diagrams on the plane and space, and next on polyhedral terrains are given.

Algorithms to compute Voronoi diagrams on the plane or space are useful not only for the information of proximity that they store. They are also useful to obtain the Delaunay triangulation of the set of sites defining the diagram. In fact, the Delaunay triangulation is the dual graph of the Voronoi diagram, and two sites of S are joined by an edge of the Delaunay triangulation when their Voronoi regions are adjacent.

2.2.9.1 Voronoi diagram of a set of points

The Voronoi diagram of a set, $S = \{s_1, \ldots, s_r\}$, of r point sites in $\mathbb{R}^d$ with $d = 2, 3$ is the subdivision of the plane into Voronoi regions which are associated to the sites. In fact, each site has a Voronoi region associated. The Voronoi region of site s_i contains the points of $\mathbb{R}^d$ closer to s_i than to any other site s_j

$$V(s_i) = \{x \in E^d \mid d(s_i, x) \leq d(s_j, x)\}$$

where $d(x, y)$ denotes the Euclidean distance. The locus of equidistant points from two sites s_i and s_j of S is named the *bisector* between sites s_i and s_j and is denoted β_{ij}, thus, $\beta_{ij} = \{x \in E^d \mid d(x, s_i) = d(x, s_j)\}$. Notice that β_{ij} is a plane in $\mathbb{R}^3$ and a line in $\mathbb{R}^2$. The Voronoi diagram of S is the family of Voronoi regions of all the sites of S. In practice, we often work with a bounded region K and the part of each Voronoi region contained in K is considered. In Figure 2.7 a) a Voronoi diagram of a set of points in the plane considering Euclidean distance is shown.

When S contains points of $\mathbb{R}^d$ in *general position*, no $d + 2$ points are cospherical. The vertices of the Voronoi diagram are the centers of the spheres passing through $d + 2$ points. The complexity of the Voronoi diagram is $O(r^{\lceil \frac{d}{2} \rceil})$, [145]. Practical and robust optimal $O(r \log r)$ and $O(r^2)$ algorithms for computing the exact Voronoi diagram of a set of points in 2D and 3D have been developed in Computational Geometry and related areas. There is also an incremental algorithm in existence that works in $\mathbb{R}^2$ and $\mathbb{R}^3$ [22], with complexity $O(r^{\lceil \frac{d}{2} \rceil + 1})$.

2.2.9.2 Generalized Voronoi diagram

The Voronoi diagram concept has been extended according to its practical applications [12, 13]. With this aim, sites of different shape or nature, weights associated to the sites, different distance functions for each site, etc. have been considered yielding to the Generalized Voronoi diagrams [17, 18, 32, 115].

The set of sites $S = \{s_1, \ldots, s_r\}$ is now a set of r different geometric objects in $\mathbb{R}^d$. Each site is associated to a distance function $d_{s_i}(x)$. The Generalized Voronoi diagram has as Voronoi region of a site s_i the set of points $x \in \mathbb{R}^d$, such that, $d_{s_i}(x) \leq d_{s_j}(x)$ $\forall j \neq i$. The bisector between two different sites s_i and s_j is defined, again, as $\beta_{ij} = \{s \in E^d \mid d_{s_i}(x) = d_{s_j}(x)\}$. If a set of line segments with Euclidean distance is considered, the bisectors are line and parabolic segments [145]. In Figure 2.7 b) and c) we show

an example of Voronoi diagrams on the plane considering a set of segments and a set of generalized sites, respectively.

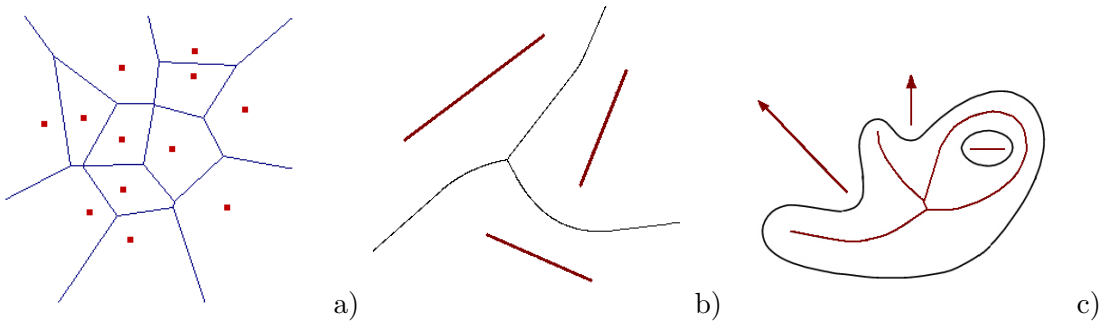


Figure 2.7: Voronoi diagram in the plane, considering Euclidean distance and a set of: a) point sites; b) segment sites; c) generalized sites: a segment, a curve and two semi-line sites.

The Voronoi diagram of a set of line segments with Euclidean distance can be exactly obtained in $O(r \log r)$ time by using the optimal algorithm provided by Yap [156]. Discrete generalized Voronoi diagrams on $\mathbb{R}^2$ and $\mathbb{R}^3$ and the distance functions defined by the sites can be computed by using graphics hardware (Section 2.3.3).

2.2.9.3 Higher order Voronoi diagram

A Voronoi diagram gives information of the closest site and a order-k Voronoi diagram informs us of the k closest sites at each point. Let $S = \{s_i\}$ be a set of r sites with associated distances $\{d_{s_i}\}$ and $S' \subset S$ a subset of k sites, $k \in \{1, \ldots, r-1\}$. The set of points of $\mathbb{R}^d$ closer to each site of S' than to any other site of S is possibly an empty region called the order-k Voronoi region of S'. The set of order-k Voronoi regions of all subsets of k sites of S is called the order-k Voronoi diagram of S. The order-1 Voronoi diagram is called the *closest Voronoi diagram* or simply the *Voronoi diagram*, and the order-$(r-1)$ Voronoi diagram is called the *furthest Voronoi diagram*.

The size of the order-k Voronoi diagram of a set of r sites in the plane is $O(k(r-k))$, the first proof of this result was given by Lee, [92], some years latter an alternative proof was provided by Shmitt and Spehner, [130]. The size of all the Voronoi diagrams of order lower or equal to k is, consequently, $O(rk^2)$. In higher dimensions, the size of all the Voronoi diagrams of order lower or equal to k is $O(k^{\lceil \frac{d+1}{2} \rceil} r^{\lceil \frac{d+1}{2} \rceil})$ this was proved by Cole and Sharir, [30]. See Figure 2.8 to see the first, second, third and forth order Voronoi diagram of a set of point sites.

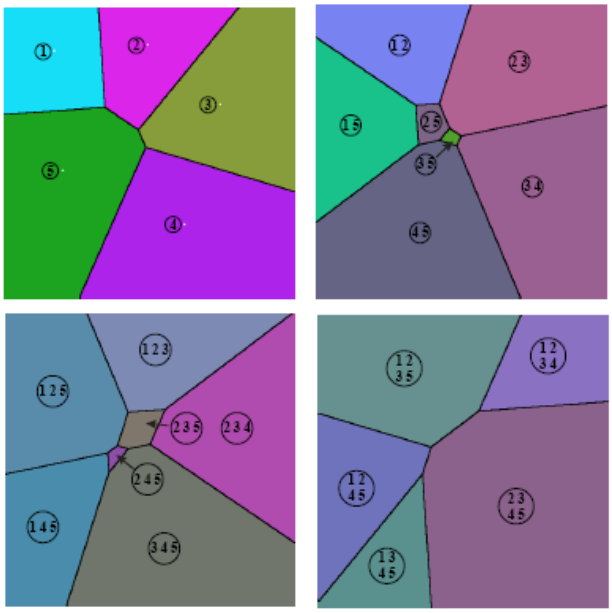


Figure 2.8: a) Voronoi diagram of a set of points. b) order-2 Voronoi diagram. c) order-3 Voronoi diagram. d) order-4 Voronoi diagram.

2.2.10 Facility location problems

Classical facility location problem determine the location X where a new facility should be located so as to minimize an objective function [38, 102] when a set of existent facilities represented by r sites is given. The set of points may represent customers, plants to be serviced, schools, markets, distribution sites, etc. and as an objective function we can use the Euclidean distance from X to its furthest customer. The two specific facility location problems that are considered are the 1-Center and the 1-Median. The former minimizes the maximal distance to an existent facility, and the latter, the sum of the distance to all the existent facilities.

When a set S of generalized sites is considered, the 1-*Center* is the point that minimizes the maximum distance to the sites. This point lies at a vertex or on an edge of the furthest Voronoi diagram and is the center of the smallest enclosing disc (the disc with the smallest radii containing S). The 1-Center of a set of r point sites in the Euclidean plane can be determined in $O(r)$ time [101, 154]. Agarwal et al. [5] present an approach to efficiently solve the 1-Center problem for a set of points P in the plane using graphics hardware.

The 1-*Median* of the set S of point sites is the point that minimizes the sum of distances to the sites. No polynomial-time algorithm for computing the 1-Median, also known as

the Fermat-Weber point, of a set of r point sites in the Euclidean plane is known, nor has the problem been shown to be NP-hard. The most common approximation algorithm is Weiszfeld's algorithm [153], an iterative procedure that converges to the 1-Median. Chandrasekaran and Tamir [26] present a polynomial time approximation scheme based on the standard ellipsoid method. Bose et al. [19] give a linear-time randomized algorithm and an $O(r \log r)$ time deterministic one for ε approximations of the Euclidean median. Agarwal et al. [5] present an algorithm for obtaining the 1-Median of a set of points P in the plane using graphics hardware.

2.3 Graphics hardware

The increasing programmability and high computational rates of graphics processing units (GPUs) make them attractive as an alternative to CPUs for general-purpose computing. Recently, different algorithms and applications that exploit the inherent parallelism, easy programmability and vector processing capabilities of GPUs have been proposed [117, 122]. In Computational Geometry and GIS fields there are several algorithms that have a fast hardware-based implementation [5, 43, 67, 76, 86, 87, 111].

In this section the graphics hardware pipeline (Section 2.3.1) is explained. The concept of a planar parameterization which is used when working with models in $\mathbb{R}^3$ and GPU is presented (Section 2.3.2). In addition graphics hardware applications presenting some general algorithms and finally some specific ones related to visibility and proximity (Section 2.3.3).

2.3.1 Graphics pipeline

The graphics pipeline [132] is divided into several stages, which are implemented as separate pieces of hardware on the GPU. The input to the pipeline is a list of 3D geometric primitives expressed as vertices defining points, lines, or polygon corners with attributes associated such as 3D coordinates, color, texture coordinates, etc. The output is an image in the frame buffer. The frame buffer is a collection of several hardware buffers corresponding to two dimensional grids whose cells are called pixels. Each pixel in the frame buffer is a set of some number of bits grouped together. There exist different buffers: the stencil buffer, the depth buffer and the color buffer which store stencil, depth, and RGB color values, respectively.

In order to store information in arrays in the GPU textures, arrays in the GPU, are used and the information they store is accessed by using texture coordinates. At present, and depending on the GPU, the maximal size of a texture is 2048×2048 or 4096×4096 and at most eight textures can be simultaneously used. There are several types of textures: RGBA-textures which store four numbers of a maximum of 16 bits in each position, one in each of the RGBA channels (making 64 bits per position); depth textures store depth values, a single number per position, which can have up to 32 bits.

In the first stage of the pipeline, per-vertex operations take place. Each input vertex is transformed from 3D coordinates to window coordinates. The next stage is rasterization. When it is finished, a fragment, with its associated attributes, for each pixel location covered by a geometric primitive is obtained. Fragment attributes such as depth, color, texture coordinates, etc. are obtained from the attributes associated to the vertices by linear interpolation. The third stage, the fragment stage, computes the color for each pixel according to the fragments corresponding to it. Each fragment passes a series of tests (scissor, alpha, stencil, depth) and per-fragment operations (updating, blending, logical operations, etc.) to avoid rendering or modifying the appearance of some fragments before being placed into its corresponding pixel of the frame buffer. Per fragment operations can use values from textures to modify its depth, color, etc. Finally, the fragments that pass the tests and the operations of the previous stage are drawn or rendered on the screen or on a specified texture with the appropriate color. The information of a user defined rectangle of the color buffer can be easily transferred to the CPU or directly to a texture. The graphics pipeline is schematically represented in Figure 2.9.

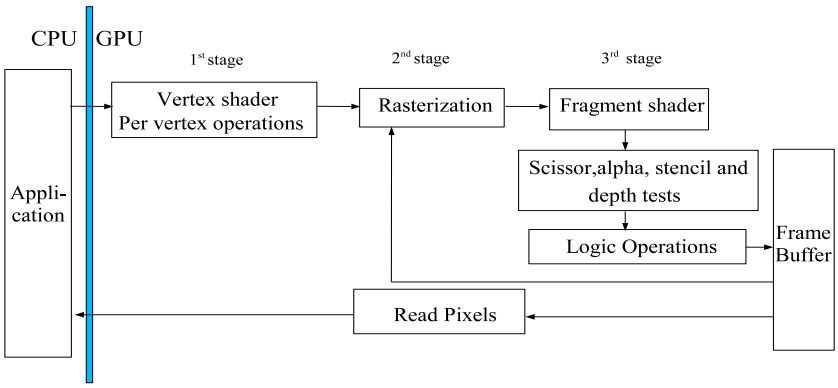


Figure 2.9: The graphics pipeline

The unique programmable parts of the graphics pipeline are the vertex and fragment shaders. Vertex shaders are executed on a per-vertex basis and fragment shaders, also

called pixel shaders, are executed on a per-fragment basis. Fragment shaders can change the appearance of the pixels by combining fragment values, such as color and depth, with values stored in the fragment attributes or in textures, which are sent to them as parameters. If necessary, a fragment shader can output data to different render targets in a single pass by using MRT (multiple-render-target) buffers.

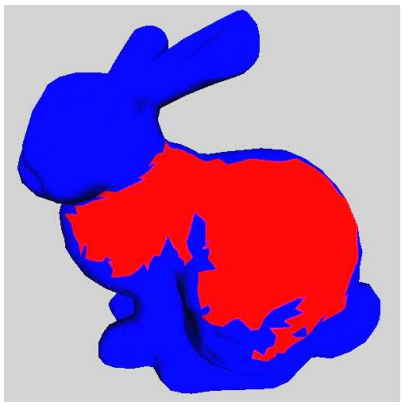
2.3.2 Planar parameterization

A basic use of graphics hardware is rendering scenes, generally defined by polyhedra, in $\mathbb{R}^3$. These scenes have to be mapped totally or partially to two dimensional arrays, textures, to provide some rendering effects on them. With this aim, each polygonal face of the scene is then mapped to a polygon in the plane with approximately the same shape (angles and area) as that in the scene. Two different polygons of $\mathcal{P}$ are mapped so that the image of the polygons do not overlap (See Figure 2.10). These mappings from $\mathcal{R}^3$ to $\mathcal{R}^2$ are known as *planar parameterizations* of a scene or surface. Computing a parameterization means finding the image also referred to as planar coordinates of each polygon of a scene. It is generally assumed that all the polygons are triangles. However, this is not a restriction because any polygon can be triangulated. Planar parameterizations are well known and used in various problems such as: texture mapping, geometry processing, remeshing, etc. [23, 50].

The problem of obtaining a parameterization maintaining the shape of the triangles is also known as the *triangle packing* problem. The algorithm proposed by Carr et al. [23] translates and rotates the mesh triangles so that they are placed in the $xy-$plane with the longest edge aligned to the $x-$axis. After sorting the triangles by increasing altitude, each other triangle is flipped, triangles are grouped into equal length sections and finally triangle groups are stacked vertically. At the end of the process, the triangles are packed into an axis-parallel rectangular region $\mathcal{R}$. The time cost of the algorithm is $O(n \log n)$. These algorithms work with polyhedral surfaces that are not necessarily homeomorphic to spheres.

2.3.3 Graphics hardware applications

In the next sections we present examples of applications of the graphics hardware in several areas such as solving problems related to: visibility, Voronoi diagrams, facility location, etc. Notice that graphics hardware uses pixels, thus, if we use the pixels to


a)

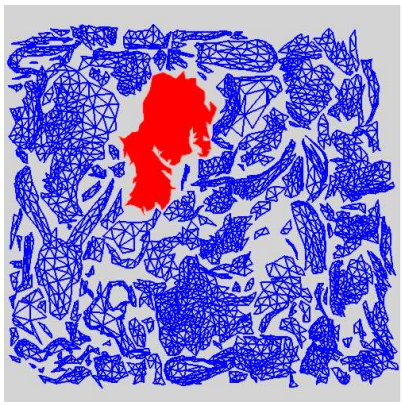

b)

Figure 2.10: a) A triangulated surface representing a rabbit; b) A planar parameterization of the surface presented in a). The red in image b) corresponds to the red area in image a).

represent a plane, it is discretized and the output is a discrete representation of what we are computing.

2.3.3.1 Upper and lower envelopes

Hardware graphics capabilities can be used to obtain easily a discrete representation of the upper or lower envelopes of $\Phi = \{ z = \pi_i(x, y) : i = 1 \dots n \}$ a given set of n planes in space. The functions defining the planes are painted one after the other, and the z value is stored at the depth value. The depth test is used to store, in each pixel, the minimum or maximum depth value depending on whether we are interested in the lower or upper envelope, respectively. If each function is painted in a different color, the final color of each pixels is the one associated to the function giving the minimum or maximum z value.

2.3.3.2 Depth peeling

Depth peeling or order-independent transparency is a recently developed technique in GPU programming first implemented by Everitt [42], and used in several algorithms [45, 112]. The depth peeling, is used to obtain the *k-th level* of an arrangement in $\mathbb{R}^3$ (Section 2.1). This is a multi-pass algorithm, which "peels" off one level of an arrangement at each step. At each step, all the functions conforming the arrangement are rendered. Before writing to the frame buffer, the current depth is compared against the closest depth found in the previous pass. Thus, at pass k the frame buffer contains the kth level of the arrangement.

This is done by using two depth buffers, one for the previous level closest depth, and one for the current level closest depth. At each step, the fragments with depths smaller than or equal to the closest depth for the previous level are rejected. At the first stage they are compared to zero. Among the not discarded fragments, the one with the smallest depth value is stored in each pixel. Since graphics hardware typically does not possess more than one depth buffer, Everitt [42] recommended a shadow mapping hardware function as a second, read-only depth buffer. Liu et al. [94] presented a faster algorithm which pells multiple layers of pixels per rendering pass. They sort and write multiple fragment colors and depths at each step via multiple-render-target (MRT) buffers.

2.3.3.3 Reduction-type operations

Reductions, map large textures of size $M \times M$ to a texture of size 1×1. Equivalently, they receive (several) input vector(s) of length $N \ll 1$, N a power of two, and output a scalar. Applications for such an operation are the computation of the maximum or the minimum of a given floating vector; vector norms, dot products, etc [60, 97]. The algorithm to compute the minimum of a set of $M \times M$ floating numbers recursively reduces the output size by computing the local maximum of each 2×2 group of elements and storing it in the corresponding output location [60]. The vector length decreases from $M \times M$ to $M/2 \times M/2$ at each step, until a 2×2 texture is reduce to the final 1×1 "scalar" texture. Consequently, the solution is obtained in a logarithmic number of iterations.

2.3.4 Visibility problems

In Computer Graphics, one of the most used rendering techniques is ray tracing. It is used to check whether a ray intersects the scene, and where the scene is intersected. Ray tracing problems can be easily, and quickly solved in the GPU for instances with the algorithm of Purcel et al. [122].

Koltum et al. [86] present an approximate algorithm which combines the hardware parallelism and a bounded dual ray space to compute the visible parts of a segment s' conforming the scene in the plane from a view segment s. The bounded dual space is represented by the framebuffer and the pixels are dual points. Each pixel represents a ray from the view segment s to s'. The pixels representing rays that intersect the other segments conforming the scene are marked in the framebuffer by considering one by one all the segments conforming the scene (See Figure 2.11). Finally, the algorithm checks

whether there is a pixel that has not been marked. Such a pixel shows that there exists a ray from s to s' joining a point of s' that is visible from s. Then, s' is assumed to be visible from s.

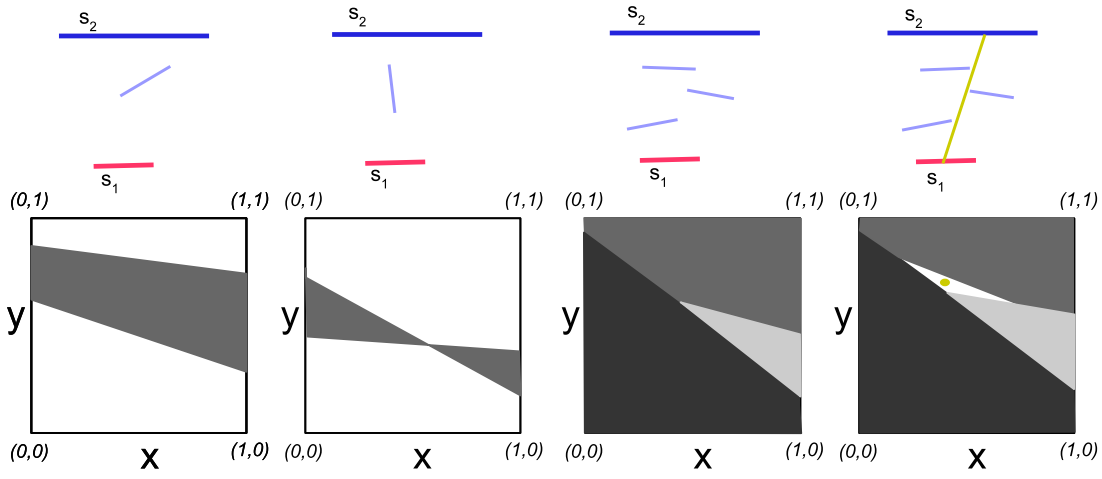


Figure 2.11: Four different scenes (top). Their dual ray space (bottom) where dark pixels represent rays from s_1 to s_2 intersecting another segment of the scene.

2.3.5 Proximity problems

Exist algorithms using graphics hardware can approximately solve proximity problems. By using GPU we can obtain any k-closest Voronoi diagram and some facility location problem such as the 1-center or 1-median can be solved.

Several algorithms to approximately compute Voronoi diagrams already exist [5, 76, 138, 142]. Amongst them, Agarwal et al. [5] provide an algorithm to compute a discrete generalized Voronoi diagram by rendering the distance functions of the generalized sites and obtaining their lower envelope by using the depth buffer [5]. In the same way, the furthest Voronoi diagram can be obtained by finding the upper envelope of the distance functions determining the fragment with bigger depth value for each pixel.

Fischer and Gotsman [45] use the depth-peeling technique to extract, in k passes, the top k levels of their arrangement of n distance functions of n point sites in the plane. Each ith-level corresponds to pixels painted in a maximum of n different colors whose depth value is the distance from the pixel to the ith nearest site. At the kth step, the algorithm obtains the *kth-nearest point diagram*, and the k-th order Voronoi diagram is obtained

by summing the colors of the ith-nearest diagrams, $i = 1 \dots k$, or overlaying them with transparency $1/k$. If the former strategy is used, colors have to be chosen in the way that all the sums give distinct colors. Coding theory studies this problem [21]. However, they observed that random 8-bit colors per site suffices to be able to distinguish between the cells. It is important to mention that Agarwal et al. [5] provide a way to compute the furthest Voronoi diagram of a set of generalized sites by rasterizing the distance functions and using the depth buffer to compute their upper envelope.

There is also an algorithm to approximately obtain the 1-center of a set of sites. Agarwal et al. [5] compute for each pixel $\sum d(p, q)$, encoding the distance function in the color component and blending the colors. Finally, they find the overall minimum.

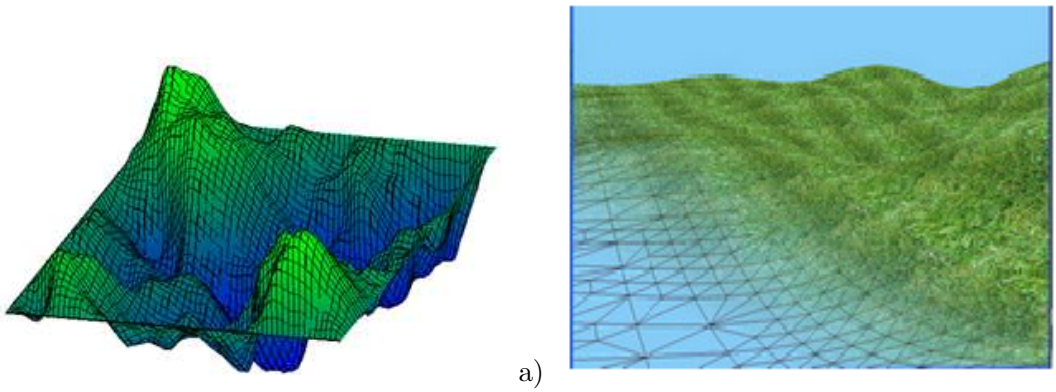
2.4 Terrain representation

Informally, a terrain is a surface that is intersected by any vertical line at most once. Consequently, a terrain is a surface in $\mathbb{R}^3$ that can be modelled as the graph of a bidimensional function defined over a simply-connected subset of the $xy-$plane, the *domain*. Thus a terrain is defined on a subdivision Σ of the domain D as a collection of planar regions $\mathcal{R} = \{t_1, \dots, t_n\}$ and a family Φ of continuous functions $z = \phi_i(x, y),\ i = 1 \dots n$, defined on t_i and such that, on the common boundary of two adjacent regions t_i and t_j, ϕ_i and ϕ_j take the same value. Consequently, a pair $(\mathcal{R}, \Phi)$ defines a terrain $\mathcal{T}$. The graph of each function ϕ_i is called *face* and is denoted f_i. The restriction of each function ϕ_i to an edge or vertex of Σ is called *edge* and *vertex* of the terrain, respectively.

Terrains are usually modelled as *Regular Square Grids* or *Polyhedral Terrain Models*. Regular Square Grid models have regular rectangular grids subdividing the domain and Bezier curves or Bezier patches as functions [44, 137] (Figure 2.12 a). They are usually used as smoothing techniques rather than to obtain accurate terrain representations. Polyhedral Terrain Models, PTM, have a domain subdivision consisting of a straight line plane graph and and linear interpolating functions. Consequently, the graph of a triangulated polyhedral terrain model consists of a network of polygonal planar faces (Figure 2.12 b)).

2.4.1 Triangulated Irregular Model

A special PTM is a Triangular Irregular Network (TIN) which is characterized by a triangular subdivision of the domain [14, 150, 127]. TIN models are used to deal with



a) b)

Figure 2.12: a) A terrain modelled with Bezier curves. b) A terrain modelled with a triangulated irregular network.

irregularly distributed data because they preserve the roughness of the terrains and may include special features such as points (peaks, pits, passes) or lines (ridges, valleys). These models have deserved interest from Computational Geometry specialists working on GIS problems and have been widely used in this field [14, 35, 150, 139]. A TIN modelling a set of points representing a terrain can be obtained by using: a Delaunay triangulation of the terrain vertices projected on the domain; a constrained Delaunay triangulation when some important features need to be preserved; etc. [35, 37, 66, 127, 139]

From now on $\mathcal{T}$ denotes a triangulated polyhedral terrain modelled by a TIN and D its triangulated domain.

2.4.2 Weighted terrains

Sometimes it is necessary to take into account the difference between a smooth terrain, a rocky terrain, a terrain consisting of different types of regions (e.g. forest, rocks, desert, lake, river), etc. With this in mind, a weight is associated to each face of the polyhedral surface [7, 8, 105, 143, 159], these special kinds of terrains are called *weighted terrains*. When we work with a weighted terrain each face $f_1, \ldots, f_n$ has a positive weight associated to it, $w_1, \ldots, w_n$, respectively. The weight associated to an edge, is the minimum of the weights of the two neighboring faces. Notice that non-weighted terrains can be seen as a special case of a weighted terrain considering all the weights equal to one.

2.4.3 Realistic terrains

Realistic terrains are considered in Computational Geometry to be able to obtain more accurate complexity analysis studies. If arbitrary TIN models are considered, there is a big gap between the worst case time complexity and the actual running time of the algorithms. Realistic input terrain models are considered to reduce this difference. A terrain is a realistic terrain if it fulfills the following characteristics [107] (See Figure 2.13):

1. the triangulation of the domain is a k-low-density triangulation,

2. the boundary of the domain is a rectangle with side lengths 1 and q.

3. the longest edge of the domain triangulation is at most d times as long as the shortest one.

where α, c, d are positive constants and planar triangulation is a *k-low-density triangulation* if for any square Q, the number of edges intersected by Q of length greater or equal to q is at most k, where q is the side length of Q. When working with distances on terrains, the following extra property has to be considered:

4. the dihedral angle of the supporting plane of any triangle in $\mathcal{T}$ with the xy-plane is at most β, where $\beta < \frac{\pi}{2}$ is some constant.

A property of these terrains is that any line traversing the terrain domain intersects at most $O(\sqrt{n})$ triangles.

2.4.4 Terrains as polyhedral surfaces

A terrain can be seen as a special case of a polyhedral surface. In fact, a triangulated terrain with n faces can be transformed to a polyhedral surface by adding $O(n)$ triangles. The obtained polyhedral surfaces is homeomorphic to a sphere and has the original terrain on top [10, 11] (Figure 2.13). This property is important because it ensures that algorithms for polyhedral surfaces can be used when working on terrains.

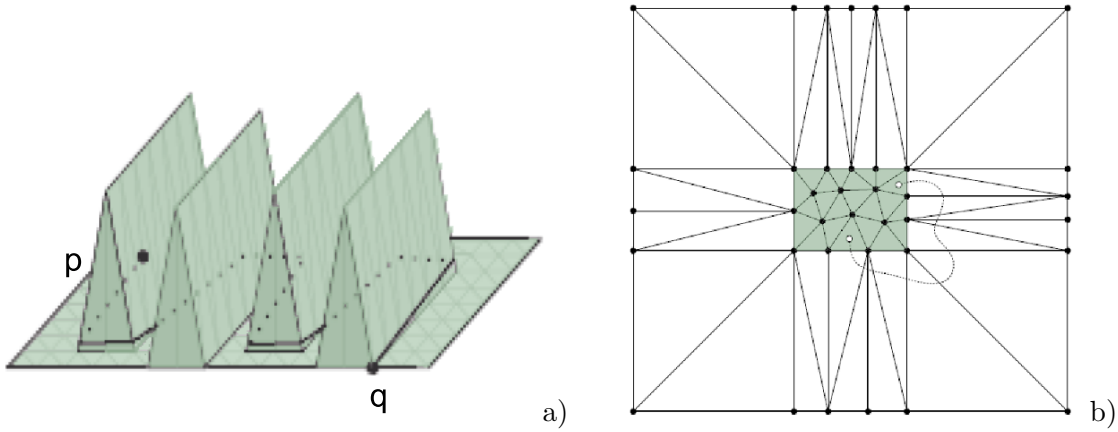


Figure 2.13: a) A realistic terrain fulfilling properties 1, 2 and 3. b) A polyhedron containing a terrain on its top.

2.5 Visibility on triangulated terrains

Visibility computations on terrain models have their main applications in Geographic Information Systems (GIS) [48, 53, 55, 57, 88, 108, 121]. They are useful for many applications related to terrain analysis such as site or path selection, path planning, mobile phone networks design, line-of-sight communication, orientation and navigation on terrains, environmental modelling, etc. Visibility problems, *viewshed analysis problems* in GIS context, have deserved interest from Computer Graphics, Computational Geometry and also specialists working on GIS problems. Existent studies dealing with visibility on terrains consider view points, view segments and view terrain faces. Some of them are designed to work with terrains, but those that can be used with general scenes in space are also presented.

First, some specific definitions related to visibility on terrains are given. Next some algorithms to solve the visibility problem considering first a view point and next a view segment are provided.

2.5.1 Basic definitions

A *view element* v is a generalized element (point, segment, polygonal chain, polygon, etc.) placed on or above the terrain, i.e. its projection onto the xy-plane is contained in domain D and any point $(x, y, z) \in v$ is such that $z \geq z'$ where (x, y, z') is the point of the terrain. A point P on the terrain is *visible* from a view point Q if the interior points of the line

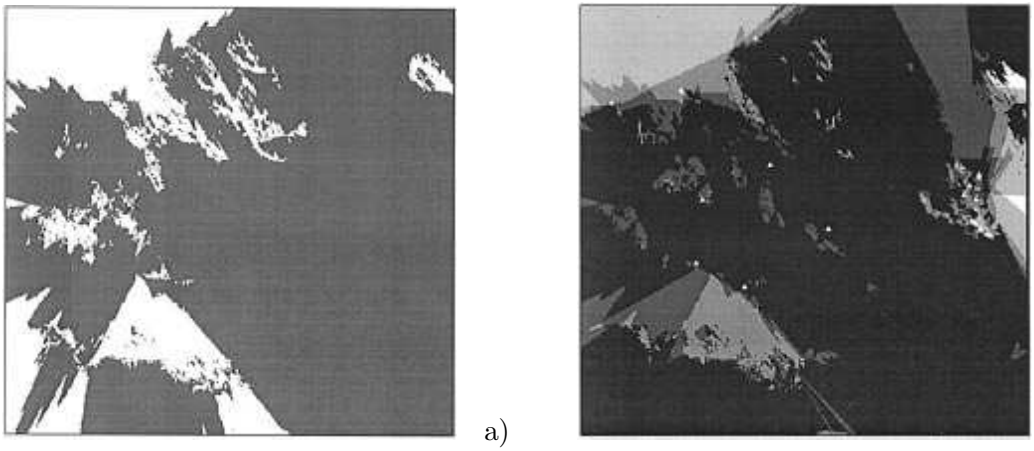
segment QP with endpoints Q and P lies above the terrain. When dealing with a view segment, view polygonal chain or view polygon v, it is said that a point P is *weakly visible* if P is visible at least from a point of v and it is said that it is *strongly visible* if P is visible from every single point of v. From now on the term visibility will refer to weak and strong visibility whenever it is not specified.

The *visibility map* of a view element v is the partition of the domain D into visible and not-visible maximal connected regions. A region of the domain is labelled as visible (not-visible) when all its corresponding points on the terrain are visible (not-visible) from v (see Figure 2.14).

Figure 2.14: In white the projection on the xy-plane of the visible points from a view point.

Multi-visibility maps related to a set $V = \{v_1, \ldots, v_i, \ldots, v_r\}$ of view points are obtained combining the visibility maps of the elements in V. A multi-visibility map is a subdivision of the domain D of the terrain into regions according to different criteria. A multi-visibility map can be addressed to answer different questions: which points of the terrain are visible from every single view element?, which points are visible from at least one view element?, from how many view elements is each point visible?, from which view elements is each point visible?, etc. Typical multi-visibility maps are: the *union*, the *intersection*, the *count* and the *overlay*. The union map subdivides the domain into maximal regions containing points that are visible from at least one view element. The intersection map yields to maximal regions of points that are visible from every view element. The count map subdivides the domain according to the cardinal of the set of view elements from which a region is visible. Finally, the overlay map is obtained superimposing the visibility maps of all view elements and labelling each region in the resulting partition of

the domain with the set of indexes of the view elements from which that region is visible (see Figure 2.15).



a) b)

Figure 2.15: a) Union map: Visible areas are painted in white and non visible areas in grey b) Counting map: lighter red areas are visible from larger numbers of view points, and white triangles show the position of the cairns. (Image taken from [**?**])

2.5.2 View point

Floriani et al. [54] present a survey of existent algorithms to compute the visible parts of a terrain from a view point V, the most important ones in the Computer Graphics area are summarized in Table 2.1. In GIS context visibility problems have been widely studied generally using Digital Elevation Models. Algorithms tend to provide discrete visibility maps, include uncertainty in the initial terrain model and study error and probability in the results [46, 48, 49, 57, 88, 121, 140]. Multi-visibility problems for a set of view points have also been studied [9, 58, 59, 83, 96, 116]. .

Next a brief overview of the most common techniques which use: a front-to-back order, lines of sight, concentric rings, a radial sweep and finally radial sectors is provided.

- *Front-to-back*: Terrain faces are processed in front-to-back order with respect to V. Face f is in front of face f' if both are intersected by a ray originated at V, and f is nearer from V than f'. This method works with sortable terrain models, where a front-to-back order exists. Delaunay TINs and TINs from regular grids are always sortable [52], and a general TIN becomes sortable by splitting some of its triangles [30].

Approach	**Authors**	**Input**	**Output**
Front-to back	De Floriani et al., 1989 [51]	TIN	Continuous Vis. Map
Front-to back	Lee 1991 [93]	TIN	Discrete Vis. Map
Line-of-sight	Shapira, 1990 [133]; Blellock 1990 [16]	RSG	Discrete Vis. Map
Line-of-sight parallel algor.	Mills, Fox and Heimback, 1992 [103]; Teng, De Menthon and Davis 1993 [146]	RSG	Inter-visibility Visibility Map
Line-of-sight	Sorensen and Lanter, 1993 [140]	RSG	Continuous Vis. Map
Sector-based	Stewart,1998 [141]	RSG; TIN*	Approx horizon
Concentric ring approx. algor.	Franklin and Ray, 1994 [57]	RSG	Discrete Visibility Map
Line-of-sight	Fisher, 1996 [47]	RSG	Extended viewshed
Radial sweep	Van Kreveld, 1996 [149]	RSG	Extended viewsheds
Line-of-sight parallel algor.	Rallings et al., 1998 [123]	RSG	Visibility counts

TIN*: only considering the TIN vertices

Table 2.1: Summary of visibility algorithms for a view-point

- *Line-of-sight*: This strategy takes into account a discrete set, $\mathcal{P}$, of points of $\mathcal{T}$. Lines-of-sight (rays) emanating from V and going to $P \in \mathcal{P}$ are intersected with the faces, in fact, usually the interior of the faces are ignored and only the edges are considered. Point P is visible if its line-of-sight does not intersect a face before reaching P.

- *Concentric rings*: This algorithm starts at the ring adjacent to V, the faces adjacent to V, and keeps on exploring all the faces computing the visibility of a point P in the i-th ring by using the previous ring information.

- *Radial sweep*: A line-of-sight rotating around V is considered. The line stops at certain events, points where visibility changes, and visibility information for all the faces intersected by the line is computed.

- *Sector-based*: A set of radial sectors is considered for every view point and each sector is processed individually. Thus a parallel implementation can be used with a

per-sector basis.

Some other algorithms that use hardware graphic to compute visibility from a view point [34, 65, 77, 158] can be found. Some of these algorithms consider a grid covering the space and for each grid cell compute and store the visible areas of the scene, consequently the storage cost is huge. However, the most general technique is named *Hidden Surface Removal* [68, 100]. This is used for visualizing $\mathbb{R}^3$ general scenes on the screen of a computer, and is a practical way to obtain a visibility map of a TIN [68]. The scene is projected onto a view-plane, represented by the screen, and only the visible areas are seen, thus, their projection on D is the visibility map (see Figure 2.16). Instead of rendering all the scene, they have some tricks to avoid rendering, or reject some geometry that can not be visible [56]. The most used ones are the following:

- *Back-face culling*: avoids rendering geometry that faces away from the viewer.
- *Viewing-frustum culling*: avoids rendering geometry placed outside the part of the scene projected on the screen, the viewing frustum.
- *Occlusion culling*: avoids rendering geometry that is occluded by some other parts of the scene (a specific method for terrains is given in [155]).
- *Visibility culling*: rejects the invisible geometry using (hopefully tight) estimations of the visible regions, it uses the previous ones.

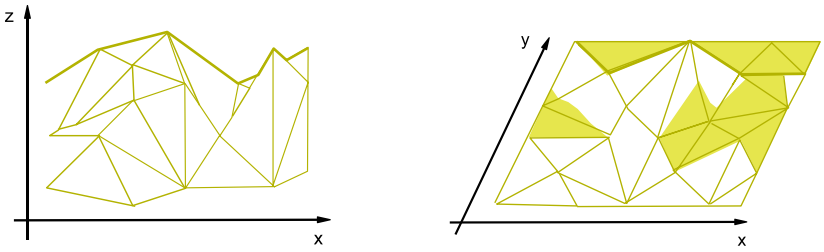


Figure 2.16: The visibility map and the visible image on the xy-plane and xz-plane, respectively. The not visible region of the domain is the green one.

2.5.3 View segment

Algorithms to compute visibility from a view segment, reduce the problem to compute visibility from a moving view point placed on the segment. A distinction is made between

algorithms that provide exact and approximate solutions.

Among the exact algorithms an explanation of the algorithm of De Berg et al. because it is the basis of our algorithm [15]. De Berg et al. [15] present an algorithm obtains exactly the visible parts of the scene from a moving view point V on s. They give a general algorithm for scenes in $\mathbb{R}^3$ which is particularized to work on terrains and finally on terrains and vertical view segments. They consider a view point V which moves along s and compute the critical points where topological change occurs. Only some points produce visible topological changes. When this happens, the visible scene is updated. To efficiently handle the updates they use some auxiliary information obtained in an initialization step where the initial visibility cycles are also computed with an algorithm provided by McKenna [100]. When general scenes in $\mathbb{R}^3$ are considered, critical points are obtained by using the transparent and opaque visibility cycles. For general scenes, the worst case complexity of the algorithm is $O(n^2 + l \log n)$ time and $O(n^2)$ space, where l is the number of transparent topological changes which is at most $n^3/3$. The computational complexity becomes $O(n^2 \log n + l)$ time when $l \notin O(n^2)$. For the special case of terrains, they use the fact that terrain edges are sortable: edge e_i is smaller than edge e_j if there exists a forward ray from V that intersects first e_i and next e_j. By using this property, critical points can be obtained with an algorithm that uses the first $i - th$ sorted edges conforming the horizon line. It reduces the computational complexity to $O((n+k)\lambda_3(n) \log n)$ time and $O(n\lambda_3(n))$ space, where k is the number of opaque topology changes and $\lambda_3(n) \in O(n\alpha(n))$, $\alpha(n)$ is the very slowly growing inver Ackerman function [71]. When s is a vertical view segment the number of possible visible topological changes and consequently the computational cost of the algorithm becomes $O(n\lambda_4(n) \log n)$ time and $O(\lambda_4(n))$ space ($\lambda_4(n) = \Theta(n2^{\alpha(n)})$) [2]. Finally there are already exist some algorithms to compute the visible parts of a terrain from a moving point of view following a not prefixed general trajectory [75, 157]. Notice that these algorithms compute the visible parts of the scene or a terrain from each point of s. However they do not obtain the visibility map.

Approximate solutions are provided by algorithms that use Computer Graphics techniques, such as the general Hidden Surface Removal algorithm [56]. Koltun et al. present an algorithm to compute the visibility of a scene with urban building [86] in an approximate fashion. These algorithms compute the visible parts from a moving point of view placed on the view segment. In GIS context, visibility from a segment is simulated as visibility from a set of points placed on the segment. Multi-visibility problems for a set of view points has been studied by several authors in this context [9, 58, 59, 83, 96, 116].

2.5.4 View polygon: a view terrain face

Concerning view region elements, Krevel et al. [108] provided an algorithm to compute strong inter-region visibility between two terrain regions in $O(n^{2+\varepsilon})$ worst case time complexity. Wang and Zen [152] present an algorithm to compute the weak visible parts of a general scene in $\mathbb{R}^3$ from a view triangle. The computational complexity of the algorithm is $O(n^8)$ time and $O(n^6)$ space, which is reduced to $O(n^6)$ time and $O(n^4)$ space algorithm for the special case of terrains. In the paper the following Lemma, which is proved using the fact that any vertical line intersects the terrain at most once, is stated.

Lemma 2.5.1 (Wang and Zen 2000) *Let us consider a terrain $\mathcal{T}$ and a face of $\mathcal{T}$ as a view polygon $\mathcal{P}$. The weakly visible parts of $\mathcal{T}$ from $\mathcal{P}$ are the weakly visible parts of $\mathcal{T}$ from $\partial\mathcal{P}$.*

Thus, if we consider a terrain face as a view polygon, $\mathcal{P}$, we have to study the visibility from the boundary segments of $\mathcal{P}$. By using this observation, the weakly visible parts of $\mathcal{T}$ from a face of the terrain, $\mathcal{P}$, can be determined by studying the boundary segments of $\mathcal{P}$ and using the algorithms to compute weak visibility for view segments.

There are also approximate algorithms in existence that use hardware graphic to compute visibility from view regions [31, 40, 129]. Some of these algorithms consider a grid covering the space and for each grid cell compute and store the visible areas of the scene, consequently the storage cost is huge.

2.5.5 Visibility maps complexity

The problem of computing the visibility map of a terrain from a view point using TIN models is studied in [15, 53, 55, 150]. To obtain a visibility map we need the visible parts of the scene, which are computed by using the algorithms given in Section 2.5.

Combinatorial results related to the complexity of the visibility map are given in [106, 107, 152]. Wang and Zhu proved [152] that given a general scene in $\mathbb{R}^3$, the complexity of the weakly visible regions of a triangle from a view triangle is $\Theta(n^6)$, which is reduced to $O(n^4)$ for the special case of a terrain. Moet et al. [106] proved that the complexity of the strong visibility map, the visibility map when strong visibility is considered, for a view segment is $\Omega(n^2)$ and for a view triangle is $O(n^2 \log n)$. They also proved that the complexity of the weak visibility map of a view segment is $\Omega(n^4)$ and for a view triangle

is $O(n^5)$. Finally, Moet et al. [107] proved that the visibility map of a view point when considering a realistic terrain has complexity $\Omega(n\sqrt(n))$.

2.6 Shortest paths on triangulated polyhedral surfaces

Computing shortest paths on polyhedral surfaces, which are assumed to be triangulated, is a fundamental problem in Computational Geometry with important applications in geographical information systems (GIS), Computer Graphics and robotics. Consequently, the shortest path problem in polyhedral surfaces when a point source s is given have been widely studied considering both non-weighted and weighted surfaces. In Section 2.4.4 it is explained that a terrain can be seen as a polyhedral surface by using an algorithm of Aronov et al. [10]. Aronov et al. also prove that the shortest path between two points on the faces of the original terrain will not leave the original terrain. Thus, the shortest path problem on terrains can be solved by using more general algorithms for computing shortest paths on polyhedral surfaces.

We can find two different types of solutions depending on whether: a) we are given a target point t and we are interested in finding the shortest path from s to t; b) we are not given any specific target and we are interested in solving the *single source to any target problem*. Shortest paths on a polyhedral surface solving the single source to any target problem define a distance function. For a given point source s, the distance from s to any point p of the polyhedral surface, $D_s(p)$, is given by the length of the shortest path.

In the next sections we give some basic definitions, characterize the shortest paths in weighted and non-weighted settings, and provide an overview of the most important algorithms to compute shortest path.

2.6.1 Shortest paths

Algorithms for obtaining shortest path on (non-weighted) polyhedral surface have been widely studied [28, 79, 80, 95, 104, 131, 144]. These algorithms can be subdivided into two categories, those providing exact solutions and those giving approximate solutions. We start with some basic definitions and properties of shortest paths on non-weighted polyhedrons, and finally overview the existent exact and approximate algorithms for obtaining shortest paths to point sources.

2.6.1.1 Basic definitions

Let $\mathcal{P}$ be a possibly non-convex polyhedral surface represented as a mesh consisting of n triangular faces $f_1, \ldots, f_n$.

The *length* of a path π on $\mathcal{P}$ is defined as $|\pi| = \sum_{i=1}^{n} |\pi_i|$, where $|\pi_i|$ denotes the Euclidean length of the path lying inside the face f_i. The *shortest path* between two points p and q on $\mathcal{P}$ is the path between them of least length and is denoted $\pi(p, q)$. When a shortest path crosses an edge at point p, it is said that the shortest path has a *bending point* on p, or, equivalently, that p is a bending point of the shortest path.

For a given point source s, the *distance from s to a point p* of the polyhedral surface $\mathcal{P}$, $D_s(p)$, is given by the length of the non-weighted shortest path from p to s, $\pi(s, p)$.

A path that is locally a shortest path is called a *geodesic*. All the shortest paths are geodesics, but not all the geodesics are shortest path. Geodesic properties have been the basis of different algorithms to compute shortest paths [104].

2.6.1.2 Geodesic properties

Mitchell et al. [104] studied the geodesics on non-convex triangulated surfaces. After the study they highlighted the three following properties, which are represented in Figure 2.17, as the characterization of this type of geodesic:

Property 2.6.1 *In the interior of a triangle a geodesic is a straight line.*

Property 2.6.2 *When crossing an edge a geodesic corresponds to a straight line if the two adjacent triangles are unfolded or placed into a common plane.*

Property 2.6.3 *A geodesic can go through a vertex if and only if the total angle of the vertex is at least 2π.*

Following the previous terminology, the operation of obtaining the plane containing adjacent triangles is called the *planar unfolding*. A vertex whose total angle α is at least 2π i called a *saddle* vertex, and a saddle vertex is *hyperbolic* if $\alpha > 2\pi$ and *parabolic* if $\alpha = 2\pi$. Apart from the previous three properties any geodesic fulfills the following Property.

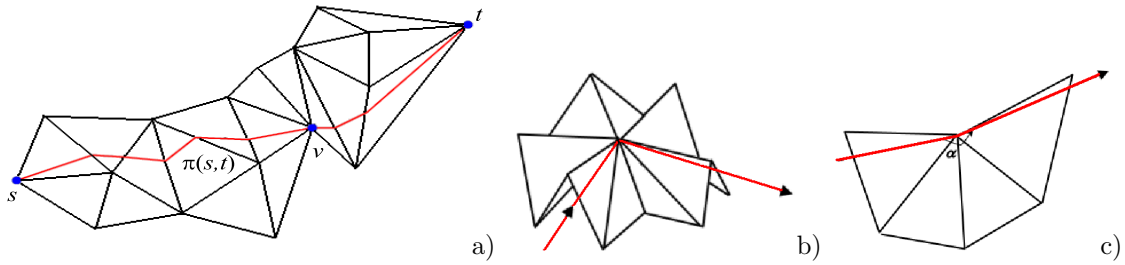


Figure 2.17: A planar unfolding containing $\pi(s,t)$. b) A geodesic going through a saddle vertex. c) The angle defined by the geodesic shown in b) at the saddle vertex.

Property 2.6.4 *Two shortest paths originating at the same source point cannot intersect in the interior of any triangle.*

Concerning the complexity of non-weighted shortest paths, Mitchell et al. [104] proved that the shortest path on a non-convex polyhedral surface with $O(n)$ triangles can traverse up to $\Omega(n)$ faces. Moet et al. [107] proved that when considering realistic terrains this complexity decreases to $\Omega(\sqrt{n})$.

2.6.1.3 Exact algorithms

Mitchell et al. [104] present an algorithm for solving the single point source shortest path problem by using a "continuous Dijkstra" method which propagates distances from the source to the rest of $\mathcal{P}$. The algorithm constructs, in $O(n^2 \log n)$ time, a data structure called the *shortest path map* that implicitly encodes the shortest paths from a given source point to the rest of points of $\mathcal{P}$. The structure allows for single-source shortest path queries, providing the length of the path and the actual path in $O(\log n)$ and $O(\log n + \overline{n})$ time respectively, where $\overline{n}$ is the number of mesh edges crossed by the path. Different improvements of this algorithm have been proposed. Surazhsky et al. [144] described a simple way to implement the algorithm and showed that it run much faster on most polyhedral surfaces than the $O(n^2 \log n)$ theoretical worst case time. Recently, Liu et al. [95] gave some important implementation details of the algorithm of Surazhsky et al. to properly obtain the desired result.

Chen and Han [28], using a rather different approach, improved the $O(n^2 \log n)$ complexity to an $O(n^2)$ time algorithm. Their algorithm constructs a search tree and works by unfolding the facets of the polyhedral surface. The algorithm also answers single-source shortest path queries in $O(\log n + \overline{n})$ time. Kaneva and O'Rourke [79] implemented Chen

and Han's algorithm and reported that the implementation is difficult for non-convex polyhedral surfaces and that memory size is a limiting factor of the algorithm.

Kapoor [80] presented an algorithm following the "continuous Dijkstra" paradigm that computes a shortest path from a source point to a target point in $O(n \log^2 n)$ time. However, it is an obscure algorithm from which no extended version or implementation is known. Recently, Schreiber and Sharir [131] provided an $O(n \log n)$ worst case time algorithm for convex polyhedral surfaces.

Table 2.2: Results on exact Shortest Paths on Polyhedral Surfaces

Polyhedral Surface	Cost Metric	Approx. Ratio	Time Complexity	Reference	
Convex	Euclidean	1	$O(n^3 \log n)$	Sharir and Schoor 1986	[136]
Convex	Euclidean	1	$O(n \log n)$	1986	Schreiber and Sharir [131]
Non-convex	Euclidean	1	$O(n^2 \log n)$	Mitchell et al. 1987	[104]
Non-convex	Euclidean	1	$O(n^2)$	Chen and Han 1996	[28]
Non-convex	Euclidean	1	$O(n \log^2 n)$	Kapoor 1999	[81]

2.6.1.4 Approximate algorithms

Surazhsky et al. [144] propose an approximate version of their exact algorithm. They use the continuous Dijkstra and proceed as in their exact method but virtually moving the sources to simplify the algorithm. Many other approximate algorithms can be used to obtain approximate shortest paths on polyhedral surfaces by considering non-weighted shortest path as a particular case of weighted shortest paths. These algorithms are explained in next section where weighted shortest paths are considered.

2.6.2 Weighted shortest paths

Weighted terrains are often used, and consequently it is necessary to compute shortest paths on weighted terrains. The problem is solved on weighted triangulated polyhedral surfaces which is a general case of weighted polyhedral terrains.

Table 2.3: Results on Shortest Paths on Polyhedral Surfaces

Polyhedral Surface	Cost Metric	Approx. Ratio	Time Complexity	Reference	
Convex	Euclidean	2	$O(n)$	Herhberger et al. 1995	[73]
Convex	Euclidean	$1+\varepsilon$	$O(n \log \frac{1}{\varepsilon} + \frac{1}{\varepsilon^3})$	Agarwal et al. 1997	[4]
Convex	Euclidean	$1+\varepsilon$	$O(n + \frac{\log n}{\varepsilon^{1.5}} + \frac{1}{\varepsilon^3})$	Har-Peled 1999	[69]
Convex	Euclidean	$1+\varepsilon$	$O(\frac{n}{\sqrt{\varepsilon}} + \frac{1}{\varepsilon^4})$	Agarwal et al. 2002	[6]
Convex	Euclidean	$1+\varepsilon$	$O(\frac{\sqrt{n}}{\varepsilon^{1.25}} + f(\varepsilon^{-1.25}))^{(*)}$	Chazelle et al. 2003	[27]
Non-convex	Euclidean	$1+\varepsilon$	$O(n^2 \log n + \frac{n}{\varepsilon} \log \frac{1}{\varepsilon} \log \frac{n}{\varepsilon})$	Har-Peled 1999	[70]
Non-convex	Euclidean	$7(1+\varepsilon)$	$O(n^{5/3} \log^{5/3} n)$	Vardarajan et al. 2000	[151]
Non-convex	Euclidean	$15(1+\varepsilon)$	$O(n^{8/5} \log^{8/5} n)$	Vardarajan et al. 2000	[151]

* This uses the Las Vegas algorithm, which approximates the distance between two points on the surface of a convex polytope, but not the path itself. The term $f(\varepsilon^{-1.25})$ represents the time required to compute an exact shortest path on the surface of a convex polytope consisting of $\varepsilon^{-1.25}$ triangular faces.

2.6.2.1 Basic definitions

Let $\mathcal{P}$ be a possibly non-convex weighted polyhedral surface represented as a mesh consisting of n triangular faces $f_1, \ldots, f_n$ with associated positive weights $w_1, \ldots, w_n$. The weight associated with an edge, is the minimum of the weights of the two neighboring faces.

Given a polyhedral surface $\mathcal{P}$, we define the *cost* of a path Π on $\mathcal{P}$ as $\|\Pi\| = \sum_{i=1}^{n} w_i|\pi_i|$, where $|\pi_i|$ denotes the Euclidean length of the path Π lying inside the face f_i. Given two points p and q on $\mathcal{P}$, the path between them of least cost is called the *weighted shortest path* and denoted $\Pi(p, q)$.

Let ε be a parameter in $(0, 1)$ a $(1+\varepsilon)$ *approximation* of a weighted shortest path between points p and q is a path between p and q whose cost is at most $(1+\varepsilon)$ times the cost $\Pi_{p,q}$.

2.6.2.2 Weighted geodesic properties

Mitchell and Papadimitriou [105] studied the properties of geodesics on weighted surfaces, and provided a characterization. According to Mitchell and Papadimitriou geodesics on weighted surfaces fulfill the following properties:

Property 2.6.5 *Two shortest paths originating at the same source point cannot intersect*

in the interior of any triangle.

Property 2.6.6 *In the interior of a face a geodesic defines a set of line segments which can contain from* 0 *to up to* $O(n)$ *segments.*

Property 2.6.7 *Geodesics cross and go along edges according to Snell's law.*

Snell's law: Snell's law states that the following equality is fulfilled:

$$w' \, sin\alpha = w \, sin\alpha'.$$

Where s and s' are two adjacent segments defining a geodesic on faces f and f', respectively; p is the common endpoint of s and s'; e is the edge containing p; w is the weight of f; w' is the weight of f'; α is the *in-angle* in p which is the counterclockwise angle between s and the normal to e; and α' is the out-angle in p defined as the counterclockwise angle between s' and the normal to e (see Figure 2.18).

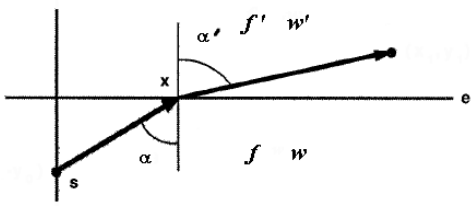


Figure 2.18: The in-angle and out-angle of a geodesic.

According to the previous properties, weighted shortest paths consist of a sequence of *edge-using* and *face-crossing* segments with endpoints, the *bending points*, on the polyhedral edges. Edge-using segments are edge subsegments and face-crossing segments join points on different edges of the same face.

Concerning the complexity of weighted shortest paths, Mitchell and Papadimitriou [105] proved that the shortest path on a non-convex weighted polyhedral surface with $O(n)$ triangles can cross up to $\Omega(n^2)$ faces.

2.6.2.3 Weighted shortest path computation

To the best of our knowledge, no exact algorithm for solving the weighted shortest path problem exists. The vast majority of algorithms, which are summarized in Table 2.4, for computing approximate shortest paths employ a discretization method and are designed to work with triangulated polyhedral surfaces homeomorphic to a sphere (convex polyhedra,

Table 2.4: Results on Shortest Paths on Polyhedral Surfaces

Polyhedral Surface	Cost Metric	Approx. Ratio	Time Complexity	Reference	
Non-convex	Weighted	Additive	$O(n^3 \log n)$	Lanthier et al. 2001	[89]
Non-convex	Weighted	$1+\varepsilon$	$O(n^8 \log(\frac{n}{\varepsilon}))$	Mitchell et al. 1991	[105]
Non-convex	Weighted	$1+\varepsilon$	$O(\frac{n}{\varepsilon^2} \log n \log \frac{1}{\varepsilon})$	Aleksandrov et al. 1998	[1]
Non-convex	Weighted	$1+\varepsilon$	$O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon}(\frac{1}{\sqrt{\varepsilon}} + \log n))$	Aleksandrov et al. 2000	[7]
Non-convex	Weighted	$1+\varepsilon$	$O(\frac{n}{\varepsilon} \log \frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$	Reif and Sun 2000	[124]
Non-convex	Weighted	$1+\varepsilon$	$O(\frac{n}{\sqrt{\varepsilon}} \log \frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$	Aleksandrov et al. 2005	[8]

or possible non-convex hole-free polyhedra) except for that given by Alexandrov et al. [8] which works with polyhedral surfaces not necessarily homeomorphic to a sphere.

In 1991 Mitchell and Papadimitriou [105] gave the characterization of the shortest paths on weighted surfaces and presented an algorithm that uses the continuous Dijkstra method to provide an $(1+\varepsilon)$ approximate shortest path, which runs in $O(n^8 \log n/\varepsilon)$ time and $O(n^4)$ space. It is the only algorithm that computes a $(1+\varepsilon)$ approximation of the shortest paths without discretizing the polyhedral surface.

Most algorithms discretize the polyhedral surface and reduce the problem to the determination of shortest paths on weighted graphs. Shortest paths on the graph are typically found by using the *Dijkstra algorithm* [1, 7, 8], which can be accelerated by using the geodesic properties presented in section 2.6. The discretization scheme places Steiner points on the triangle edges or on the bisectors of the triangle vertices. Between the former, the best algorithm was proposed by Aleksandrov et al. [7]. This follows a logarithmic discretization scheme and has time complexity $O(n/\varepsilon \ \log 1/\varepsilon \ (1/\sqrt{\varepsilon} + \log n))$. An algorithm of $O(n/\sqrt{\varepsilon} \ \log n/\varepsilon \ \log 1/\varepsilon)$ time complexity that uses the latter approach is presented again by Aleksandrov et al. [8].

Recently, an alternative strategy to Dijkstra algorithm called *Bushwhack* strategy has been used. It was proposed by Sun and Reif [143] and uses the property that two shortest paths do not intersect in a triangle. Considering an improved version of the discretization scheme introduced by Aleksandrov et al. [7] they obtained an approximate shortest path in $O(n/\varepsilon \ \log 1/\varepsilon \ \log n/\varepsilon)$ time. In the next sections, we explain how the bushwack strategy works.

The previous algorithms compute $(1+\varepsilon)$-approximate shortest paths. However, there are several approximate algorithms [82, 84, 90, 98, 109, 114, 144] in existence whose output

is a path which is supposed to be a shortest path, without having a bound on the committed error. These algorithms usually discretize the domain by considering only the polyhedron vertices. Some of them use a hierarchical representation of the triangulation which contains several triangulations of different levels of detail. The algorithms start by computing an initial shortest path in a simpler graph which is progressively improved by using a selective refinement of the triangulation considering only the triangles the path intersects. Some others use the *Fast Marching Method* [84, 98, 114]. They compute approximate distances to the vertices of the triangulation by propagating a wavefront using a typical discrete Dijkstra. When the distance to a vertex is to be recomputed, they find the direction of the shortest path using a gradient term obtained solving an equation which depends on the method.

2.6.2.4 Bushwack strategy

Sun and Reif [143] proposed the Bushwhack strategy to compute distances from a vertex source, s_v, on the graph nodes using the fact that no shortest path intersects in the interior of a face. The basic idea of this strategy is to track and keep together groups of shortest paths by partitioning each face edge into a set of (discrete) intervals, containing nodes, so that all shortest paths that cross an interval have the same structure. In the initialization step, Bushwhack creates discrete intervals, encoding the distance function D_{v_s} of the vertex source v_s, on the edges of the triangles containing v_s. Shortest paths are propagated across mesh triangles in a lazy and best-first propagation scheme. When a node v on an edge e is first visited, an interval I is created. An interval is denoted $I_{v,e,e'}$ when it contains nodes of edge e', where e' is opposite to v when v is a vertex or is adjacent to e otherwise (See Figure 2.19). Interval $I_{v,e,e'}$ contains those contiguous nodes of e' whose shortest path from v_s to $v' \in I_{v,e,e'}$ may use node v before arriving at v' via an edge-using or face-crossing segment contained on the face determined by e' and v. Node v is also called the *virtual source* of interval $I_{v,e,e'}$ and denoted I^s. Such intervals do not overlap and on average each interval will contain $O(1)$ Steiner points. The Bushwhack strategy complexity is of $O(mn \log mn)$ time and $O(mn)$ space when a graph with m nodes per edge on a polyhedral surface of n faces is used.

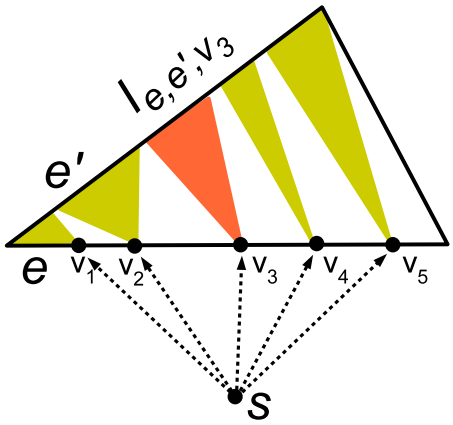


Figure 2.19: A point source s with approximate shortest paths to vertices $v_1, \ldots, v_5$ and the interval $I_{v_3,e,e'}$ associated to v_3.

2.6.3 Voronoi diagrams on triangulated surfaces

Several studies related to Voronoi diagrams on non-weighted polyhedral surfaces considering a set S of r point sites and the shortest path distance already exist. Before studying the closest [11, 104, 110] and the furthest [11, 126] Voronoi diagrams on a polyhedral terrain, some properties of the equidistant points between two sites s and $t \in S$ when considering shortest paths on polyhedral surfaces are given. The terminology is the one introduced by Aronov et al. [11]. Results related to realistic terrains are provided by Moet et al. [107].

2.6.3.1 Basic definitions and properties

The *bisector* between point site s_i and $s_j \in S$, β_{ij}, is the set of points p on the terrain equidistant from s_i and s_j and they have the following properties [110] .

Property 2.6.8 *A bisector consists of $O(n^2)$ straight-line segments and hyperbolic arcs for general terrains.*

Property 2.6.9 *Two bisectors, β_{ij} and β_{ik}, intersect at most twice, where s_i, s_j, $s_k \in S$ and are different.*

Property 2.6.10 *If two sites of S are located at the same distance to a vertex of the polyhedral surface $\mathcal{P}$, then two-dimensional bisectors exist.*

It is assumed that the previous degeneracy does not happen.

An *edge* of the Voronoi diagram is the interior of the boundary between two adjacent regions; thus, each edge lies on a bisector.

A *vertex* of the Voronoi diagram is the non-empty intersection of the closures of three or more regions of the Voronoi diagram. It is assumed that all vertices have degree three; otherwise, a degeneracy is present.

A *breakpoint* is an intersection point between two adjacent segments or arcs conforming a Voronoi edge, or a point where a bisector crosses an edge of $\mathcal{P}$. These points are named breakpoints as opposed to the vertices of the diagram.

The *total complexity* of a Voronoi diagram is the sum of the number of vertices and breakpoints in the diagram.

2.6.3.2 Closest Voronoi diagram

Some existent exact algorithms for computing shortest paths from a fixed point s can be generalized to the multiple source case when a set of point sites S of r sites is given. When the generalization is possible, the algorithm computes for any point $p \in \mathcal{P}$ the shortest path that joins the closest site of S to p and p, and its corresponding distance. It yields the construction of the Voronoi diagram. The most important algorithm to obtain the closest site Voronoi diagram of a set of point sites on a polyhedral surface is the one proposed by Mitchell et al. [104] by using the implementation details given by Surazhsky et al. [144]. The algorithm proposed by Mitchell et al. [104] to obtain the closest Voronoi diagram requires $O(\tilde{r}^2)$ space and $O(\tilde{r}^2 \log \tilde{r})$ time, in the worst case, where $\tilde{r}$ is the maximum between n and r.

Aronov et al. [11] presented a study of the Closest Voronoi diagrams properties, some of them were previously given by Mount et al. in [70, 104]. Among them the following ones are highlighted.

Lemma 2.6.1 (Aronov et al. 2003) *Voronoi regions are path-connected.*

Theorem 2.6.1 (Aronov et al. 2003) *The total complexity of the closest Voronoi diagram is, in worst case,* $\Omega(\tilde{r}^2)$ *with* $\tilde{r}$ *is the maximum of* r *and* n

Closest Voronoi diagrams on realistic terrains

When considering realistic terrains some bound can be reduced. Moet et al. [107] present some interesting results that are presented next. To obtain some of these results they introduce the concept of a trace, there are three different types of traces. The first one contains points that have two shortest paths having the same edge sequences except for edges incident to v from source s to them, it is the *trace of a vertex* v with total angle $< 2\pi$. The second one is the maximal connected set of points on the boundary of the geodesic region of vertex v with total angle $\geq 2\pi$. The *geodesic region* of a vertex v contains the points of the terrain whose shortest paths from to s pass through v as the last vertex. The last one is the *junction trace* of a point q, where q is a point where two traces end, is the set of points with two different shortest paths from s, and such that the edge sequences of these two shortest paths are the same starting at one edge of the triangle that contains q. Traces do no intersect and no trace intersects a shortest path from source s.

Some of the special results for the case of realistic terrains are the ones presented next:

- A shortest path between two points intersects $O(\sqrt{n})$ triangles.
- A bisector has $O(n\sqrt{n})$ and $\Omega(n)$ breakpoints instead of $O(n^2)$.
- Any trace has complexity $O(\sqrt{n})$
- There are at most $O(n+m)$ traces in the closest Voronoi diagram.
- The complexity of the closest Voronoi diagram is $O((n+r)\sqrt{n})$ and $\Omega(n+r\sqrt{n}))$.

2.6.3.3 Furthest Voronoi diagram on terrains

As far as we know there are no studies related to the furthest site Voronoi diagrams on polyhedral surfaces. However, we can find some studies on the furthest site Voronoi diagrams on polyhedral terrains [11, 126] can be found. Aronov et al. [11] studied and presented an algorithm for obtaining the furthest site Voronoi diagram on a polyhedral surface, $\mathcal{P}$, when a set S of point sites is given. It uses a divide and conquer technique on S. They stop subdividing S when $|S| = 2$, when S has only two sites, the furthest and closest Voronoi diagrams have the same regions and only differ in the site each region is associated to. The closest Voronoi diagram of the two sites is computed by using the Mitchell et al. [104] technique, and by using a merging step, the furthest site Voronoi

diagram of the initial set S is obtained. Their algorithm provides the furthest Voronoi diagram in $O(r\,n^2 \log^2 r \log n)$ expected time.

Apart from the algorithm to obtain the furthest Voronoi diagram, Aronov et al. [11] showed some interesting results. The following ones are highlighted:

Lemma 2.6.2 (Aronov et al. 2003) *The furthest site Voronoi diagram of a set S with r sites has $O(r)$ path connected cells, $O(r)$ vertices, and $O(r)$ edges.*

Theorem 2.6.2 (Aronov et al. 2003) *The maximum total complexity of the furthest site Voronoi diagram S is $\Theta(r\,n^2)$.*

In the paper they provide an intermediate result which is used to prove the complexity of the furthest Voronoi diagram that we will also use:

Lemma 2.6.3 (Aronov et al. 2003) *Let B be a set of pseudocircles on the surface of a simple polyhedron $\mathcal{P}$. If the common interior of the pseudocircles in B is non-empty, then their common exterior is path-connected.*

2.6.4 Facility location problems on a triangulated terrain

The 1-Center problem has also been solved on a triangulated polyhedral terrain considering the shortest path distance from a set of r point sites. The 1-Center is located at a vertex or on an edge of the furthest Voronoi diagram. By using this property, Aronov et al. [11] give an algorithm for locating the 1-Center of the set of r point sites on a triangulated terrain in $O(rn^2)$ time.

Lanthier et al. [91] to the best of our knowledge, provide the only algorithm available to obtain an approximate 1-Center of r point sites on the surface of a terrain with n triangular faces. The theoretical worst case running time of the algorithm is $O(r\,n^2 \log rn + rn^3)$ for non-weighted terrains, and $O(r\,n^3 \log rn + rn^5)$ for weighted ones. However, for typical terrain data, the expected running time is $O(r\,n \log rn)$ for both weighted and non-weighted terrains.

Chapter 3

Multi-visibility on terrains

In this chapter the problem of approximately reconstructing an unknown multi-visibility map for a terrain modelled by a TIN with respect to an heterogeneous set of view elements containing points, segments, polygonal chains and polygons is addressed. An algorithm whose input is a terrain modelled by a TIN, a set V of view elements and a specified visibility criterion, weak or strong, for each view element in V is presented. The output is the representation of the desired approximated multi-visibility map obtained by using an algorithm for approximately reconstructing unknown planar subdivisions (Section 2.2.7). After obtaining the multi-visibility map, point and polygonal region multi-visibility queries can be answered in an approximate fashion.

In order to obtain the approximated multi-visibility map, visibility information is not computed on all the points of the domain, but on a set of line probes contained on it. Line probes are intersected with the triangulation of the terrain domain yielding a set of segments that are lifted onto terrain faces. The visibility of these segments is computed, after decomposing the view elements into view segments (Section 3.1), by using the segment-segment visibility algorithm (Section 3.3). With this information the desired multi-visibility map is obtained (Section 3.3.5). The algorithm can be restructured in such a way that, after a preprocessing stage the algorithm provides any multi-visibility map we are interested in (Section 3.5).

3.1 Basic properties

Before describing the method, some definitions and properties that are used to facilitate the visibility computation process are presented. In fact, these allow us to decompose any view element to a set of view segments.

A *view element* (point, segment, polygonal chain, polygon, etc.) v is an element placed on or above the terrain, i.e. its projection onto the xy-plane is contained in domain D and any point $(x, y, z) \in v$ is such that $z \geq z'$ where (x, y, z') is the point of the terrain. A point p on the terrain is *visible* from a view point q if the interior points of the line segment $\overline{qp}$ with endpoints q and p lie above the terrain.

When dealing with a view segment, view polygonal chain or view polygon v, we can consider either *weak visibility* (point p is visible when it is visible from at least a point of v) or *strong visibility* (point p is visible when it is visible from every single point of v). From now on the term visibility will refer to weak and strong visibility whenever it is not specified.

Lemma 3.1.1 *A view polygon P can be represented as the polygonal chain conforming its boundary (∂P).*

Proof. We first prove the result for weak visibility and then for strong visibility.

Let $p \in \mathcal{T}$ be a weakly visible point from P. Then, there is a point $u \in P$ where the segment $\overline{up}$ has all its points above or on $\mathcal{T}$. Let us consider π the vertical plane through $\overline{up}$, and u' the highest intersection point between π and P. Point u' is necessarily in ∂P, and p is also visible from u'. Consequently if p is visible from a point of P, it is also visible from a point of ∂P.

Now, let $p \in \mathcal{T}$ be a non strongly visible point from P, then, there is a point $u \in P$ where the segment $\overline{up}$ has some points below $\mathcal{T}$. Let π be the vertical plane through $\overline{up}$ and $P_\pi = \pi \cap P$. If P_π is a point, it is in ∂P, and p not strongly visible from ∂P. If P_π is a segment, let u' be the lowest endpoint of P_π (see Figure 3.1). Then, segment $\overline{u'p}$ contains, necessarily, points below $\mathcal{T}$, and p is not visible from $u' \in \partial P$.

Consequently p is weakly or strongly visible from P, if and only if it is weakly or strongly visible from ∂P, respectively. □

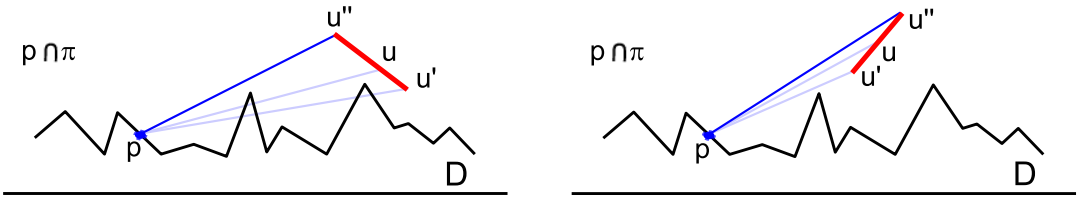


Figure 3.1: The section of a terrain given by the vertical plane delimited by point p and a point of the view polygon P.

Lemma 3.1.2 *A view polygonal chain can be represented as a set of view segments conforming a unique view element.*

Proof. A point is weakly visible from a view polygonal chain if it is weakly visible from at least one of its segments and it is strongly visible if it is strongly visible from every segment. Then, by computing the visibility information of each segment and adequately combining it, we are able to work with view polygonal chains as sets of view segments. □

A view point can be considered as a degenerate segment. Lemma 3.1.1 proved that the visibility from a view polygon is given by the visibility of the polygonal line conforming its boundary. Lemma 3.1.2 shows that the visibility from a polygonal line can be obtained considering the segments conforming it and combining, adequately, their visibility information. Thus we can state the following proposition.

Proposition 3.1.1 *Visibility from a generalized view element can be computed by appropriately using an algorithm to compute visibility from a view segment.* □

Consequently, to work with points, segments, polygons and polygonal lines it suffices to design an algorithm to compute visibility from view segments.

3.2 Algorithm overview

The method we present uses an approach of Coll et al. [33] that reconstructs an approximation of an unknown planar subdivision with exact visibility information gathered from linear probes of the subdivision (Section 2.2.7).

The input of the algorithm is a terrain $\mathcal{T}$ with domain $\mathcal{D}$ modelled by a TIN with n triangular faces, a set V of view elements, a specified visibility criterion, weak or strong, for each view element in V and the kind of multi-visibility map we are interested in. The output is the structure of the desired multi-visibility map stored in a DCEL (Section 2.2.1).

Visibility information is exactly computed in a quite complex algorithm that uses opaque and transparent visibility cycles (Section 2.2.8). We consider several lines, *line probes*, randomly chosen on the terrain domain. Each line is intersected with the triangulation of the terrain domain yielding to a set of segments that are lifted onto terrain faces. The visibility of these segments is computed after decomposing the view elements into view segments (Section 3.1), using the segment-segment visibility algorithm (Section 3.3). The desired multi-visibility information of the first line probe, obtained by merging the visibility information of all the view elements (Section 3.3.5), is used to obtain the first approximated multi-visibility map. After obtaining the multi-visibility of a new line probe, the approximated multi-visibility map is updated and its error is recomputed (Section 2.2.7). The process ends when either the committed error in the multi-visibility map is smaller than a threshold or a preestablished maximal number of line probes have been studied.

When the algorithm finishes, we have an approximated multi-visibility map represented in a DCEL (Section 2.2.1). Consequently, some queries such as point or region queries can be answered by using standard methods for planar subdivisions. A scheme of the process is given in Figure 3.2.

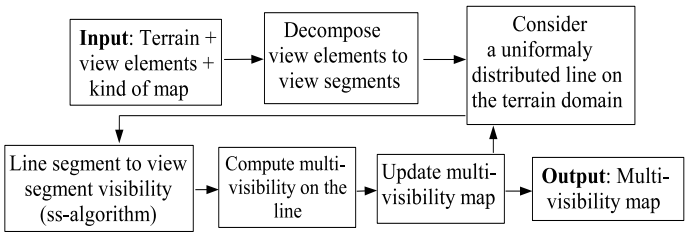


Figure 3.2: A schematic process overview of our algorithm

3.3 Visibility computation

In this section how the visible parts of a segment s, placed on a terrain face, from a view segment v are exactly computed is explained. This algorithm is referred to as the *exact segment-segment* visibility algorithm. In order to compute segment-segment visibility, a

projection from $\mathbb{R}^3$ to $\mathbb{R}^2$ called skew projection is used (Section 3.3.1). An overview of the segment-segment visibility algorithm (Section 3.3.2), which first computes critical points (Section 3.3.3) and next determines the visible parts of segment s from segment v is provided (Section 3.3.4).

The segment-segment algorithm is repeatedly used in each segment of each line probe L. Since the computation of the visibility on each s is completely independent from the computation on any other segment, this process can be parallelized by considering different segments s in parallel.

3.3.1 Skew projection

Let s and v be two non coplanar segments in $\mathbb{R}^3$ so that they determine a solid tetrahedron $\mathbb{T}_{v,s}$. Consider segments v and s parameterized by t and u from 0 to 1, respectively. We denote v_t the point of v with parameter value t and s_u the point of s with parameter value u. Denote $\mathcal{U} = \{\ (t,u) \mid 0 \leq t,u \leq 1\}$ the unit square in $\mathbb{R}^2$. Segment $s = \overline{v_t s_u}$ with endpoints $v_t \in v$ and $s_u \in s$ determines the point $(t,u) \in \mathcal{U}$ (see Figure 3.3).

For each point $p \in \mathbb{T}_{v,s}$ let $\pi_{p,v}$ and $\pi_{p,s}$ be the planes through point p and segments v and s, respectively. Let $w_p = \pi_{p,v} \cap \pi_{p,s} \cap \mathbb{T}_{v,s}$ be the unique segment through p with endpoints in v and s.

A mapping $sk : \mathbb{T}_{v,s} \mapsto \mathcal{U}$ that maps point $p \in \mathbb{T}_{v,s}$ to the point in $\mathcal{U}$ that corresponds to segment w_p is defined. Notice that all the points $q \in w_p$ satisfy $D(q) = D(p)$. This mapping is a restriction of the *skew projection* introduced by Jaromczyk and Kowalukin in [78], and it is continuous in the interior of $\mathbb{T}_{v,s}$.

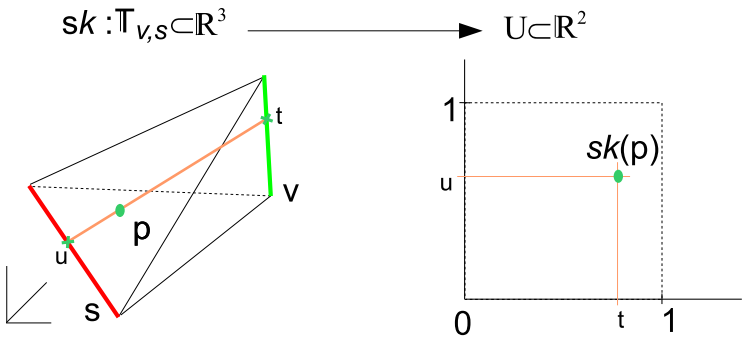


Figure 3.3: Given two segments s, v and a point p, the image of the p for the skew projection is $sk(p) = (t,u)$

3.3.2 Exact segment-segment visibility algorithm

Let v be a view segment with its corresponding visibility criterion (weak or strong) and s a segment on a terrain face. To compute the visible parts of s from v we consider $\mathcal{H}$ the convex hull of segments s and v, which, depending on the relative position of the segments is a line segment, a polygon or a tetrahedron. We assume that s and v are not coplanar, otherwise we are considering a problem in $\mathcal{R}^2$. Denote F the set of terrain faces that may prevent v from seeing s. In Figure 3.4 a segment s and several view segments v with the corresponding convex hulls $\mathcal{H}$ and some terrain faces contained in F can be seen. Set F is obtained choosing the faces intersecting $\mathcal{H}$ among those whose projections onto $\mathcal{D}$ intersect $\mathcal{H}_\mathcal{D}$, the projection of $\mathcal{H}$ onto $\mathcal{D}$. The faces whose projection intersects $\mathcal{H}_\mathcal{D}$ are obtained with the optimal algorithm for windowing queries and subdivision traversal given in Section 2.2.6. According to this algorithm the following lemma can be provided.

Lemma 3.3.1 *The set of terrain faces that may prevent v from seeing s can be obtained in $O(n')$ time and storage, where n' is the number of terrain faces intersecting $\mathcal{H}_\mathcal{D}$.* □

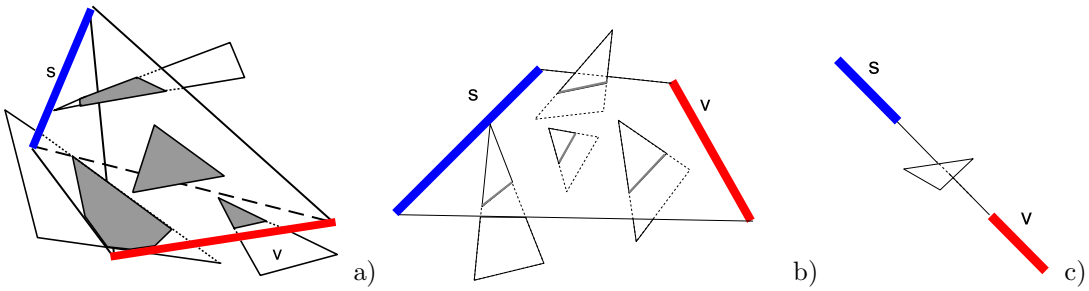


Figure 3.4: A segment s on the terrain and a view element v, their convex hull $\mathcal{H}$ and some terrain faces contained in F.

In order to obtain the visible parts of s from v, we first place a moving point on v and obtain the list of critical points of v where an opaque topology change occurs (Section 3.3.3). As stated in Section 2.2.8 these are the points where the visibility of s can change. Next, the visible parts of s by considering v as a linear light source and using the critical points where opaque topology changes occur are determined (Section 3.3.4). With this aim, segments v and s are parameterized by parameters t and u in $[0, 1]$, respectively. The point in v with parameter t is denoted v_t; accordingly, s_u denotes the point in s with parameter u. Point v_1 is chosen to be higher than v_0 and s_0 and s_1 so that the orthogonal projections onto the terrain domain D of segments $\overline{v_0 s_0}$ and $\overline{v_1 s_1}$ do not intersect.

3.3.3 Critical points computation

To compute critical points we use a triangle-sweep algorithm on $\mathcal{H}$. The view point v_t is moved along v, from $t = 0 \ldots 1$, and maintain segment s fixed. We use the transparent and opaque visibility cycle, which consist of two ordered lists of terrain edges intersecting the sweeping triangle are used (Section 2.2.8.3). The transparent visibility cycle contains all the terrain edges, however, the opaque one only contains visible edges from v_t. Algorithms using visibility cycles to compute visibility information use a pre-process stage which is avoided by storing some extra information in the cycles and adding auxiliary edges and faces in the terrain.

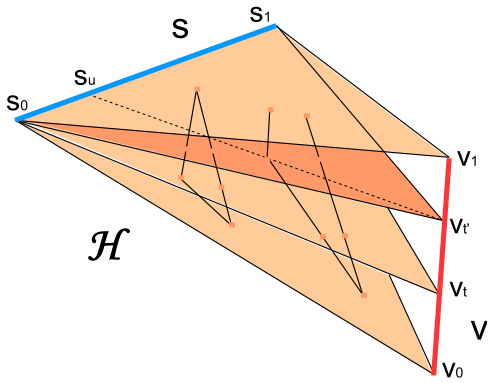


Figure 3.5: The convex hull $\mathcal{H}$ swept by the triangle T_t with several TIN edges and their pierce points depicted on several view triangles.

Auxiliary edges and faces are defined by considering, the set E of segments obtained by intersecting $\mathcal{H}$ with the triangles in F (Section 3.3.2). Every vertical segment joining an endpoint of a segment in E with its orthogonal projection onto the terrain domain $\mathcal{D}$ is an auxiliary edge. Auxiliary faces are the quadrilaterals determined by a segment of E and its orthogonal projection onto $\mathcal{D}$, i.e. they are contained on a vertical plane through edge e. From now on and whenever it is not specified, faces and edges mean both, TIN and auxiliary edges. The following lemma bounds the number of auxiliary edges and faces that have been introduced.

Lemma 3.3.2 *$O(n')$ auxiliary edges and faces are used.* □

At a given time t we denote: T_t the sweep-triangle defined by s and v_t; $p_{e,t}$ the intersection point, *pierce point*, of edge e and T_t; $f_{e,t}$ the *face behind* e, the first face intersected by the ray $\overline{v_t p_{e,t}}$ after going through $p_{e,t}$, if it gets to s without intersecting any face $f_{e,t} = s$; tvc_t the transparent visibility cycle; ovc_t the opaque visibility cycle; and F_t an ordered list of faces intersecting $\overline{v_t u_1}$. Faces in F_t are ordered by increasing distance from v_t. In the cycles, edges are ordered according to the angular order of their pierce points.

This algorithm consists of an initialization step where structures are initialized and then the sweep starts. During the sweep the list of faces F_t, the transparent visibility cycle tvc_t, the opaque visibility cycle ovc_t, and the face behind the edges stored in tvc_t are maintained. We say that a *change* occurs when some of the structures or information stored has to be updated. Thus, during the sweep, we have to determine the points v_t where a change occurs. These points are named *critical points* and are characterized by *events*. Critical points producing opaque changes are also called *shadow points*. Notice that auxiliary edges produce critical points, however, they do not produce shadow points because auxiliary edges are not visible from v_t.

The general idea is that: the transparent visibility cycle is needed to compute the critical points; the opaque one to determine when a critical point is a shadow point; the faces behind the edges, $f_{e,t}$, to determine where the changes are produced and the list of faces F_t to easily update the faces behind the edges. Next we will characterize the critical points, and handle the updating of the stored information.

3.3.3.1 Critical points characterization

In the next Lemma we will show that all the critical points cause transparent topology changes.

Lemma 3.3.3 *Any critical point causes a transparent topology change, and viceversa.*

Proof. Let p be a critical point, if it is defined by a change on the opaque or transparent topology cycle the lemma is obvious. If there is a change in F_t, there is a change in tvc_t and a transparent topology change occurs. Finally, if $f_{e,t}$, the face behind an edge e, has to be updated; again two edges intersecting T_t exchange their angular position in tvc_t. □

The point where a topology change occurs can be easily determined according to the following properties:

(1) A transparent topology change occurs if and only if there are two edges e and e' and a line segment going through v_t, e, e' and s. It also occurs when an edge starts or stops intersecting $\mathcal{H}$

(2) A transparent topology change is an opaque topology change if the line segment from v to s that goes through edges e and e' does not pass through the interior of any face.

Therefore, we can provide the following Property 3.3.1 which ensures that any transparent topology change can be characterized by a 4-tuple (t, e, e', u) where s_u $(/u_t)$ is the intersection point between s $(/v)$ and the segment through v_t $(/u_t)$, e, e' and s $(/v)$. Coordinates u and t are obtained by using the skew projection (Section 3.3.1) of segments e and e'. Points where an edge e starts or stops intersecting $\mathcal{H}$ are also considered, these points are characterize by using the 4-tuple (t, e, e, u).

Property 3.3.1 *Point v_t is a critical point if and only if there are two edges e and e' and a line segment ℓ going through v_t, e, e' and s or an edge e starts or stops intersecting $\mathcal{H}$. It is a shadow point if ℓ does not intersect the interior of any face.*

Critical points are encoded by using the 4-tuple, but only those with t and u within $[0, 1]$ are considered, the rest are rejected because they are not produced in segments v and s. The unrejected critical points are stored in a priority queue by increasing order. Events are ordered by the lexicographical order defined by the following four real numbers: $(t, u, |\overline{pv_t}|, |\overline{qv_t}|)$ where p and q are, respectively, the intersection points of T_t with e and e' and $|\overline{pv_t}|$ and $|\overline{qv_t}|$ the Euclidean distance from p and q to v_t, respectively. It defines a partial ordering on the events. Notice that two events are not comparable if and only if the 4-tuples are equal, and consequently their four edges define the same pierce point on T_t (this may happen with the edges incident to a vertex). When we have two events not comparable with the previous partial order, events with $e \neq e'$ are handled before those with $e = e'$, if $e = e'$, those with a TIN edge are considered first and then, those with auxiliary edges.

In practice, critical points are obtained by considering three different types of events. Events of type (1) and (2) are obtained in an initialization-step where we also obtain some

events of type (3), but we keep on obtaining these kind of events during the whole sweep using new adjacent pairs of ordered edges in tvc_t. The three types are the following:

(1) T_t contains a pierce point which is an endpoint of an edge.

(2) A TIN edge intersects the boundary of T_t.

(3) Two edges exchange their order in tvc_t.

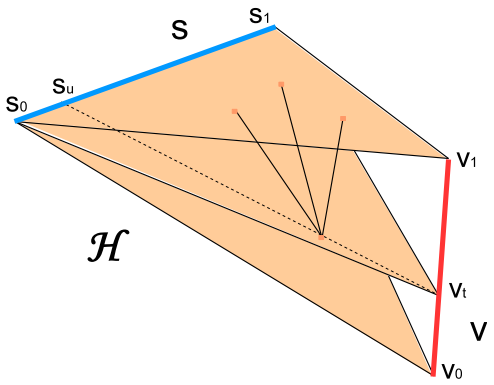


Figure 3.6: In the figure we can see different events of type (1) at the same t.

3.3.3.2 Initialization step

First of all, the priority queue of events E is initialized. In the initialization stage, all the events of type (1) and (2) are computed and stored in the priority queue by increasing t. Events of type (1) are obtained by considering the edges e intersecting $\mathcal{H}$, whose 4-tuple are (t, e, e, u). Coordinate t is computed by intersecting the plane delimited by s and the corresponding endpoint of e, p, with segment v. Coordinate u is obtained intersecting s with the line defined by segment $\overline{v_t p}$. Events of type (2) are obtained by considering the TIN edges that intersect the boundary of $\mathcal{H}$, $\partial\mathcal{H}$. Their 4-tuple is (t, e, e, u) with t or u equal to 0 or 1. Parameters u and t are computed as in the previous case considering the endpoint of e instead of the intersection point $\partial\mathcal{H} \cap e$.

The initial transparent and opaque visibility cycle tvc_0, ovc_0 is computed. The transparent visibility cycle tvc_0 is given by the edges e producing events with 4-tuple $(0, e, e, u)$. These edges have to be ordered by increasing u. Therefore, the order in tvc_0 coincides with the increasing order of the events in E. To build tvc_0 the events of the priority queue

of type $(0, e, e, u)$, which are deleted from E are considered. For each edge e, $f_{e,0}$, the face behind e, by using a line-sweep that sweeps T_0 around v_0 is computed. Finally edge e with $f_{e,0}$ is stored in tvc_0. The opaque visibility cycle ovc_0 contains the edges of tvc_0 that are visible from v_0, thus for each edge e we check whether e is visible from v_0. This can be done by using, again, the sweep on T_0. In the case that e has to be stored in ovc_0, event $(0, e, e, u)$ is stored in the list of shadow points $\mathcal{C}_s$ whenever $f_{e,0} = s$. The list of faces intersecting $\overline{v_0 s_1}$, F_0, is obtained at the last step of the sweep used to compute the faces behind the edges.

Finally events of type (3) obtained from tvc_0 are inserted in the priority queue E. All the pairs of adjacent edges in tvc_0 are considered. Using the skew projection (Section 3.3.1), the coordinates t and u where these points exchange their angular position are computed.

Lemma 3.3.4 *The complexity of the initialization step is $O(n' \log n')$ time and $O(n')$ storage, in the worst case.*

Proof. In the initialization step at most $O(n')$ events of types (1) and (2) (Lemma 3.3.2) are inserted in E. Each 4-tuple defining such an event is obtained in $O(1)$ time, therefore the time needed to compute and insert the events in E is $O(n' \log n')$. Obtaining tvc_0, ovc_0 and F_0 takes $O(n' \log n')$ time and $O(n')$ storage: at most $O(n')$ events from E are deleted and we use a sweep algorithm on T_0, which takes $O(n' \log n')$ time and $O(n')$ storage. At most $O(n')$ events of type (3) are obtained from tvc_0 and added to E in $O(n' \log n')$ time. Thus, the total complexity of the initialization step is $O(n' \log n')$ time, and $O(n')$ storage. □

3.3.3.3 Sweep algorithm

During the sweep, more than one event at the same time t can be found. Since events of type (3) are found during the sweep, it is important to avoid inserting events that have already been handled. Do to this we do not insert new events with time smaller than t, the current time, and before inserting an event with time t we check whether it has already been handled. This is done by using a list E_t containing all the handled events at time t. A list V_t of visible points at time t, is used to avoid repeating some computations.

While the priority queue E is not empty, the first event of E, $\nu = (t, e, e', u)$, is taken and deleted from E. Assume that t' is the time of the previously handed event. Thus we

have to update $tvc_{t'}$, the faces behind its edges, $ovc_{t'}$, $F_{t'}$, $E_{t'}$, $V_{t'}$ and E. If ν produces an opaque topology change on s, event ν is stored in the list of shadow points on s, $\mathcal{C}_s$. Notice that this happens when it produces a change in $ovc_{t'}$ and s is the old or new face behind edge e or e', i.e. $s \in \{f_{e,t'}, f_{e',t'}, f_{e,t}, f_{e',t}\}$. In the case that t is not equal to t' lists $E_{t'}$ and $V_{t'}$ are emptied, the rest of the updates are handled depending on the type of event we are considering:

A) An auxiliary edge e appears or disappears: Since we are considering auxiliary edges, opaque topology changes do not occur. Consequently, we have to update $tvc_{t'}$, the face behind edge e and if $u = 1$ list $F_{t'}$.

- The updated *transparent visibility cycle*, tvc_t, is obtained by deleting e from $tvc_{t'}$ if e is in $tvc_{t'}$, or adding e to $tvc_{t'}$, otherwise.
- If $u = 1$, the updated *list of faces* F_t is obtained by considering all the auxiliary faces having pierce point $p_{e,t}$ as a vertex. The auxiliary faces contained in $F_{t'}$ are deleted from it and the auxiliary faces not contained in $F_{t'}$ are added to it.
- When e is added to $tvc_{t'}$ we have $u = 1$ and the *face behind* e is found by using F_t. Priority queue E is updated according to new adjacent pairs of edges in tvc_t, this produces at most two new events.
- The *priority queue* is updated by considering the possibly new events of type (3).

Lemma 3.3.5 *An event where an auxiliary edge appears or disappears can be handled in $O(\log n')$ worst case time.*

Proof. We need $O(\log n')$ time to locate e in $tvc_{t'}$ and update it and $O(\log n')$ time to update $F_{t'}$. Auxiliary faces adjacent to this pierce-point, which is an endpoint, can be obtained in $O(1)$. We need $O(\log n')$ worst case time to obtain the face behind e. Finally, at most two new events are created in $O(1)$ time and inserted in $O(\log n')$ time in E. □

B) A TIN edge e appears or disappears: A TIN edge can appear or disappear elsewhere, thus we may have to update $tvc_{t'}$, the face behind e, $V_{t'}$, $ovc_{t'}$ and if $u = 1$ list $F_{t'}$.

- The updated *transparent visibility cycle*, tvc_t, is obtained by deleting e from $tvc_{t'}$ if e is in $tvc_{t'}$, or adding e near an edge $e' \in tvc_{t'}$ defining the same pierce point as e, which always exists, otherwise.

- Priority queue E is updated according to new adjacent pairs of edges in tvc_t, which produce at most two new events.

- When e is inserted in tvc_t near e' the *face behind* e is the face behind e'.

- The set of *visible points* V_t is updated when e appears in $ovc_{t'}$. In this case, pierce point $p_{e,t}$ is visible and is stored in V_t by using the tuple $(t, u, |\overline{p_{e,t}v_t}|)$ that encodes $p_{e,t}$.

- The updated *opaque visibility cycle*, ovc_t, is obtained by deleting e from $ovc_{t'}$ if e is in $ovc_{t'}$, or adding e to $ovc_{t'}$ if e is not in $ovc_{t'}$ and the tuple encoding $p_{e,t}$ is in V_t.

- List $F_{t'}$ is updated when $u = 1$. The TIN faces are updated by determining the faces having the endpoint of e as a vertex. We check whether the faces contained in $F_{t'}$ have to be deleted, and whether the faces not contained in $F_{t'}$ have to be added.

Lemma 3.3.6 *An event where a TIN edge appears or disappears can be handled in* $O(\log n')$ *worst case time.*

Proof. We need $O(\log n')$ time to locate e in $tvc_{t'}$, in $ovc_{t'}$ and to update Ft'. The new face behind e is determined in $O(1)$ time. Notice that finding the faces adjacent to the endpoint of e can be done in $O(1)$ time by using a DCEL structure and due to the fact that each vertex is contained in a constant number of faces. At most two new events are created in $O(1)$ time and inserted in $O(\log n')$ time. □

C) Two edges e and e' exchange their angular position: Two edges can exchange their angular position elsewhere, thus we may have to update $tvc_{t'}$, the faces behind its edges, $V_{t'}$ and $ovc_{t'}$. In this case event ν can only be handled if edges e and e' are adjacent in $tvc_{t'}$. Priority queue E always contains an event with time t fulfilling the previous property, thus we keep on taking events from E until we obtain one fulfilling it. The removed events that have not been handled are reinserted in E.

- The updated *transparent visibility cycle* is obtained by exchanging the angular position of e and e'.

- Priority queue E is updated according to new adjacent pairs of edges in tvc_t, it produces at most two new events.

- The *opaque visibility cycle* is updated when e and e' appear in $ovc_{t'}$. In this case the edge further from v_t is deleted. In the case, that only one edge, assume that it is edge e, appears in $ovc_{t'}$, insert e' in ovc_t whenever e' is an edge of $f_{e,t'}$ or of $f_{e,t}$.
- The *faces behind* the edges are updated depending on:
  - i) If $f_{e,t'} = f_{e',t'}$, let e be the edge with $|\overline{p_{e,t}v_t}| < |\overline{p_{e',t}v_t}|$, then $f_{e',t} = f_{e',t'}$ and $f_{e,t} = f_i$ with f_i the face containing e' minimizing the clock-wise angle $\widehat{p_{e,t}p_{e',t}q}$, where $q \in T_t \cap f_i$.
  - ii) If $f_{e,t'} \neq f_{e',t'}$ and $f_{e,t'}$ contains e', let f be the other face adjacent to e' different from $f_{e,t'}$, if the angle determined by f and the ray $\overline{v_t s_u}$ is bigger than π, $f_{e,t} = f_{e',t'}$ and $f_{e',t} = f_{e',t'}$. In the case that the angle is at most π: $f_{e,t} = f$ and $f_{e',t} = f_{e',t'}$
  - iii) If $f_{e,t'} = f_{e',t'}$ and $f_{e',t'}$ contains e proceed analogously to the previous case.
  - iv) Otherwise nothing is done.

Lemma 3.3.7 *An event where two edges exchange their angular position can be handled in $O(\log n')$ worst case time.*

Proof. The time needed to obtain an appropriate event can be considered to be $O(1)$. We may obtain an event (t, e, e', u) with e and e' not adjacent in $tvc_{t'}$ when $p_{e,t} = p_{e't}$ is a vertex of e which is incident to some other edges. The number of vertices contained in T_t can be considered $O(1)$ and each vertex is incident to a constant number of edges. Thus in constant time, an appropriate event is obtained. Consequently, the time needed to reinsert the events in E is $O(\log n')$. Once the event is found, we need $O(\log n')$ time to locate e in $tvc_{t'}$ which can be updated in $O(1)$ time, the updating of $f_{e,t}$ and $f_{e',t}$ can also be done in $O(1)$ time. Locating e in $ovc_{t'}$ is done in $O(\log n')$ worst case time, and it is updated in $O(1)$ time. Therefore, these events can be handled in E in $O(\log n')$ worst case time. At most two new events are created in $O(1)$ time and inserted in $O(\log n')$ time, after checking whether they are stored in E_t in $O(1)$ time. □

If Figure 3.7 we can see several steps of the sweep algorithm when a segment s on the terrain and a view segment v are considered. In the Figure 3.7 the TIN faces contained in F have been painted, auxiliary faces and edges are not represented.

Proposition 3.3.1 *The shadow points producing topological changes on s can be obtained in $O(n'^2 \log n')$ time and $O(n'^2)$ storage, in the worst case.*

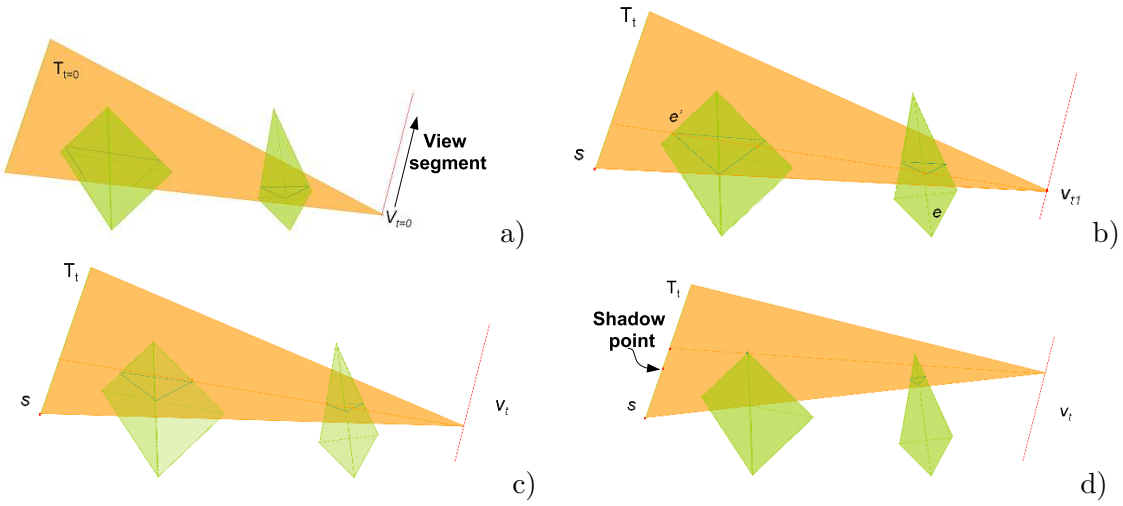


Figure 3.7: Representative steps of the sweep algorithm, segment s is on the terrain and v the view element. Faces in F and the convex hull $\mathcal{H}$ are painted. (a) The initialization step. (b) Two critical points at the same time t: e intersects $\partial\mathbb{T}_{v,s}$ producing a shadow point, and e_1 and e_3 exchange their angular position. (c) Edges e_2 and e_4 produce a transparent topology change. (d) The vertex produces an opaque topology change, it is a shadow point.

Proof. Since according to Lemma 3.3.3 all the critical points cause transparent topology changes which are associated with pairs of edges due to Property 3.3.1, we may handle up to $O(n'^2)$ events, one per edge pair and one per edge endpoint. The initialization step is done in $O(n' \log n')$ time and $O(n')$ storage due to Lemma 3.3.4. The storing of events in E takes $O(n'^2 \log n')$ time and $O(n'^2)$ storage. Any event can be handled in $O(\log n')$ time according to Lemma 3.3.5, Lemma 3.3.6 and Lemma 3.3.7. Therefore, the sweep process takes, in the worst case, $O(n'^2 \log n')$ time and $O(n'^2)$ storage. □

3.3.4 Visibility computation on a segment

To obtain the visible parts of segment s from segment v, we act as if we placed a linear light along v and determine the illuminated parts of s. We use the list of events producing shadow points on s, $\mathcal{C}_s$. They are then stored by increasing u coordinate and partition s into fragments. Each of these subsegments is such that all the points contained in it are visible from the same set of points of v. In other words, all the points receive the same amount of light. By computing the visible part of v from s_0 and sweeping s, we can determine the visible parts of v at each subsegment of s and consequently compute the weakly and strongly visible parts of s from v.

To compute the visible parts of v from s_0, the triangle tr_0 defined by v an s_0 is considered. The maximal visible subsegments of v from s_0, by using an angular sweep around s_0 is determined. The endpoints of the visible parts of v are delimited by a line segment emanating from s_0 to v going through an edge e that intersect tr_0. Thus, each endpoint of a visible subsegment of v can be characterized by an edge $\tilde{e}$ and each visible subsegment of v by a pair of edges $(\tilde{e}, \check{e})$. Visible subsegments of v are stored in a list $\mathbb{V}_0$, which, at the end of the process, contains the maximal visible subsegments of v from s_0.

To compute the visible parts of s from v, s_0 is moved along s and $\mathbb{V}_0$ is updated stopping at each shadow point of $\mathcal{C}_s$. Assume that $\nu = (t, e, e', u)$ is the shadow point to handle, and u' the coordinate of the last shadow point considered. We proceed as follows:

i) If $(e, e') \in \mathbb{V}_{u'}$, subsegment (e, e') is deleted from $\mathbb{V}_{u'}$. The visible subsegment of v delimited by edges e and e' is a point which will not be visible for $u' > u + \varepsilon$ with $0 < \varepsilon \ll 1$.

ii) If $(e, e'') \in \mathbb{V}_{u'}$ with $e'' \neq e'$ an edge, the visible subsegment (e, e'') is replaced by (e', e''). Edges e and e' exchange their angular position and the visible subsegment will have an endpoint at e' and not at e.

iii) If $(e', e'') \in \mathbb{V}_{u'}$ with $e'' \neq e$ an edge, the visible subsegment (e', e'') is replaced by (e, e''). Again, e and e' exchange their angular position and the visible subsegment will have an endpoint at e and not at e'.

iv) If neither e nor e' delimit a visible segment of $\mathbb{V}_{u'}$, the visible subsegment of v delimited by (e, e') is inserted in $\mathbb{V}_u$. From now on these two edges will determine a visible subsegment of v.

An edge e can not delimit two different visible subsegments of v at u'. Each edge e that defines an endpoint of a visible subsegment, defines, necessarily, an endpoint of a non-visible subsegment. Thus, the previous cases are exclusive and exhaustive. In other words, for each shadow point one and only one of the four previous cases is satisfied.

Finally, the *weakly visible parts* of s are those parts that are visible from at least a point of v, which are the subsegments of s having $\mathbb{V}_u$ not empty. To determine the weakly visible subsegments, the first and last point of s having $\mathbb{V}_u$ not empty, until we get s_1 is stored alternatively. The *strongly visible parts* of s are those parts that are visible from every point of v, thus $\mathbb{V}_u$ has to contain the whole segment v. When $\mathbb{V}_u$ contains a single subsegment, $\mathbb{V}_u = \{(e, e')\}$, we check whether it is the whole s. Assume that $\mathbb{V}_u$ is then

modified at coordinate u' and that $v_0 < e < e' < v_1$ considering the angular order on the plane delimited by v and $s_{(u+\widehat{u})/2}$. Let π be the plane delimited by v_0 and e, π' that delimited by e' and v_1, and denote $s_p = \pi \cap s$ and $s_{p'} = \pi' \cap s$. If π ($/\pi'$) does not separate $p'(/p)$ and $e'(/e)$, the subsegment $\overline{s_p s_{p'}} \cap \overline{s_u, s_{u'}}$ is strongly visible, otherwise $\overline{s_u s_{u'}}$ is not strongly visible.

Lemma 3.3.8 *The total complexity of obtaining the visible parts of s by using the list of shadow points causing opaque topology changes on s is $O(n'^2 \log n')$ time and $O(n'^2)$ space, in the worst case.*

Proof. The visible parts of v from s_0 are computed in $O(n' \log n')$ time and $O(n')$ storage. The visible parts of s from v are obtained by sweeping s considering at most $O(n'^2)$ shadow points. Since each update is handled in $O(\log n')$, $O(n'^2 \log n')$ time and $O(n'^2)$ storage is needed, in the worst case. □

The next proposition provides the complexity of obtaining the visible parts of s from v, which is the complexity of the segment-segment algorithm. It is a consequence of Proposition 3.3.1 and Lemma 3.3.8.

Proposition 3.3.2 *Given a view segment v and a segment s, the worst case complexity of the segment-segment algorithm is $O(n'^2 \log n')$ time and $O(n'^2)$ space, where $n' = O(n)$ but generally $n' \ll n$.* □

In the next figure we represent how the visible parts of segment $\overline{pq}$ from $\overline{ab}$ are determined, when the depicted triangles occlude the visibility. The pairs of vertices written below line segment $\overline{pq}$ are the shadow points produced by the triangles, $\overline{ab}$ and $\overline{pq}$, and those vertices written above $\overline{pq}$ represent the visible parts of $\overline{ab}$. By using the shadow points we sweep $\overline{pq}$ and maintain the visible parts of $\overline{ab}$ using the previous algorithm.

3.3.5 Multi-visibility map computation

The aim of our algorithm is to obtain the multi-visibility map specified in the input of the algorithm. By using the exact segment-segment visibility algorithm, the visibility on each segment s of line probe L from a view element is computed. Once it has been done for every view element the multi-visibility on s is computed. When each segment s has

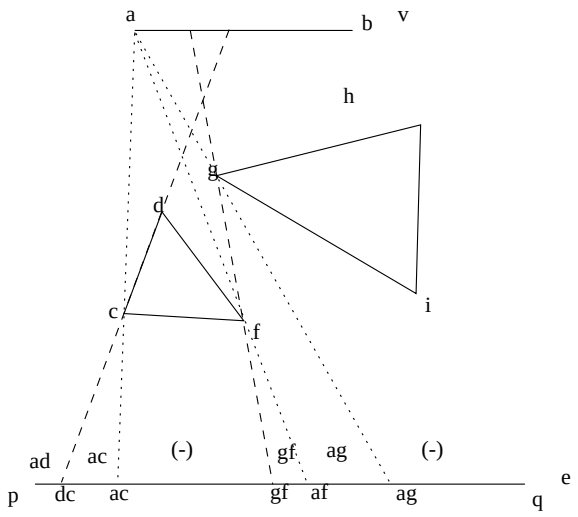


Figure 3.8: The visible parts of segment $\overline{pq}$ from $\overline{ab}$ are the parts delimited by the dashed lines having above a non empty list $(-)$.

been considered, the multi-visibility on L is exactly obtained and finally the approximated multi-visibility map is reconstructed.

Multi-visibility information is stored in a copy of line probe L, L_M, which is progressively updated after considering each view element. Let s be a segment of the line probe L, then, the visibility of s from the first view element v_1 according to its visibility criterion: weak or strong is studied. The visible parts of s from v_1 are stored in L_M. Next, the visibility of s from v_2 is computed, and L_M is updated according to the kind of multi-visibility map we are interested in obtaining. The process is repeated until all the segments have been considered. When all the segments conforming L have been studied for all the view objects, L_M contains the multi-visibility information of L.

To obtain the *overlay map* the information of all the view objects is superimposed. Then the subsegments of s with the set of view elements from which they are visible are labelled. To obtain the *union map*, the union of all the visible subsegments of s are considered and labelled as visible. The *intersection map* is obtained by considering the intersection of all the visible subsegments of s and labelled as visible. Finally, the *count map* is obtained by superimposing the information of all the view objects and labelled with the number of view elements each segment is visible from.

Notice that the computation of the multi-visibility map on each segment s is independent from the computation to any other segment s'. We have already mentioned that the segment-segment algorithm can be parallelized, the process of storing the visibility

information in order to obtain the multi-visibility information can be also parallelized considering different segments s and s' in parallel.

Let us assume that we are given a set of r view elements that are decomposed into R view segments. Next lemma specifies the complexity of the algorithm to obtain the multi-visibility information on a segment s ignoring the time needed in the segment-segment algorithm which is used R times.

Lemma 3.3.9 *The worst case extra time complexity to obtain the multi-visibility of a segment s form a the set of view elements V is $O(n^2R)$.*

Proof. To obtain the multi-visibility on segment s, visibility from the R view elements needs to be overlayed. Each view segment s may be subdivided into $O(n'^2)$ subsegments for each view segment, n' depends on the view element. Thus we use that in the worst case $n' \in O(n)$. Therefore the complexity is $O(n^2R)$ time, and $O(n^2R)$ space. □

Once we have studied all the segments s conforming a line probe, the line probe is used to update the multi-visibility map reconstruction by adding the information provided by the new line probe. The committed error on the new approximation is computed and if desired, a new line probe is considered to keep on refining the reconstructed multi-visibility map. At the end of the process, an approximate reconstruction of planar subdivision induced by the desired multi-visibility map is obtained. In fact it is obtained in a DCEL structure. Then typical algorithms for answering point or region queries on a planar subdivision can be used to answer multi-visibility queries (Section 2.2.6). In Chapter 4 more details are given about which queries can be answered, and the type of multi-visibility map required.

3.4 Computational cost

Finally,the computational cost of the whole algorithm is analyzed. It has been partially analyzed, but the complexity of the whole algorithm has not been calculated.

Let us assume that we are working with a terrain with n faces. Let V be the set of r view segments, and R the total number of view segments conforming the r view elements, $r \leq R$. Let us assume that during the whole algorithm we have used m line probes that define M line segments when they are intersected with the terrain domain

triangulation, in the worst case $M \in O(mn)$. Consequently the segment-segment visibility algorithm has been used $O(RM)$ times. The total time used by the segment-segment visibility algorithm is $O(RMn^2 \log n)$ and $O(n^2)$ space. Notice that, in the segment-segment visibility algorithm only the visible parts of each view segment are stored. These visible parts are then used to obtain the multi-visibility information (Proposition 3.3.2). The multi-visibility information of all the line probes is obtained in $O(MRn^2)$ time. At each step we only store the multi-visibility of the current line probe, which defines at most $O(n)$ line segments and is used to update the multi-visibility map. Therefore the space complexity is $O(n^3R)$. The reconstruction of the approximate multi-visibility map is obtained (Section 2.2.7) in $O(m \log m)$ mean time. Thus the following theorem can be stated.

Theorem 3.4.1 *The time complexity of obtaining a desired multi-visibility of a set of view elements conformed by R view segments on a terrain $\mathcal{T}$ of n faces by using m line probes defining M segments on $\mathcal{T}$ is $O(RMn^2 \log n + m \log m)$.* □

It can be said say that the time complexity of obtaining the desired multi-visibility of a set of view elements, defined by R view segments on a terrain $\mathcal{T}$ of n faces, with the algorithm to approximately reconstruct planar subdivisions is $O(Rmn^3 \log n + m \log m)$. This is obtained by using the fact that $M \in O(nm)$. If a realistic terrain is considered, the number of faces intersected by a line is $O(\sqrt{n})$ (Section 2.4.3). Therefore the multi-visibility map is obtained in $O(RMn n^2 \log n + m \log m)$ time with $M = O(n\sqrt{n})$.

3.5 Obtaining any desired multi-visibility map

Given a terrain $\mathcal{T}$ and a set of r view elements V, with associated weak or strong visibility criterion, the previous algorithm can be modified to compute any desired multi-visibility map after a pre-processing stage.

The previous algorithm is divided into two stages. In the first one visibility information on a preestablished set of uniformly distributed lines on $\mathcal{D}$ for every view element is computed. In the second one the desired multi-visibility map is reconstructed using the already computed visibility information. We could also compute in the pre-processing stage an approximate reconstruction of the overlay map which contains all the visibility information. Next, from the overlay map any desired multi-visibility map can be obtained.

As before, at the end of the process, an approximate reconstruction of planar subdivision induced by the desired multi-visibility map in a DCEL structure is obtained. Therefore, multi-visibility queries, can be answered by using typical algorithms for answering point or region queries on a planar subdivision (Section 2.2.6).

3.6 Inter-region multi-visibility on terrains

Algorithms to compute visibility from generalized view elements can be used to compute weak or strong inter-region visibility on terrains. To compute inter-region visibility we have to be given two terrain regions $\mathcal{R}_1$ and $\mathcal{R}_2$, and the aim is to obtain the mutually visible parts of $\mathcal{R}_1$ and $\mathcal{R}_2$. This can be done by using our algorithm.

Region $\mathcal{R}_1$ is considered as a view area. Since we are interested in visibility on $\mathcal{R}_2$, line probes are restricted on the projection of $\mathcal{R}_2$ onto the terrain domain. Visibility on $\mathcal{R}_2$ is computed by considering all the terrain triangles and according to the specified visibility criterion, using the guidelines provided next. By doing so we, the visible parts of $\mathcal{R}_2$ from $\mathcal{R}_1$ can be obtained. Then, $\mathcal{R}_2$ is considered as the view area, the line probes are restricted on the projection of $\mathcal{R}_1$ onto the terrain domain to obtain the visible parts of $\mathcal{R}_1$ from $\mathcal{R}_2$.

Depending on whether weak or strong visibility is considered, we proceed in one of the two following ways:

- **weak visibility**: the set of view elements contains the polygons obtained by intersecting the terrain faces with $\mathcal{R}_1$ with weak visibility criterion associated. Their union multi-visibility map restricted on $\mathcal{R}_2$ is computed. Since two points are mutually visible or mutually not-visible, we can consider as set of view elements the polygons obtained by intersecting $\mathcal{R}_2$ with the terrain faces containing weakly visible points from $\mathcal{R}_1$. Now the union multi-visibility map restricted on $\mathcal{R}_1$ is computed.

- **strong visibility**: the set of view elements contains the polygons obtained by intersecting the view area $\mathcal{R}_1$ with the terrain faces, they have associated strong visibility criterion. The intersection multi-visibility map restricted to $\mathcal{R}_2$ is computed. Next, we consider as view area $\mathcal{R}_2$ and the intersection multi-visibility map restricted to $\mathcal{R}_1$ is computed. Previously computed information cannot be used because a visible point of $\mathcal{R}_1$ has to be visible from every single point of $\mathcal{R}_2$.

Thus, the weak and strong inter-region visibility problem on a terrain can be solved.

3.7 Conclusions

A way to exactly compute the visible parts of a segment from a general view element which can be a view point, segment, polygon or polygonal is given. This algorithm is the first to compute an approximation of a multi-visibility map on a terrain domain when considering generalized view elements and strong or weak visibility.

However, it is expensive both in time and storage and is difficult to turn into a practical algorithm. Bearing these problems in mind, another approach based on graphics hardware capabilities is being developed from a practical point of view.

Chapter 4

Multi-visibility on terrains by using Graphics Hardware

In this chapter we address the problem of visualizing approximate multi-visibility maps, obtained by using the GPU, for a terrain modelled by a TIN with respect to an heterogeneous set of view elements containing points, segments, polygonal chains and polygons. We handle both weak and strong visibility, and we also efficiently solve, approximately, different point and polygonal region multi-visibility queries. The method presented here differs from the one presented in the previous chapter in how visibility information is computed and how multi-visibility maps are represented.

We consider a grid covering the domain and, in a preprocessing stage, the approximate visibility map of each view element is computed (Section 4.2). Visibility information is obtained by repeatedly using an algorithm that approximately computes segment-segment visibility (Subsection 4.2.1). Then we can visualize a discrete approximation of any multi-visibility map from the visibility maps previously obtained (Section 4.3). We can also efficiently answer point and polygonal region multi-visibility queries (Section 4.4) including multi-visibility inter-region queries. Finally, we present experimental results obtained with the implementation of our algorithms (Section 4.5).

4.1 Process overview

The input of the algorithm is a terrain modelled by a TIN, a set V of view elements and a specified visibility criterion, weak or strong, for each view element in V. The output

is the visualization of the desired approximate multi-visibility maps. After visualization, point and polygonal region multi-visibility queries can be answered.

Visibility information is computed approximately in a preprocessing stage with a robust and easy-to-program method by using graphics hardware. In this stage the terrain domain is discretized into a rectangular grid and the discrete visibility map of every view element is computed (Section 4.2). Each grid column is represented by the vertical line through the centers of its cells. A visibility map is obtained by scanning the lines representing the grid columns from left to right. Lines are intersected with the triangulation of the terrain domain yielding a set of segments that are lifted onto terrain faces. The visibility of these segments is computed, after decomposing the view elements into view segments (Section 3.1), by using the segment-segment visibility algorithm (Section 4.2.1). Finally, the rectangular grid with all visibility information is transferred to the GPU using textures.

After the preprocessing stage, we can visualize any multi-visibility map (Section 4.3) and answer queries related to it (Section 4.4). A scheme of the process is given in Figure 4.1.

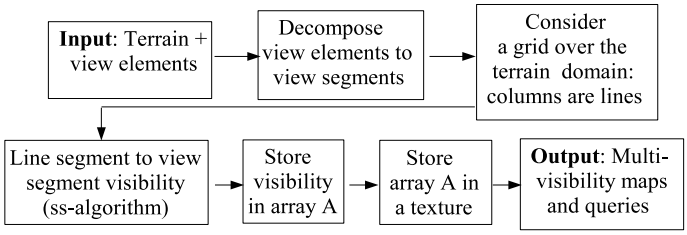


Figure 4.1: A schematic process overview of our algorithm

4.2 Visibility information computation

Visibility information is computed in the preprocessing stage where the domain of the terrain is discretized into a rectangular grid of size $M \times N$. This size is chosen according to a parameter μ representing the number of grid points per length unit of the terrain domain. Notice that the bigger the μ, the more precision we have in the obtained results.

Visibility maps are obtained by scanning the lines representing the grid columns. As explained in Section 4.1, visibility in each line is determined by repeatedly using an algorithm that approximately computes segment-segment visibility (Section 4.2.1). In the following step, the discrete visibility map of each view element of V is properly stored by

using a true/false (visible/not visible) criterion in a single bit of each grid position.

Finally, at the end of this stage, the grid with the visibility information is transferred to the GPU in a RGBA-texture (see Section 4.3). The initial grid covering the domain is accordingly mapped into a unidimensional array $\mathcal{A}$ of size $4MN$, which is later transferred to a texture. Every four consecutive elements of $\mathcal{A}$ represent the same grid point and are stored in the four channels of a texture position. Therefore, we can represent up to 64 visibility maps, one per bit, in a single array $\mathcal{A}$ (texture). In the case when we have more than 64 view elements, we need an array for every 64 view elements or fraction. When a texture is not sufficient to cover the grid ($M, N > 4096$) more than one is used and the information is appropriately transferred to them.

4.2.1 Approximate segment-segment visibility algorithm

Given a view segment v with its corresponding visibility criterion (weak or strong) and a segment s on a face of the terrain, we want to compute the visible parts of s from v.

Let $\mathcal{H}$ be the convex hull of segments s and v which, depending on the relative position of the segments, is a line segment, a polygon or a tetrahedron. Denote F the set of terrain faces that may prevent v from seeing s. The set F is obtained by choosing from among the faces, whose projection onto the xy-plane intersect the projection of $\mathcal{H}$, the ones intersecting $\mathcal{H}$ and front-facing at least one endpoint of v. A face f is front-facing a point p if the dot product of the upward normal of f and a vector from a point on f to p is positive. The faces whose projection intersects the projection of $\mathcal{H}$ are obtained using an optimal algorithm for windowing queries and subdivision traversal presented in Section 2.2.6. Consequently we can provide the following Lemma.

Lemma 4.2.1 *The set F of faces that may prevent v from seeing s can contain $O(n)$ faces and is obtained in $O(n)$ worst case time.*

We parameterize $v(/s)$ by $t(/u)$ in $[0, l_v](/[0, l_s])$ where $l_v(/l_s)$ are the lengths of $v(/s)$. The point in $v(/s)$ with parameter $t(/u)$ is denoted $v_t(/s_u)$. Let Sk_{vs} be the mapping that maps each segment $s_{t,u}$ of endpoints v_t and s_u to the pair $Sk_{vs}(s_{t,u}) = (t, u)$ of the bounded rectangular $\mathcal{R} = [0, l_v] \times [0, l_s]$. This mapping coincides with the skew projection (Section 3.3.1) but having as image $[0, l_v] \times [0, l_s]$ instead of $[0, 1] \times [0, 1]$. Notice that Sk_{vs} is a bijection between $\mathcal{R}$ and the set of segments with endpoints at v and s. For each face

$f \in F$, denote $Sk_{vs}(f)$ the set of pairs $(t,u) \in \mathcal{R}$ such that the segment $s_{t,u}$ intersects f, this set is named the image of f by Sk_{vs}.

We denote $U = \cup_{f \in F} Sk_{vs}(f) \subset \mathcal{R}$, the union of all the images of the faces of F by Sk_{vs}. Pairs in U correspond to segments from v to s that intersect at least one face of F. Consequently, pairs in $\mathcal{R}$ not covered by U correspond to *non blocked* segments, i.e. segments with endpoints at v and s that do not intersect any face of the terrain, and hence, the point of segment s is visible from the one of v. A point $s_{u_0} \in s$ is weakly visible from v if there exists at least one non blocked segment connecting s_{u_0} with v. This happens if and only if there is an existing pair $(u_0,t) \in \mathcal{R}$ contained in $\mathcal{R} - U$. Therefore, the weakly visible points of s can be obtained by computing $pr_u(\mathcal{R}-U)$, which is the projection of the pairs not contained in U on the $u-$axis. On the other hand, point $s_{u_0} \in s$ is not strongly visible from v if there are blocked segment from s_{u_0} to v. This happens if and only if there are already pairs $(u_0,t) \in \mathcal{R}$ contained in U. Consequently, the strongly visible parts of s are the parts of $[0, l_s]$ not contained in $pr_u(U)$, or equivalently the not strongly visible parts are given by $pr_u(U)$ (see Figure 4.2). These results are stated in Lemma 4.2.2.

Lemma 4.2.2 *The strongly and weakly visible points of s are given by $pr_u(U)$ and $pr_u(\mathcal{R}-U)$, respectively.* □

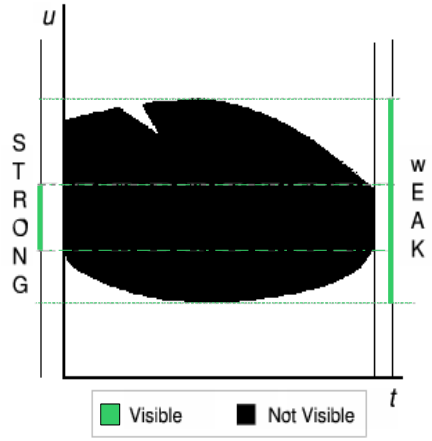


Figure 4.2: A union U painted white over a black background. On the left, the image is horizontally projected to obtain the strongly visible parts of s. On the right, the projection is done according to the definition of weak visibility. Grey subsegments correspond to the visible parts.

In Subsection 4.2.1.1, we propose an approximate method to compute U with a simple and easy-to-program algorithm that avoids analytic computations and robustness problems

common in geometric algorithms.

4.2.1.1 Visibility computation using graphics hardware

In order to compute the union U associated to a pair of segments s and v, we discretize into a grid of $w \times h$ pixels the bounded region $\mathcal{R}$ maintaining a specific number of pixels, μ', per unit of length. The value μ' is a parameter and affects the accuracy of the obtained results, the bigger the μ' the lower the error. We render each region $Sk_{vs}(f)$, by using graphics hardware, on a rectangle of the frame buffer representing the grid. If a single frame buffer does not suffice to represent all the grid, or equivalently if the grid resolution, $\mu' l_e \times \mu' l_v$, exceeds the frame buffer resolution, we partition $\mathcal{R}$ in subrectangles so that each part can be represented in the frame buffer. In such a case the region of U contained in each part is computed independently, making necessary more than one rendering step to compute the whole union U.

Regions $Sk_{vs}(f)$ are painted white on an initial black background. To paint $Sk_{vs}(f)$ we proceed as follows. For each face f in F we render the whole rectangle $[0, l_v] \times [0, l_s]$ obtaining wh fragments. For each fragment, whose associated pixel location defines a segment with endpoints on v and s, we use a fragment shader to check whether the segment intersects f. If this is so, the fragment is colored white and discarded otherwise, thus, white fragments correspond to $Sk_{vs}(f)$. Thus, the final result is that pixels contained in at least one $Sk_{vs}(f)$ are colored white and the rest are black, whenever the screen was initially black. Union U may involve up to $O(n)$ faces, thus, the time complexity of this part of the algorithm is the one provided in the next Lemma 4.2.3. In it the time needed to render a pixel is referred as the rendering time. Notice that w and h are taken proportional to the length of segments s and v, and therefore they are, in general, much smaller than M and N which are the size of the initial grid discretizing the terrain domain.

Lemma 4.2.3 *For a given face f, $Sk_{vs}(f)$ is rendered in $O(wh)$ rendering time. Union U is rendered in $O(nwh)$ worst case rendering time.m* □

To determine whether segment $s_{t,u}$ intersects face f, which is a triangle, we send the vertices of f, and the origin and direction vectors of segments v and s to the fragment shader. In the fragment shader: first, we use the position of each fragment on the screen, its coordinates in $\mathcal{R}$, to determine t and u; second, and by using the information related to v and s we compute v_t and s_u, the segment endpoints; finally, we check whether the

segment intersects f, by using a strategy similar to the one used in rendering techniques to handle ray tracing problems in the GPU (Section 2.3.4). The whole process is fast thanks to the GPU parallelism capabilities.

After painting all the regions, the visible parts of s are obtained by adequately projecting the obtained image according to Lemma 4.2.2. Hence, the subsegment of s contained in a pixel is weakly visible if in the corresponding row there is a black pixel, and it is strongly visible if there are no white pixels in it (Figure 4.2). Projections can be obtained transferring the image to the CPU (using the OpenGL *glReadPixels* function) and keeping track of the black pixels in each row. However, it is faster to obtain using the following reduction-type algorithm (Section 2.3.3.3). The initial window $w \times h$ is progressively reduced to a column $1 \times h$ where visibility information is projected. At step i for each row, we store at column i a white(/black) pixel if at columns $2i$ or $2i+1$ there is a white(/black) pixel when considering weak(/strong) visibility. After $\log w$ steps, we have a single column with the desired projection. This column contains the visibility information of s and is transferred to the CPU (using the OpenGL *glReadPixels* function). In the next Lemma we present the time needed to obtain the visible parts of s, where, the time needed to transfer a pixel from the GPU to the CPU is to referred as the reading time.

Lemma 4.2.4 *Visibility information on s can be obtained after rendering U in $O(2hw)$ rendering time and $O(h)$ reading time.*

Proof. Concerning rendering time, we have to realize $\log w$ rendering steps. At step i we render a rectangle of size $\frac{w}{2^i} \times h$. Consequently, the total number of rendered pixels is $wh + \frac{wh}{2} + \frac{wh}{4} + \cdots + \frac{wh}{2^{\log w}} = \sum_{i=0}^{\log w} \frac{wh}{2^i} = wh \sum_{i=0}^{\log w} \frac{1}{2^i}$ which is a harmonic series with scale factor $2wh$ and $wh \sum_{i=0}^{\log w} \frac{1}{2^i} = 2wh$. Concerning the reading time, we have to read a single column of length h, this is $O(h)$ time. □

Notice that when the grid resolution, $\mu' l_e \times \mu' l_v$, exceeds the frame buffer resolution and $\mathcal{R}$ is partitioned in subrectangles, union U is obtained in different parts that need to be collectively projected.

4.2.1.2 Acceleration

We avoid repeating some face dependent operations by grouping, for each face f_0 of the terrain, the set of segments S_{f_0} on f_0 induced by the grid columns covering the region

of the domain associated to f_0. Before studying the visibility of the segments in S_{f_0} we need to: 1) delete the part of the view segment v whose points do not front-face f_0 (we still denote v the remaining part of the view segment); 2) compute F_{f_0}, the set of terrain faces that intersect $\mathcal{H}_{f_0}$, the convex hull of f_0 and v. Note that F_{f_0} are the faces that may prevent v from seeing f_0. Set F_{f_0} is contained in the set of faces whose projection onto the xy-plane intersects the projection of $\mathcal{H}_{f_0}$. This last set can be efficiently obtained by using the algorithm provided in Section 2.2.6. Given view segment v and set F_{f_0} we study the visibility of segments $s \in S_{f_0}$ using the segment-segment visibility algorithm.

Lemma 4.2.5 *Given a view segment v the time needed, during the whole algorithm, to compute the faces that may prevent v from seeing a segment s is $O(n^2)$.*

Proof. We obtain a set F_f of faces that may prevent v from seeing a specific segment s for each terrain face f. The time needed to obtain such a set F_f is, in the worst case, $O(n)$, the terrain has n faces, thus the time needed in computing faces is $O(n^2)$ in the worst case. □

To study the segment-segment visibility for a pair of segments (v and s with $s \in S_{f_0}$) we paint the corresponding $U = \cup_{f \in \mathcal{F}_{f_0}} Sk_{vs}(f) \subset \mathcal{R}$ white on an initial black frame buffer, i.e. we use a true/false code that can be stored in a bit. Since each pixel of the frame buffer has 48 color bits (16 bits per three color channels), we can simultaneously represent the unions corresponding to a collection of pairs of segments (v, s_1) ,, (v, s_k), with $1 \leq k \leq 48$ in it, using a different bit for each pair. To merge the color of the already rendered fragments (unions) with the incoming ones we use the per-fragment *glLogicOp* operation, specifying the *or_logic* option. Observe that the use of *or_logic* and the black color of the pixels not contained in the regions guarantee that the union is properly painted. Then we use the explained reduction-type algorithm by using a fragment shader in the rendering steps to appropriately merge the bits. Thus by using this observation, Lemma 4.2.3 and Lemma 4.2.4 we can provide the following result.

Lemma 4.2.6 *The complexity of obtaining the visible parts of a cluster of 48 segments from a view segment is $O(48\ n\bar{h}\bar{w} + 2\bar{h}\bar{w})$ worst case rendering time and $O(\bar{h})$ reading time, where $\bar{w}$ and $\bar{h}$ are the maximal used width and height.* □

These considerations do not contradict the fact that the algorithm can be parallelized. This can be done by using the fact that the visibility study of each cluster of 48 pairs

of segments is independent. Thus, the process can be accelerated by parallelizing these computations.

4.2.2 Computational cost

Let us consider a terrain $\mathcal{T}$ with n triangles and a set of r view segments, let R be the total number of view segments conforming the r input view elements, $r \leq R$. Assume that the grid discretizing the terrain domain is of size $M \times N$. We assume that the hidden constant in the reading time is K, the constant hidden in the rendering time is not specified as it is much smaller. Then we can provide the following Theorem:

Theorem 4.2.1 *Visibility information concerning terrain $\mathcal{T}$ and the set of view elements V is obtained in $O(R(n^2 + nN\bar{h}(n\bar{w} + \frac{1}{48}(\bar{w} + K)) + MN))$.*

Proof. Let us first consider a single view segment v. The total number of vertical segments determined by the terrain faces is $O(nN)$ in the worst case. The total time needed to obtain the faces that may prevent a view segment from seeing segment s is $O(n^2)$ (Lemma 4.2.5). The time needed to obtain all the unions U is $O(Nn^2wh)$ worst case rendering time, while the unions are projeted once for each cluster of 48 segments. Consequently this takes $O(\frac{Nn}{48}(\bar{h}\bar{w} + \bar{h}K))$ considering both the rendering and the reading worst case time (Lemma 4.2.6). Multi-visibility information is stored in the CPU in an array of size $M \times N$ which is updated for each view segment and this is done in $O(MN)$ time time. Thus the time needed for each view segment is $O(n^2 + nN\bar{h}(n\bar{w} + \frac{1}{48}(\bar{w} + K)) + MN)$. Since we have R view elements, the worst case total time is $O(R(n^2 + N\bar{h}(n\bar{w} + \frac{1}{48}(\bar{w} + K)) + MN))$. □

If we consider a realistic terrain (Section 2.4.3) the number of faces intersected by a line is $O(\sqrt{n})$. Therefore the multi-visibility map is obtained in $O(R(n^2 + \sqrt{n}N\bar{h}(n\bar{w} + \frac{1}{48}(\bar{w} + K)) + MN))$.This is stated in the following Proposition.

Proposition 4.2.1 *Visibility information concerning a realistic terrain $\mathcal{T}$ and the set of view elements V is obtained in $O(R(n^2 + \sqrt{n}N\bar{h}(n\bar{w} + \frac{1}{48}(\bar{w} + K)) + MN))$* □

Notice if we are not interested in storing the visibility information in the CPU we can directly store it in a texture in the GPU by using an appropriate fragment shader. Once the projection of a cluster of 48 unions has been done, it can be transferred to a

$M \times N$ texture. We do not proceed in this way because it is quite complicated, due to: the clustering of 64 bits we consider; the scale factor ($\mu' l_s \times \mu' l_v$ which is the resolution where the visibility on s is obtained and it is completely independent from $M \times N$); the fact that we have to "rebuild" the view elements from the view segments, etc.

4.3 Multi-visibility maps visualization

After transferring the visibility map of the r view elements of V to textures in the GPU, which is done in $O(rMN)$ time, a fragment shader is used to visualize, on the screen, any multi-visibility map of any specified region of the terrain in a window. The fragment shader obtains the textures, the type of multi-visibility map desired and the number of view elements in V as input parameters.

Depending on the grid dimension and the window size, each pixel may contain more than one grid position or, each grid position may correspond to more than one pixel. The maximal precision in a multi-visibility map is obtained when each pixel represents a grid position, i.e. the grid and screen resolution match. When the grid resolution is exceeded, several pixels represent the same grid position and when it is not reached only some information is visualized. To color each pixel we use the information stored in the grid position containing the pixel center.

Next we describe how the union, intersection and count maps can be obtained. To obtain the union map, the fragment shader colors a pixel whose center corresponds to a position of the texture storing a value with a bit equal to 1 in white (visible) and in black otherwise (not visible). The intersection map is obtained coloring a pixel white corresponding to a position of the texture when its value has all the used bits equal to 1 (visible) and black otherwise (not visible). Notice that when we have $k \leq r$ view elements of V in a texture, $1 \leq k \leq 64$, the fragment shader only uses the corresponding k bits of the texture values to compute the union and intersection maps. To obtain the count map it colors each pixel in the appropriate color according to the number of bits equal to 1 of the value stored in the corresponding position of the texture.

Visualizing the union, intersection and count maps for a subset of view elements or also the overlay map of subsets of V is not difficult. Since in the screen we can only represent eight bits per color channel, the maximum size of the subset is 24 to guarantee that each region of the overlay has a different color. The overlay map of such a subset

of V is obtained by associating a color to each view element of the subset, painting each visibility map accordingly and obtaining the merged image. The visibility map of each single view element is obtained by coloring a pixel if its corresponding texture value has the bit associated to the view element equal to 1. Therefore, we have to send the bit we have to check and the corresponding texture as input parameters to the fragment shader. As we have said, once all the visibility maps have been painted, different colors are merged and each pixel is colored accordingly. This merged image is obtained by using the *glLogicOp* operation specifying the *or_logic* constant and painting the visibility map of each view element with its color on an initial black background. As an example in Section 4.5 we can find the overlay of three view elements (a view point, a view segment and a view polygonal chain) represented by using different gray levels in Figure 4.11.

Proposition 4.3.1 *Once the visibility information is transferred to the GPU in* $O(\frac{r}{64}MN)$ *time, the time needed to obtain any desired multi-visibility map in a window of size* $H \times W$ *is* $O(HW)$. □

Notice that the time needed to visualize a multi-visibility map depends on the number of pixels of the window where it is visualized and the time needed by the fragment shader. This time is not only independent from the number of terrain faces and view elements, but also from the dimensions of the grid covering the domain.

A zoom option can be easily handled. A region of interest of the domain can be chosen and any of its multi-visibility maps can be visualized by using the whole screen. Moreover, once we have obtained a multi-visibility map on the terrain domain, we can transfer the image in a texture and use this information to paint the multi-visibility map on the terrain, in $\mathbb{R}^3$. Each terrain point is painted according to the color stored in the corresponding texture position.

4.4 Multi-visibility queries

We can be interested in obtaining multi-visibility information of a specific point of the terrain, in answering a point multi-visibility query. Among the point multi-visibility queries we can answer, there are the following ones: is the point visible from every view element?, is the point visible from a view element?, how many view elements is the point visible from?, which view elements is the point visible from?, is the point visible from a specific view element?, etc. Multi-visibility point queries are answered by considering a suitable

multi-visibility map. For example, in order to answer the previously mentioned queries the multi-visibility maps to be considered are: the intersection map, the union map the, count map, the overlay map and the visibility map of the specific view element, respectively.

Each multi-visibility query can be answered according to the information visualized on the screen, or with maximal precision according to the visibility information stored in the $M \times N$ initial grid. To obtain the answer we use the *glReadPixels* OpenGL function or we find the grid position containing the queried point, respectively. In both cases, the time needed to answer a point multi-visibility query is constant. However, notice that in the second case we do not need to visualize the multi-visibility map.

All the previous questions can also be answered when, instead of a point, a polygonal region is specified. However, the question may be focused on existence or quantity. For instance: is there any point in the region that is visible from every view element?, which percentage of the region contains points that are visible from every view element? When using the visualized information, the answer is obtained by reading the minimal axis-parallel bounding box covering the region with the *glReadPixels* OpenGL function (see Figure 4.3). When using the information stored in the initial grid, all the grid positions contained in the region are considered. Therefore, polygonal region multi-visibility queries can be answered in $O($ number of pixels in the window $)$ time, by using the visualized information, or in $O(MN)$ worst case time, by using the information stored in the initial grid.

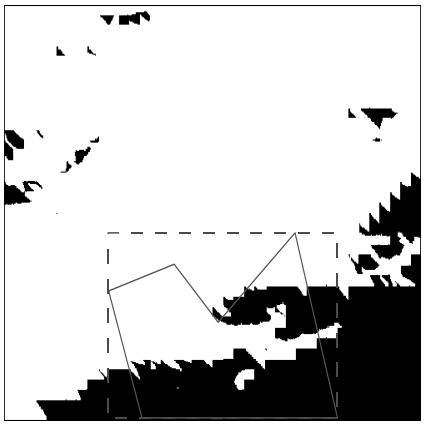


Figure 4.3: The union map and a query polygonal region in grey. The minimum axis-parallel bounding box in a dashed grey0 line.

This process can also be used to solve inter-region multi-visibility problems on terrains

using the strategy provided in Section 3.6 and the strategy to compute visibility by using graphics hardware presented in Section 4.2.

4.5 Implementation and experimental results

The algorithms proposed in this chapter have been implemented in C++ and OpenGL. In this section, we provide some experimental results to show the good performance and efficiency of these algorithms. All tests have been carried out on an Intel(R) Pentium(R) D at 3GHz with 1GB of RAM and a GeForce 7800 GTX/PCI-e/SSE2 graphics board.

Our implementation handles heterogeneous sets of view elements containing view points, segments, polygonal chains and polygons. The user can choose μ (specifying the size of the initial grid), and μ' (specifying the length per pixel used to compute the visibility information). By default, μ' is the number of grid positions per unit length induced by the initial grid. The implementation, that handles a zoom option, allows for the visualization of the union, intersection, count and overlay (of one, two, three or all the view elements, if there are less than 24 view elements) maps on the domain or in $\mathbb{R}^3$ on the terrain. It also answers multi-visibility queries.

In Tables 4.1, 4.2 and Figure 4.4 , we analyze the computational complexity of these algorithms according to experimental results. We start analyzing the complexity of the preprocessing stage by checking the relationship between time and the values of n, r and $M = N$, the number of faces, view segments and columns of the grid discretizing the domain, respectively. Then, we analyze the time needed to visualize a multi-visibility map.

Preprocessing stage

In Table 4.1 and Table 4.2 we give the number of pairs of segments and triangles considered, as well as the colored and read pixels, in order to show the number of operations done. Table 4.1 presents the time used in the preprocessing stage for several terrains with different n values, two view segments and several $M \times N$ grids. Table 4.2 refers to terrains with $n = 800$ and $n = 3200$, a grid of size 500×500 and varying r. We can also find in the Table 4.1 and Table 4.2 the number of triangles sent to render, as well as the number of colored and read pixels. In Figure 4.4, where the time spent in the preprocessing

stage is compared to n considering several grid dimensions, we observe that time depends quadratically on n, as was shown (Section 4.2.2). In the three cases the adjusted coefficient of multiple determination for the quadratic fit is 0.999. Figure 4.4 is obtained using the results presented in Tables 4.1 and 4.2.

In Tables 4.1 and 4.2 we can see that the number of studied pairs has the same behavior as the number of read pixels, and the number of considered triangles the same as the colored pixels. Notice that, in the results obtained with the scenes considered, the number of studied pairs and read pixels decreases when n increases from 7200 to 12800. This happens because we only study those segments on faces that do face the view segment. Thus, the number of studied pairs does not depend directly on n, but also on the position of the view segment.

Table 4.1: Preprocessing stage tests. In this table we show SS the number of segment-segment visibility computations, T painted triangles, $Cpix$ colored pixels, $Rpix$ read pixels and execution time in the preprocessing stage for two view elements and various terrains depending on the number n of terrain faces and the size $N \times N$ of the initial grid covering the domain.

n	N	SS	T ($\times 10^6$)	$Cpix$ ($\times 10^6$)	$Rpix$ ($\times 10^6$)	time (s)
800	500	18300	0.113824	192.855	2.219	7
800	1000	36600	0.229899	385.571	3.74102	9
800	4096	150508	0.950526	1578.89	11.3329	20
3200	500	34957	0.700112	1436.73	5.28961	35
3200	1000	67225	1.40049	2873.65	9.69502	42
3200	4096	277509	5.84242	11750.7	27.9186	82
7200	500	73485	2.06396	5342.64	13.5641	168
7200	1000	141539	4.1712	10686.5	26.5291	187
7200	4096	565035	17.2496	43770.1	79.2885	319
12800	500	62727	2.44284	7667.97	12.0897	560
12800	1000	116493	4.88513	15328.4	24.7562	576
12800	4096	465972	20.3909	62868.7	76.1341	743
20000	500	105465	4.51306	16947.0	26.3929	1303
20000	1000	210930	9.51831	33831.4	55.2668	1336
20000	4096	884653	40.5104	133442	178.058	1686

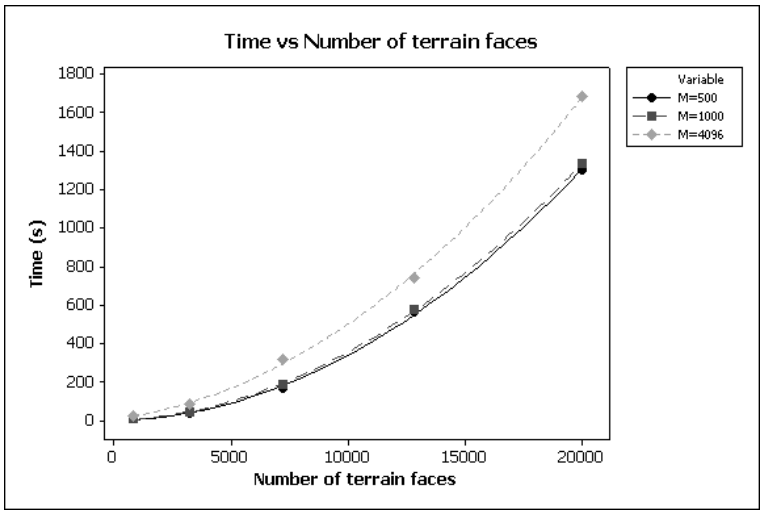


Figure 4.4: Execution time versus number of terrain faces for several grid dimensions.

Multi-visibility map visualization

Finally some multi-visibility maps are shown. With this aim, we use the terrain and the three view elements (a point, a segment and a polygonal chain) depicted in Figure 4.5. For the segment and polygonal chain we use the weak visibility criterion. The results are visualized on the domain and also in $\mathbb{R}^3$ on the terrain. To facilitate the understanding of $\mathbb{R}^3$ figures we use illumination. Consequently, the image colors are slightly altered according to the light position.

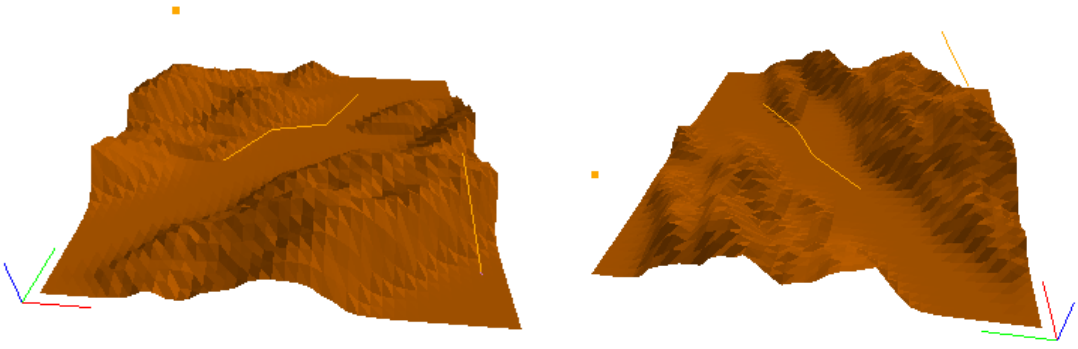


Figure 4.5: Two different views of the considered terrain with three view elements over the terrain: a point, a polygonal chain and a segment

The union map is given in Figure 4.6, where visible points from at least one view

Table 4.2: Preprocessing stage tests. In this table we show SS the number of segment-segment visibility computations, T painted triangles, $Cpix$ colored pixels, $Rpix$ read pixels and execution time in the preprocessing stage for various terrains depending on the number n of terrain faces and the number r of view segments considering a 500×500 grid covering the domain.

n	r	SS	T ($\times 10^6$)	$Cpix$ ($\times 10^6$)	$Rpix$ ($\times 10^6$)	time (s)
800	1	12900	0.144298	167.519	1.07429	4
800	5	37975	0.317551	363.719	3.01567	15
800	10	81950	0.667980	672.105	5.76114	28
800	15	142575	1.13747	1294.02	11.405	46
800	20	203250	1.79864	1915.43	15.6315	60
3200	1	15509	0.303405	488.587	1.85078	16
3200	5	98683	1.96699	3828.61	14.2134	94
3200	10	201279	4.14537	7751.3	26.6047	188
3200	15	311415	6.50442	11861.2	39.9502	296
3200	20	435058	9.77866	20508.8	62.3632	422

element are painted lighter and the not visible ones darker. The intersection map is given in Figure 4.7, where lighter points are visible from every single view element.

Figure 4.8 shows the count map, where points are colored in a white to black gradation according to the number of view elements they are visible from (0 black - 3 white).

The visibility map of the view polygonal chain is shown in Figure 4.9, where points weakly visible from the view polygon are colored lighter. A two view elements (the segment and the point) overlay map is given in Figure 4.10 where the points are colored according to the subset of view elements they are visible from. The three view elements overlay map is given in Figure 4.11, where, again, the points are colored according to the subset of view elements they are visible from.

With our implementation, we achieve the visualization of the union, intersection, count, overlay of one view element and overlay of all the view elements maps in 0.006 seconds. The overlay map of two view elements takes 0.011 seconds and of three view elements 0.017 seconds. Once the information is computed and transferred to the GPU, the time depends on the fragment shader and the number of images we have to merge. Notice that when obtaining overlays of more than one view element, the number of merged images is more than one, except for the overlay of all the view elements where all the information stored in the texture is used.

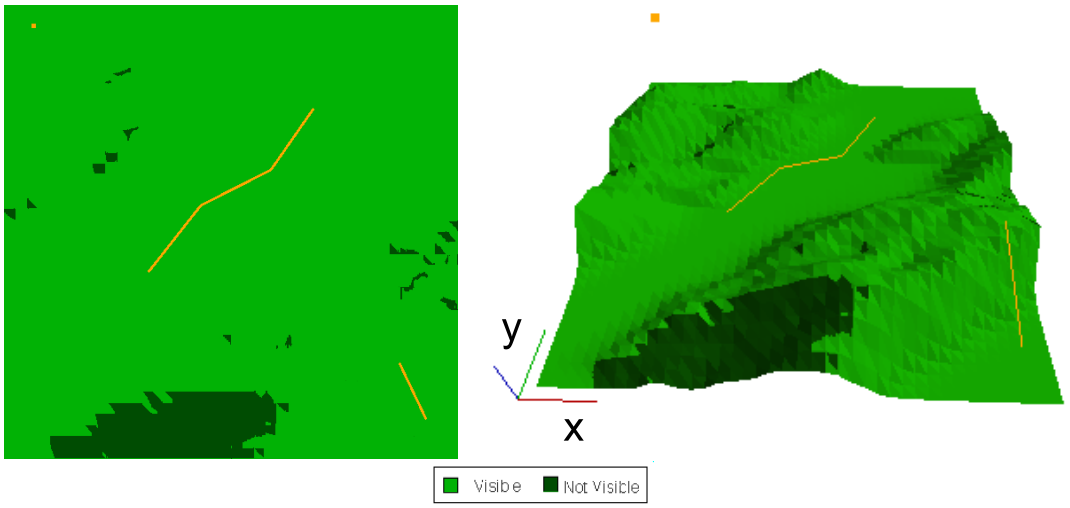


Figure 4.6: On the left, the union map of a view point, a view segment and a view polygonal chain; the points of the domain are colored according to whether they are visible from at least one view element or not; the projection of the view elements is also shown. On the right, the view elements and the union map on the terrain are shown.

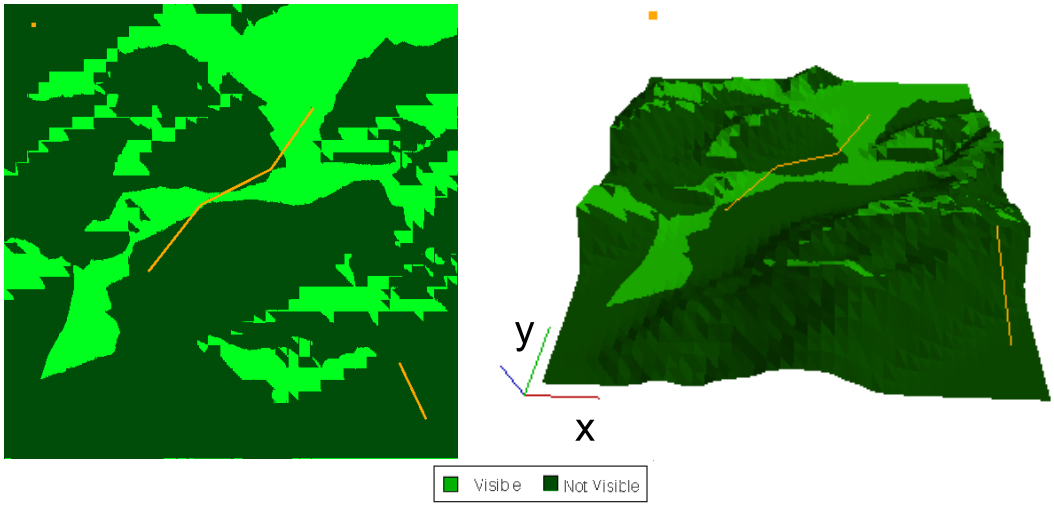


Figure 4.7: On the left, the intersection map of a view point, a view segment and a view polygonal chain; the points of the domain are colored according to whether they are visible from all three view elements or not; the projection of the view elements is also shown. On the right, the view elements and the intersection map on the terrain are shown.

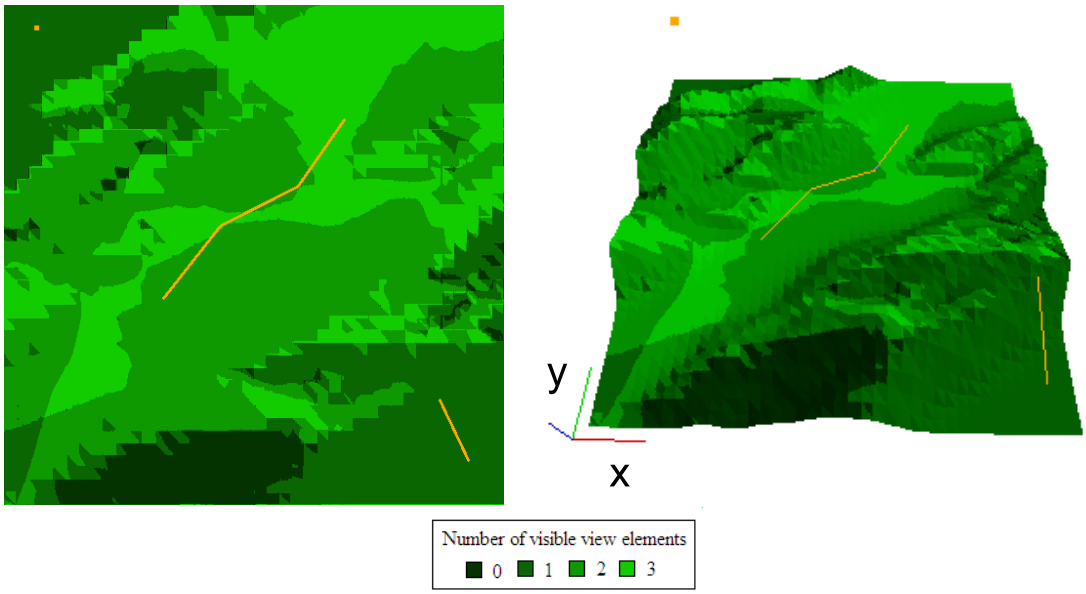


Figure 4.8: On the left, the count map of a view point, a view segment and a view polygonal chain; the points of the domain are colored according to the number of view elements they are visible from; the projection of the view elements is also shown. On the right, the view elements and the count map on the terrain are shown.

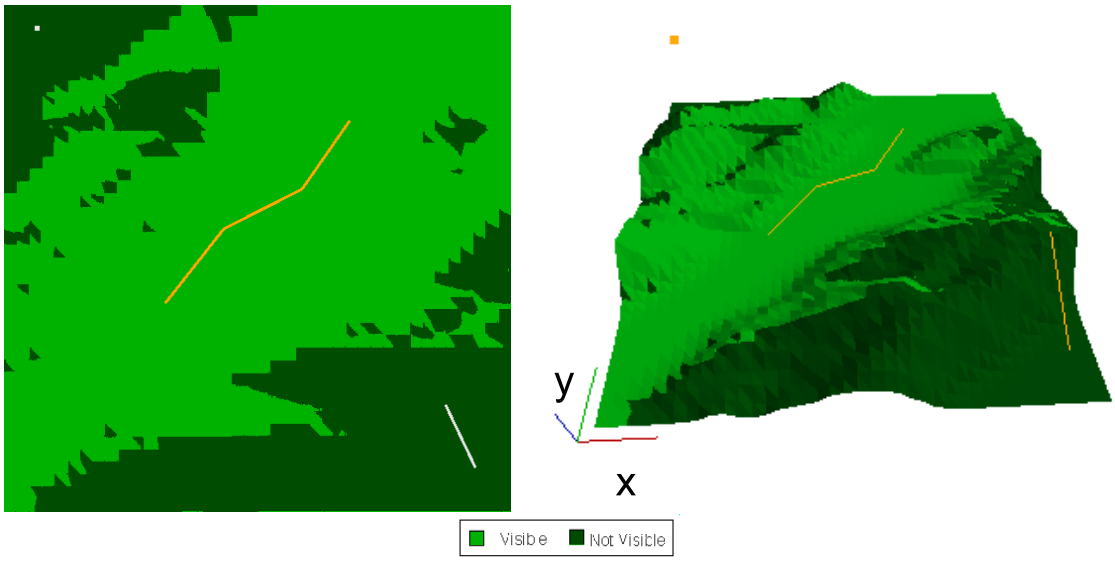


Figure 4.9: On the left, the visibility map of the view polygonal chain; the points of the domain are colored according to whether they are visible or not; the projection of the view polygonal chain is also shown. On the right, the polygonal chain and its visibility map on the terrain are shown.

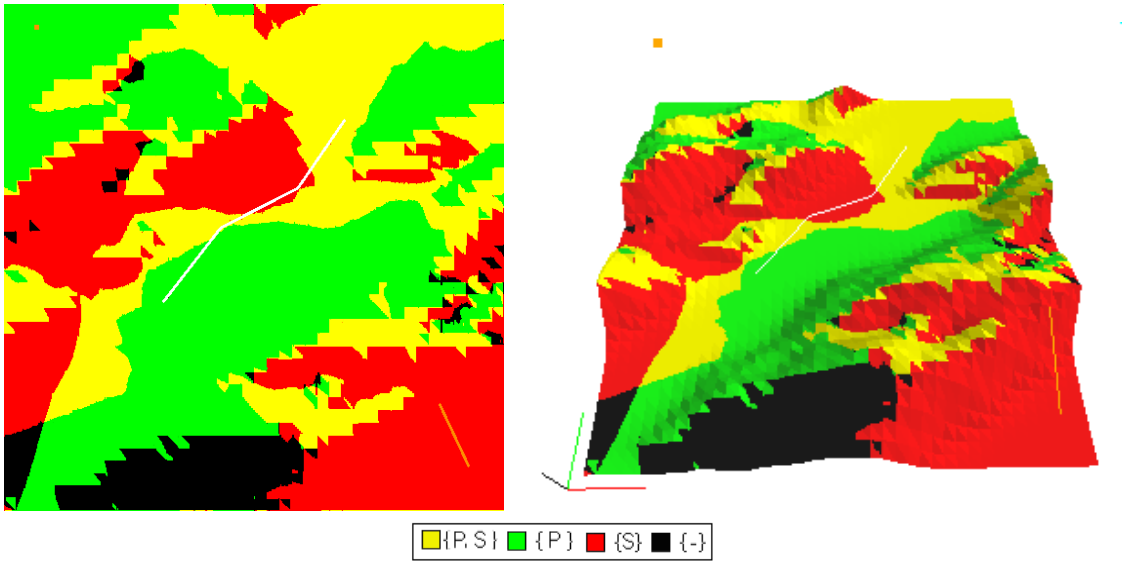


Figure 4.10: On the left, the overlay map of the view point, P, and the view segment, S; the points of the domain are colored according to the view elements they are visible from; the projection of P and S is also shown. On the right, P, S and their overlay map on the terrain are shown.

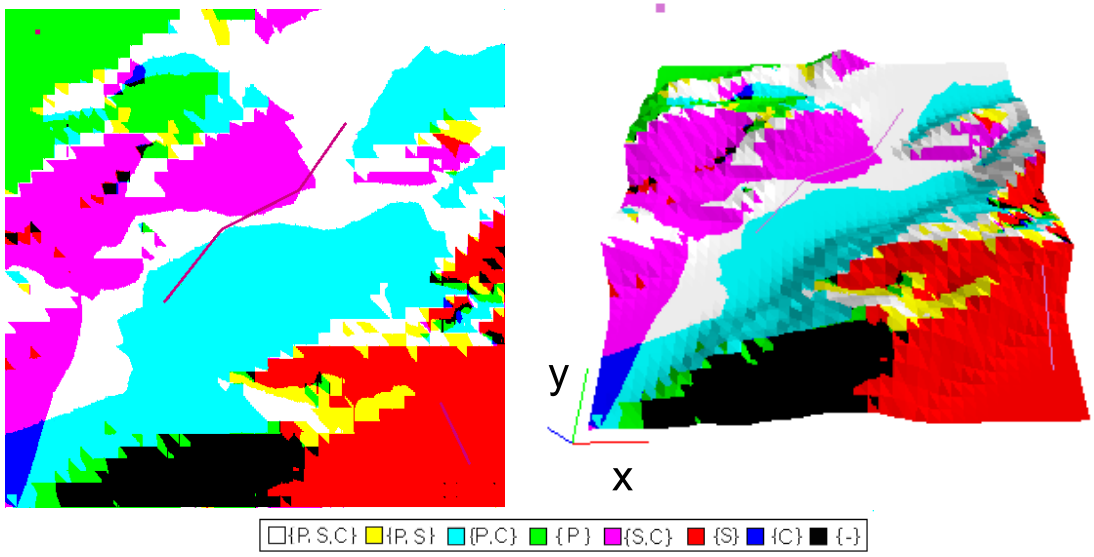


Figure 4.11: On the left, the overlay map of the view point, P, and the view segment, S, and the view polygonal chain L; the points of the domain are colored according to the view elements they are visible from; the projection of the view elements is also shown. On the right, the view elements and their overlay map on the terrain are shown.

4.6 Conclusions

We have presented a method to visualize multi-visibility maps of a triangulated terrain containing an heterogeneous set of view elements (points, segments, polygonal chains and polygons) for weak and strong visibility. We compute approximate visibility information in a pre-processing stage by repeatedly using graphics hardware and then from this information visualize any multi-visibility map on the screen with a zoom option. We also answer point and polygonal region multi-visibility queries. The results obtained with our implementation show that our approach is practical, robust and efficient.

Chapter 5

Distances on polyhedral surfaces

We present an exact algorithm for computing shortest paths, and consequently distances, from a generalized source (point, segment, polygonal chain or polygonal region) on a triangulated non-convex polyhedral surface in which polygonal chain or polygon obstacles are allowed. It extends the ideas developed by Surazhsky et al. [144] for the implementation of the algorithm by Mitchell et al. [104] to the case of generalized sources and a polyhedral surfaces with obstacles. In this chapter, we present an algorithm to obtain an implicit representation of the distance function defined by shortest paths on $\mathcal{P}$ to a generalized source (Section 5.1). The algorithm easily extends to the case of several sites providing their distance field, which intrinsically encodes the closest Voronoi diagram of the set of generalized sites (Section 5.2). From the implicit representations we obtain the distance or the shortest path to any point $p \in \mathcal{P}$ (Section 5.3). Finally we give a theoretical study on the complexity of higher-order Voronoi diagrams of a set of generalized sites on a non-convex polyhedral surface assuming non-degenerate position (Section 5.4). The chapter ends with some conclusions (Section 5.5). The algorithms described in this chapter have been implemented. In Chapter 7 we provide execution times for different terrains, which are special triangulated surfaces, and sets of generalized sites.

We consider $\mathcal{P}$ a polyhedral surface represented as a mesh consisting of n triangular faces and a generalized element on $\mathcal{P}$ playing the role of a source s. Obstacles on $\mathcal{P}$ are modelled by a set of non-punctual generalized elements on $\mathcal{P}$. A shortest path from a generalized source to a point on $\mathcal{P}$ is a path from the source and the point with minimal length such that the path stays on $\mathcal{P}$ and avoids the obstacles. The shortest path distance function defined by a source point p on $\mathcal{P}$ is a function d_p such that for any point $q \in \mathcal{P}$,

$d_p(q)$ is the Euclidean length of the shortest path along $\mathcal{P}$ from q back to point p. The shortest path distance function defined by a generalized source s is a function d_s such that for any point $q \in \mathcal{P}$, $d_s(q)$ is the length of the shortest path from q back to source s.

5.1 Implicit distance function

A quick overview of our algorithm to compute shortest path distances from a generalized source on a polyhedral surface with obstacles is provided first. The shortest path problem from a generalized source s is solved by tracking together groups of shortest paths and partitioning each triangle edge into a set of intervals. All shortest paths that cross an interval are encoded locally using a parameterization of the distance function d_s. After an initialization step, where intervals encoding d_s in the edges of the triangles containing s are created, the distance function is propagated across mesh triangles in a "continuous Dijkstra" fashion by repeatedly using an interval propagation process. A complete intrinsic representation of d_s is obtained when the propagation process ends. From this representation, the shortest path from any point q to source s is computed by using a "backtracing" algorithm.

5.1.1 Point and segment sources

A point can be considered as a degenerated segment whose endpoints coincide and consequently, an algorithm for computing the distance function of a segment source can be used for the special case of a point source. Consequently, we center our study on segment sources. We start by providing some properties of shortest paths for segment sources and next explain the algorithm steps.

Lemma 5.1.1 *Shortest paths from a segment source s to any destination point q fulfill the following four properties:*

1. *In the interior of a face a geodesic is a straight line*
2. *When two neighboring faces are unfolded in a common plane, a geodesic becomes a straight line.*
3. *A geodesic can only go through a vertex v if it is a boundary vertex or if its total angle is bigger than 2π.*

4. *A geodesic starting at an interior point of s is orthogonal to s in f, where f is a face of $\mathcal{P}$ containing s.*

Proof. The shortest path distance function defined by a segment source s is given by

$$d_s(q) = \min_{p \in s} d_p(q).$$

Since s is a closed interval and $d_s(q)$ is a continuous function the minimum is reached at a point of s. Consequently these shortest path are shortest paths to a point source and necessarily fulfill Property 2.6.1, Property 2.6.2 and Property 2.6.3, which correspond to the first three points. Finally the fourth property characterizes the fact that each point is connected with the closest point of segment s, thus a path obeys the the properties of shortest paths to segments in the plane. □

To compute the distance function d_s for a segment source s we track together groups of geodesic paths by partitioning the edges of $\mathcal{P}$ into intervals. Geodesic paths that cross an interval go through the same triangles and bend at the same vertices of $\mathcal{P}$. In an initialization step, intervals defining d_s in the triangle(s) containing s are created. Then the distance function is propagated across triangles in a Dijkstra-like sweep in such a way that, over each interval, d_s can be represented in a compact way by using an appropriate codification.

5.1.1.1 Distance function codification

The distance function and the shortest paths defining it are encoded and propagated by using intervals, subsegments of the edges of $\mathcal{P}$. We use two different types of intervals: 1) intervals associated to the interior of the segment source s; 2) intervals associated to either an endpoint of s or a vertex of $\mathcal{P}$.

Consider a shortest path from the source segment s to some point q on an edge e, and let us assume that this path does not bend at any mesh vertex. When all the triangles intersecting the path are unfolded in a common plane, the path forms a straight line (Lemma 5.1.1). The set of neighboring points of q on e whose shortest paths back to s pass through the same sequence of triangles form: a) a pencil of rays emanating from an endpoint of s; b) a pencil of orthogonal rays emanating from the interior of s (see Figure 5.1). In both cases we represent the group of shortest paths over an interval on the edge e.

Assume now that the shortest path from $p \in s$ to q bends at one or more vertices on its way to the source s, and let v be the nearest such vertex to q. Consider the set of neighboring points on e whose shortest paths back to v go through the same strip of triangles. In the unfolding of the strip between e and v, these shortest paths will form a pencil of rays emanating from the *pseudosource* vertex v, as seen in Figure 5.1. As before, we represent this group of shortest paths over an interval on the edge e. Consequently, we can provide the following Property.

Property 5.1.1 *Shortest path emanating from a segment source s can be grouped by using intervals on the edges of $\mathcal{P}$.*

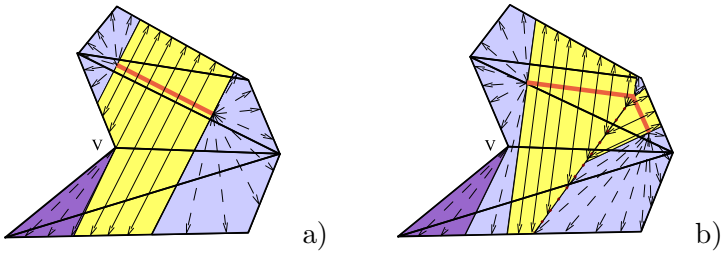


Figure 5.1: An unfolded strip of triangles with: a) a segment source; b) a polygonal chain source.

Intervals representing a group of geodesics emanating from a punctual source p (an endpoint of s, a punctual source, or a mesh pseudosource vertex) are called *p-intervals*, and intervals representing a group of geodesics emanating from interior points of s are called *s-intervals*. Intervals not only group shortest paths, but also encode them.

p-intervals The group of geodesics associated to a p-interval I originated at a point p (an endpoint of s or a pseudosource vertex) is locally encoded by using a 6-tuple $(b_0, b_1, d_0, d_1, \sigma, \tau)$. Where b_0, $b_1 \in [0, |e|]$ measure the Euclidian distance from the endpoints of I to the origin (the lexicographically smallest endpoint) of e. Distances d_0 and d_1 measure the Euclidean distance from the endpoints of I to p, direction τ specifies the side of e on which p lies, and σ encodes the distance from p to s.

Lemma 5.1.2 *A source p defining a p-window can be positioned in $O(1)$ on the plane of a triangle t containing I by using the 6-tuple $(b_0, b_1, d_0, d_1, \sigma, \tau)$. The distance function within I can be recovered in $O(1)$ time.*

Proof. Point p is positioned by simulating the planar unfolding adjacent to e in the rectangular coordinate system $\mathcal{S}_e$ that aligns e with the x-axis as it is shown in Figure 5.2 a). To obtain the position of $p = (p_x, p_y)$ we have to solve the following system

$$\begin{cases} (p_x - b_0)^2 + p_y^2 = d_0^2 \\ (p_x - b_1)^2 + p_y^2 = d_1^2 \end{cases}$$

The solution is $p_x = \frac{b_1^2 - d_1^2 - (b_0^2 - d_0^2)}{2(b_1 - b_0)}$ and $p_y = \pm\sqrt{d_0^2 - (p_x - b_0)^2}$. The sign of s_y is chosen according to parameter τ. Once the source has been located the distance from source s to a point $q \in I$ is

$$d_s(q) = ||\overline{qp}|| + \sigma.$$

The operations involved in both processes take constant time. □

The point source obtained from $(b_0, b_1, d_0, d_1, \sigma, \tau)$ in the coordinate system $\mathcal{S}_e$ is referred as the *virtual source* of I and noted I^s.

s-intervals The group of geodesics associated to an s-interval I is locally encoded by using a 5-tuple $(b_0, b_1, d_0, d_1, \phi)$. Where $b_0, b_1 \in [0, |e|]$ are the distances from the endpoints of I to the origin of e, d_0 and d_1 measure the distance of the endpoints of I to s, finally the angle determined by e and the rays emanating from s is stored in $\phi \in [0, 2\pi]$.

Lemma 5.1.3 *Given an s-interval, the part s' of s from which geodesics to I emanate can be positioned from the 5-tuple $(b_0, b_1, d_0, d_1, \phi)$ in $O(1)$ time. The distance function within I can be recovered in $O(1)$ time.*

Proof. Segment s' is positioned by simulating the planar unfolding adjacent to e in the rectangular coordinate system $\mathcal{S}_e$ that aligns e with the x-axis as it is shown in Figure 5.2 b). According to the parameters stored in the 5-tuple and using $\mathcal{S}_e$, the virtual segment source is the segment delimited by point $(b_0 + d_0 \cos\phi, d_0 \sin\phi)$ and point $(b_1 + d_1 \cos\phi, d_1 \sin\phi)$, which are obtained by using basic trigonometry. The distance from s' to a point $q = (q_x, p) \in I$ is

$$d_s(q) = d(q, s') = d_0 + \frac{(d_1 - d_0)(x - b_0)}{b_1 - b_0},$$

this distance is computed by using Tales rule. The operations involved in both process take constant time. □

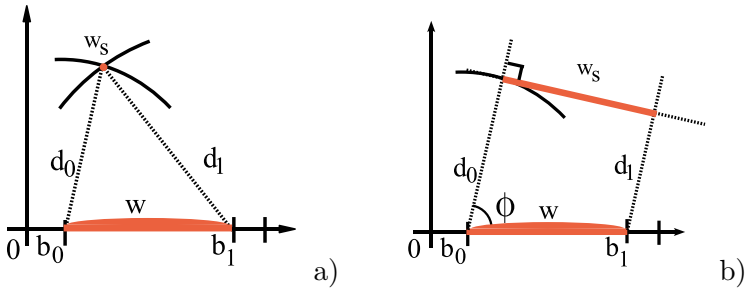


Figure 5.2: The virtual source s is positioned using the information stored in: a) a p-interval; b) a s-interval.

The obtained segment source is referred again as the *virtual source* of I and noted I^s.

According to Observation 5.1.1 shortest paths can be grouped by using intervals, Lemma 5.1.2 and Lemma 5.1.3 prove that the distance function can be recovered from the information stored in the intervals. Therefore, we can provide the following Proposition.

Proposition 5.1.1 *Distance function d_s can be encoded by using p-intervals and s-intervals.* □

5.1.1.2 Interval propagation

We propagate the distance function d_s encoded in an interval on an edge e to the next adjacent triangle t in the unfolding by creating new (potential) intervals on the two opposing edges of t. They are potential intervals because they may overlap previously computed intervals. Consequently, we must intersect the potential interval with previous intervals and determine the combined minimum distance function.

Lemma 5.1.4 *The distance function encoded in an interval I can be propagated in $O(1)$ time.*

Proof. Given a p-interval or s-interval I on an edge e, we propagate d_s by computing how the pencil of straight rays representing geodesics associated to I extends across one more unfolded triangle t adjacent to e. New potential intervals can be created on the opposing edges of t (see Figures 5.3 a) and b)). To encode d_s in a new potential interval on e' we first obtain the position of the source in the coordinate system $\mathcal{S}_e$. Then, we consider the rays emanating from the source through the endpoints of I to determine the

new interval $[b'_0, b'_1]$ on e'. New distances d'_0 and d'_1 from the interval endpoints to the source are computed. For p-intervals σ' does not change and for s-intervals the angle ϕ' is the angle defined by the rays and edge e'. When the interval I is adjacent to a saddle vertex v, geodesics may go through it. Vertex v is a new pseudosource and generates new potential p-intervals with σ' given by d_0 or d_1 when I is an s-intervals and σ' given by $d_0 + \sigma$ or $d_1 + \sigma$ when I is a p-intervals (see vertex v of Figure 5.3 c).

Since I_s is obtained in $O(1)$ time (Lemma 5.1.2 and Lemma 5.1.3), and computations involved in this process take constant time, the whole cost is $O(1)$.

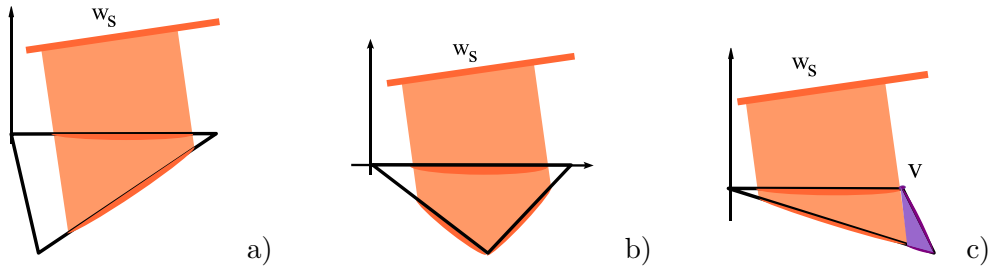


Figure 5.3: Examples of s-interval propagation resulting in: a) a new single interval; b) two new intervals; c) a new single interval and a pseudosource vertex v.

□

5.1.1.3 Interval overlapping

After the propagation, each new potential interval I' on edge e' may overlap with previously created intervals. Let I be a previously created interval which overlaps I', notice that both I and I' can either be s-intervals or p-intervals. We have to decide which interval defines the minimal distance on the overlapped subsegment $\delta = [b_0, b_1] \cap [b'_0, b'_1]$. In this process, interval I' or I can be deleted or trimmed. The most interesting case is when they are trimmed but not deleted. To correctly obtain the intervals we have to compute the point q in δ where the two distance functions coincide. Notice that in this stage we discard those geodesics encoded in I and I' that cannot be shortest paths.

Lemma 5.1.5 *The overlapping of two intervals is computed in $O(1)$ time.*

Proof. To obtain the point q where the distance functions defined by I and I' coincide, we define the rectangular coordinate system $\mathcal{S}_{e'}$ that aligns e' with the x-axis. In this

coordinate system point q is $(q_x, 0)$, and let the virtual sources be $I^s = (i_x, i_y)$ and $I'^s = (i'_x, i'_y)$. To obtain q we have to solve the equation obtained when we make that the distance functions coincide at q. Depending on wether I and I' are p-intervals or $s-$intervals, we have to distinguish between the three following cases.

- Two p-intervals: with $I = (d_0, d_1, b_0, b_1, \sigma, \tau)$ and $I' = (d'_0, d'_1, b'_0, b'_1, \sigma'\tau')$. We solve the equation $|\overline{I^s q}| + \sigma = |\overline{I'^s q}| + \sigma'$, or equivalently
$$\sqrt{(i_x - q_x)^2 + i_y^2} + \sigma = \sqrt{(i'_x - q_x)^2 + i'^2_y} + \sigma'.$$
This equation can be reduced to a quadratic equation with a single root in δ. This equation is $Aq_x^2 + Bq_x + C = 0$ with $A = \alpha^2 - \beta^2$, $B = \gamma\alpha + 2i'_x\beta^2$ and $C = 1/4\,\gamma^2 - ||I'^s||^2\beta^2$, where $\alpha = i'_x - i_x$, $\beta = \sigma' - \sigma$ and $\gamma = ||I^s||^2 - ||I'^s||^2 - \beta^2$.

- An s-interval and a p-interval: with $I = (d_0, d_1, b_0, b_1, \phi)$ and $I' = (d'_0, d'_1, b'_0, b'_1, \sigma', \tau')$ an s-interval and a p-interval respectively. In this case I^s is a virtual segment source and I'^s a virtual punctual source. We solve the equation $d(I^s, q) = |\overline{I'^s q}| + \sigma'$, where $d(I^s, q)$ is the Euclidean distance from q to segment I^s, using the correspondent formulas we obtain:
$$d_0 + \frac{d_1 - d_0}{b_0 - b_1}(q_x - b_0) = \sqrt{(i'_x - q_x)^2 + i_y^2} + \sigma'.$$
This equation can be reduced to the quadratic equation $Dq_x^2 + Eq_x + F = 0$ with a single solution in δ. The quadratic equation coefficients are $D = \eta^2 - 1$, $E = 2\,(\,\eta\,\kappa\, +\, i'_x\,)$, $F = \kappa^2 - ||I'^s||$ where $\eta = \frac{d_1 - d_0}{b_0 - b_1}$ and $\kappa = (d'_0 - \eta b'_0 - \sigma')$.

- Two s-intervals: with $I = (d_0, d_1, b_0, b_1, \phi)$ and $I' = (d'_0, d'_1, b'_0, b'_1, \phi')$. We have two virtual segment sources I^s and I'^s and the equation we have to solve is $d(I^s, q) = d(I'^s, q)$, consequently we obtain the equation:
$$d_0 + \frac{d_1 - d_0}{b_0 - b_1}(q_x - b_0) = d'_0 + \frac{d'_1 - d'_0}{b'_0 - b'_1}(q_x - b'_0),$$
which is a linear equation with a single solution $q_x = \frac{\eta b_0 - d_0 - (\eta' b'_0 - d'_0)}{\eta - \eta'}$ where $\eta = \frac{d_1 - d_0}{b_0 - b_1}$ and $\eta' = \frac{d'_1 - d'_0}{b'_0 - b'_1}$.

In all three cases the virtual sources are determined in constant time (Lemma 5.1.2 and Lemma 5.1.3), and the solution of the obtained equations is computed in $O(1)$ time. Consequently, the overlapping between two intervals takes $O(1)$ time. □

Notice that in this process some already existent intervals can be deleted but only two already existent intervals can be trimmed but not deleted.

5.1.1.4 Continuous Dijkstra propagation strategy

The algorithm uses a Dijkstra-like propagation strategy. In the initialization step, we create intervals encoding the distance function on the edges of the triangle(s) containing the segment source s. If the closest point of s to point $q \in \mathcal{P}$ is an interior point of s, point q is contained in an s-interval. On the other hand, if the closest point of s to q is an endpoint of s, point q is contained in a p-interval.

When intervals are created they are stored in a priority queue which contains both s-intervals and p-intervals. Intervals are stored by increasing distance to the source. The minimum distance from an s-interval to s is $\min(d_0, d_1)$. For p-intervals we use $\min(d_0, d_1) + \sigma$ as their weight in the priority queue, even though it may not be the minimal distance. This can be done because the solution obtained does not depend on the order in which intervals are removed from the queue, however, by using the priority queue a wavefront is simulated and the process is accelerated.

The first interval of the priority queue is selected, deleted and propagated. Next, overlappings are checked, intersections are computed and the new intervals are added to the priority queue. Notice that when there is an overlap, intervals may be modified or deleted and the priority queue has to be updated accordingly.

Now, we present some Lemmas to determine the time and space complexity needed to obtain the shortest path distance functions from a segment source. We first bound the number of intervals generated on each mesh edge. Remember that we work with a non-convex polyhedral surface $\mathcal{P}$ represented as a mesh consisting of n triangles and a segment source placed on $\mathcal{P}$.

Lemma 5.1.6 *The algorithm creates, at most, $O(n)$ intervals per mesh edge.*

Proof. Assume that on the edge e there are $\hat{n}$ intervals. Consider $\hat{n}$ shortest paths starting at an interior point of each interval of e and arriving at s. There may be shortest paths arriving at interior points of s and others at the endpoints of s. We sort the $\hat{n}$ shortest paths clock-wise around e and consider pairs of consecutive shortest paths. There may exist at most four pairs of consecutive shortest paths that traverse exactly the same triangles. This may only happen when: 1) one path arrives at an interior point of s and the other to an endpoint of s; 2) the first path arrives at an endpoint of s and the second path to the other endpoint of s. The other pairs of consecutive shortest paths are associated to a triangle-vertex pair (t,v), where t is the last triangle traversed by both shortest paths,

and v is the vertex separating the pair of paths. Observe that the vertex v may be the first pseudosource of one of the shortest paths. It is not difficult to see that at most two different pairs of shortest paths can be associated to the same (t,v) pair (when v is a pseudosource). Since there exists a bijection between triangle-vertex pairs and edges, $\hat{n} \in O(n)$. □

Lemma 5.1.7 *At most $O(n^2)$ intervals can be deleted in the overlapping step.*

Proof. Once an interval is deleted it can not be propagated and an interval that has been propagated can not be deleted in the future because it defines the minimum distance at some point. Any interval creates at most two new intervals on the opposite edges. These two new intervals may end up being deleted before being propagated. Thus, according to Lemma 5.1.6 there are at most $O(n^2)$ created intervals that can be deleted. □

Theorem 5.1.1 *The distance function defined by a segment source can be propagated on a non-convex triangulated surface with n faces in $O(n^2 \log n)$ time and $O(n^2)$ space.*

Proof. At most $O(n^2)$ intervals are created and propagated, thus the time needed to propagate and create them is $O(n^2)$ (Lemma 5.1.4, Lemma 5.1.7). The time needed in the overlapping process is $O(n^2)$ because a most $O(n^2)$ intervals are deleted in $O(1)$ time each (Lemma 5.1.7), and at most two intervals are trimmed at each step in $O(1)$ time (Lemma 5.1.5). This yields a $O(n^2)$ complexity. Consequently, the time complexity for computing the shortest path distance function is bounded by the maximum between the number of created intervals and the time needed to create and maintain the priority queue. In the worst case the total complexity is $O(n^2 \log n)$ time and the $O(n^2)$ space. □

5.1.2 Polygonal sources

The distance function defined by a polygonal chain is obtained by simultaneously considering all the segments of the polygonal chain in the initialization step. For each segment s of the polygonal chain we create potential s-intervals in the triangle(s) containing s and for each vertex we create potential p-intervals. We handle one segment/point conforming the polygonal line after the other, new potential intervals are intersected with the already created ones to ensure that they define the actual distance function. The other parts of the algorithm do not need changes.

The distance function defined by a polygonal region r, a connected region of $\mathcal{P}$ whose boundary is a closed polygonal chain, is the distance function defined by its boundary in the complementary of r, and is 0 in the interior of r. We compute the distance function defined by its boundary polygonal chain creating, in the initialization step, intervals only in the complementary of r. From now on $\tilde{r}$ denotes the number of segments conforming the generalized source s, it will be different from 1 when s is a polygon or polygonal line.

Lemma 5.1.8 *At most $O(n+\tilde{r})$ intervals per mesh edge are created.*

Proof. We proceed as in the proof of Lemma 5.1.6. Now we have at most four shortest paths traversing the same edges and faces for each segment conforming the polygon or polygonal source. Consequently the number of intervals is now $O(n+\tilde{r})$. □

Putting together this Lemma and the previous observations on how we have to modify the algorithm to compute shortest paths from segment sources allow us to provide the following Theorem.

Theorem 5.1.2 *The distance function defined by a polygonal or polygon source can be propagated on a non-convex triangulated surface with n faces in $O(n(n+\tilde{r})\log(n+\tilde{r}))$ time and $O(n(n+\tilde{r}))$ space.*

Proof. The algorithm provided to compute shortest paths from a segment source works with minor changes. These changes do not affect the time or space complexity of the algorithm which is again determined by the number of created intervals. Lemma 5.1.8 bounds the number of created intervals by $O(n(n+\tilde{r}))$, and consequently the time complexity of the algorithm is $O(n(n+\tilde{r})\log(n(n+\tilde{r})))$ which equals $O(n(n+\tilde{r})\log(n+\tilde{r}))$. The space complexity is $O(n(n+\tilde{r}))$. □

5.1.3 Polygonal obstacles

Given a, possibly non-convex, polyhedral surface $\mathcal{P}$ (which may represent a terrain), we model obstacles as polygonal chains or polygonal regions (which may represent rivers, lakes, etc) on $\mathcal{P}$. The polyhedral surface is retriangulated so that the obstacles are represented as several mesh triangles, edges and vertices. We keep n as the number of triangles of the new mesh. We assume that paths cannot traverse the polygonal obstacles, but we

let paths go along them. Now, geodesic paths can go through a vertex not only if it is a saddle vertex but also when it is an obstacle vertex. To compute shortest paths we only need to make two modifications in the interval propagation process. On the one hand intervals on an obstacle edge are not propagated. On the other, obstacle vertices are new pseudosource vertices regardless of their total angle. These modifications do not increase the time or space complexities of the algorithm. Thus we can claim:

Theorem 5.1.3 *The distance function defined by a generalized element on a polyhedral surface with obstacles can be computed in $O(n^2 \log n)$ time and $O(n^2)$ space.* □

5.2 Implicit distance field computation

The algorithm for computing the distance function from a generalized source extends naturally to the case of several sites. In this case we obtain a generalized *distance field*, which for any point of $\mathcal{P}$ gives the shortest path distance to its nearest site. Notice that the implicit distance field provides the implicit generalized Voronoi diagram.

In the initialization step we generate intervals for each single site and store them in a unique priority-queue. Thus, we propagate the distance functions defined by all the sites simultaneously. This way we obtain a codification of the distance field that yields an implicit representation of the Voronoi diagram of the set of generalized sites.

From now on we denote by $\tilde{r}$ the total number of segments conforming the set S of generalized sites.

Theorem 5.2.1 *The implicit distance field of a set of r generalized sites can be obtained in $O(n(n+\tilde{r})\log(n+\tilde{r}))$ time and $O(n(n+\tilde{r}))$ space.*

Proof. When the distance field of a set of generalized sites is computed, the maximum number of created intervals per edge is $O(n+\tilde{r})$, this can be proved similarly to Lemma 5.1.8. It gives at most $O(n(n+\tilde{r}))$ created and deleted intervals providing a $O(n(n+\tilde{r})\log(n+\tilde{r}))$ time and $O(n(n+\tilde{r}))$ space complexity. □

5.3 Distance and shortest path computation

When the propagation of the distances function has concluded, the distance from any point of a triangle of $\mathcal{P}$ to the source can be obtained from the implicit distance function. If we are given a set of sites S, we can compute the shortest path distance to the closest site and the actual path but using the same technique by using the implicit distance field of S instead of the distance function of a source s.

5.3.1 Influence regions

To make the computation of the distance and shortest paths easier, we first determine which points of $\mathcal{P}$ can be reached by the geodesics encoded in an interval.

Let I be an interval on a mesh edge e and t be a triangle adjacent to e. We define the *influence region of interval* I, denoted R_I, as the set of point of t that can be reached by geodesics encoded in I. According to the geodesic properties, a point $q \in t$ is reached by a geodesic associated to I in the planar unfolding adjacent to e when: 1) the triangle t and the virtual source of I, I^s, are placed in opposite sides of e; 2) the point q belongs to a line emanating from I^s or orthogonal to I^s depending on whether I is a p-interval or s-interval, respectively. Therefore, each interval I defines a unique influence region R_I which is a polygon of at most five vertices contained in one of the two adjacent triangles to e.

When I has an endpoint in a saddle vertex v of t, the interval I can also define a pseudosource. The points of t that are not contained in R_I and that can be reached by a line segment emanating from the pseudosource v without intersecting R_I determine the *influence region of pseudosource* v, P_v, which is a convex polygon of at most three vertices.

Lemma 5.3.1 *The influence region R_I of an interval can be computed in $O(1)$ time.*

Proof. To compute the influence region we have to propagate interval I and determine the convex hull defined by the endpoints of the obtained intervals. Since the propagation is done in $O(1)$ time (Lemma 5.1.4) and since R_I has at most five vertices, region R_I can be obtained in $O(1)$. □

5.3.2 Distance computation

The shortest path distance from any point q on a triangle t to the source s can be obtained by finding the interval I_m on the edges of t or the pseudosource v defining the minimum distance value.

If d_I denotes the distance function defined by the interval I in R_I, we have $d_I(q) = d_s(q') + |\overline{qq'}|$, where q' is a point on I such that the line segment $\overline{qq'}$ from q' to q is contained in a geodesic emanating from I^s. Notice that to determine I_m, those intervals whose influence area do not contain q' can be directly discarded. Therefore, we only take into account an interval I on an edge e if its virtual source I^s and triangle t are located in different sides of e, and $q \in R_I$. If q belongs to the influence region of a pseudosource v, the distance function defined by v, $d_v(q) = d_s(v) + |\overline{qv}|$, needs also to be considered. If we denote Ω the set of not discarded intervals and the possible pseudosources, we have that $d_s(q) = \min_{\iota \in \Omega} d_\iota(q)$. Thus we claim the following Proposition.

Proposition 5.3.1 *The length of the shortest path from a point of $\mathcal{P}$ to the source s or to its nearest source in S can be obtained by standard methods in $O(n + \tilde{r})$ and $O(n + \tilde{r})$ time, respectively.* □

5.3.3 Shortest path computation

The shortest path from any point q on a triangle t to the source s can be obtained by using a backtracing technique.

Proposition 5.3.2 *The shortest path from a generalized source s to point p can be obtained in $O(n + \tilde{r} + \overline{n})$ time, where $\overline{n}$ is the number of crossed triangles. The shortest path to the closest site of S is obtained in $O(n + \tilde{r} + \overline{n})$ time.*

Proof. We first determine the element, interval or vertex, ι_m defining the minimum distance to q. There exist two possibilities: 1) If ι_m is an interval I_m, we jump to the adjacent triangle t' by using the direction τ when I_m is a p-interval or the angle ϕ when I_m is an s-interval; 2) If ι_m is a pseudosource, it is an endpoint of an interval I_m which is used to jump to the adjacent triangle. Then, we keep on jumping to the adjacent triangle until we get s.

According to Proposition 5.3.1 the distance can be obtained in $O(n + \tilde{r})$ or $O(n + \tilde{r})$

time. Once ι_m is obtained we jump to a previous interval in constant time by locating the virtual source ι_m^s and finding the new interval when ι_m is an interval, or checking the adjacent edges to the virtual source ι_m otherwise. Both searchings can be done in constant time. □

5.4 Voronoi diagram complexity

We consider a triangulated polyhedral surface $\mathcal{P}$ with n triangles, homeomorphic to either the plane or the sphere, and a set $S = \{s_1, \ldots, s_r\}$ with r generalized sites containing points, line segments, polygonal chains or a polygons. The distance $d(p, q)$ between two points $p, q \in \mathcal{P}$ of the surface is the shortest path distance given by the shortest path from p to q which is given by $d_p(q)$.

In this section, a *(generalized) site* s on $\mathcal{P}$ is a point or a line segment contained in $\mathcal{P}$. As far as our results are concerned, there is no advantage in considering more complicated sites (polygonal chains or polygons) instead of the family of segments that defines them. In this section, considering objects of constant description complexity is relevant for the results below. Otherwise some demonstrations, for instance those using the random sampling fall apart because some objects are more complex than others.

Let $S = \{s_1, \ldots, s_r\}$ be a set of r generalized sites in $\mathcal{P}$. We assume that the sites are pairwise interior-disjoint. A *bisector* β_{ij} defined by sites s_i and s_j is the locus of points at the same distance from s_i and s_j. When a vertex of the surface $\mathcal{P}$ is equidistant from two sites, it may happen that a bisector contains two-dimensional pieces. We exclude this case as degenerate, during only the complexity analysis, and assume henceforth that no vertex of the surface is equidistant from two sites. With this assumption, a bisector consists of straight-line, parabolic and hyperbolic segments for being defined by weighted distances from point sources or non weighted distances from segment sources. We assume that no three bisectors intersect in a point, the case when three bisectors intersect in a point is also excluded as a degenerate case. A *breakpoint* is the intersection between two adjacent segments or arcs on a Voronoi edge, the point where a bisector crosses an edge of $\mathcal{P}$ is also considered a breakpoint.

We are interested in order-k Voronoi diagrams of S. We next recall its definition, together with other related concepts. For a subset S' of the sites, the Voronoi region or

cell $Vor(S')$ of S' is defined as

$$Vor(S') = \{p \in \mathcal{P} \mid d_{s'}(p) \leq d_s(p)\ \forall s' \in S', s \in S\}.$$

For each integer $1 \leq k \leq r-1$, the order-k Voronoi diagram of S is the family of Voronoi regions of all subsets of k sites of S. When $k = 1$ and $k = r-1$ the order-k Voronoi diagrams are called the *(closest) Voronoi* and the *furthest Voronoi* diagram, respectively. A *k-region* is a Voronoi region in the order-k Voronoi diagram. A *k-edge* is a connected component in the intersection of two k-regions. A k-edge always lies on the bisector of two sites. A *k-breakpoint* is a breakpoint on a k-edge. A *k-vertex* is the intersection point of three or more k-regions.

The terms *edges* and *vertices* of the order-k Voronoi diagram are chosen because of their resemblance with a graph embedded in the surface P. The removal of edges from the surface leaves a set of *faces*. The closure of a face corresponds to a order-k Voronoi cell, and vice versa. Finally, let us remark the difference between vertices and breakpoints: while vertices correspond to endpoints of edges, breakpoints can only appear along the relative interior of edges.

The complexity of the order-k Voronoi diagram is the total number of k-regions, k-vertices, k-breakpoints and k-edges. The purpose of this work is to study the complexity of order-k Voronoi diagrams.

We start by providing some results referred to the closest Voronoi diagram (Section 5.4.1). Then we study order-k Voronoi diagrams on triangulated surfaces homeomorphic to a sphere (Section 5.4.2 to Section 5.4.4). Finally, we provide bounds for the special case of a Realistic terrain (Section 5.4.5).

During this section, we will provide some examples using terrain models instead of polyhedral triangulated surfaces. Notice that it can be done due to the fact that any terrain can be represented by a polyhedral surface (Section 2.4.4). Notice that when we consider a terrain with n triangular faces, the obtained polyhedral surface has $O(n)$ faces. Thus complexities on triangulated terrains or on polyhedral surfaces are exactly the same.

5.4.1 Properties of the closest and furthest Voronoi diagram

Lemma 5.4.1 *In the closest Voronoi diagram the cell of a site $s \in S$, C_s, is path-connected.*

Proof. Let us consider two points p and q in C_s, shortest paths $\pi_{p,s}$ from p to s and shortest path $\pi_{s,q}$ from s to q. Path $\pi = \pi_{p,s} \cup \pi_{s,q}$ connects p and q. We will prove that π is contained in C_s for the sake of contradiction. Assume that there exists a point t in $\pi_{p,s} \subset \pi$ that is not contained in C_s, then there exist a site s' closer to t than site s. Consequently the following inequality holds $d_{s'}(p) \leq d_{s'}(t) + d_t(p) < d_s(t) + d_t(p) = d_s(p)$ because distances are given by the shortest path lengths. It contradicts the fact that $p \in C_s$. □

Lemma 5.4.2 *Bisector β_{s_i,s_j} is connected and homeomorphic to a circumference.*

Proof. Consider the Voronoi diagram of $S = \{s_i, s_j\}$, C_{s_i} and C_{s_j} cover the whole surface $\mathcal{P}$ and they are path connected (Lemma 5.4.1). By definition $C_{s_i} \cap C_{s_j} = \beta_{i,j}$ which is the boundary of C_{s_i} and C_{s_j}. Since $\mathcal{P}$ is homeomorphic to a sphere, $\beta_{i,j}$ is homeomorphic to a circumference. □

Lemma 5.4.3 *Any vertex of a closest Voronoi diagram has at least degree three.*

Proof. Consider v to be a vertex of the closest Voronoi diagram defined as the intersection point v of bisector $\beta_{i,j}$ and $\beta_{i,k}$. Then v is also a point of $\beta_{t,u}$ because $d_{s_i}(v) = d_{s_j}(v) = d_{s_k}(v)$, and consequently v is a vertex of degree at least three. □

Lemma 5.4.4 *Each bisector $\beta_{i,j}$ consists of $O(n(n + \tilde{r}))$ straight-line segments and hyperbolic arcs, where $\tilde{r}$ is the number of segments conforming the set of sites S.*

Proof. The complexity of a bisector on each face is $O(n+\tilde{r})$ due to Lemma (Lemma 5.1.8). Since a bisector intersects at most O(n) faces, the total complexity is $O(n(n + \tilde{r}))$ □

Lemma 5.4.5 *For any three distinct sites s_i, s_j and s_k, bisectors $\beta_{i,j}$ and $\beta_{i,k}$ can not intersect more than twice.*

Proof. Consider the closest Voronoi diagram of $\{s_i, s_j, s_k\}$. Let v be an intersection point of $\beta_{i,j}$ and $\beta_{i,k}$. Then, v is also a point of $\beta_{j,k}$ because $d_{s_i}(v) = d_{s_j}(v) = d_{s_k}(v)$, and consequently v is a vertex of degree at least three of the closest Voronoi diagram.

Suppose for the sake of contradiction that bisectors $\beta_{i,j}$ and $\beta_{i,k}$, and consequently $\beta_{j,k}$ intersect at least at three different points v_1, v_2 and v_3. Connect each site s_i, s_j, s_k to each point v_1, v_2 and v_3 by a shortest path. These paths have the following properties: i) No two of the shortest paths sharing an end point cross. Consider a pair of paths not sharing an end point, say $\pi_{s_i v_1}$ and $\pi_{s_j v_2}$. The former is contained in C_{s_i} and the later in C_{s_j}, the Voronoi regions associated to s_i and s_j, respectively, and their intersection lies in $\beta_{i,j}$. In particular the two paths cannot cross. ii) A shortest path can not cross a site. In fact if shortest path π_{s_i,v_l} crosses s_j: $d_{s_i}(v_l) > d_{s_j}(v_l)$ which is impossible for being v_l a point of β_{ij}. Using this property sites s_i, s_j and s_k can be homeomorphicaly transformed to point sites without changing the topology defined by the mentioned shortest paths.

Consequently we obtain six points (three sites and three intersection points) and nine interconnecting paths forming a non-crossing embedding of $K_{3\times3}$, the 3×3 complete bipartite graph, on the topological sphere which is a contradiction. □

Lemma 5.4.6 *The furthest Voronoi diagram of a set S of r generalized sites has $O(r)$ cells, vertices and edges.*

Proof. Let us consider $R_{i>j}$ the region of points that are further from s_i than s_j. Thus the furthest Voronoi region associated to s_i, C_{s_i} is $\cap_{j=1\ldots r|j\neq i} R_{i>j}$ which is the common exterior of a set of pseudo-disks that contain s_i. By using Lemma 2.6.3 the intersection is path connected so there is at most one region per site. Since each vertex has at least degree three, it has $O(r)$ cells, vertices and edges. □

Theorem 5.4.1 *The maximum complexity of a furthest Voronoi diagram of a set of r generalized sites is $O(rn(n+\tilde{r}))$*

Proof. It has at most r bisectors of complexity $O(n(n+\tilde{r}))$ (5.4.4) yielding a complexity of $O(rn(n+\tilde{r})$. For a set of point sites, the maximum complexity of the furthest Voronoi diagram is $\Theta(rn^2)$ (Theorem 2.6.2), thus the provided bound is tight when $\tilde{r} < n$. □

5.4.2 Properties of order-k Voronoi diagrams.

Let v be a k-vertex and s_1, s_2, s_3 be the three sites defining v: v is at the intersection of the bisectors β_{12}, β_{13}, β_{23}. A disk centered at v with radius $d_{s_1}(v)$ contains s_1, s_2, s_3 and possibly some other sites, which we denote by S_v.

Lemma 5.4.7 *The cardinality of S_v with v a k-vertex is either $k-2$ or $k-1$.*

Proof. A vertex v is a point that is equidistant to three sites, thus there exists a closed circle centered at v containing l sites, $l-3$ of these sites are interior to the circle. We will prove that such a point is a $(l+1)$-vertex and a $(l+2)$-vertex proving the previous result. This is true because in a neighborhood of v there exist points contained in a: i) $(l+1)$-edge, because these points are centers of circles containing two points in the boundary and $l-1$ points in the interior. ii) $(l+2)$-edge because they are centers of circles containing 2 points in the boundary and $l-2$ points in the interior. Since distance functions are continuous it is always fulfilled. □

The following Lemma states intuitively that when a k-edge is traversed, the sets of the k nearest sites differ in a single site. We denote C_i a cell of a order-k Voronoi diagram, S_i the set of k sites defining C_i and β_{kl} the bisector defined by sites s_k and s_l.

Lemma 5.4.8 *Given two k-cells, C and C' with a common k-edge $e \subset \beta_{ij}$, The intersection of sets S_C and $S_{C'}$ contains $k-1$ sites, in fact, sets S_C and $S_{C'}$ only differ in the sites defining β_{ij}.*

Proof. Assume that k-edge e is on bisector β_{ij}, points on e stay at the same distance from s_i and s_j. If we consider a neighborhood U of a point $p \in \beta_{ij}$ such that U does not intersect any other bisector, we can sort the sites of $S \setminus \{s_i, s_j\}$ according to $d_{s_k}(p)$ with $s_k \in S \setminus \{s_i, s_j\}$. Since distance functions are continuous and U does not intersect any other bisector the inequalities hold for every $q \in U$. If we also consider sites s_{i_1} and s_{i_2} we have in p the same ordered chain but with the equality $d_{s_i}(p) = d_{s_j}(p)$. When an arbitrary point is considered, the inequality will depend on the side of β_{ij} where q is placed. When e is a k-edge it also changes the set of the k-nearest sites, thus in one side of e site s_i is the k-nearest site and s_j the $(k+1)$-nearest one, while in the other is the other way round. Consequently, given two k-regions C and C' with a common edge e, edge e is in the bisector defined by the sites in $\{S_C \cup S_{C'}\} \setminus \{S_C \cap S_{C'}\}$.

□

Lemma 5.4.9 *A k-vertex has at least degree three.*

Proof. Suppose for the sake of contradiction that there exists a k-vertex v, the intersection point v of bisector $\beta s, t$ and $\beta s, u$, which has degree two. This necessarily contradicts

Lemma 5.4.8. □

Lemma 5.4.10 *The union of the k-cells having site s_i as one of the k closest sites, $\mathcal{U}_{s_i}$, is path connected.*

Proof. Let us consider any two points p and q of $\mathcal{U}$, shortest paths π_{p,s_i} from p to s_i and shortest path $\pi_{s_i,q}$ from s_i to q. Path $\pi = \pi_{p,s_i} \cup \pi_{s_i,q}$ connects p with q. Assume that there exists a point t in $\pi_{p,s_i} \subset \pi$ that is not contained in $\mathcal{U}_{s_i}$, then there exist k sites closer to t than site s_i. The distance from any of these k sites s'_j to p fulfills the following inequality: $d_{s_j}(p) \leq d_{s'_j}(t) + d_t(p) < d_{s_i}(t) + d_t(p) = d_{s_i}(p)$. It contradicts the fact that p has s_i as one of the k closest sites. □

Lemma 5.4.11 *Let C_{k+1} be a $(k+1)$-cell and C_k a k-cell, then $C_{k+1} \nsubseteq C_k$.*

Proof. Assume that, on the contrary, $C_{k+1} \subset C_k$. Let S_{C_k} and $S_{C_{k+1}}$ be the sets of sites defining C_k and C_{k+1}, respectively. Since $C_{k+1} \subset C_k$ are a $k+1$-cell and a k-cell, respectively, there exists a site s_{k+1} such that $S_{C_{k+1}} = S_{C_k} \cup \{s_{k+1}\}$. If site s_{k+1} is located in C_k, it is the closest site to some points of C_k and $s_{k+1} \in S_{C_k}$ reaching a contradiction. Thus, s_{k+1} is not located in C_k. Let us consider the shortest path, π, from s_{k+1} to a point $p \in C_{k+1}$ (see Figure 5.4). Due to the not degenerate position of the sites, path π contains a point q in $C_k \setminus C_{k+1}$. Point q is such that $d_{s_{k+1}}(p) = d_q(p) + d_{s_{k+1}}(q)$ and if the $k+1$th closest site to q is $s' \neq s_{k+1}$ then $d_{s'}(p) \leq d_q(p) + d_{s'}(q) < d_q(p) + d_{s_{k+1}}(q) = d_{s_{k+1}}(p)$, thus s' is closer to p than s_{k+1} getting a contradiction. □

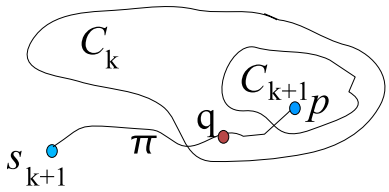


Figure 5.4: Cells C_k and C_{k+1} are cells of a k-cell and a $(k+1)$-cell, respectively, π is the shortest path from site s_{k+1} to a point $p \in C_{k+1}$.

Property 5.4.1 *Each face of the order-$(k+1)$ Voronoi diagram contains an edge of the order-k Voronoi diagram.*

Proof. A face of the order-$(k+1)$ Voronoi diagram which does not contain any edge of the order-k one is contained in a face of the order-k Voronoi diagram. This contradicts Lemma 5.4.11. □

5.4.3 Pathologies of order-k Voronoi diagrams.

5.4.3.1 Edges without vertices

Although the definition of edges and vertices resemble those of graphs embedded in $\mathcal{P}$, the situation may be quite different.

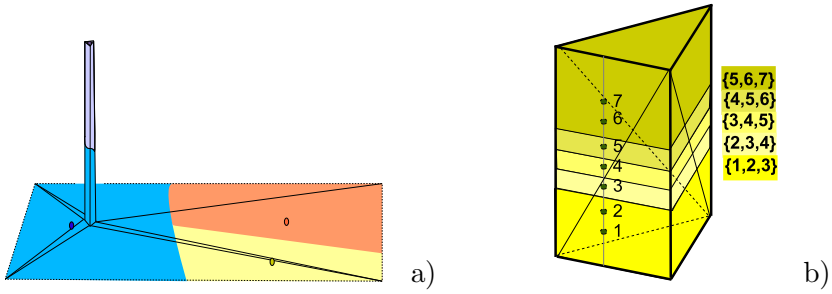
Lemma 5.4.12 *There exist order-k Voronoi diagrams with edges that form closed curves in P and are not adjacent to any vertex.*

Proof. Let us consider a large plane with a truncated triangular pyramid on it, as shown in Figure 5.5. The base of the pyramid is an equilateral triangle with edges of length l, the edges of the top triangle are of length $0.9l$ and the pyramid height is $1000l$. We place a single site on the top of the truncated triangular pyramid and several sites on the plane: one near the pyramid base and some other ones far from it, as in Figure 5.5a). The region corresponding to the site on the top of the pyramid has no vertices and its edge is a closed curve.

Another example is obtained by considering a prism with triangular base. Consider a line ℓ parallel to an edge of the prism from one base to the other, as in Figure 5.5 b). We place r sites on ℓ, no pair of bisectors intersect and some of them define the boundary a k-cell. Consequently k-cells may have closed curve edges without vertices. □

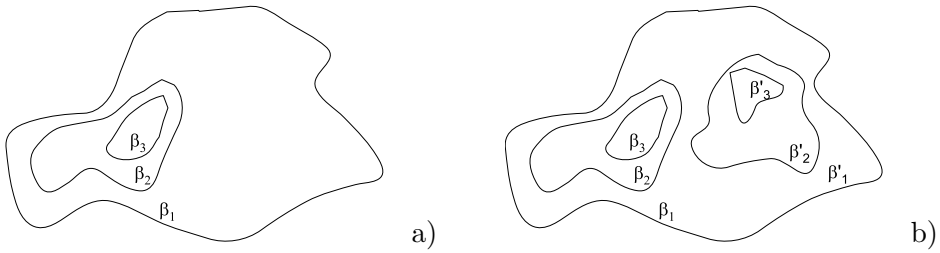
We want to prove that there exist at most $O(r)$ k-edges without vertices. This is proved by showing that we have a different site $s \in S$ in the interior of each such a region, or equivalently that we can associate a different site to each such a region. All the edges and bisectors considered in this section are k-edges without vertices that are defined by a bisector.

In the following Lemmas we consider a set of bisectors $\mathcal{B} = \{\beta_i : i = 1 \dots l\}$ defining a chain of nested regions of the order-k Voronoi diagram (see Figure 5.6). Let R_i be the interior of the region delimited by β_i so that $R_1 \supset R_2 \supset \dots \supset R_l$ and let S_i be the set of k closest sites to the points of the region $R_i \setminus R_{i+1}$.



a) b)

Figure 5.5: Example of: a) closest Voronoi diagram with a region, the one containing the points of the truncated pyramid top, without vertices; b) order-3 Voronoi diagram with all the regions without vertices.



a) b)

Figure 5.6: Nested regions of a order-k Voronoi diagram defining: a) a chain of bisectors $\mathcal{B} = \{\beta_1, \beta_2, \beta_3\}$. b) two chains of closed bisectors $\mathcal{B} = \{\beta_1, \beta_2, \beta_3\}$ and $\mathcal{B}' = \{\beta'_1, \beta'_2, \beta'_3\}$.

Each bisector β_i is defined by the equidistant points to sites $s_{i,1}$ and $s_{i,2}$, one of these sites is contained in R_i and the other in its complementary.

Lemma 5.4.13 *Given $\mathcal{B}$, a chain of nested regions, let s be a site defining bisectors β_i and β_j with $i < j$. Then site s is: a) located in $R_i \setminus R_j$. b) in S_l for $i \leq l \leq j-1$. c) does not define any other bisector containing an edge.*

Proof. Assume that $s \in R_j$, then necessarily $s \in S_j$ and consequently $s \notin S_{j-1}$, let U_j be a neighborhood of β_j, then s is one of the k-closest sites in $U_j \cap R_j$ and it is not one of the k-closest sites in $U_j \cap R_j^c$, where R_j^c is the exterior of R_j (Lemma 5.4.8). Analogously, let U_i be a neighborhood of β_i, then s is one of the k-closest sites of points either in $U_i \cap R_i$ or $U_i \cap R_i^c$. In any of the two possible cases we obtain a non path connected region of points having s as one of the k-closest sites, which contradicts Lemma 5.4.10. Thus, s can not

be contained in R_j. Therefore, $s \notin S_j$ and $s \in S_{j-1}$. If we assume that s is contained in the exterior of R_i, and we proceed in a similar way we obtain a contradiction again. Thus, $s \in R_i \setminus R_j$, $s \notin S_j$, $s \in S_{j-1}$ and $s \in S_i$.

To prove that $s \in S_l$ for $i < l < j-1$ we proceed in the same way. We obtain, by using Lemma 5.4.8, two not path connected regions containing s as one of the k-closest sites. This contradicts Lemma 5.4.10 and proves item c). □

Lemma 5.4.14 *If region R_i contains l nested regions in its interior, it contains, at least, $l+1$ sites.*

Proof. This result is proved by induction from $l = 0$ to $l = h$. When $l = 0$, region R_i necessarily contains one of the two sites defining β_i, thus R_i contains one site. Now we use induction, assume that if R_i contains $l = h-1$ regions and h sites. We prove it for $l = h$: when region R_i contains h regions, it contains $h+1$ sites. Let s be the site involved in β_i defining R_i, the biggest region, so that $s \in R_i$. If s has not been involved in any other bisector delimiting a region interior to R_i, it is not in the h sites obtained by induction, thus R_i contains $h+1$ sites. If s is also involved in another bisector β_j so that $R_j \subset R_i$, site $s \in R_i \setminus R_j$ (Lemma 5.4.13 *a*)), and s has not been counted yet, for being located in the complementary of R_j. According to Lemma 5.4.13 c) no more possibilities exist, and consequently R_i contains $l+1$ sites. □

Let us now consider two sets of closed bisectors $\mathcal{B} = \{\beta_i : i = 1 \dots h\}$ and $\mathcal{B}' = \{\beta'_i : i = 1 \dots h'\}$ defining two chains of nested regions of the order-k Voronoi diagram such that there exist an integer $i_0 \geq 1$ with $\beta_i = \beta'_i$ for $i < i_0$ and $R_i \cap R'_i = \emptyset$ for $i \geq i_0$ (see Figure 5.6b))

Lemma 5.4.15 *If region R_i contains l regions delimited by closed curve edges, then, R_i contains $l+1$ sites.*

Proof. We consider $\mathcal{B}$ and $\mathcal{B}'$, two sets of nested regions, having $\beta_i = \beta'_i$ for $i < i_0$ and $R_{i_0} \cap R'_{i_0} = \emptyset$. According to Lemma 5.4.14, if region $R_{i_0}(/R'_{i_0})$ contains $h_{i_0}(/h'_{i_0})$ regions, it contains $h_{i_0} + 1(/h'_{i_0} + 1)$ sites.

By using this notation R_0 contains $(i_0 - 2) + (h_{i_0} + 1) + (h'_{i_0} + 1) = i_0 + h_{i_0} + h'_{i_0}$ regions, thus it has to contain $i_0 + h_{i_0} + h'_{i_0} + 1$ sites. Thus, we have to prove that apart from the already counted sites, R_0 contains $i_0 - 1$ extra sites.

This is proved by induction and considering each bisector $\beta_j = \beta'_j$ for $j = i_0 - 1$ to $j = 1$. Let us start with $j = i_0 - 1$ and let s be the site defining β_j so that $s \in R_j$, we show that site s has not been counted before, if: i) s does not define any other bisector β_k nor β'_k with $k > j$, s has not been counted before (Lemma 5.4.14); ii) s defines one other bisector β_k, then $s \in R_j \setminus R_k$ (Lemma 5.4.13 a)) and we have only counted sites s defining a bisector β so that $s \in R$ (Lemma 5.4.14), thus s is not counted yet; iii) s defines one other bisector β'_k, it is analogous to case ii). iv) s defines two other bisectors β_k and $\beta'_{k'}$, then, $s \in (R_j \setminus R_k) \cap (R_j \setminus R'_{k'}) = R_j \setminus \{R_k \cup R''_k\}$ (Lemma 5.4.13), since s can not define any other bisector of $\mathcal{B}$ nor $\mathcal{B}'$ (Lemma 5.4.13 b)) the same reasoning of ii) shows that s has neither been counted yet.

The induction step uses exactly the same reasoning. Finally, by doing $j = 0$ we obtained the remaining $i_0 - 1$ more sites. □

Combining Lemmas 5.4.14 and Lemma 5.4.15 we conclude the following proposition.

Proposition 5.4.1 *There exist $O(r)$ edges without vertices in a order-k Voronoi diagram.* □

For an example of a order-1 Voronoi diagram with $O(r)$ cells see Figure 5.7 a). It is obtained by using $r - 1$ truncated triangular pyramids with a site s on the plane and one site on the top of each pyramid. The region corresponding to the site on the plane contains all the points of the plane, and the regions of the other $r - 1$ sites are delimited by edges without vertices. Several nested regions can be obtained by using the scheme provided in Figure 5.7 b), order-k Voronoi diagrams with such regions can be obtained by replacing the pyramids by prisms as it is done in Figure 5.5 and Lemma 5.4.12.

5.4.3.2 Path-connectivity of the cells

We want to show that order-k Voronoi diagrams in polyhedral surfaces are substantially more complex than in the plane, and also more complex than closest Voronoi diagrams. For example, the cells of order-k Voronoi diagrams in the plane or the cells of closest or furthest Voronoi diagram in polyhedral surfaces are path-connected (Section 2.6.3.3). This is not true for the general case.

Lemma 5.4.16 *For any given $k \geq 2$, there exists a polyhedral surface and a set of sites such that the order-k Voronoi diagram has some disconnected cells.*

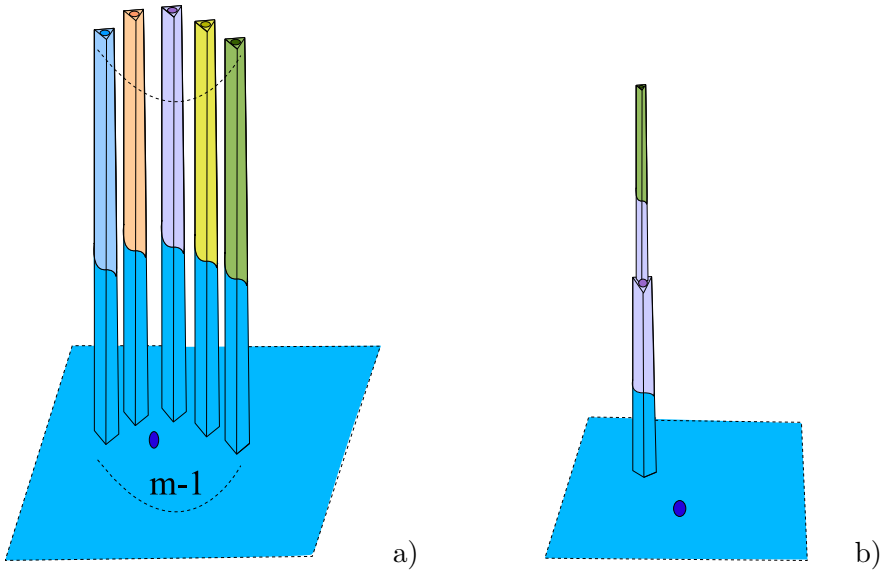


Figure 5.7: a) not nested cells without vertices. b) Concentric regions without vertices

Proof. We describe a polyhedral surface $\mathcal{P}$ embedded in 3-dimensional Euclidean space. Consider three squares, Q_i, $i = 1 \dots 3$, of sides of length 1.1, 1 and 0.9, respectively. They are placed in planes parallel to the xy-plane, with their centers on the line $x = y = 0$, and with their sides parallel to the x, y coordinate axis. Squares Q_1 and Q_3 are placed at height zero and Q_2 at height 1000. Let $\mathcal{P}$ be the polyhedral surface obtained by gluing xy-plane minus the interior of Q_1, the pyramid with base Q_1 and Q_2 (without the base), the pyramid with base Q_2 and Q_3 (without the base), and the square Q_3. Let ℓ be the curve obtained by intersecting $\mathcal{P}$ with the vertical plane $y = 0$. See Figure 5.8a) for a projection onto the xy-plane.

We place $3k$ point-sites along ℓ, as follows. First, we place k sites in Q_3, symmetrically respect to its center. We then place k sites in each side of the portion of the xy-plane outside Q_1, distributed in four groups: S_1^l, S_2^l, S_1^r and S_2^r. Groups S_1^l, S_1^r contain $\lceil k/2 \rceil$ points, while groups S_2^l, S_2^r contain $\lfloor k/2 \rfloor$ points. Within each group, the points are regularly spaced at distance ϵ, where $0 < \epsilon \ll 1/k$. The points S_1^l are placed to the left of Q_1, along ℓ, at distance ϵ from Q_1, and set S_2^l at distance 1 from S_1^l. We do the same on the right of Q_1 but placing S_2^r closer to Q_1 than S_1^r (see Figure 5.8a)).

Consider the Voronoi region R having $S_1^l \cup S_2^r$ as closest sites. First, note that R is nonempty because it contains the point $(0, 1.1, 0)$ in its interior, and hence R is a Voronoi cell of the order-k Voronoi diagram. However, no point on ℓ is in R, and since the construction is symmetric, this means that R has at least two path-connected components.

The construction for $k = 2$ is shown in Figure 5.8b) (xy-projection) and in 5.8 c) (3D scene).

We can also obtain a polyhedral surface having the same properties by considering a cube with two such constructions in two opposite faces. □

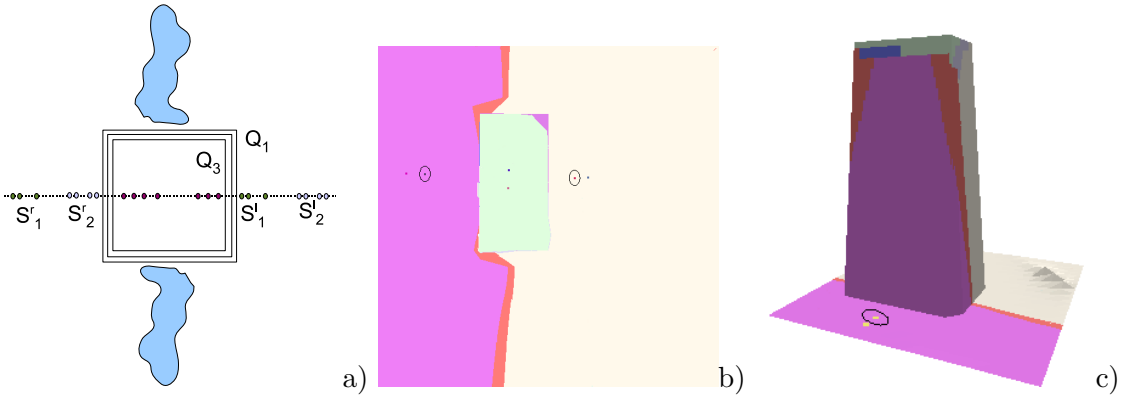


Figure 5.8: Example of a order-k Voronoi diagram with disconnected cells. a) general scene b) a order-2 Voronoi diagram projected on the xy-plane, the disconnected cell is the red one and corresponds to the two sites marked with a circle. c) the order-2 Voronoi diagram of figure b) on the 3D scene.

5.4.4 Complexity of order-k Voronoi diagrams

For any given k, bounds for the complexity of all the order-i Voronoi diagrams together, for $i = 1 \dots k$, are presented. We use a similar approach to the one provided by Clarkson and Shor [30] and Sharir [135]. Let B_k be the number of k-breakpoints, and let $B_{\leq k}$ be the number of i-breakpoints for $i = 1 \dots k$. Similarly, let V_k be the number of vertices

Lemma 5.4.17 *The complexity of the order-k Voronoi diagram is proportional to* $B_k + V_k$.

Proof. Each edge that is not adjacent to a vertex (c.f. Lemma 5.4.12) contains some breakpoints. Therefore, the number of k-edges that are not adjacent to any vertex is bounded by B_k. The number of k-edges that are adjacent to some vertex is bounded by $O(V_k)$, as follows by applying Euler's formula to the planar graph defined by the k-vertices and the k-edges adjacent to some k-vertices. Finally, the number of k-cells is bounded by the number of k-edges because of Euler's formula. □

Lemma 5.4.18 *$B_{\leq k} = O(kn(kn+r)) = O(k^2n^2 + knr)$ for any $1 \leq k \leq r-1$.*

Proof. Draw a random sample R of S, by independently drawing each element of S with probability p. We will set below p to an appropriate value. Let $B(R)$ be the number of breakpoints of the order-1 Voronoi diagram of R in $\mathcal{P}$. From the bounds on the complexity of order-1 Voronoi diagrams, we know that $B(R) = O(n^2 + |R|n)$. Since $|R|$ follows a binomial distribution, we have

$$\mathbb{E}\left[|R|\right] = rp,$$

and therefore

$$\mathbb{E}\left[B(R)\right] = O(n^2 + nrp). \tag{5.1}$$

Consider a breakpoint b in some j-edge e of the order-j Voronoi diagram of S. Let s, s' be the two sites defining the bisector that contains b. Note that a disk centered at b and with radius $d_s(b)$ contains s, s' and $j-1$ sites, which we denote S_b. The point b is a breakpoint in the order-1 Voronoi diagram of the random sample R if and only if s, s' are in R and any other of the $j-1$ sites S_b are not in R. Therefore, the probability that b is a breakpoint in the order-1 Voronoi diagram of R is precisely $p^2(1-p)^{j-1}$. By linearity of expectation we then have

$$\begin{aligned}
\mathbb{E}\left[B(R)\right] &= \sum_{j=1}^{r-1} B_j p^2 (1-p)^{j-1} \\
&\geq p^2 \sum_{j=1}^{k} B_j (1-p)^{j-1} \\
&\geq p^2 (1-p)^{k-1} \sum_{j=1}^{k} B_j \\
&= p^2 (1-p)^{k-1} B_{\leq k}.
\end{aligned}$$

Manipulating and substituting equation (5.1) we see

$$B_{\leq k} \leq \frac{\mathbb{E}\left[B(R)\right]}{p^2(1-p)^{k-1}} = \frac{O(n^2+nrp)}{p^2(1-p)^{k-1}}.$$

Finally, setting $p = 1/k$ we obtain

$$B_{\leq k} = O\left(\frac{n^2 + nr/k}{(1/k)^2(1-1/k)^{k-1}}\right) = O\left(k^2n^2 + knr\right),$$

where we have used that

$$\frac{1}{(1-1/k)^{k-1}} = \left(\frac{k}{k-1}\right)^{k-1} \leq e.$$

□

Lemma 5.4.19 *We have $V_{\leq k} = O(k^2 r)$ for any $1 \leq k \leq r-1$.*

Proof. This proof is very similar to the previous. Draw a random sample R from S, by independently drawing each element of S with probability p. Let $V(R)$ be the number of vertices of the order-1 Voronoi diagram of R in $\mathcal{P}$. From the known bounds we have $V(R) = O(|R|) = O(rp)$.

Consider a j-vertex v, and let s_1, s_2, s_3 be the three sites defining v: v is in the intersection of the bisectors β_{12}, β_{13}, β_{23}. A disk centered at v and with radius $d_{s_1}(v)$ contains s_1, s_2, s_3 and possibly some other sites, which we denote by S_v. The cardinality of S_v is either $j-2$ or $j-1$. The point v is a vertex in the order-1 Voronoi diagram of the random sample R if and only if s_1, s_2, s_3 are in R and any other of the sites S_v are not in R. Therefore, the probability that b is a breakpoint in the order-1 Voronoi diagram of R is precisely $p^3(1-p)^{|S_v|} \geq p^3(1-p)^{j-1}$. By linearity of expectation we then have

$$\begin{aligned}
\mathbb{E}[V(R)] &= \sum_{j=1}^{r-2} V_j p^3 (1-p)^{j-1} \\
&\geq p^3 \sum_{j=1}^{k} V_j (1-p)^{j-1} \\
&= p^3 (1-p)^{k-1} V_{\leq k}.
\end{aligned}$$

Like in the previous proof, we manipulate, substitute $V(R) = O(rp)$, and set $p = 1/k$ to get

$$V_{\leq k} \leq \frac{\mathbb{E}[V(R)]}{p^3(1-p)^{k-1}} = \frac{r/k}{(1/k)^3(1-1/k)^{k-1}} = O(k^2 r).$$

□

Combining Lemmas 5.4.17–5.4.19 we can conclude our main result:

Theorem 5.4.2 *Let $\mathcal{P}$ be a polyhedral surface with n triangles, let S be a set of r sites, where each site is either a segment or a point in $\mathcal{P}$, and let k be an integer $1 \leq k \leq r-1$. The complexity of all the order-i Voronoi diagrams of S together for $i = 1 \ldots k$ is $O(k^2 n^2 + k^2 r + knr)$.* □

5.4.4.1 Discussion

In the plane, where $n = O(1)$, it is known that the order-k Voronoi diagram of r has complexity $\Omega(k(r-k))$ [92, 145]. If we add the complexity of all the order-i Voronoi

diagrams for $i = 1 \dots k$, we obtain that the total complexity is $\Omega(rk^2)$. Thus, the bound in Theorem 5.4.3 is tight when term $k^2 r$ dominates.

We can build an example with complexity $O(krn)$. We consider two symmetric semi-circles each of which contains $r/2$ sites as is shown in Figure 5.9 a) and a triangulation placed as the one in Figure 5.9 b). Each order-i Voronoi diagram has $r - 2$ edges-i that intersect $O(n/8)$ triangles. Consequently the total complexity is $\Omega(knr)$ and the bound in Theorem 5.4.3 is tight when term knr dominates.

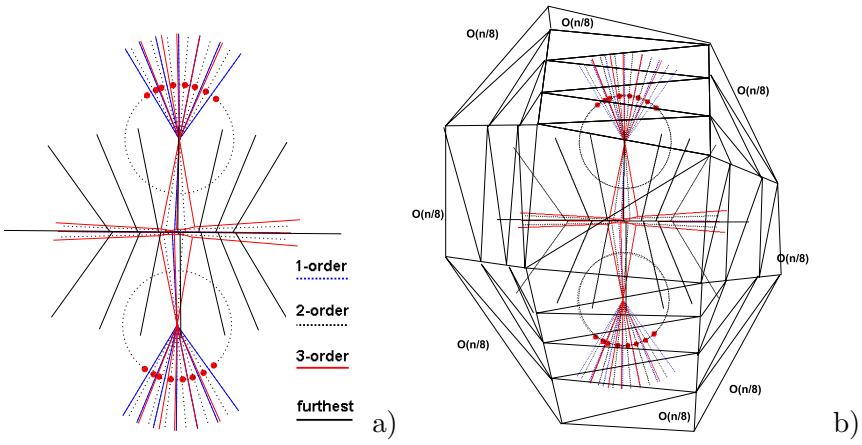


Figure 5.9: a) A set of 18 red sites and their closest, order-2, order-3 and furthest Voronoi diagrams. b) The previous set of sites with their Voronoi diagrams and a triangulation producing $O(knr)$ breakpoints.

We have not been able to find an example with complexity $O(n^2k^2)$. We have only been able to provide lower bounds to show that the total complexity of the order-k Voronoi diagrams of generalized sites on a polyhedral surface is $\Omega(k^2r + krn)$. We have also proved that the furthest Voronoi diagram has complexity $\Omega(rn^2)$.

5.4.5 Complexity of order-k Voronoi diagrams on Realistic Terrains

As we have already said a terrain is a special polyhedral surface, it can be transformed to a polyhedral surface by using a technique provided in Section 2.4.4. However, in this section, we are interested in the complexity on a realistic terrain. Thus, we only count the vertices, edges or breakpoints not placed on the original terrain. As we show, when considering realistic terrains we can reduce some bounds related to the complexity of the Voronoi diagrams.

Lemma 5.4.20 *The shortest path from a generalized source s to a point on a realistic terrain crosses $O(\sqrt{n})$ faces.*

Proof. Let $\pi_{s,p}$ be the shortest path from the generalized site s to point p and $q = \pi_{s,p} \cap s$. Since $\pi_{s,p}$ is the shortest path from point q to point p it traverses up to $O(\sqrt{n})$ faces (Section 2.4.3). □

Lemma 5.4.21 *The number of breakpoints of a bisector between two generalized sites on a realistic terrain is $O((n + \tilde{r})\sqrt{n})$.*

Proof. Let $\beta_{i,j}$ be the bisector defined by sites s_i and s_j, and p and q two consecutive points whose shortest paths to the sites cross a different sequence of triangle edges. Notice that these points have two shortest paths that go from the sites to them. There may exist up to $O(n+\tilde{r})$ such points. The part of the bisector joining p to q intersects at most $O(\sqrt{n})$ triangles. This is proved in the same way that Moet et al. proved the result for the special case of point sites [106]. Let us consider $\Pi_{s_i,p}$, $\Pi_{s_i,q}$, $\Pi_{s_j,p}$, and $\Pi_{s_j,q}$, the shortest paths from s_i and s_j to points p and q. If there exits more than one shortest path those enclosing smallest area are considered. Since $\Pi_{s_i,p}$, $\Pi_{s_i,q}$ are the shortest path from points p and q to a point of s_i and s_j, respectively, they cross $O(\sqrt{n})$ triangles (Section 2.4.3). Points p and q are two consecutive points of $\beta_{i,j}$ where the edges crossed by the shortest path change, thus, in the enclosed area by the paths there are no terrain vertices and the bisector also intersects $O(\sqrt{n})$ triangles. □

Proposition 5.4.2 *The closest Voronoi diagram of a set S of r generalized sites defining $\tilde{r}$ line segments has complexity $O((n+\tilde{r})\sqrt{n})$.*

Proof. Shortest paths from a segment source can be homeomorphicaly transformed to shortest paths from three points representing the two segment endpoints and the segment interior. Moet et al. (Section 2.6.3.2) proved that when $\tilde{r}$ point sources are considered there are $O(n + \tilde{r})$ discontinuity points, points where there exists a discontinuity on the edge sequence traversed by shortest paths. Since each segment delimited by two such points intersects $O(\sqrt{n})$ triangles, the complexity of the is $O((n+\tilde{r})\sqrt{n})$. □

The Voronoi diagram of a set of r sites on a Realistic terrain has complexity $O((n + r)\sqrt{n})$ (Section 2.6.3). Considering this bound the provided Lemmas and the Theorem for the special case of a realistic terrain are the following ones.

Lemma 5.4.22 *We have $B_{\leq k} = O(k\sqrt{n}(kn + r))$ for any $1 \leq k \leq r - 1$.*

Proof. This proof is analogous to proof of Lemma 5.4.18 considering $B(R) = O(n\sqrt{n} + |R|\sqrt{n})$.

Since $|R|$ follows a binomial distribution, we have $\mathbb{E}[|R|] = rp\sqrt{n}$, and therefore

$$\mathbb{E}[B(R)] = O(\sqrt{n}(n + rp)). \tag{5.2}$$

By using the reasoning provided in Lemma 5.4.18 we have that

$$\mathbb{E}[B(R)] = p^2(1-p)^{k-1}B_{\leq k}.$$

Manipulating, substituting equation (5.2) and setting $p = 1/k$ we obtain

$$B_{\leq k} = O\left(\frac{n\sqrt{n} + \sqrt{n}r/k}{(1/k)^2(1-1/k)^{k-1}}\right) = O\left(k^2 n\sqrt{n} + kr\sqrt{n}\right) = O(k\sqrt{n}(kn + r)).$$

□

The number of vertices is not modified so the following lemma can be given

Lemma 5.4.23 *We have $V_{\leq k} = O(k^2 r)$ for any $1 \leq k \leq r - 1$ when a realistic terrain is considered.*

Combining Lemmas 5.4.22–5.4.22 we can conclude the following theorem.

Theorem 5.4.3 *Let $\mathcal{P}$ be a realistic terrain with n triangles, let S be a set of r sites, where each site is either a segment or a point in $\mathcal{P}$, and let k be an integer $1 \leq k \leq r - 1$. The complexity of all the order-i Voronoi diagrams of S together for $i = 1 \ldots k$ is $O(k\sqrt{n}(knr) + k^2 r)$.* □

5.5 Conclusions

We have presented an algorithm for computing exact shortest paths from generalized sources (point, segment, polygonal line and polygon sources) on triangulated polyhedral surfaces with several generalized obstacles. The algorithm takes $O(n(n + \tilde{r})\log(n + \tilde{r}))$ time and $O(n(n+\tilde{r}))$ space, where $\tilde{r}$ is the number of segments conforming the generalized source. The algorithm is extended to the case of several generalized sites when their implicit distance field is obtained in $O(n(n + \tilde{r})\log(n + \tilde{r}))$ time and $O(n(n + \tilde{r}))$ space

where $\tilde{r}$ is the total number of segments conforming the generalized sites. The output of algorithm is a codification of the distance function or distance field. From this codification the distance from a point on $\mathcal{P}$ to the site and the actual shortest path can be obtained in $O(n + \tilde{r} + \overline{n})$ time, where $\overline{n}$ is the number of faces the path goes through. We can also obtain the shortest path distance and the actual shortest path to the closest site of a set S in in $O(n + \tilde{r} + \overline{n})$ time.

We want to mention that the algorithm of Chen and Han, a typical algorithm for computing exact shortest paths on polyhedral surfaces, can also be adapted to support generalized sites without increasing the time and storage complexity of the original algorithm. This algorithm works when the one angle one split observation holds, and it holds when generalized sites are considered.

Finally we provide a theoretical study of high-order Voronoi diagrams of a set of generalized sources a polyhedral surface $\mathcal{P}$. We generalize some basic properties of order-k Voronoi diagrams such as that a k-cell and can not contain a $(k+1)$-cell, or that a k-vertex has at least degree three. We prove that the furthest Voronoi diagram has $O(r)$ path connected cells. We then present some pathologies of order-k Voronoi diagrams, for instances: two bisectors may cross more than twice, there exist up to $O(r)$ k-cells without vertices, and k-cells ar not necessarily path-connected. Concerning general polyhedral surfaces we end with a study the complexity of the order-k Voronoi diagrams by using a probabilistic proof which is $O(k^2n^2 + k^2r + krn)$ and $\Omega(krn + rk^2)$. This section is ended by providing some bounds and properties of shortest paths and Voronoi diagrams of arbitrary order on realistic terrains which can be considered as a special case of triangulated polyhedral surfaces.

Chapter 6

Weighted distances on polyhedral surfaces

In this chapter, the problem of computing approximate distance functions from a generalized source on a possibly non-convex weighted polyhedral surface $\mathcal{P}$ with obstacles is addressed.

We present an algorithm to compute $(1+\varepsilon)$-approximate weighted shortest paths, and consequently, the distances from a generalized source (point, segment, polygonal chain or polygonal region) on a non-convex weighed polyhedral surface in which polygonal chain or polygon obstacles are allowed (Section 6.1). The algorithm easily extends to the case of several sources providing their distance field, which intrinsically encodes the closest Voronoi diagram of the set of generalized sites (Section 6.2). From the implicit representations, the distance or the shortest path to any point $p \in \mathcal{P}$ is obtained (Section 6.3). Finally we end with some conclusions (Section 6.4). Algorithms described in this chapter are implemented and implementation results are presented in Chapter 7 (Section 4.5).

Let $\mathcal{P}$ be a possibly non-convex polyhedral surface represented as a mesh consisting of n triangular faces $f_1, \ldots, f_n$ with associated positive weights $w_1, \ldots, w_n$, respectively, with $w_i \geq 1$. The weight associated with an edge is the minimum of the weights of the two neighboring faces. The cost of a path Π on $\mathcal{P}$ is defined as $\|\Pi\| = \sum_{i=1}^{n} w_i|\Pi_i|$, where $|\Pi_i|$ denotes the Euclidean length of the path lying in face f_i. Notice that a non-weighted polyhedral surface can be considered a weighted polyhedral surface with $w_i = 1$ for $i = 1 \ldots n$.

6.1 Implicit distance function computation

We compute $(1+\varepsilon)$ approximate distance functions for generalized sources by extending the weighted discrete graph and the Bushwack strategy proposed by Sun and Reif [143] to handle generalized sources. Sun and Reif use Bushwhack strategy to compute distances from a vertex source to the graph nodes, on a graph that provides $(1+\varepsilon)$-approximate weighted shortest paths from a source vertex to any vertex.

Bushwack strategy (Section 2.6.2.4) can only be used if two shortest path do not intersect in the interior of a face. This strategy tracks and keeps together groups of shortest paths by partitioning each face edge into a set of (discrete) intervals so that all the shortest path that cross an interval have the same structure. Given two edges e and e' of a face f and v a node on e, interval $I_{v,e,e'}$ codifies the shortest paths emanating from node v to nodes on edge e' and the distance they define, for further details see Section 2.6.2.4. Node v, also denoted I^s, is called the *virtual source* of interval $I_{v,e,e'}$.

We start this section by describing our discretization scheme which provides $(1+\varepsilon)$-approximate distances from a point or segment source to a graph node. Next, we show that the Bushwack strategy can be used for the case of a point source in general position and for a segment source. We consider polygonal line and polygon sources and end by placing generalized obstacles on the surface.

6.1.1 Discretization scheme

We define a logarithmic discretization scheme which places Steiner points on the edges of the triangulated polyhedral surface $\mathcal{P}$ according to a given parameter $0 < \varepsilon \leq 1$. The scheme we propose adapts the one provided by Sun and Reif [143] to take into account a point or segment source s which is not necessarily a vertex nor an edge.

We start with some definitions and notation. We denote E the set of edges of $\mathcal{P}$, f a face, e an edge, v a vertex and x an arbitrary point. We define $F(f)$ as the union of the faces without empty intersection with f, face f included, $F(x)$ as the union of the faces containing point x. Let $E(x)$ denote the set of edges containing x and $S(x)$ the set of sources contained in $F(x) \setminus \{x\}$. In the current case $S(x)$ will be empty or $\{s\}$. Let D_x be the minimal distance between x and $S(x) \cup E \setminus E(x)$, $D_e = sup\{D_v | v \in e\}$ and $D_{v_e} = D(e)$. For each point x we define the radius $r'(x)$ to be $D_x/5$, and the weighted radius $r(x)$ of x to be $\frac{w_m}{w_M} r'(x)$, where w_m, w_M are the minimal and maximal weights of

the faces in $F(x)$. Notice that these radius take into account not only the proximity of the edges in $E \setminus E(x)$ but also the proximity of s to x, when x is in $F(s)$. Finally $V(x)$ is the vicinity of point x. Vicinity $V(x)$ is defined as having radius $r_\varepsilon(x) = \varepsilon r(x)$. It contains all the points around x at a distance of at most $r_\varepsilon(x)$. When instead of a point x, a source s is considered $F(s)$, $S(s)$, $E(s)$, D_s, $r'(s)$, $r(s)$, $V(s)$ and $r_\varepsilon(s)$ are analogously defined.

Steiner points are placed on the edges of $\mathcal{P}$ considering that source s and each vertex v has a vicinity. For a given vertex v_i Steiner points $v_{i,1}, v_{i,2}, \ldots, v_{i,k_i}$ are on edge $e = \overline{v_i v_j}$ and outside the vicinities. They are chosen according to the following criteria:

- When e is not in $F(s)$, the first node is placed so that $|\overline{v_i v_{i,0}}| = r_\varepsilon(v_i)$, the rest using the equality $|\overline{v_{i,k} v_{i,k+1}}| = \varepsilon\ D_{v_{i,k}}$, until placing v_{i,k_i} where $\overline{v_{i,k_i} v_i} + \varepsilon\ D_{v_{i,k_i}} \geq |\overline{v_i v_e}|$.
- When $e \in F(s)$ we take into account the vicinity of s, $V(s)$. If $e \cap V(s) = \emptyset$ we proceed as in the previous case. Otherwise, when $e \cap V(s) \neq \emptyset$, $e \setminus V(s)$ defines two subedges $\overline{v_i v}$ and $\overline{v v_j}$, since we are considering v_i we choose Steiner points on $e' = \overline{v_i v}$. Steiner point v_{i_0} is placed so that $|\overline{v_i v_{i,0}}| = r_\varepsilon(v_i)$, Steiner points for $i = 1 \ldots j_i$ so that $|\overline{v_{i,k} v_{i,k+1}}| = \varepsilon\ D_{v_{i,k}}$ until $k = j_i$ where $\overline{v_{i,j_i} v} + \varepsilon D_{v_{i,j_i}} \geq |\overline{v_i, v_{e'}}|$. The rest of the points are placed considering endpoint v, Steiner point $v_{i,k_i} = r_\varepsilon(v)$, for $k = k_i \ldots j_i + 1$ according to $|\overline{v_{i,k} v_{i,k+1}}| = D_{v_{i,k}}$ until $\overline{v_{i,j_i+1} v} + \varepsilon D_{v_{i,j_i+1}} \geq |\overline{v, v_{e'}}|$.

Lemma 6.1.1 *The number of Steiner points placed on each edge is in* $O\left(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$.

Proof. The logarithmic scheme places $O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ Steiner points per edge. The hidden constant when the first case is considered is $C(e) = O\left(\frac{|e|}{D_e} \log \frac{|e|}{\sqrt{r(v_1) r(v_2)}}\right)$. When the second case is used the constant becomes $C(e) = O\left(\frac{(D_{e'} + D_{e''})|e|}{D_{e'} D_{e''}} \log \frac{|e|}{\sqrt{r(v_1) r(v_2)}}\right)$ where e' and e'' are the sub-edges defined by $e \setminus V(s)$.

□

A graph whose nodes are the vertices of $\mathcal{P}$ and the Steiner points is built. Graph edges consist of face-crossing segments joining pairs of Steiner points of the same face, and edge-using segments joining consecutive nodes along an edge (see Figure 6.1), and each vertex v to the first node of the edges having v as incident vertex.

Notice that when s is a vertex, this scheme is the logarithmic scheme provided by Sun and Reif [143]. We will prove that we have adapted the logarithmic scheme to guarantee

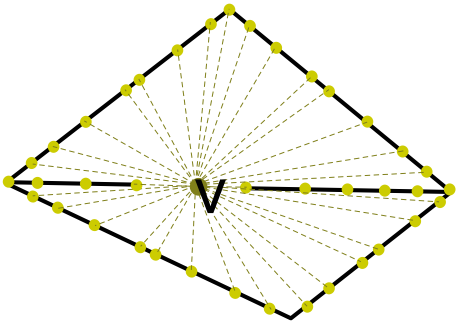
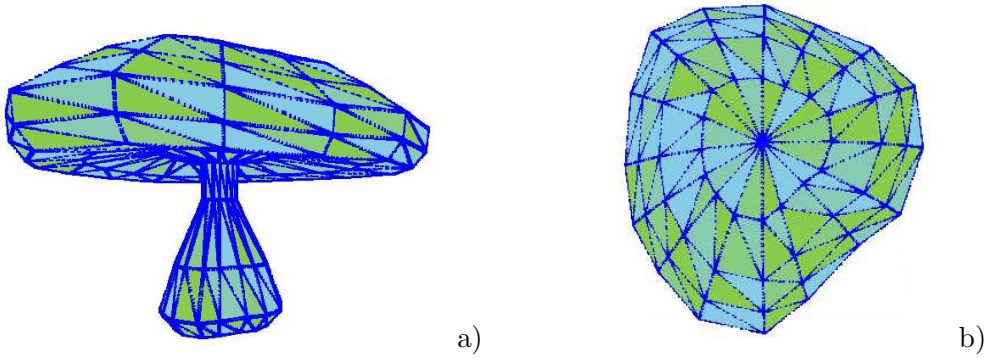


Figure 6.1: Steiner points on two surface faces with the edges emanating from node v.

an $(1+\varepsilon)$-approximate shortest path from a point or source s to the graph nodes with s in an arbitrary position. In this chapter we will show that this scheme, can be used to handle any generalized source.

In Figure 6.2 we show a triangulated polyhedral surface representing mushroom. The polyhedral has 448 faces, we have considered $\varepsilon = 0.5$ and the figure shows the Steiner points obtained by using this discretization scheme. Faces are painted in a blue gradation according to the face weight. Green faces have weight around one and blue faces around five.



a) b)

Figure 6.2: Steiner points on the faces of a triangulated polyhedral surface representing a mushroom.

6.1.1.1 Approximation analysis

We are interested in bounding the committed error when using the graph presented in Section 6.1.1 to compute weighted distances from s: a point source in arbitrary position

or a segment source. With this aim different Lemmas are presented.

From now on we denote $\tilde{w}$ and w', the largest and smallest weight of the faces of $\mathcal{P}$ and we assume $\varepsilon \leq 1$.

Lemma 6.1.2 *For any path p from source $s \in f_s$ to node $t \in f_t$, which does not intersect the vertex vicinities of f_t, there is a normalized path $\hat{p}$ so that $||\hat{p}|| = (1 + \frac{\varepsilon}{2})||p||$.*

Proof. Assume that p passes through a vertex vicinity, $V(v)$. We distinguish between two situations depending on whether v is a vertex of f_s or not (See Figure 6.3).

A) Let us assume that v is a vertex of f_s. We denote u_1, u_2, the first bending point of p in $V(v)$ and u_2'' the last bending point of p in $F(v)$ (See Figure 6.3 a)). By using the definition of D_v and the fact that the radius of $V(v)$ is $\frac{\varepsilon}{5}D_v$ we obtain:

On the one hand, that $|p[u_2'', u_2]| + |\overline{u_2 v}| \geq |\overline{u_2'' v}| \geq D_v$ and $|p[u_2'', u_2]| \geq D_v - \varepsilon\frac{D_v}{5}$ for being $|\overline{u_2 v}| \leq \varepsilon\frac{D_v}{5}$. Therefore,

$$\frac{|\overline{u_2 v}|}{|p[u_2'', u_2]|} \leq \frac{\varepsilon \cdot D_v/5}{D_v - \varepsilon \cdot D_v/5} = \frac{\varepsilon}{5-\varepsilon} \leq \frac{\varepsilon}{4} \qquad (1)$$

On the other hand, that $|p[s, u_1]| + |\overline{u_1 v}| \geq |\overline{sv}| \geq D_v$ and $p[s, u_1]| \geq D_v - \varepsilon\frac{D_v}{5}$ for being $|\overline{u_1 v}| \leq \varepsilon\frac{D_v}{5}$. Therefore,

$$\frac{|\overline{u_1 v}|}{|p[s, u_1]|} \leq \frac{\varepsilon \cdot D_v/5}{D_v - \varepsilon \cdot D_v/5} = \frac{\varepsilon}{5-\varepsilon} \leq \frac{\varepsilon}{4} \qquad (2)$$

We denote $r_1(/r_2)$ the region with minimum weight traversed by $p[s, u_1](/p[u_2, u_2''])$ and $u_1'(/u_2')$ the last point of p in $r_1(/r_2)$. Notice that u_1' may be s. Finally we denote $w_{r_1}(/w_{r_2})$ the weight of $r_1(/r_2)$ (See Figure 6.3 a)). Let us consider the normalized path $\hat{p}[s, u_2''] = \{p[s, u_1'], \overline{u_1' v}, \overline{v u_2'}, p[u_2', u_2'']\}$. By comparing the costs of $\hat{p}[s, u_2'']$ and $p[s, u_2'']$ and using inequalities (1) and (2) we obtain that:

$$\begin{aligned}
||\hat{p}[s, u_2'']|| - ||p[s, u_2'']|| &= w_{r_1}|\overline{u_1' v}| + w_{r_2}|\overline{v u_2'}| - ||p[u_1', u_1]|| - ||p[u_1, u_2]|| - ||p[u_2, u_2']|| \\
&\leq (w_{r_1}|\overline{u_1' v}| - ||p[u_1', u_1]||) + (w_{r_2}|\overline{v u_2'}| - ||p[u_2, u_2']||) \\
&\leq w_{r_1}|\overline{u_1 v}| + w_{r_2}|\overline{v u_2}| \leq w_{r_1}\frac{\varepsilon}{4}|p[s, u_1]| + w_{r_2}\frac{\varepsilon}{4}|p[u_2, u_2'']| \\
&\leq \frac{\varepsilon}{4}||p[s, u_1]|| + \frac{\varepsilon}{4}||p[u_2, u_2'']|| \leq \frac{\varepsilon}{4}||p[s, u_2'']||
\end{aligned}$$

Therefore,

$$||\hat{p}[s, u_2'']|| \leq (1 + \frac{\varepsilon}{4})||p[s, u_2'']|| \qquad (3)$$

B) Now, we assume that v is not a vertex of f_s. We proceed in a similar way. Let u_1'', and u_2'' be the first and last bending points of p in $F(v)$ (See Figure 6.3 b)). Point u_1'' plays the role of s, so we define $\hat{p}[u_1'', u_2''] = \{p[u_1'', u_1'], \overline{u_1'v}, \overline{vu_2'}, p[u_2', u_2'']\}$ (Figure 6.3 b)). It can be proved that

$$|\overline{u_1v}|/|p[u_1'', u_1]| \leq \frac{\varepsilon}{4} \qquad (4)$$

(equivalent to (1)), again using the same reasoning, a result equivalent to (3) is obtained:

$$||\hat{p}[u_1'', u_2'']|| \leq (1 + \frac{\varepsilon}{4})||p[u_1'', u_2'']|| \qquad (5)$$

Assume that p passes through l vertex vicinities, $V(v_1)$, $V(v_1)$, ..., $V(v_l)$. For each vertex we replace the subpath p_i of p that passes through $V(v_i)$ for the normalized path $\hat{p}_i$. Using the correspondent inequality ((3) or (5)) for each vicinity, we find that $||\hat{p}|| \leq ||p|| + \frac{\varepsilon}{4} \sum_{i=1}^{l} ||p_i|| \leq (1 + \frac{\varepsilon}{2})||p||$. □

Notice that in the previous Lemma we have used a vertex vicinity $V(x)$ of radius $\varepsilon r'(v)$. However, we have defined vicinities of radius $\varepsilon r(v) = \varepsilon \frac{w_m}{w_M} r'(v)$ which is necessary in the following Lemma.

Lemma 6.1.3 *For any path p from source $s \in f_s$ to node $t \in f_t$ there is a normalized path $\hat{p}$ so that $||\hat{p}|| = (1 + \frac{\varepsilon}{2})||p||$.*

Proof. Assume that p passes through a vertex vicinity, $V(v)$. In Lemma 6.1.2 we have studied the cases when v is not a vertex of f_t, thus we assume that v is a vertex of f_t. By using the notation introduced in Lemma 6.1.2 we define the normalized path $\hat{p}[u_1'', t] = \{p[u_1'', u_1'], \overline{u_1'v}, \overline{vu_2'}, p[u_2', t]\}$ (See Figure 6.3 c)). In this case $|\overline{vu_2}| + |p[u_2, u_1'']| \geq |\overline{vu_1''}| \geq D_v$ and $|p[u_2, u_1'']| \geq D_v - \varepsilon \frac{D_v}{5}$ for being $|\overline{vu_2}| \leq \varepsilon \frac{w_m}{w_M} \frac{D_v}{5}$. Therefore,

$$\frac{|\overline{u_2v}|}{|p[u_2, u_1'']|} \leq \frac{\varepsilon \cdot w_m/w_M \cdot D_v/5}{D_v - \varepsilon \cdot D_v/5} = \frac{w_m}{w_M} \frac{\varepsilon}{5 - \varepsilon} \leq \frac{w_m}{w_M} \frac{\varepsilon}{4} \qquad (6)$$

Let us now bound $||\hat{p}[u_1'', t]|| - ||p[u_1'', t]||$, by proceeding as in Lemma 6.1.2 and using inequalities (4) and (5):

$$\begin{aligned}||\hat{p}[u_1'', t]|| - ||p[u_1'', t]|| &\leq \ldots \leq w_{r_1}|\overline{u_1 v}| + w_{r_2}|\overline{v u_2}| \\ &\leq w_{r_1}\frac{\varepsilon}{4}|p[u_1'', u_1]| + w_{r_2}\frac{w_m}{w_M}\frac{\varepsilon}{4}|p[u_1'', u_2]| \\ &\leq \frac{\varepsilon}{4}||p[u_1'', u_1]|| + \frac{w_m}{w_M}\frac{\varepsilon}{4}\frac{w_{r_1}}{\min(w_{r_1}, w_{r_2})}||p[u_1'', u_2]|| \\ &\leq \frac{\varepsilon}{4}||p[u_1'', t]||.\end{aligned}$$

Therefore,

$$||\hat{p}[u_1'', t]|| \leq (1 + \frac{\varepsilon}{4})||p[u_1'', t]|| \qquad (7)$$

If path p passes through l vertex vicinities, $V(v_1)$, $V(v_1)$, $\ldots$, $V(v_l)$, then for each vertex, we replace the subpath p_i of p that passes through $V(v_i)$ for the normalized path $\hat{p_i}$. Using the corresponding inequality ((3),(4) or (7)) for each vicinity we find that

$$||\hat{p}|| \leq ||p|| + \frac{\varepsilon}{4}\sum_{i=1}^{l}||p_i|| \leq (1 + \frac{\varepsilon}{2})||p||.$$

□

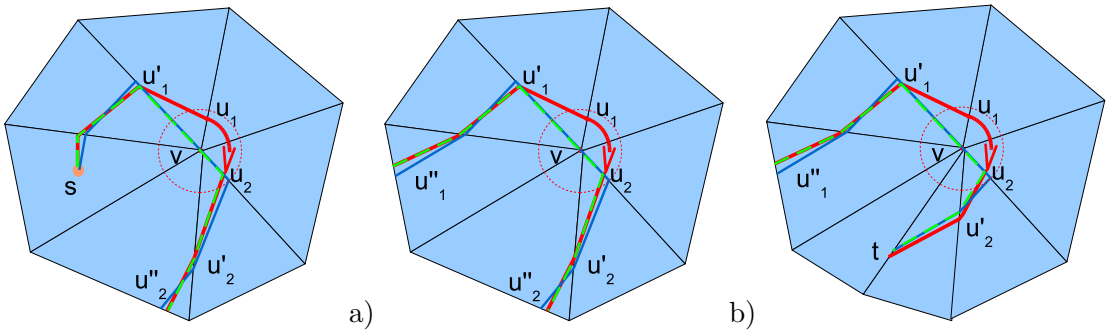


Figure 6.3: A subpath of path p from $s \in f_s$ to $t \in f_t$ in red, in dashed green a normalized path $\hat{p}$ and in blue a path on the graph. Path p goes through vertex vicinity $V(v)$, where v is: a) a vertex of the face containing s; b) a vertex of a face not containing s nor t. c) a vertex of a face containing t.

Theorem 6.1.1 *The obtained graph contains a* $(1 + 3\varepsilon)$*-approximation of the shortest path* Π *from point source* s *to an arbitrary node* t*.*

Proof. According to Lemma 6.1.3 a normalized path $\hat{\Pi}$ such that $||\hat{\Pi}|| \leq (1 + \frac{1}{2}\varepsilon)||\Pi||$ exists.

Let $\overline{v} = \overline{v_1 v_2}$ be a segment of $\hat{\Pi}$ contained in a face f. Endpoint $v_1(/v_2)$ of $\overline{v_1 v_2}$ is on a segment $\overline{u_{1,1}, u_{1,2}}(/\overline{u_{2,1}, u_{12,2}})$ which is delimited by either two Steiner points or a vertex and a Steiner point. They are named a *pure Steiner segment* and a *half Steiner segment*, respectively. Segments conforming $\hat{\Pi}$ belong to one of the following three categories: (1) Both endpoints are on pure Steiner segments; (2) An endpoint is on a pure Steiner segment and the other on a half Steiner segment; (3) Both endpoints are on half Steiner segments. For each of the three cases, it can be proved that $|\overline{u_{1,i}, u_{2,j}}| \leq (1+2\varepsilon)|\overline{v_1 v_2}|$ for $i, j \in \{1, 2\}$:

$$|\overline{v_1 v_2}| \leq |\overline{v_1 u_{1i}}| + |\overline{u_{1i} u_{2j}}| + |\overline{u_{2j} v_2}| \leq$$

$$\leq \varepsilon|\overline{v_1 v_2}| + |\overline{u_{1i} u_{2j}}| + \varepsilon|\overline{v_1 v_2}| = |\overline{u_{1i} u_{2j}}| + 2\varepsilon|\overline{v_1 v_2}|.$$

Consequently, we can construct a path Π' such that

$$||\Pi'|| \leq (1+2\varepsilon)||\hat{\Pi}|| \leq (1+2\varepsilon)(1+\frac{\varepsilon}{2})||\Pi|| \leq (1+\frac{5\varepsilon}{2})||\Pi||.$$

□

6.1.2 From point source to node

Let us consider an arbitrary point source s in a face of $\mathcal{P}$. We provide a way to obtain approximate shortest paths from s to any node of the graph considering the discrete graph presented in Section 6.1.1 by adapting the Bushwack strategy.

6.1.2.1 Distance function propagation

We have proved that the obtained graph provides $(1+\varepsilon)$-approximate shortest paths from a point source s to the graph nodes. The fastest way to compute distances on discrete graphs is by using Bushwack strategy. However, this strategy can only be used when shortest paths do not intersect in the faces interior.

Since according to Property 2.6.5 (Section 2.6.2), two shortest paths originating from the same source point cannot intersect in the interior of any face, we can use Bushwhack strategy to compute distances from a point source s.

Now s is not a vertex, thus, we have to adapt the initialization step where intervals $I_{s,-,e}$ associated to the source s are created. These intervals encode distance function d_s of source s, on the edges e of the triangle(s) containing s. Bushwack strategy propagates the distance function across mesh triangles in a lazy and best-first propagation scheme. When

a node v on an edge e is first visited, several intervals $I_{v,e,e'}$ are created, with e' opposite to v when v is a vertex or adjacent to e otherwise. Interval $I_{v,e,e'}$ contains those contiguous nodes of e', whose shortest path from s to $v' \in I(v, e, e')$, may use node v before arriving at v' via an edge-using or face-crossing segment contained on the face determined by e' and v (See Figure 6.4).

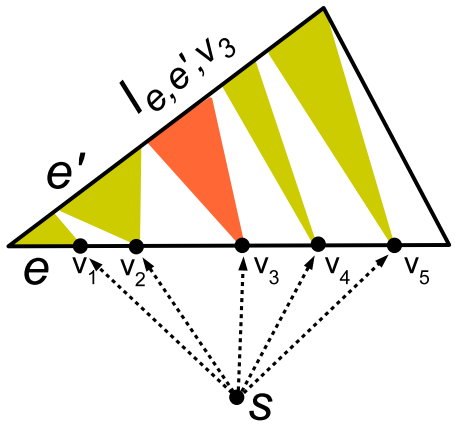


Figure 6.4: A point source s with approximate shortest paths to vertices $v_1, \ldots, v_5$ and the interval $I_{v_3,e,e'}$ associated to v_3.

According to the observations given in this section, we can use Bushwack strategy whose complexity is $O(mn \log(mn))$ time and $O(mn)$ space when we consider a surface with n edges and m nodes per edge. Thus according to Lemma 6.1.1 and Theorem 6.1.1 we state the following Theorem.

Theorem 6.1.2 *$(1+3\varepsilon)$-approximate distances from a point source s to the graph nodes can be obtained in $O(mn \log(mn))$ time and $O(mn)$ space, where $m \in O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$.* □

6.1.3 From segment source to node

Let us now consider a segment source s on $\mathcal{P}$. We compute approximate weighted distances from s to $\mathcal{P}$ building the graph by using the provided discretization scheme. We show that we obtain $(1+\varepsilon)$-approximate distances which can be computed by using Bushwack strategy.

We use the discretization scheme presented in Section 6.1.1. It is important to note that D_s is the minimal distance between s and the edges of $\mathcal{P}$ that do not intersect s. This

is used to define the radius of $V(s)$, the vicinity of s, which is $r_\varepsilon(s) = \varepsilon \frac{w_m}{w_M} \frac{D_s}{5}$. Given point x, D_x is the minimal distance between x and $S(x) \cup E \setminus E(x)$, where $S(x) = \{s\} \setminus \{x\}$ when $s \in F(x)$ or $S(x) = \emptyset$, otherwise. Consequently the discretization scheme takes into account the proximity of segment s and provides a $(1+3\varepsilon)$-approximation of the optimal path from the segment source s to a node t.

6.1.3.1 Distance function propagation

Again, to be able to compute approximate shortest paths by using the Bushwhack strategy, we have to prove that two shortest paths from s to arbitrary points on $\mathcal{P}$ do not intersect in the interior of a face (Lemma 6.1.4) and adapt the initialization step where we create intervals on the edges of the face(s) containing s. Notice that these intervals will contain both edge-using and face-crossing segments from s to the nodes. Each node will be joined to the point of s defining minimum distance. Consequently, in the faces containing f we can find two different types of edges: those emanating from the endpoints of s and the rest that are perpendicular to segment s (See Figure 6.5).

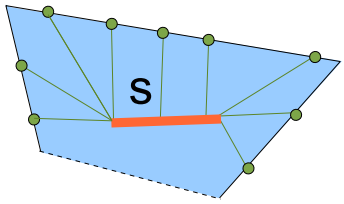


Figure 6.5: Shortest paths from the orange segment source $s \in f_s$ to some green nodes on f_s.

Lemma 6.1.4 *Let s be a segment source and p, p' two points of $\mathcal{P}$. Denote $\Pi_{s,p}$ and $\Pi_{s,p'}$ the shortest paths from s to p and to p', respectively. Shortest paths $\Pi_{s,p}$ and $\Pi_{s,p'}$ do not intersect in the interior of any face, except for in s.*

Proof. To prove the lemma, we will assume that $\Pi_{s,p}$ and $\Pi_{s,p'}$ intersect in a point $q \notin s$ in the interior of a face f. We will define two new paths $\Pi'_{s,p}$ and $\Pi'_{s,p'}$ such that $||\Pi'_{s,p}|| + ||\Pi'_{s,p'}|| < ||\Pi_{s,p}|| + ||\Pi_{s,p'}||$, this contradicts the fact that $\Pi_{s,p}$ and $\Pi_{s,p'}$ shortest paths. Let v_1 and v_2 be the intersection points of $\Pi_{s,p}$ with f (v_1 closer to s than v_2). Points v'_1 and v'_2 are analogously defined for $\Pi_{s,p'}$. In Figure 6.6 we can easily see that if we define two new paths replacing the segments containing q (the shortest path intersection point) for the dotted segments, we obtain two paths $\Pi'_{s,p} = \{\Pi_{s,v_1}, \overline{v_1 v'_2}, \Pi_{v'_2,p'}\}$ and $\Pi'_{s,p'} = \{\Pi_{s,v'_1}, \overline{v'_1 v_2}, \Pi_{v_2,p}\}$ fulfilling the previously mentioned inequality. In fact, if w is the weight

of f, $||\Pi_{s,p}|| = ||\Pi_{s,v_1}|| + w|v_1v_2| + ||\Pi_{v_2,p}||$, and $||\Pi_{s,p'}|| = ||\Pi_{s,v'_1}|| + w|v'_1v'_2| + ||\Pi_{v'_2,p'}||$. Since $|v_1v_2| + |v'_1v'_2| > |v_1v'_2| + |v'_1v_2|$, then $||\Pi_{s,p}|| + ||\Pi_{s,p'}|| > ||\Pi'_{s,p}|| + ||\Pi'_{s,p'}||$. Consequently, two shortest paths can not cross in the interior of a face f. □

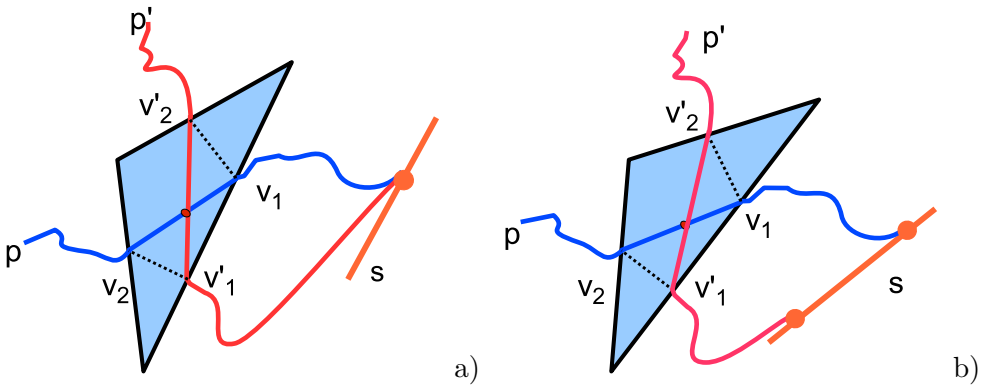


Figure 6.6: Shortest paths from segment source s to points p and p', when p and p' are closer to: a) the same point of s. b) different points of s.

We can use Bushwack strategy (Lemma 6.1.4) to compute distances from s on the graph to provide $(1+3\varepsilon)$-approximate distances (Theorem 6.1.1). The complexity of Bushwack strategy is $O(mn\log(mn))$ time and $O(mn)$ space when we consider a surface and n edges with m nodes per edge with $m \in O(\frac{1}{\epsilon}\ \log\frac{1}{\epsilon})$ (Lemma 6.1.1). Therefore, the following Proposition can be stated.

Proposition 6.1.1 *We can obtain a $(1+3\varepsilon)$-approximate distance defined by a segment source s in $O(mn\log(mn))$ time and $O(mn)$ and space, where $m \in O(\frac{1}{\epsilon}\ \log\frac{1}{\epsilon})$.* □

6.1.4 From polygonal line source to node

The distance function defined by a polygonal chain s defined by r' segments, is obtained by considering s as the union of all the segments s_i defining it. In this case we define a vicinity around s as the union of the vicinities of the segments defining s.

The discretization scheme follows the same guidelines. However, now $V(s)$ may intersect two or more times the same edge e. Consequently when we consider $e \setminus V(s)$ we can obtain more than two sub-edges, and some of them are not incident to a vertex of $\mathcal{P}$. In subedges $e' = \overline{uv}$ not incident to a vertex we place one Steiner point at each endpoint u and v and then we use the logarithmic scheme along e' using first u and next v, in both

cases until we get point $v_{e'}$ (See Figure 6.7). All the Steiner points on these edges should be placed the first time that e is considered.

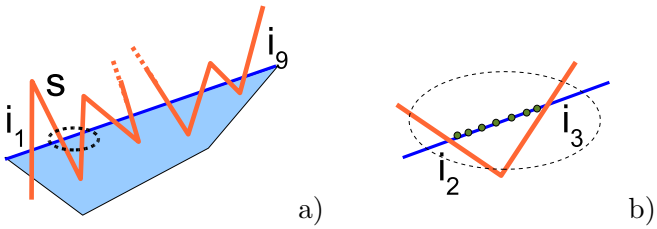


Figure 6.7: a) A polygonal line source intersecting an edge e, in dark blue, nine times. b) Detail of subedge $\overline{i_2 i_3}$ with its corresponding Steiner points.

Consequently, when s intersects r'-times edge e, we obtain $r'+1$ subedges and we place $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ nodes per subedge. The total number of Steiner points in this case is $O(r'\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$. This result is stated in the following Lemma.

Lemma 6.1.5 *The number of Steiner points generated on an edge intersected r' times by the Vicinity of s is $O(r'\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ Steiner points.* □

We assume that the intersection between s and the edges of $\mathcal{P}$ is computed when s is placed on $\mathcal{P}$. By properly storing the intersection points, the r' subedges can be obtained in $O(r')$ time. Previously proved results related to normalized paths remain true, and therefore we can provide the following Theorem.

Theorem 6.1.3 *The graph contains a $(1+3\varepsilon)$-approximation of any optimal path from a polygonal line source s to a node t.* □

A polygonal region s is a connected region of $\mathcal{P}$ whose boundary, ∂s, is a closed polygonal chain. The distance defined by s is 0 in s and the distance function defined by ∂s in $\mathcal{P}\setminus s$. Therefore, the distance defined by a polygonal region can be computed considering the polygonal line source defining its boundary and propagating its distance to $\mathcal{P}\setminus s$ and giving distance 0 in s. This is summarized in the next result.

Observation 6.1.1 *The distances from a polygonal region s can be computed by considering its boundary polygonal line $\partial(s)$.* □

6.1.4.1 Distance function propagation

We will show that any two shortest paths from s to two different points of $\mathcal{P}$ can not intersect in the interior of any face, except in a point of s.

Lemma 6.1.6 *Let s be a polygonal line source and p, p' two points of $\mathcal{P}$. The shortest paths from p, p' to s do not intersect in the interior of any face, except for perhaps in s.*

Proof. A polygonal line s can be seen as a set of segment sources, with the segments defining the polygonal line. Lemma 6.2.2 in the next section proves that two shortest paths emanating from different generalized sources of a set of sources S do not intersect in the interior of a face except for in S. The proof of the current Lemma is analogous to that one considering a set S containing only the segments conforming the polygonal source s. □

Therefore, we can use Bushwack strategy by adapting the initialization step where we consider each segment s_i conforming the polygonal line s independently. Let $s_1, \ldots, s_k$ b the segments conforming s. We create intervals $I_{s_i,-,e'}$ containing the nodes closer to s_i than to any other already considered segment s_j, $j < i$. When a new segment s_j is considered the previously computed intervals may be modified which is what happens during the propagation step.

This method can also be used to compute distances from a polygonal region according to Observation 6.1.1. When considering a polygonal region, in the initialization step we will only define intervals in $\mathcal{P} \setminus s$.

Thus by using Lemma 6.1.6, Lemma 6.2.1 and making the logical assumption that the polygonal s intersects each edge a constant number of times, we can state the following Proposition.

Proposition 6.1.2 *We can obtain a $(1+3\varepsilon)$-approximate distance defined by a polygonal line or polygon source s in $O(mn\log(mn))$ time and $O(mn)$ space, where $m \in O(\frac{1}{\epsilon}\ \log\frac{1}{\epsilon})$.* □

6.1.5 Polygonal obstacles

Given the polyhedral surface $\mathcal{P}$, with obstacles represented as several surface faces, edges and vertices (Section 5.1.3), we can adapt the algorithm to compute approximate weighted

shortest paths that can go along obstacle edges, but not through them. Obstacle edges have their nodes duplicated somehow. A copy of the nodes is used for each face. The shortest path arriving at a node on an obstacle edge can not go through the edge. It can only be propagated along the edge or back to the face it comes from. Thus the only thing that we have to change are the edges of the obtained graph.

These modifications do not affect the discretization scheme nor the proofs of the provided Lemmas. Consequently we can state the following Theorem which summarizes the results obtained until now, assuming $0 < \varepsilon \leq 1$ and that source s intersects each edge a constant number of times, if it is not on an edge.

Theorem 6.1.4 *Let $\mathcal{P}$ be a weighted triangulated polyhedral surface with generalized obstacles and s be a generalized source on $\mathcal{P}$, a $(1+3\varepsilon)$-approximate distances from s can be obtained by using Bushwack strategy in $O(mn\log(mn))$ time with $m \in O(\frac{1}{\epsilon}\ \log\frac{1}{\epsilon})$.* □

Notice that we have not specified the cost when a realistic terrain is considered because no changes exist. The number of Steiner points is exactly the same and consequently the complexity or the Bushwack strategy too.

6.2 Implicit distance field computation

When we consider a set of sites S we can use the graph to obtain their distance field, which for any node gives the approximate shortest path distance to its nearest site of S.

The scheme provided in Section 6.1.1 contemplates the case when more than one site is considered. Thus, it can be used to obtain a graph where we will obtain a $(1+3\varepsilon)$ - approximation of the distance field. In this case we have to proceed as explained in the case of polygonal sites in Section 6.1.4. When we place Steiner points on an edge e that is intersected by $V(S) = \cup_{s\in S}V(s)$ we have to handle each sub-edge of e apart and use the logarithmic scheme more than twice. Consequently the number of Steiner points, may increase.

Lemma 6.2.1 *The number of Steiner points generated on an edge intersected r' times by vicinity $V(S)$ of the sources in S contains $O(r'\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ Steiner points.* □

The results related to the ε-approximation provided in previous sections remain true. Consequently, we can state the following Theorem.

Theorem 6.2.1 *The graph allows the computation of a* $(1+3\varepsilon)$*-approximate distance field defined by a set of generalizes sites* S*.* □

In Figure 6.8 we show the triangulated polyhedral surface representing a mushroom with a polygonal source on the right and a line segment source on the right. Steiner points are colored in a from blue to green gradation according to the distance to the closest site.

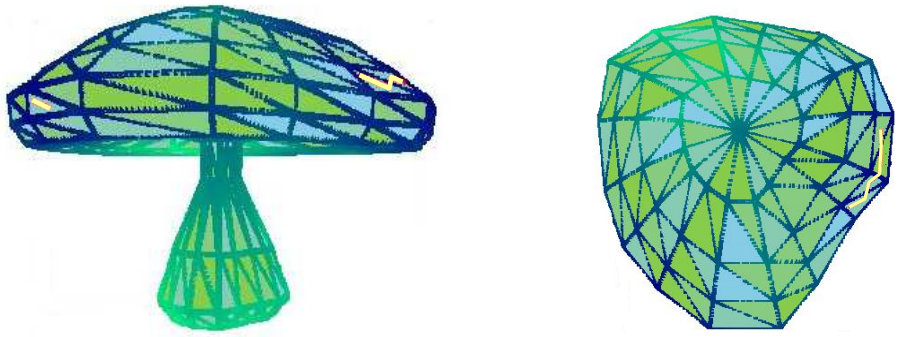


Figure 6.8: A triangulated polyhedral surface with two generalized sites. Steiner points are painted in a from blue to green gradation according to the distance to the closest site.

6.2.1 Distance field propagation

Bushwhack algorithm can be used to obtain the distance field of a set of generalized sites as this is proved in the following Lemma.

Lemma 6.2.2 *Let us consider a set of generalized sites* S *and two points of* $\mathcal{P}$*,* p *and* p'*. Denote* $\Pi_{S,p}$ *the shortest paths that joins* p *to the closest site of* S *to* p*, and* $\Pi_{S,p'}$ *the one joining* S *with* p'*. Then* $\Pi_{S,p}$ *and* $\Pi_{S,p'}$ *do not intersect in the interior of a face.*

Proof. Assume that $\Pi_{S,p}$ and $\Pi_{S,p'}$ are the shortest paths form p and p' to S, respectively. Then, the sum of their costs is smaller than the sum of the costs of any other two paths from p and p' to S. We will assume that $\Pi_{S,p}$ and $\Pi_{S,p'}$ intersect in the interior of a face f and get a contradiction. Let $s \in S$ be the site where $\Pi_{S,p}$ originates, v_1 and v_2 the intersection points of $\Pi_{p,S}$ with f (v_1 closer to s than v_2). Site $s' \in S$ and points v_1' and v_2' and analogously defined for $\Pi_{S,p'}$ (see Figure 6.9). According to this notation the cost of $\Pi_{S,p}$, $\|\Pi_{S,p}\|$, is $d(s,v_1)+w|v_1v_2|+d(v_2,p)$, and $\|\Pi_{S,p'}\| = d(s',v_1')+w|v_1'v_2'|+d(v_2',p')$.

Let us consider now two alternative paths $\Pi'_{S,p}$ and $\Pi'_{S,p'}$ defined by the set of points $\{s', v'_1, v_2, p\}$ and $\{s, v_1, v'_2, p'\}$ that join S with p and p', respectively. Since $\|v_1v_2\| + \|v'_1v'_2\| > \|v_1v'_2\| + \|v_2v'_1\|$, then $\|\Pi_{S,p}\| + \|\Pi_{S,p'}\| > \|\Pi'_{S,p}\| + \|\Pi'_{S,p'}\|$ which contradicts the fact that $\Pi_{s,p}$ and $\Pi_{s,p'}$ are shortest paths. Consequently, two shortest paths can not cross in the interior of a face f. □

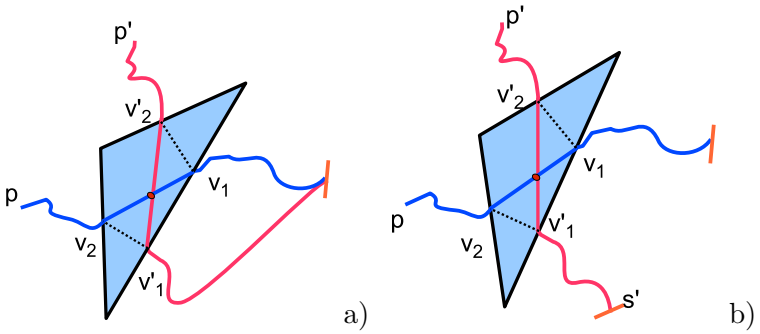


Figure 6.9: Shortest paths from points p and p' to: a) segment site s, when p and p' are closer to the same point of s. b) different points segment sources s and s'.

The Bushwack strategy is adapted to compute the distance field by propagating the distance function of all the sites at once. This is achieved by considering all the sites in the initialization step. We consider the first site and define intervals on the faces it intersects. Next, we consider the second site and again define its intervals taking into account the previously defined ones which can be modified. Since intervals emanating from different sites are stored in the same list, we propagate their distance field. Similar to Bushwack, the shortest path with minimal cost is propagated at each step. The difference is that, now, the paths may come from different initial sites. The time and space complexity does not increase, and is again $O(mn \log(mn))$ and $O(nm)$, whenever each edge of e is intersected by S a constant number of times.

Theorem 6.2.2 *A $(1 + 3\varepsilon)$-approximate distance filed defined by a set S of generalized sites on a triangulated weighted polyhedral surface $\mathcal{P}$ with obstacles can be obtained by using the Bushwack in $O(mn \log(mn))$ time where $m \in O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$.* □

6.3 Distance and shortest path computation

When the propagation of the distance function from a generalized source to the nodes has concluded, the distance and also the shortest path to any point on $\mathcal{P}$ can be obtained. If we are given a set of sites S, we can compute the shortest path distance to the closest site and the actual path by using the distance field defined by S.

6.3.1 Influence regions

The approximate distance function in the interior of the faces is computed by propagating the shortest paths arriving at the nodes of the discrete graph to the face points. Given a node v contained on edge e, Bushwhack algorithm defines intervals $I_{v,e,e'}$ associated to v, e and the edges $e' \neq e$ of the face(s) containing v or opposite to v when v is a vertex. These intervals contain the nodes (Steiner points and vertices) that, according to the previously visited nodes, may be reached by a shortest path that leaves from v. Now, for each interval $I_{v,e,e'}$ we define the *influence region* of I, denoted R_I, on the face f containing e, e' and v, as the set of all points of f that, according to $I_{v,e,e'}$ can be reached by a shortest path emanating from v.

To compute R_I for a given interval $I_{v,e,e'}$ on face f with edges e, e' and e'' (see Figure 6.10), we consider: a) the previously visited nodes v_l, v_r, of e placed contiguously on the left and right of v, respectively; b) the nodes v'_l, v'_r of e' placed contiguously on the left and right of $I_{v,e,e'}$. If $I_{v,e,e'}$ contains the leftmost(/rightmost) vertex of e', $v'_l(/v'_r)$ is the corresponding vertex of e'. Region R_I is the polygonal region of vertices $\{v_l, v'_l, v'_r, v_r\}$. Consequently R_I is a region of at most four vertices. Notice that the influence region R_I of an interval can be computed in $O(1)$ during the Bushwack algorithm.

Property 6.3.1 *The influence region R_I of an interval can be computed in $O(1)$ with the information computed during Bushwack algorithm.* □

6.3.2 Distance computation

The shortest path distance from any point $q \in f$ to the source s can be obtained by finding the node v_m on the edges of f defining the minimum distance value. If $d_{s,v}$ denotes the distance function defined by s and node v, we have $d_{s,v}(q) = d_s(v) + w|\overline{vq}|$, where w is the

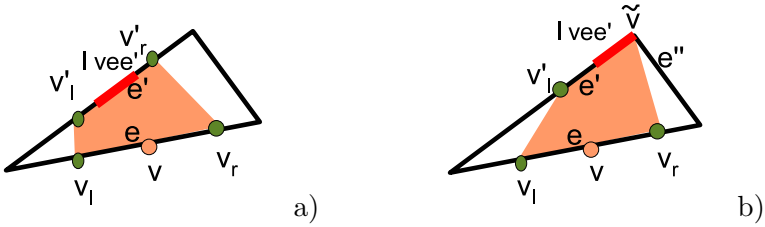


Figure 6.10: The shadowed region is the influence region of v when v'_l and v'_r belong to: a) the same edge. b) different edges.

weight of f. Notice that to determine v_m, those nodes whose influence region does not contain q can be directly discarded. The cost of the shortest path from a point of $\mathcal{P}$ to its nearest site can be obtained by standard methods.

6.3.2.1 Approximation analysis

We have provided a way to compute distances from a generalized source s to all the points on a surface. We use a graph that provides $(1+3\varepsilon)$-approximate distances from s to the nodes. Let us now bound the error produced when obtaining the distance from s to an arbitrary point t of $\mathcal{P}$.

Proposition 6.3.1 *We can obtain $(1+4\varepsilon)$-approximate shortest paths from a generalized source s to an arbitrary target point $t \in \mathcal{P}$.* □

Proof. As we are interested in obtaining a path on the graph, normalized paths are considered. Let Π'' be a normalized path such that $||\Pi''|| \leq (1+\frac{1}{2}\varepsilon)||\Pi||$. Let us assume that $\{s, p_1, \ldots, p_k, t\}$ are the bending points of Π'' and let v_i be the closest Steiner point to p_i is v_i. We will prove that path $\hat{p} = \{s, v_1, \ldots, v_k, t\}$ is so that $||\hat{p}|| \leq (1+3\varepsilon)||\Pi''||$. Using this inequality, we find that $||\hat{p}|| \leq (1+3\varepsilon)(1+\frac{1}{2}\varepsilon)||\Pi|| = (1+4\varepsilon)||\Pi||$, and consequently $||\Pi'|| \leq ||\hat{p}|| \leq (1+4\varepsilon)||\Pi||$.

Let us denote $s_0 = \overline{s, p_1}$, $s_i = \overline{p_i p_{i+1}}$, $s_k = \overline{p_k, t}$, $\hat{s_0} = \overline{s, v_1}$, $\hat{s}_i = \overline{v_i v_{i+1}}$ and $\hat{s}_k = \overline{v_k, t}$. In Lemma 6.1.1 we have proved that $|\hat{s_i}| = (1+2\varepsilon)|s_i|$ for $i = 0 \ldots k-1$. The inequality $(|\hat{s}_k| - |s_k|) \leq \varepsilon \frac{w_m}{w_M}|s_{k-1}|$ is fulfilled because $(|\hat{s}_k| - |s_k|)$ is smaller than the distance between two Steiner points. Consequently,

$$||\hat{p}|| - ||\Pi''|| = \sum_{i=0}^{i=k-1} w_i(|\hat{s_i}| - |s_i|) + w_{k+1}(|\hat{s}_k| - |s_k|) \leq \sum_{i=0}^{i=k-1} 2\varepsilon \; ||s_i|| + \varepsilon||s_{k-1}|| =$$

$$= 2\varepsilon \; ||\Pi''(s, p_k)|| + \varepsilon||s_{k-1}|| \leq 2\varepsilon||\Pi''|| + \varepsilon||\Pi''|| = 3\varepsilon||\Pi''||$$

□

When a set of sites S is given, the distance to the closest site can be computed by using the $(1+4\varepsilon)$-approximate distance filed. By using this result and a proof similar to that of the previous Proposition, we can provide the following Theorem which summarizes all the obtained results.

Theorem 6.3.1 *Given a set of generalizes sites S, we obtain a $(1+4\varepsilon)$-approximation of their distance field. When $S=\{s\}$ we obtain a $(1+4\varepsilon)$-approximation of its distance function.* □

Distance fields are important because they define Voronoi diagrams. When distances are exactly computed, the points that are equidistant from two sites define the Voronoi diagram bisectors. Since now we are working with approximate distances, we should study where the exact bisectors are when the approximate distance is used. Let d_s denote the $(1+4\varepsilon)$-approximate distance function and $\tilde{d}_s$ the exact distance function for a source s. Let p be a point of an exact bisector determined by sites s_0 and s_1, thus, $\tilde{d}_{s_0}(p)=\tilde{d}_{s_1}(p)$. Therefore the following Proposition can be stated.

Proposition 6.3.2 *The error produced on a Voronoi diagram bisector tends to 0 when ε tends to 0.*

Proof.

$$|d_{s_0}(p)-d_{s_1}(p)| \leq |d_{s_0}(p)-\tilde{d}_{s_0}(p)| + |\tilde{d}_{s_0}(p)-\tilde{d}_{s_1}(p)| + |\tilde{d}_{s_0}(p)-d_{s_1}(p)| \leq$$

$$\leq 3\varepsilon\tilde{d}_{s_0}(p) + 3\varepsilon\tilde{d}_{s_1}(p) = 6\varepsilon\tilde{d}_{s_0}(p).$$

□

6.3.3 Shortest path computation

After computing the distance function for a source s, we can obtain the shortest path from s to an arbitrary point q on a face f of $\mathcal{P}$. It suffices to use a backtracing technique and store, during the Bushwhack algorithm, in each node v a pointer to the previous node in the path that goes from s to v. Once the distance function is computed, the path is obtained by first determining the node v of f providing the minimum distance at q. From

node v we go back to node v' stored in the pointer associated to v. We keep jumping until we arrive at source s. The shortest path can be obtained in $O(m+\overline{n})$ time where $\overline{n}$ is the number of segments defining the path. To obtain the path we store the pointers to the nodes.

When a set of sources S is considered, the shortest path to the closest site can be obtained by using the same strategy by using the distance field instead of the distance function.

6.4 Conclusions

We presented an algorithm to compute $(1+\varepsilon)$-approximate shortest paths from generalized sources (point, segment, polygonal line and polygon sources) on triangulated weighted polyhedral surfaces with generalized obstacles. First, a way to build a discrete graph on $\mathcal{P}$ taking into account a generalized source is proposed. By using this graph and the Bushwack strategy we obtain the $(1+\varepsilon)$-approximate distance from the source to the graph nodes in $O(nm\log(nm))$ time and $O(nm)$ space with $m \in O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$.

The algorithm is extended to the case of several generalized sources when their implicit distance field is obtained, with the same time and space complexity whenever each edge is intersected a constant number of times by the set of sites. The output of the algorithm is a codification of the distance function or distance field on the discrete graph.

From the distances to the graph nodes a $(1+\epsilon)$-approximate distance from any point on $\mathcal{P}$ to the source and the actual shortest path can be obtained in $O(m+\overline{n})$ time, where $\overline{n}$ is the number of faces the path goes through. We can also obtain the shortest path distance and the actual shortest path to the closest site of a set S in in $O(\log m+\overline{n})$ time.

Chapter 7

Discrete distance function and applications

In this chapter we provide a method to obtain an explicit discrete representation of the distance function of generalized sources on a polyhedral surface with obstacles (Section 7.2), by using graphics hardware. During the discretization process we use distance vectors (Section 7.1), a planar parameterization of $\mathcal{P}$ (Section 2.3.2), and the implicit distance functions obtained with the algorithms described in Chapter 5 and Chapter 6. We also provide four different applications of this discretization. The first one uses the distance field of a set of sites S (see Figure 7.1), while the rest use the distance functions of each site of S (see Figure 7.2). They consist of the following four algorithms to compute discrete:

1. Closest Voronoi diagrams from the implicit distance field (Section 7.3).

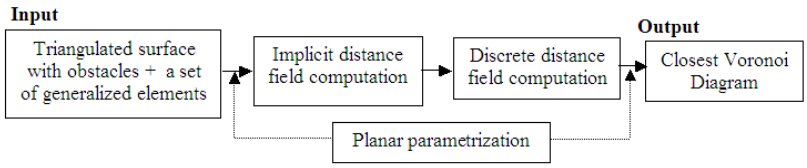


Figure 7.1: Closest Voronoi diagram computation from the distance field defined by the sites.

2. Closest and furthest Voronoi diagrams from the distance functions (Section 7.4.1);

3. Any order-k Voronoi diagrams from the distance functions (Section 7.4);

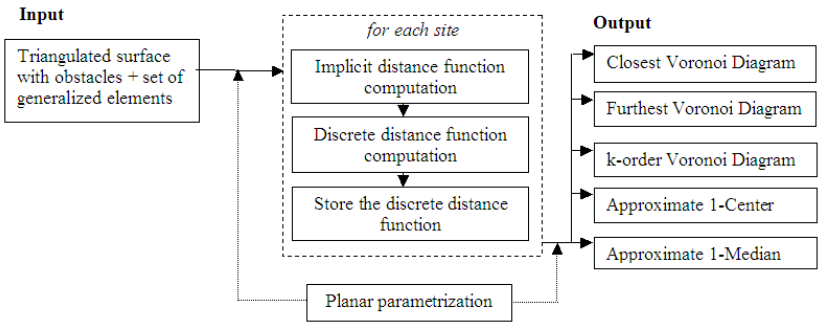


Figure 7.2: Voronoi diagrams, 1-Center and 1-Median computation from the set of distance functions of the sites.

4. An approximate 1-Center (Section 7.6) and 1-Median from the distance functions (Section 7.7).

We present experimental results (Section 7.8) and end with some conclusions (Section 7.9).

7.1 Distance vectors

The distance vector $\overrightarrow{d_q}$ of a point q with respect to a virtual source I^s (a point or a segment) is the vector joining the closest point to q of I^s with q.

Distance vectors are useful to compute distances due to the fact that $d(I^s, q) = |\overrightarrow{d_q}|$. Consequently, if we consider an interval I and a point $q \in R_I$, distance $d_I(q)$ is given by $d_s(I^s) + |\overrightarrow{d_q}|$ in the non-weighted case, and $d_s(I^s) + w|\overrightarrow{d_q}|$, in the weighted case, where w is the weight of the face containing R_I.

Next, we provide some properties that allow us to compute distance vectors from I^s to points on R_I by using distance vectors from the vertices of R_I to I^s when R_I has been triangulated.

7.1.1 Punctual source

Consider a triangulation of R_I and assume that the triangle containing q has vertices p_1, p_2, p_3. Denote $\overrightarrow{d_1}, \overrightarrow{d_2}, \overrightarrow{d_3}$ the distance vectors of p_1, p_2, p_3 associated to a punctual virtual source I^s. Point q can be univocally expressed as $q = \alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3$, with

$\alpha_1 + \alpha_2 + \alpha_3 = 1$ and $\alpha_1, \alpha_2, \alpha_3 \geq 0$. Consequently $\overrightarrow{d_q} = \overrightarrow{I^s q} = q - I^s$. Since $q - I^s = \alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3 - (\alpha_1 + \alpha_2 + \alpha_3) I^s$ the following equality is fulfilled $\overrightarrow{d_q} = \alpha_1 \overrightarrow{d_1} + \alpha_2 \overrightarrow{d_2} + \alpha_3 \overrightarrow{d_3}$ (See Figure 7.3a). Thus the distance vector $\overrightarrow{d_q}$ can be obtained by using linear interpolation from $\overrightarrow{d_1}$, $\overrightarrow{d_2}$, and $\overrightarrow{d_3}$. This is summarized in the following lemma.

Lemma 7.1.1 *The distance vector from a punctual source to a point q in the influence region R_I can be obtained by using linear interpolation from the distance vectors associated to the vertices of R_I.* □

7.1.2 Segment source

Assume that R_I has been triangulated and that p_1, p_2, p_3 are the vertices of the triangle containing the point q of R_I. Denote $\overrightarrow{d_1}, \overrightarrow{d_1}, \overrightarrow{d_3}$ the distance vectors of p_1, p_2, p_3 relative to a segment virtual source I^s, which are parallel and orthogonal to I^s. As before point q can be univocally expressed as $q = \alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3$, with $\alpha_1 + \alpha_2 + \alpha_3 = 1$ and $\alpha_1, \alpha_2, \alpha_3 \geq 0$. We want to prove that $\overrightarrow{d_q} = \alpha_1 \overrightarrow{d_1} + \alpha_2 \overrightarrow{d_2} + \alpha_3 \overrightarrow{d_3}$ (See Figure 7.3b).

Let us first consider the case where one α_i is equal to 0, for example $\alpha_3 = 0$. Then, the vector $\alpha_1 \overrightarrow{d_1} + \alpha_2 \overrightarrow{d_2}$ is the vector perpendicular to I^s obtained by joining the point q to $\alpha_1 p'_1 + \alpha_2 p'_2$ where p'_1 and p'_2 are the points of I^s defining $\overrightarrow{d_1}$ and $\overrightarrow{d_2}$ respectively. When $\alpha_i \neq 0$, $i = 1, 2, 3$, we can consider a line joining q and p_3, and determine the point q' on the segment for which the parameter $\alpha_3 = 0$. The distance vector $\overrightarrow{d_{q'}}$ can be obtained from $\overrightarrow{d_1}$ and $\overrightarrow{d_2}$ as explained for the first case and, finally, $\overrightarrow{d_q}$ can be similarly obtained from $\overrightarrow{d_{q'}}$ and $\overrightarrow{d_3}$. This is summarized in the following lemma.

Lemma 7.1.2 *The distance vector from a segment source to a point q in the influence region R_I of an interval can be obtained by using linear interpolation from the distance vectors associated to the vertices of R_I.* □

7.2 Discrete distance function computation

In this section we provide a way to obtain a discrete representation of the distance function by using the methods given to obtain implicit distance functions (Chapter 5 and Chapter 6).

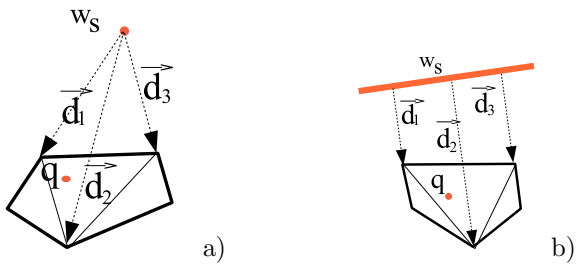


Figure 7.3: The distance vectors associated to the vertices of a convex polygon when dealing with: a) a point source; b) a segment source. In both figures the distance vector to point q can be obtained by the distance vectors shown.

We use a planar parameterization (Section 2.3.2) that maps $\mathcal{P}$ to a bounded planar region $\mathcal{R}$ on the xy-plane. For the special case of a polyhedral terrain, which is a polyhedral surface such that its intersection with any vertical line is either empty or a point, we can alternatively use the terrain projection on the xy-plane as a planar parameterization. We represent the axis-parallel rectangular region bounding the terrain projection on the xy-plane by $\mathcal{R}$. In this case the shape of the projected triangles do not correspond with its shape on the polyhedral terrain.

We discretize the rectangular region $\mathcal{R}$ of the xy-plane as a rectangular grid of size $W \times H$ that induces a discretization on the triangles of $\mathcal{P}$. This discretization and the planar parameterization is used to obtain a discrete representation on $\mathcal{R}$ of the distance function defined by shortest paths on $\mathcal{P}$. This is achieved by keeping track of the explicit representation of the distance function while it is propagated along the surface $\mathcal{P}$ with the presented algorithms (Chapter 5, Chapter 6).

For each interval I we compute the distance it defines to the grid points contained in its influence region(s) R_I. In the non-weighted case we also have to consider influence regions R_v defined by pseudosources and compute the distance defined by v in R_v. In Section 7.2.1 we explain how these distances are obtained by using a planar parameterization of the polyhedral surface, distance vectors and graphics hardware. Notice that grid points may be contained in the influence region of different intervals, thus, during the process we may obtain several distances for the same grid point. Since we are interested in the shortest path distance, only the minimal one is stored.

The planar parameterization maps the points on $\mathcal{P}$ to points on the rectangular region $\mathcal{R}$ of the xy-plane. The OpenGL pipeline triangulates input polygons, processes the tri-

angle vertices and rasterizes the triangles into fragments by using interpolation. Within the rasterization step all the parameters associated to vertices such as texture coordinates, colour, normal vectors, etc. are also interpolated from those associated to the triangle vertices. Consequently, the value obtained in these channels in a fragment is the interpolation of the values associated to the triangulated polygon vertices. Then the fragment shader computes the distance defined by the current interval or pseudosource at each point. It also sets this distance normalized into [0,1] as the depth value of the rasterized fragments. Finally, the depth test is used to keep the minimum distance obtained at each position stored in the depth buffer.

7.2.1 Discrete distance function computation

To compute the discrete distance function we represent the rectangular region $\mathcal{R}$ by a grid of $W \times H$ pixels, using the depth buffer.

The discrete representation of the distance function is obtained during the distance function propagation process. In the initialization process of the propagation method used to compute the implicit distance function, we initialize the depth buffer to the maximal depth value (1). When an interval I is propagated(/created) or a pseudo-source is found, in the non-weighted(/weighted) case, we compute its distance function in it(/s) influence region(/s) R_I. This is done by mapping R_I into the plane by using the planar parameterization obtaining PR_I. Next, the planar region PR_I representing R_I is painted and the distances to the pixels it covers are computed by using a fragment shader. The non-weighted(/weighted) distance given by I in a point $q \in R_I$ is obtained by adding the distance from q to the virtual source I^s and the distance from s to I^s. The distances to these points are computed and stored in the depth value of the fragments. The depth buffer is updated whenever we still have not obtained a smaller distance value. At the end of the process the value stored in the depth buffer is the minimum depth (distance) defined by all the processed fragments.

The previous process correctly computes the distance values: a) observations related to the influence regions given in Section 5.3.1 and Section 6.3.1 show that we will consider the actual shortest path. b) its length or cost is correctly computed according to Lemma 7.1.1 and Lemma 7.1.2. c) the minimum distance value is stored in the depth buffer. Consequently, the shortest path distance function is correctly computed and the algorithm is correct. These observations are summarized in the following Observation.

Observation 7.2.1 *The discrete representation of the distance function can be exactly obtained by using graphics hardware and the algorithms to compute the distance function from a generalized source presented in Chapter 5 and Chapter 6.*

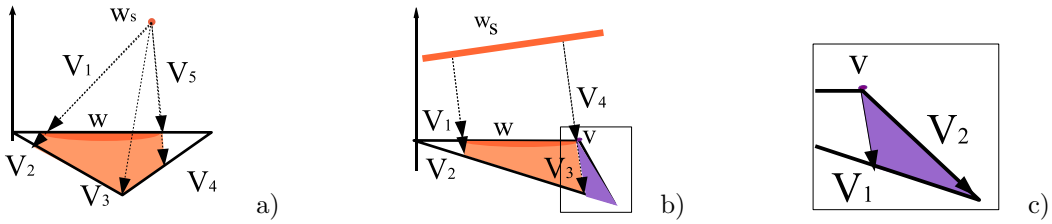


Figure 7.4: Examples of influence regions and distance vectors of: a) a p-interval; b) a s-interval; c) the vertex pseudosource obtained in b).

Lemma 7.2.1 *The distance field defined by an interval I in its influence region R_I can be computed in O(number of pixels in R_I)*

Proof. To compute the distance in R_I, we compute the distance vectors from the virtual source I^s to the vertices of R_I. Posteriorly the polygon PR_I is painted by using OpenGL. We associate to the polygon, in texture coordinate channels, the distance from the virtual source to the initial source and, in the weighted case, the weight of the face where it is contained. We also associate to each vertex of PR_I its distance vector using another texture coordinate channel. In the non-weighted case we have to consider, and handle in the same way, the influence regions P_v of pseudosources. At the end of the process, we have in the depth buffer the distance function defined by the source. □

Therefore, a discretization of the distance function is obtained by painting all the influence regions, one after the other and storing the minimal distance value at each pixel.

Theorem 7.2.1 *A discrete representation of the distance function of a generalized source s is obtained in $O(n^2 \log n + nHW)$ time in the non-weighted case and $O(mn \log(mn) + mHW)$ in the weighted case, which represents an extra time of $O(nHW)$ and $O(mHW)$, respectively.*

Proof. The previously explained process to obtain the discrete representation of the distance field does not increase the time of the Continuous Dijkstra nor the Bushwack algorithm used for the non-weighted and weighted case, respectively. In fact, for each interval

I we have to compute the influence regions and the distance vectors from the virtual source I^s to the influence region vertices, which can be done in constant time.

Since each influence region R_I is contained in a face f, the extra time needed to paint the influence regions can be bounded as follows. In the non-weighted case, we have at most n intervals per edge and each face is painted, at most, $O(n)$ times. The extra time needed to paint all the influence regions in the non-weighted case is $O(nHW)$ time. In the weighted case we have an interval for each node and we have at most m nodes per edge, consequently, each face can be painted $O(m)$ times providing a time complexity of $O(mHW)$. □

7.3 Discrete closest Voronoi diagrams

In this Section we propose a way to compute the discrete closest Voronoi diagram for a set of generalized sources S on the polyhedral surface $\mathcal{P}$ with obstacles. Although in Section 7.4 a general procedure to compute order-k Voronoi diagrams is given, for the special case of the closest Voronoi diagram ($k = 1$) this process is much more efficient, in both, time and space complexity. In this section a discrete Voronoi diagram is obtained by using graphics hardware and the algorithms to compute the distance field of a set of generalized sources (Chapter 5, Chapter 6).

To obtain the closest Voronoi diagram, we discretize the generalized distance field of a set of generalized sources S, and properly determine the Voronoi regions. The discretization of the distance field is obtained by slightly modifying the algorithm to compute the discrete distance functions described in Sections 7.2. We keep on using a planar parameterization of the triangulated polyhedral surface $\mathcal{P}$ that maps $\mathcal{P}$ to $\mathcal{R}$, and the depth and color buffers to discretize $\mathcal{R}$. In this case we use the depth buffer to store distances and the color buffer to obtain the Voronoi diagram regions painted in different colors. We now use the algorithm to compute the implicit distance field. Again, the discrete representation of the distance field is obtained by painting the corresponding influence regions, but now they are painted in different colors. We associate a different color to each source in S and the influence regions are colored with the color of the generalized source where the interval originates from. To identify this source, each interval stores an extra parameter that gives the index of the source in S where is originated. When a fragment is processed, the color of the color buffer is updated if and only if the depth value is updated. At the end of the process, the values stored in the depth buffer are the values of the distance field

and the obtained image in the color buffer is the discrete closest Voronoi diagram. This is summarized in the following Observation:

Observation 7.3.1 *The process to obtain a discrete representation of the distance function can be adapted to obtain a discrete Voronoi diagram of a set of sites S.*

We want to mention that there may exist one or two dimensional regions equidistant to two different sites. One dimensional regions are easily detected as being the boundary separating points of different color. Two dimensional regions are, somehow, lost and will be painted in one or the other colors depending on the behavior of the used method to propagate the distance field and the order in which the regions are painted.

Theorem 7.3.1 *A discrete Voronoi diagram can be obtained from the implicit distance field in extra $O((n+r)HW)$ in the non-weighted case and $O(mHW)$ in the weighted case.*

Proof. The extra time needed to obtain the closest Voronoi diagram while using the algorithm to implicitly obtain the distance field is the time spent by painting influence regions. Each influence region is painted in constant time. Therefore, the time complexity in the non weighted case is $O((n+r)HW)$ because each edge contains at most $O(n+r)$ intervals and each face is painted, in the worst case, $O(n+r)$ times. In the weighted case the extra time is $O(mHW)$. Each face contains at most $O(m)$ time and consequently each face is painted at most $O(m)$ time. □

7.4 Discrete high order Voronoi diagrams

In this section we describe how the discrete order-k Voronoi diagrams, $k = 1 \ldots r-1$, for a set of r generalized sources S on the polyhedral surface $\mathcal{P}$ can be obtained. Since obtaining the distance function of a source is expensive, we assume that we have pre-computed and stored the discrete distance function of each source. The time needed to compute the r discrete distance function in the non-weighted case is $O(r(n^2 \log\ n + HW))$ and in the weighted case it is in $O(r((mn+m'n')\log(mn+m'n')+HW))$ time. Previously computed discrete distance functions can be stored by rendering the depth buffer in a depth texture or stored in the CPU.

7.4.1 Closest Voronoi diagram

The closest Voronoi diagram can be obtained by associating to each site of S a color and rendering, one after the other, each distance function in the appropriate color.

To obtain the closest Voronoi diagram the r discrete distance functions are transferred to a texture one after the other. Once a distance function is active in the fragment shader, a rectangular region covering $\mathcal{R}$ is painted in the color of the site. The distance of the distance function is stored as the depth of the fragments and the depth test is used to store the smallest depth value. Consequently, when the r distance functions are painted their lower envelope is computed and the pixel is painted in the color of the closest site. Finally, the Voronoi diagram is obtained in the color buffer and the distance field in the depth buffer. The time needed to compute this lower envelope is $O(rHW)$. Thus we can provide the following Proposition.

Proposition 7.4.1 *The closest Voronoi diagram of r discrete distance functions can be computed in extra $O(rHW)$ time from the previously computed discrete distance functions.* □

7.4.2 Furthest Voronoi diagram

The furthest Voronoi diagram is obtained as the upper envelope of the distance functions, consequently it is obtained by using a process similar to the one provided to obtain the closest Voronoi diagram.

The furthest Voronoi diagram is computed by rendering, one after the other, each distance function in their corresponding colors and storing the distance values as the depth of the fragments. The depth test is used to store, in each pixel, the maximum depth value. Each point is accordingly painted with the color of the fragment with maximal depth value, which is the color associated to the furthest site to each point. The furthest Voronoi diagram is obtained in the color buffer and the furthest distances in the depth buffer. The complexity analysis is the same as the one given for the closest Voronoi diagram. Consequently, the time complexity is $O(rHW)$.

Proposition 7.4.2 *The furthest Voronoi diagram of r discrete distance functions can be computed in extra $O(rHW)$ time by using the previously computed discrete distance functions.* □

7.4.3 order-k Voronoi diagram

The algorithm to compute discrete order-k Voronoi diagrams uses a depth peeling technique similar to the one described on (Section 2.3.3.2) to compute order-k Voronoi diagrams of a set of points in the plane. It is a multi-pass algorithm that, at every pass, "peels" off one level of the arrangement of distance functions. At each pass all the distance functions are painted in their corresponding colors and the minimal depth value is stored in the depth buffer. In the first pass the closest Voronoi diagram is obtained. The depth buffer is then transferred to a texture and sent to the fragment shader. In the second pass all the distance functions are again painted. The distance function that is being painted is compared in the fragment shader with the distance obtained in the previous pass at the current fragment. Only the fragments with distance bigger than the distance obtained in the previous pass are painted. The others are discarded. Therefore, the values stored in the depth buffer in the second step are the second minimal distance. When this process is repeated k times, the kth-nearest diagram is obtained. The order-k Voronoi diagram can be obtained by overlaying the ith-nearest diagrams, $i = 1 \dots k$, with transparency $1/k$.

Proposition 7.4.3 *A order-k Voronoi diagram can can be computed in extra $O(krHW)$ time by using the previously computed discrete distance functions.*

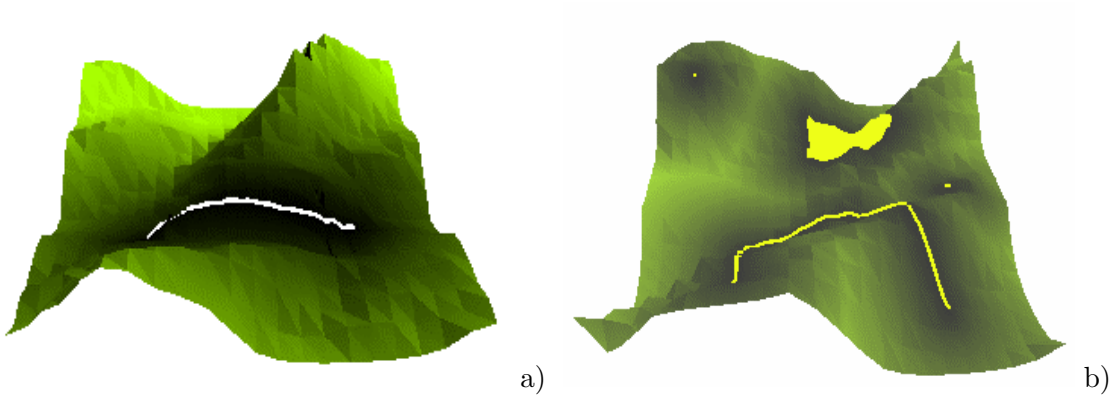
Proof. To obtain the order-k Voronoi diagram of the r already computed discrete distance we need k-steps. For each of the k peeling passes, the depth buffer is copied on a texture and the r distance functions are painted. Therefore, the algorithm time complexity is $O(krHW)$ □

7.5 Visualization on the polyhedral surface

Once we have obtained a discrete representation of any of the mentioned Voronoi diagrams in the color buffer, we can transfer the values of this buffer to a texture. The texture is an explicit discrete representation of the Voronoi diagram and by using texturing methods, with the already used planar parameterization, the Voronoi diagram can be visualized on the polyhedral surface.

Distance functions can also be visualized on the polyhedral surface by transferring the values of the depth buffer to a depth texture. Since distances vary from 0 to 1 they can be used to weight the color of the pixels. If the white color is used, pixels are painted in

a grey gradation from black (distance 0) to withe (distance 1) according to the distance function values. In Figure 7.5 we have used a green gradation to represent the distance function of a site Figure 7.5 a) or the distance field of a set of sites Figure 7.5 b), black points are closer to the sites, and green points further from them.



a) b)

Figure 7.5: a) The distance function defined by a segment source s. b) The distance field defined by a set of four generalized sites.

7.6 Approximating the 1-Center

The 1-Center of a set of r sites S is the point C that minimizes the maximum distance to the sites. Consequently it is the center of the minimum enclosing disc defined by the shortest path distance. Thus we have to find the point $q \in \mathcal{P}$ where the min $\max_{s \in S} d_s(q)$ is achieved. Notice that the value $\max_{s \in S} d_s(q)$ is the distance given by the furthest Voronoi diagram.

The algorithm given in Section 7.4.2 provides the furthest Voronoi diagram in the color buffer, and in the depth buffer $\max_{s \in S} d_s(q)$ has been stored in each pixel. To obtain the 1-Center we first obtain the furthest Voronoi diagram and next compute the point where the minimum value of the depth buffer is achieved.

The point where the minimum depth value is achieved can be obtained by using a "reduction-type" algorithm. We start by transferring the values stored in the depth buffer to a texture t of size $H \times W$ which is used in a fragment shader. At the first step we paint a rectangle on a rendering texture of size $\frac{H}{2} \times \frac{W}{2}$. When the fragment corresponding to pixel (i, j) is processed, we use a fragment shader to find the position (i_0, j_0) where the minimum of the values $\{t(2i, 2j),\ t(2i+1, 2j),\ t(2i, 2j+1),\ t(2i+1, 2j+1)\}$ is

achieved and, $t(l,m)$ is the value stored in texture t at the position (l,m). Coordinates (i_0,j_0) are codified in the color of the pixel (i,j) which is stored in the rendering texture. Next, the color texture is transferred to the fragment shader to be used in the second rendering step. In the second step we have texture t, which has not changed, and a new texture c_0 of size $\frac{H}{2}\times\frac{W}{2}$, where $c_0(i,j)$ encodes (i_0,j_0). Now, we paint a rectangle of size $\frac{H}{4}\times\frac{W}{4}$ and at fragment (i,j) and we find the position (i_1,j_1) where the minimum of $\{t(c_0(2i,2j)),\ t(c_0(2i,2j+1)),\ t(c_0(2i+1,2j)),\ t(c_0(2i+1,2j+1))\}$ is achieved. Again we codify (i_1,j_1) in the color values of fragment (i,j). We keep on doing the same, at step k we use t and c_{k-1} of size $\frac{H}{2^{k-1}}\times\frac{W}{2^{k-1}}$, we paint a rectangle of size $\frac{H}{2^k}\times\frac{W}{2^k}$ and find the position (i_k,j_k) where the minimum of $\{t(c_{k-1}(2i,2j)),\ t(c_{k-1}(2i+1,2j)),\ t(c_{k-1}(2i+1,2j+1)),\ t(c_{k-1}(2i,2j+1))\}$ is achieved. This process computes the position where the minimum depth value is achieved. The algorithm ends in $O(\log HW)$ steps, the pixel where the minimum is achieved is obtained by reading the value stored in the color of a pixel. This position gives an approximate 1-Center on $\mathcal{R}$. To obtain a point on $\mathcal{P}$ the planar parameterization is used.

The total number of pixels painted during the $O(\log HW)$ is given by $HW+\frac{HW}{4}+\frac{HW}{4^2}+\cdots+\frac{HW}{4^{\log HW}}=\sum_{i=0}^{i=\log(HW)}\frac{HW}{4^i}$ which is a geometric series of factor $\frac{1}{4}$. Consequently it gives a total number of painted pixels of $\frac{1}{3}HW$. This result is summarized in the next Proposition.

Proposition 7.6.1 *The* 1*-Center and the distance value can be obtained from the furthest Voronoi diagram by using graphics hardware in extra* $O((r+1)WH)$ *time.* □

Consequently, we take as an approximate 1-Center of S the point of $\mathcal{P}$ corresponding to the position of the depth buffer with minimal value. This value is the radius of the minimum enclosing disc.

7.7 Approximating the 1-Median

The 1-Median of a set of r sites S is the point M that minimizes $\sum_{s\in S}d_s(q)$. We first obtain $\sum_{s\in S}d_s(q)$ and approximate the 1-Median by determining the pixel where the minimum value is achieved.

To obtain $\sum_{s\in S}d_s(q)$ we use r rendering steps and we transfer at each step the information stored in the buffer to the fragment shader. After initializing the depth buffer to

0, at the first step we paint d_{s_0} and store its values in the depth buffer. In the second step we transfer the depth buffer to a texture t_0, paint d_{s_1} and store, in the depth buffer, the sum of d_{s_1} with the value stored in t_0. In the ith step we transfer the depth buffer to a texture t_{i-1}, we paint d_{s_i} and store in the depth buffer the sum of d_{s_i} with the value stored in t_{i-1}. We repeat the process until d_{s_r} is considered. At the end of the process, we have the desired $\sum_{s\in S} d_s(q)$, in the depth buffer. To obtain the pixel where the minimum is achieved we use the process explained in 7.6.1. This result is summarized in the next Proposition.

Proposition 7.7.1 *The* 1*-Median and the distance value in it can be obtained by using graphics hardware in extra* $O((r + \log(HW))WH)$ *time.* □

The results obtained in this chapter are approximated, and two different types of error can be seen. One is the discretization error, which depends on the discretization size. It can be reduced using the fact that the bigger the grid size $W \times H$, the smaller the error produced. The other is due to floating errors, which are specially related to the depth buffer and depth texture precision. The 32-bit precision is sufficient to store the normalized distances which take values in the interval $[0, 1]$. This can be specially seen when computing order-k Voronoi diagrams, where many distances have to be compared. In the rest of the applications it is not visible.

7.8 Experimental results

We have implemented the proposed methods using C++ and OpenGL for the special case of polyhedral terrains. All the images have been carried out on a Intel(R) Pentium(R) D at 3GHz with 1GB of RAM and a GeForce 7800 GTX/PCI-e/SSE2 graphics board.

In Table 7.1 we present some experimental results, obtained by considering terrains without obstacles, a set S of six sites (one point, two segments, two polygonal lines and one polygon), $\varepsilon = 0.5$ and a grid to discretize the domain of size 500×500. We present execution times for non-weighted and weighted terrains with weights randomly generated between one and five. In the table we specify the number of terrain faces, n, and the number of faces intersected by the sites n'. Concerning non-weighted terrains we have used the Continuous Dijkstra strategy. We provide the time needed to compute the distance field and to obtain the explicit Voronoi diagram of S, and finally the time needed to compute and store the six distance functions. When considering weighted terrains, we

provide the total number of Steiner points. Next, we give the time needed to compute the distance field using the Bushwack strategy and finally the time needed to compute the six distance functions. Notice that in both cases the time needed to obtain the six distance functions is about six times that needed to obtain the distance field. The extra time needed to obtain, from the already computed distance fields, the closest or furthest Voronoi diagram is 0.08(s), the 4th nearest site to each point is 0.3(s) and the 5th nearest site is 0.35(s). Finally, in 0.1(s) we obtain an approximate 1-Center when $r = 6$ and in 0.12(s) an approximation of the 1-Median. Notice that the time needed to compute the distance fields are execution times of the algorithms provided in Chapter 5 and Chapter 6.

	Non-weighted terrain		Weighted terrain			
n	D. Field	D. Functions	n'	N. Steiner Points	D. Field	D. Functions
800	0.3 (s)	2.3 (s)	73	24449	5.6 (s)	30 (s)
5000	2.3 (s)	14 (s)	145	135670	28 (s)	147 (s)
10000	5.8 (s)	39 (s)	221	260599	54 (s)	281 (s)
20000	13 (s)	96 (s)	271	533677	102 (s)	582 (s)
45000	29 (s)	184 (s)	240	1228163	268 (s)	1408 (s)

Table 7.1: Implicit distance computation.

Notice that a non-weighted surface can be seen as a weighted terrain with weights equal to 1. Thus we could use the Bushwack strategy proposed for weighted terrains to obtain approximate shortest paths on non-weighted terrains. However, according to the time performance of the algorithms, it is better to use the Continuous Dijkstra strategy to compute exact shortest paths on non-weighed terrains because it is much faster. Even though the implementation of the Bushwack strategy is more expensive and provides $(1+\varepsilon)$-approximate distances, it is much more robust than that of the Continuous Dijkstra.

Figures 7.6 to 7.12 show some examples of Voronoi diagrams for generalized sources on polyhedral terrains with n =800 faces obtained with our implementation. The generalized sources, except for the polygon sources interior, are painted on the terrain surface and the remaining points of the surface are colored according to the Voronoi region they belong to.

Figures 7.6 to 7.9 show Voronoi diagrams on non-weighted polyhedral terrains obtained by using the Continuous Dijkstra strategy. We have considered a set S of ten sites: four points, two segments, two polygonal chains and two polygon sources, and a terrain with n =800 faces. In Figure 7.6 we show the closest Voronoi diagram of S; in Figure 7.7 each point is painted in the color of the 7th nearest site, we do not show the 7th order Voronoi

because the image is not easy to understand due to the merged colors. In Figure 7.8 the furthest Voronoi diagram is shown and Figure 7.9 shows the closest Voronoi diagram of S when obstacles (two polygonal chains and a polygonal region), which are painted black, are considered.

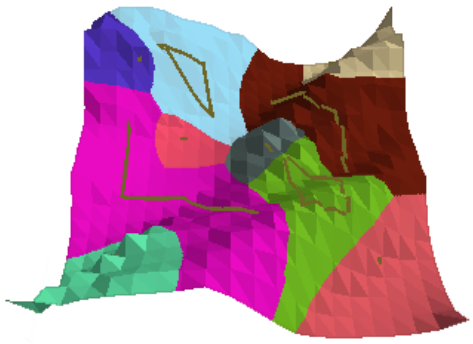


Figure 7.6: A non-weighted terrain, ten generalize sites and their Closest Voronoi diagram.

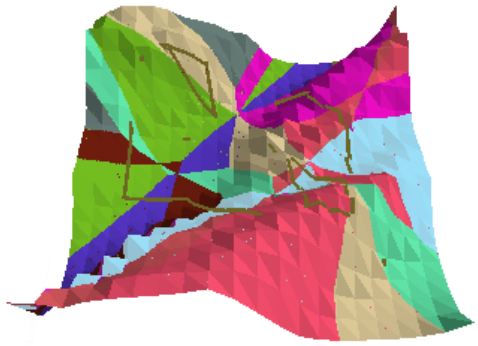


Figure 7.7: A non-weighted terrain, ten generalize sites and their 7th-nearest diagram (see site colors in Figure 7.6).

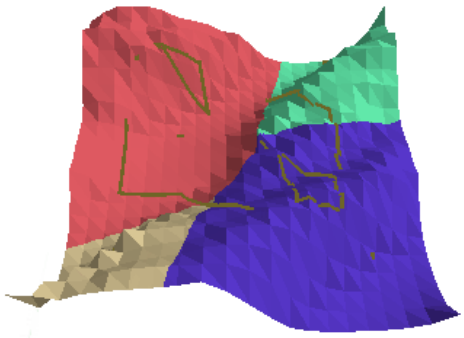


Figure 7.8: A non-weighted terrain, ten generalize sites and their furthest Voronoi diagram (see site colors in Figure 7.6).

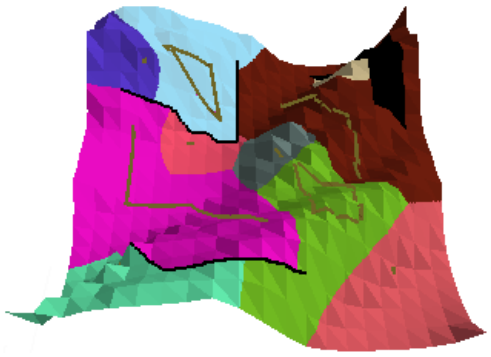


Figure 7.9: A non-weighted terrain with obstacles (in black), ten generalized sites and their closest site Voronoi diagram.

Figures 7.10 to 7.12 show some examples of Voronoi diagrams on weighted polyhedral terrains for a set S of eight sites: three points, two segments, two polygonal chains and one polygon. Figure 7.10 shows the closest Voronoi diagram of S, in Figure 7.11 each point is painted in the color of its 6th-nearest site. In Figure 7.12 the furthest Voronoi diagram is shown with the 1-Center and 1-Median of S, the 1-Center is represented by a circle and the 1-Median by a square.

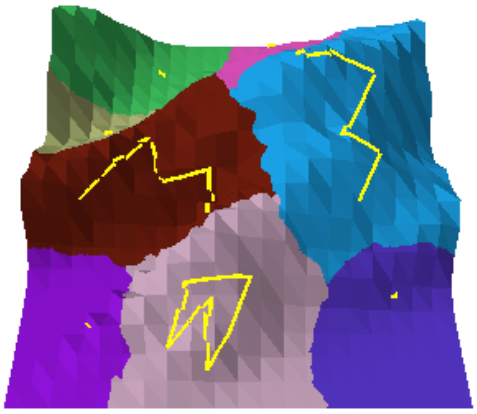


Figure 7.10: A weighted terrain, eight generalize sites and their closest Voronoi diagram.

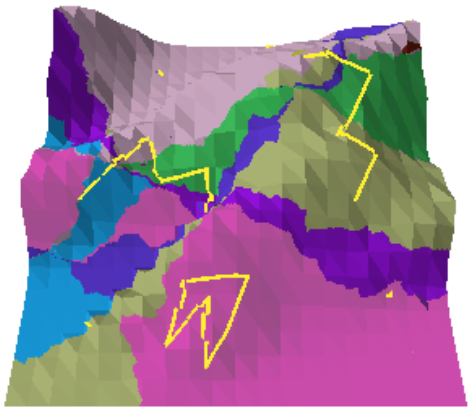
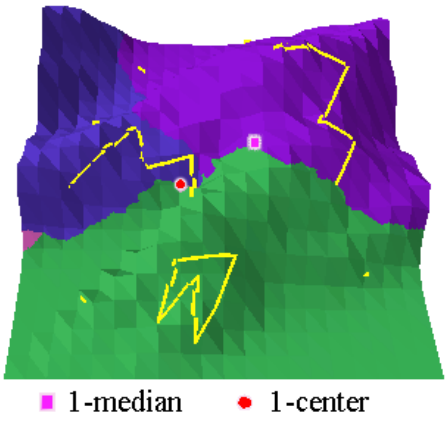


Figure 7.11: A weighted terrain, eight generalized sites and their 6th-nearest diagram (see site colors in Figure 7.10).



a)

Figure 7.12: A weighted terrain, eight generalized sites, their furthest Voronoi diagram, 1-Center and 1-Median. See site colors in Figure 7.10.

The error produced by the 32-bit precision of the depth buffer can be seen in Figure 7.7 and Figure 7.10. Isolated pixels are painted in the color of the regions adjacent to the region they belong to.

7.9 Conclusions

We have presented a way to obtain a discrete representation of the distance function or distance field from their implicit representation obtained in Chapter 5 and Chapter 6. As applications we provide a way to directly obtain a discrete Voronoi diagram from the implicit distance field, and a more general technique, which from the distance functions of all the sources in S the closest, furthest or any $k-$order Voronoi diagram can be obtained. We have also approximately solved the 1-Center and 1-Median facility location problems.

Finally some experimental results obtained with our implementation that works for polyhedral terrains with generalized sources and obstacles are presented.

Chapter 8

Conclusions, further comments and future work

8.1 Conclusions

In this thesis we solved visibility and proximity problems concerning generalized elements (points, segments, polygonal lines and polygons).

Visibility problems

Visibility problems are solved on triangulated terrains, we give a way to exactly compute the visible parts of a segment from a generalized view element considering weak or strong visibility. It is the first algorithm that computes an approximation of a multi-visibility map on a terrain domain when considering generalized view elements and concerning strong or weak visibility. The algorithm is expensive both in time and storage and is difficult to turn into a practical algorithm. Bearing these problems in mind, another approach based on graphics hardware capabilities is developed from a practical point of view.

Next, we present a method to visualize multi-visibility maps of a triangulated terrain containing an heterogeneous set of view elements for weak and strong visibility. We, by repeatedly using graphics hardware, compute approximated visibility information in a pre-processing stage and then from this information visualize any multi-visibility map on the screen with a zoom option. We also answer point and polygonal region multi-visibility queries. The results obtained with our implementation show that our approach

is practical, robust and efficient.

Proximity problems

Proximity problems are solved on a triangulated polyhedral surface where generalized obstacles are allowed. To solve proximity problems we compute shortest path distances from generalized elements. We tackle both the non-weighted and the weighted problems in an exact and approximated fashion, respectively.

We first present an algorithm to compute exact shortest paths from generalized sources (point, segment, polygonal line and polygon sources) on triangulated polyhedral surfaces with several generalized obstacles in $O(n(n+\tilde{r})\log(n+\tilde{r}))$ time and $O(n(n+\tilde{r}))$ space, where $\tilde{r}$ is the number of segments conforming the generalized source. The algorithm is extended to the case of several generalized sites when their implicit distance field is obtained in $O(n(n+\tilde{r}_S)\log(n+\tilde{r}_S))$ time and $O(n(n+\tilde{r}_S))$ space where $\tilde{n_S}$ is the total number of segments conforming the generalized sites. The output of the algorithm is a codification of the distance function or distance field. From this codification the distance from a point on $\mathcal{P}$ to the site and the actual shortest path is obtained in $O(n+\tilde{r}+\overline{n})$ time, where $\overline{n}$ is the number of faces the path goes through. We also obtain the shortest path distance and the actual shortest path to the closest site of a set S in $O(n+\tilde{r}_S+\overline{n})$ time.

We present an algorithm to compute $(1+\varepsilon)$-approximate weighted shortest paths from generalized sources. First we propose a way to build a discrete graph on $\mathcal{P}$ taking into account a generalized source. By using this graph and the Bushwack strategy we obtain the $(1+\varepsilon)$-approximate distance from the source to the graph nodes in $O(nm\log(nm))$ time and $O(nm)$ space with $m \in O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$. The algorithm is extended to the case of several generalized sources when their implicit distance field is obtained, with the same time and space complexity whenever each edge is intersected a constant number of times by the set of sites. The output of the algorithm is a codification of the distance function or distance field on the discrete graph. From the distances to the graph nodes a $(1+\epsilon)$-approximate distance from any point on $\mathcal{P}$ to the source and the actual shortest path are obtained in $O(m+\overline{n})$ time, where $\overline{n}$ is the number of faces the path goes through. We computed the shortest path distance and the actual shortest path to the closest site of a set S in in $O(\log m+\overline{n})$ time.

In order to provide some applications we present a way to obtain discrete representations of these functions. The discrete representations are then used to directly obtain a

discrete Voronoi diagram by using the distance field. With a more general technique the closest, furthest or any $k-$order Voronoi diagram are obtained from the distance functions of all the sources contained in the set of sources. We approximately solve some facility location problems, the 1-Center and 1-Median. We also present some experimental results obtained with our implementation that works for polyhedral terrains with generalized sources and obstacles.

Finally we provide a theoretical study of high-order Voronoi diagrams of a set of generalized sources a polyhedral surface $\mathcal{P}$. We generalized some basic properties of order-k Voronoi diagrams such as that a k-cell cannot contain a $(k+1)$-cell, or that a k-vertex has at least degree three. We prove that the furthest Voronoi diagram has $O(r)$ path connected cells. We also present some pathologies of order-k Voronoi diagrams, for instances: two bisectors may cross more than twice, there exist up to $O(r)$ k-cells without vertices, and k-cells are not necessarily path-connected. We also study the complexity of the order-k Voronoi diagrams by using a probabilistic proof which was $O(k^2n^2 + k^2r + krn)$ and $\Omega(krn + rk^2)$. Finally, we give some bounds and properties of shortest paths and Voronoi diagrams of arbitrary orders on realistic terrains which can be considered as a special case of triangulated polyhedral surfaces.

8.2 Further comments

We want to emphasize that we provide the implementation of most of our algorithms. A very important share of the time spent during the research work described in this thesis is dedicated to the implementation of the algorithms. In this section we present the general methodology used as well as the most salient specific problems encountered in the implementation of our algorithms.

The use of adequate programming languages is a very important issue in the implementation of any application. We decide to use the C++ programming language, as it is a broadly used, general purpose, object oriented and numerically proficient programming language. An extra asset is that C++ also provides tools such as QT for user interface, the OpenGL library for 2D and 3D visualization, CGAL for Computational Geometry tools and CG for programming hardware graphics.

From the point of view of application design, a PhD. Thesis is a difficult setting as the requirements change from day to day following the development of the algorithms, often

after they have been implemented (and sometimes because of it). We have always tried to keep our code as "open" as possible in order to be able to process changes easily, although in some cases we have not been able to avoid introducing major changes to already finished parts of the code.

The last step of the implementation test is the testing of the code. This is a long, complex and very important process as badly tested code makes experimentation difficult if not impossible. We have tested all our methods separately. After this was done we performed integration tests. All this tests have a first manual step that is carried with small examples whose execution is followed to the last instruction. Then more complicated and bigger examples are used to obtain some results. Once the implementation is tested some terrain models with generalized sites are considered to obtain some experimental results.

8.3 Future work

We present some future work we are interested in:

Visibility problems

- Generalize the method to compute multi-visibility maps in general scenes in $\mathcal{R}^3$.
- Provide a parallelized implementation of the proposed algorithm which would really improve the running time.

Proximity problems

- Obtain a tight upper-bound on the complexity of the generalized order-k Voronoi diagrams.
- Study the theoretical complexity of the weighted Voronoi diagrams. It is difficult to study because weighed shortest paths have a complicate behavior.

Further applications

- Compute Voronoi diagrams of visible sites, a point q is in the region of site s_i if s_i is the closest visible site to q.

- Compute visibility restricted to shortest path distances: a point is not visible according to distances if it is not visible in the usual sense, or it is too far from the site.

Bibliography

[1] L. A. M. L. A., Maheshwari, and J.-R. Sack. An ϵ - approximation algorithm for weighted shortest paths on polyhedral surfaces. In *SWAT'98*, pages 11–22, 1998.

[2] P. K. Agarwal. Intersection and decomposition algorithms for planar arrangements. University Press, Cambridge, 1991.

[3] P. K. Agarwal, B. Aronov, and M. Sharir. On levels in arrangements of lines, segments, planes, and triangles. In *SCG '97: Proceedings of the thirteenth annual symposium on Computational geometry*, pages 30–38, New York, NY, USA, 1997. ACM.

[4] P. K. Agarwal, S. Har-Peled, M. Sharir, and K. R. Varadarajan. Approximating shortest paths on a convex polytope in three dimensions. *J. ACM*, 44(4):567–584, 1997.

[5] P. K. Agarwal, S. Krishnan, N. Mustafa, and S. Venkatasubramanian. Streaming geometric optimization using graphics hardware. In *11th European Symp. on Algorithms*, pages 544–555, 2003.

[6] P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002.

[7] L. Aleksandrov, A. Maheshwari, and J.-R. Sack. Approximation algorithms for geometric shortest path problems. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 286–295, New York, NY, USA, 2000. ACM Press.

[8] L. Aleksandrov, A. Maheshwari, and J.-R. Sack. Determining approximate shortest paths on weighted polyhedral surfaces. *J. ACM*, 52(1):25–53, 2005.

[9] M. A. Anile, P. Furno, G. Gallo, and A. Massolo. A fuzzy approach to visibility maps creation over digital terrains. *Fuzzy Sets Syst.*, 135(1):63–80, 2003.

[10] B. Aronov, M. J. van Kreveld, R. van Oostrum, and K. R. Varadarajan. Facility location on terrains. In *ISAAC '98: Proceedings of the 9th International Symposium on Algorithms and Computation*, volume 1533, pages 19–28, London, UK, 1998. Springer-Verlag.

[11] B. Aronov, M. J. van Kreveld, R. van Oostrum, and K. R. Varadarajan. Facility location on a polyhedral surface. *Discrete & Computational Geometry*, 30(3):357–372, 2003.

[12] F. Aurenhammer. Voronoi diagrams: a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, 1991.

[13] F. Aurenhammer and R.Klein. *Handbook of Computational Geometry*, chapter Voronoi diagrams, pages 201–290. Elsevier, 2000.

[14] Bern and Eppstein. Mesh generation and optimal triangulation. In D.-Z. Du and F. Hwang, editors, *Computing in Euclidean Geometry, Edited by , World Scientific, Lecture Notes Series on Computing*, volume 1. 1992.

[15] M. W. Bern, D. P. Dobkin, D. Eppstein, and R. L. Grossman. Visibility with a moving point of view. *Algorithmica*, 11(4):360–378, Apr 1994.

[16] G. E. Blelloch. *Vector models for data-parallel computing*. MIT Press, Cambridge, MA, USA, 1990.

[17] I. Boada, N. Coll, N. Madern, and J. A. Sellarès. Approximations of 3D generalized voronoi diagrams. In *21st European Workshop on Computational Geometry*, pages 163–166, 2005.

[18] I. Boada, N. Coll, and J. Sellarès. Adaptive approximations of 2D and 3D generalized voronoi diagrams. *International Journal of Computer Mathematics*, -(-):–, 2008.

[19] P. Bose, A. Maheshwari, and P. Morin. Fast approximations for sums of distances, clustering and the fermat–weber problem. *Comput. Geom. Theory Appl.*, 24(3):135–146, 2003.

[20] P. Bose and P. Morin. An improved algorithm for subdivision traversal without extra storage. In *ISAAC '00: Proceedings of the 11th International Conference on Algorithms and Computation*, pages 444–455, London, UK, 2000. Springer-Verlag.

[21] R. Bose and S. Chowla. Theorems in the additive theory of numbers. *Comment. Math. Helv.*, 37:141–147, 1962-63.

[22] A. Bowyer. Computing dirichlet tessellations. *Comput. J.*, 24(2):162–166, 1981.

[23] N. Carr, J. Hart, and J. Maillot. The solid map: Methods for generating a 2-d texture map for solid texturing. In *Proc. Western Computer Graphics Symposium*, pages 179–190, 2000.

[24] Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *GEOMETRY: Discrete and Computational Geometry*, 16, 1996.

[25] T. M. Chan. A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1196–1202, New York, NY, USA, 2006. ACM.

[26] R. Chandrasekaran and A. Tamir. Algebraic optimization: the fermat-weber location problem. *Math. Program.*, 46(2):219–224, 1990.

[27] B. Chazelle, D. Liu, and A. Magen. Sublinear geometric algorithms. In *35th ACM Symposium of Thory of Computing*, pages 531–540, 2003.

[28] J. Chen and Y. Han. Shortest paths on a polyhedron; part i: computing shortest paths. In *Inernational Journal of Computational Geomtry & Applications*.

[29] W. Chen and K. Wada. On computing the upper envelope of segments in parallel. *IEEE Trans. Parallel Distrib. Syst.*, 13(1):5–13, 2002.

[30] K. Clarkson and P. Shor. Application of random sampling in computational geometry ii. *Discrete and Computational Geometry*, 4:387–421, 1989.

[31] D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. *Computer Graphics Forum*, 17(3):243–254, 1998.

[32] N. Coll. *Approximation and Visualization Methods for Bidimensional Geometric Objects*. PhD thesis, Universitat Politècnica de Catalunya., 2004.

[33] N. Coll, F. Hurtado, and J. A. Sellarès. Approximating planar subdivisions and generalized vornoi diagrams from random sections. In *19th European Workshop on Computational Geometry*, pages 27–30, 2003.

[34] S. Coorg and S. Teller. Real-time occlusion culling for models with large occluders. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 83–ff., New York, NY, USA, 1997. ACM Press.

[35] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Heidelberg, 1997.

[36] M. de Berg, M. J. van Kreveld, R. van Oostrum, and M. H. Overmars. Simple traversal of a subdivision without extra storage. *International Journal of Geographical Information Science*, 11(4):359–373, 1997.

[37] O. Devillers. Improved incremental randomized delaunay triangulation. In *SCG '98: Proceedings of the fourteenth annual symposium on Computational geometry*, pages 106–115, New York, NY, USA, 1998. ACM Press.

[38] Z. Drezner and H. W. Hamacher. *Facility Location: applications and theory*. Springer–Verlah, 2002.

[39] A. Dumitrescu and C. D. Tóth. On the number of tetrahedra with minimum, unit, and distinct volumes in three-space. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1114–1123, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[40] F. Durand, G. Drettakis, J. Thollot, and C. Puech. Conservative visibility preprocessing using extended projections. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 239–248, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[41] H. Edelsbrunner. *Algorithms in combinatorial geometry*. Springer-Verlag New York, Inc., New York, NY, USA, 1987.

[42] C. Everitt. Interactive order-independent transparency, 2001.

[43] Q. Fan, A. Efrat, V. Koltun, S. Krishnan, and S. Venkatasubramanian. Hardware assisted natural neighbour interpolation. In *Proc. 7th Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2005.

[44] G. Farin. *Curves and Surfaces for Computer-Aided Geometric Design — A Practical Guide*. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. Morgan Kaufmann Publishers (Academic Press), 5^{th} edition, 2002. 499 pages.

[45] I. Fisher and C. Gotsman. Fast approximation of high order voronoi diagrams and distance transforms on the GPU. *Journal of Graphics Tools*, 11(4):39–60, 2006.

[46] P. Fisher. Stretching the viewshed. In *Sixth International Symposium on Spatial Data Handling*, pages 725–738, 1994.

[47] P. Fisher. Extending the applicability of viewsheds in landscape planning. *Photogrammetric engineering and remote sensing (Photogramm. eng. remote sensing)*, 62(11):1243–1306, 1996.

[48] P. F. Fisher. Algorithm and implementation uncertainty in viewshed analysis. *International Journal of Geographical Information Science*, 7(4):331–347, July 1993.

[49] P. F. Fisher. Reconsideration of the viewshed function in terrain modeling. *Geographical Systems*, 3:33–58, 1996.

[50] M. S. Floater and K. Hormann. Surface parameterization: a tutorial and survey. In N. A. Dodgson, M. S. Floater, and M. A. Sabin, editors, *Advances in multiresolution for geometric modelling*, pages 157–186. Springer Verlag, 2005.

[51] L. D. Floriani, B. Falcidieno, G. Nagy, and C. Pienovi. Polyhedral terrain description using visibility criteria. Technical Report 17, nstitute for Applied Mathematics, National Resemvh Council, 1989.

[52] L. D. Floriani, B. Falcidieno, G. Nagy, and C. Pienovi. On sorting triangles in a delaunay tessellation. *Algorithmica*, 6(4):522–532, 1991.

[53] L. D. Floriani and P. Magillo. Visibility algorithms on triangulated digital terrain models. *International Journal of Geographical Information Systems*, 8(1):13–41, 1994.

[54] L. D. Floriani and P. Magillo. Algorithms for visibility computation on terrains: a survey. *Environment and Planning B: Planning and Design*, 30(5):709–728, 2003.

[55] L. D. Floriani, E. Puppo, and P. Magillo. *Handbook of Computational Geometry*, chapter Ch. 7, Applications of Computational Geometry to Geographic Information Systems, pages 333–388. Elsevier Science, 1999.

[56] J. D. Foley, A. van Dam, and J. F. Hughes. *Computer Graphics, Principles and Practice.* Addison-Wesley, 1990.

[57] W. R. Franklin and C. K. Ray. Higher isn't necessarily better: Visibility algorithms and experiments. In *Sixth International Symposium on Spatial Data Handling*, volume 2, pages 751–763, Edinburgh, Scotland, 1994.

[58] W. R. Franklin and C. Vogt. Efficient multiple observer siting on large terrain cells. In *GIScience 2004 Third International Conference on Geographic Information Science*, 2004.

[59] W. R. Franklin and C. Vogt. Multiple observer siting on terrain with intervisibility or lo-res data. In *XXth Congress, International Society for Photogrammetry and Remote Sensing*, pages 12–23, 2004.

[60] D. Gddeke. *GPGPU::Reduction Tutorial.*

[61] P.-L. George and H. Borouchaki. *Delaunay Triangulation and Meshing: Applications to Finite Elements.* Editions Hermes, 1998.

[62] S. Ghali. Computation and maintenance of visibility and shadows in the plane. In *Proc. 6th Int. Conf. Computer Graphics & Visualization*, pages 117–124, Feb 1998.

[63] S. Ghali and A. J. Stewart. Incremental update of the visibility map as seen by a moving viewpoint in two dimensions. In *Proceedings of the Eurographics workshop on Computer animation and simulation '96*, pages 3–13, New York, NY, USA, 1996. Springer-Verlag New York, Inc.

[64] S. Ghali and A. J. Stewart. Maintenance of the set of segments visible from a moving viewpoint in two dimensions. In *SCG '96: Proceedings of the twelfth annual symposium on Computational geometry*, pages 503–504, New York, NY, USA, 1996. ACM Press.

[65] N. Greene, M. Kass, and G. Miller. Hierarchical z-buffer visibility. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 231–238, New York, NY, USA, 1993. ACM Press.

[66] J. Gudmundsson, M. H. Hammar, and M. J. van Kreveld. Higher order delaunaytriangulations. *Computational Geometry Theory & Applications*, 23(1):85–98, Jul 2002.

[67] H. W. Guesgen, J. Hertzberg, R. Lobb, and A. Mantler. First steps towards buffering fuzzy maps with graphics hardware. In *FOIS Workshop on Spatial Vagueness, Uncertainty and Granularity*, FOIS Workshop on Spatial Vagueness, Uncertainty and Granularity, 2001.

[68] N. Gupta and S. Sen. An efficient output-size sensitive parallel algorithm for hidden-surface removal for terrains. *Algorithmica*, 31(2):179–207, 2001.

[69] S. Har-Peled. Approximate shortest paths and geodesic diameter on a convex polytope in three dimensions. *Discrete & Computational Geometry*, 21(2):217–231, 1999.

[70] S. Har-Peled. Constructing approximate shortest path maps in three dimensions. *SIAM J. Comput.*, 28(4):1182–1197, 1999.

[71] S. Hart and M. Sharir. Nonlinearity of davenport-schinzel sequences and of generalized path compressions schemes. *Combinatorica*, 6, 1986.

[72] J. Hershberger. Finding the upper envelope of n line segments in o(n log n) time. *Inf. Process. Lett.*, 33(4):169–174, 1989.

[73] J. Hershberger and S. Suri. Practical methods for approximating shortest paths on a convex polytope in R^3. In *SODA '95: Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 447–456, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.

[74] S. Hertel, M. Mntyl, K. Mehlhorn1, and J. Nievergelt. Space sweep solves intersection of convex polyhedra. *Acta Informatica*.

[75] M. Hesse and M. L.Gavrilova. An efficient algorithm for real-time 3D terrain walkthrough. *International Journal of CAD/CAM*, 3(2):111–117, 2003.

[76] K. E. Hoff III, J. Keyser, M. Lin, D. Manocha, and T. Culver. Fast computation of generalized Voronoi diagrams using graphics hardware. *Computer Graphics*, 33(Annual Conference Series):277–286, 1999.

[77] T. C. Hudson, D. Manocha, J. D. Cohen, M. C. Lin, K. E. H. III, and H. Zhang. Accelerated occlusion culling using shadow frusta. In *Symposium on Computational Geometry*, pages 1–10, 1997.

[78] J. W. Jaromczyk and M. Kowaluk. Skewed projections with an application to line stabbing in r3. In *Symposium on Computational Geometry*, pages 362–370, 1988.

[79] B. Kaneva and J. O'Rourke. An implementation of Chen and Han's shortest paths algorithm. In *Proceedings of the 12th Canadian Conference on Computational Geometry*, pages 139–146, 2000.

[80] S. Kapoor. Efficient computation of geodesic shortest paths. In *STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 770–779, 1999.

[81] S. Kapoor. Efficient computation of geodesic shortest paths. In *STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 770–779, New York, NY, USA, 1999. ACM Press.

[82] S. Kim and C. Shin. Computing the optimal bridge between two polygons. Technical Report Research Report TSCS-99-14, HKUST, 1999.

[83] Y.-H. Kim, S. Ranab, and S. Wise. Exploring multiple viewshed analysis using terrain features and optimisation techniques. *Computers and Geosciences*, 30(9–10):1019–1032, 2004.

[84] R. Kimmel and J. Sethian. Computing geodesic paths on manifolds. In *Proceedings of National Academy of Sciences*, volume 95.

[85] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM J. Comput.*, 15(1):287–299, 1986.

[86] V. Koltun, Y. Chrysanthou, and D. Cohen-Or. Hardware-accelerated from-region visibility using a dual ray space. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 205–216, London, UK, 2001. Springer-Verlag.

[87] S. Krishnan, N. H. Mustafa, and S. Venkatasubramanian. Hardware-assisted computation of depth contours. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 558–567, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[88] C. Kumsap, F. Borne, and D. Moss. The technique of distance decayed visibility for forest landscape visualization. *International Journal of Geographical Information Science*, 19(6):723–744, July 2005.

[89] M. Lanthier, A. Maheshwari, and J.-R. Sack. Approximating shortest paths on weighted polyhedral surfaces. *Algorithmica*, 30(4):527–562, 2001.

[90] M. Lanthier, D. Nussbaum, and J.-R. Sack. Parallel implementation of geometric shortest path algorithms. *Parallel Comput.*, 29(10):1445–1479, 2003.

[91] M. A. Lanthier, D. Nussbaum, and T.-J. Wang. Calculating the meeting point of scattered robots on weighted terrain surfaces. In M. Atkinson and F. Dehne, editors, *Eleventh Computing: The Australasian Theory Symposium (CATS2005)*, volume 41 of *CRPIT*, pages 107–118, Newcastle, Australia, 2005. ACS.

[92] D. T. Lee. O k-nearest neighbor voronoi diagrams in the plane. *IEEE Trans. Comput.*, 31:478–487, 1982.

[93] J. Lee. Analyses of visibility sites on topographic surfaces. *International Journal of Geographical Information Systems*, 5:413–429, 1991.

[94] B. Liu, L.-Y. Wei, and Y.-Q. Xu. Multi-layer depth peeling via fragment sort. *Microsoft Research*, (MSR-TR-2006-81):4, June 2006.

[95] Y.-J. Liu, Q.-Y. Zhou, and S.-M. Hu. Handling degenerate cases in exact geodesic computation on triangle meshes. *The Visual Computer*, 23(9-11):661–668, 2007.

[96] M. Llobera. Extending gis-based visual analysis: The concept of visualscapes. *International Journal of Geographic Information Science*, 17(1):25–48, 2003.

[97] Z. Luo, H. Liu, Z. Yang, and X. Wu. Self-organizing maps computing on graphic process unit. In *ESANN*, pages 557–562, 2005.

[98] D. Martinez, L. Velho, and P. C. Carvalho. Geodesic paths on triangular meshes. In *SIBGRAPI '04: Proceedings of the Computer Graphics and Image Processing, XVII Brazilian Symposium on (SIBGRAPI'04)*, pages 210–217, 2004.

[99] J. Matoušek. *Lectures on Discrete Geometry*. Number 212 in Graduate Texts in Mathematics. Springer-Verlag, 2002.

[100] M. McKenna. Worst-case optimal hidden-surface removal. *ACM Trans. Graph.*, 6(1):19–28, 1987.

[101] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1983.

[102] J. Miheli and B. Robi. Facility location and covering problems. In *Proc. of the 7th International Multiconference Information Society*, 2004.

[103] K. Mills, G. Fox, and R. Heimbach. Implementing an intervisibility analysis model on a parallel computing system. *Comput. Geosci.*, 18(8):1047–1054, 1992.

[104] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, 16(4):647–668, 1987.

[105] J. S. B. Mitchell and C. H. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *J. ACM*, 38(1):18–73, 1991.

[106] E. Moet, C. Knauer, and M. van Kreveld. Visibility of segments and triangles in 3D. In *Visibility of Segments and Triangles in 3D*, volume 1, 2006.

[107] E. Moet, M. van Kreveld, and A. F. van der Stappen. On realistic terrains. In *SCG '06: Proceedings of the twenty-second annual symposium on Computational geometry*, pages 177–186, New York, NY, USA, 2006. ACM Press.

[108] E. Moet, M. van Kreveld, and R. van Oostrum. Region intervisibility in terrains. *International Journal of Computational Geometry*, 17(4):331–347, 2007.

[109] D. Mould and M. Horsch. An hierarchical terrain representation for approximately shortest paths. In *PRICAI*, pages 104–113, 2004.

[110] D. M. Mount. Voronoi diagrams on the surface of a polyhedron. Technical report, University of Maryland, 1985.

[111] N. H. Mustafa, S. Krishnan, G. Varadhan, and S. Venkatasubramanian. Dynamic simplification and visualization of large maps. *International Journal of Geographical Information Science*, 20(3):273–302, 2006.

[112] Z. Nagy and R. Klein. Depth-peeling for texture-based volume rendering. In *11th Pacific Conference on Computer Graphics and Applications (PG)*, volume 00, pages 429–433, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[113] K. Nechvíle and P. Tobola. Local approach to dynamic visibility in the plane. In *Proc. 7th Int. Conf. Computer Graphics, Visualization and Interactive Digital Media (WSCG '99)*.

[114] M. Novotni and R. Klein. Computing geodesic distances on triangular meshes. In *Proc. 10th International Conference in Central Europe on Compute Graphics, Visualization and Computer Vision*, pages 341–347, 2002.

[115] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial Tessellation: Concepts and Application of Voronoi Diagrams.* John Wiley and Sons, 2000.

[116] A. Okabe, B. Boots, and K. Suihara. Nearest neighbourhood operations with generalized voronoi diagrams: a review. *International Journal of Geographical Information Science*, 8(1):43–71, 1994.

[117] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krger, A. E. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80–113, 2007.

[118] M. Pocchiola and G. Veger. The visibility complex. *International Journal of Computer geometry and applications*, 6(3):279–308, 1996.

[119] F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, 20(2):87–93, 1977.

[120] F. P. Preparata and M. I. Shamos. *Computational geometry: an introduction.* Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[121] E. Puppo and P. Marzano. Discrete visibility problems and graph algorithms. *International Journal of Geographical Information Science*, 11(2):139–161, 1997.

[122] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan. Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics*, 21(3):703–712, July 2002. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

[123] P. J. Rallings, J. A. Ware, and D. B. Kidner. Parallel distributed viewshed analysis. In *Proc. ACMGIS'98*, pages 151–156, 1998.

[124] J. Reif and Z. Sun. An efficient approximation algorithm for weighted region shortest path problem. In *Proc. of the 4th Workshop on Algorithmic Foundations of Robotincs (WAFR2000)*, pages 191–203, 2000.

[125] S. Rivière. Dynamic visibility in polygonal scenes with the visibility complex. In *3th Annual ACM Symposium on Computational Geometry*, pages 421–423, 1997.

[126] J. Robert and G. Toussaint. Computational geometry and facility location. In *International Conference on Operations Research and Management Science*, pages 1–19, 1990.

[127] A. Saalfeld. Delaunay triangulations and stereographic projections. *Cartography and Geographic Information Science*, 26(4):289–296, October 1999.

[128] J.-R. Sack and J. Urrutia. *Handbook of computational geometry.* North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands, 2000.

[129] G. Schaufler, J. Dorsey, X. Decoret, and F. X. Sillion. Conservative volumetric visibility with occluder fusion. In *Proceedings of SIGGRAPH 2000*, pages 229–238, 2000.

[130] D. Schmitt and J.-C. Spehner. Order-k voronoi diagrams, k-secions, and k-sets. In *JCDCG'98*, pages 290–304, 2000.

[131] Y. Schreiber and M. Sharir. An optimal-time algorithm for shortest paths on a convex polytope in three dimensions. In *SCG '06: Proceedings of the twenty-second Annual Symposium on Computational Geometry*, pages 30–39, New York, NY, USA, 2006. ACM Press.

[132] M. Segal and K. Akeley. *The OpenGL Graphics System: A Specification.* http://www.opengl.org/documentation/specs/version2.0/glspec20.pdf., 2004.

[133] A. Shapria. Visibility and terrain labelling. Master's thesis, NY, Rensselaer Polytechnic Institute, NY, Rensselaer Polytechnic Institute.

[134] M. Sharir. Arrangements in higher dimensions: Voronoi diagrams, motion planning, and other applications. In *WADS*, pages 109–121, 1995.

[135] M. Sharir. The clarkson-shor technique revisited and extended. *Combinatorics, Probability & Computing*, 12(2):191–201, 2003.

[136] M. Sharir(and A. Schorr. On shortest paths in polyhedral spaces. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 144–153, New York, NY, USA, 1984. ACM Press.

[137] B. Sharp. Optimizing curved surface geometry. *Game Developer*.

[138] C. Sigg, R. Peikert, and M. Gross. Signed distance transform using graphics hardware. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, pages 83–90, 2003.

[139] J. Snoeyink and M. J. van Kreveld. Linear-time reconstruction of delaunay triangulations with applications. In *ESA '97: Proceedings of the 5th Annual European Symposium on Algorithms*, pages 459–471, London, UK, 1997. Springer-Verlag.

[140] P. Sorensen and D. Lanter. Two algorithms for determining partial visibility and reducing data structure induced error in viewshed analysis. *Photogrammetric Engineering and Remote Sensing*, 59(3):1129–1132, 1993.

[141] A. J. Stewart. Fast horizon computation at all points of a terrain with visibility and shadow applications. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):82–93, 1993.

[142] A. Sud, M. Otaduy, and D. Manocha. DiFi: Fast 3D distance field computation using graphics hardware. In *Eurographics*, volume 23, pages 557–566, 2004.

[143] Z. Sun and J. Reif. On finding approximate optimal paths in weighted regions. *Journal of Algorithms*, 58(1):1–32, 2006.

[144] V. Surazhsky, T. Surazhsky, D. Kirsanov, S. J. Gortler, and H. Hoppe. Fast exact and approximate geodesics on meshes. *ACM Trans. Graph.*, 24(3):553–560, 2005.

[145] M. Teillaud. Towards dynamic randomized algorithm in computational geometry. Technical Report 1727, Unit de Recherche Inria-Sophia Antipolis, 1992.

[146] Y. Teng, D. D. Menthon, and L. Davis. Region-to-region visibility analysis using data parallel machines. *Concurrency: Practice and Experience*, 5:379–406, 1993.

[147] D. T.Lee and B. Schachter. Two algorithms for constructing delaunay triangulations. *International Journal of Computer Information Sciences*, 9(3):219–242, 1980.

[148] K. N. P. Tobola. Dynamic visibility in the plane. In *15th Spring Conf. Computer Graphics*, f1999.

[149] M. J. van Kreveld. Variations on sweep algorithms: efficient computation of extended viewsheds and clas intervals. *International Journal of Computational Geometry an dApplications*, 7(1-2):82–93, 1996.

[150] M. J. van Kreveld. Digital elevation models and tin algorithms. In *Algorithmic Foundations of Geographic Information Systems, this book originated from the CISM Advanced School on the Algorithmic Foundations of Geographic Information Systems*, pages 37–78, London, UK, 1997. Springer-Verlag.

[151] K. R. Varadarajan and P. K. Agarwal. Approximating shortest paths on a nonconvex polyhedron. *SIAM J. Comput.*, 30(4):1321–1340, 2000.

[152] C. A. Wang and B. Zhu. Three-dimensional weak visibility: Complexity and applications. *Theor. Comput. Sci.*, 234(1-2):219–232, 2000.

[153] E. Weiszfeld. Sur le point pour lequel la somme des distances de n points donnes est minimum. *Tohoku Mathematical Journal*, 43:355–386, 1936.

[154] E. Welzl. Smallest enclosing disks (balls and ellipsoids). *New Results and New Trends in Computer Science*, 555:359–370, 1991.

[155] P. Wonka and D. Schmalstieg. Occluder shadows for fast walkthroughs of urban environments. *Computer Graphics Forum*, 18(3):51–60, 1999.

[156] C.-K. Yap. An o (n log n) algorithm for the voronoi diagram of a set of simple curve segments. *Discrete & Computational Geometry*, 2:365–393, 1987.

[157] Z. Youbing, Z. Ji, S. Jiaoying, and P. Zhigeng. A fast algorithm for large scale terrain walkthrough. In *CAD/Graphics2001*, 2001.

[158] H. Zhang, D. Manocha, T. Hudson, and K. Hoff. Visibility culling using hierarchical occlusion maps. In *Proceedings of SIGGRAPH 97*, pages 77–88, 1997.

[159] J. Zhang, D. Papadias, K. Mouratidis, and Z. Manli. Query processing in spatial databases containing obstacles. *International Journal of Geographical Information Science*, 19(10):1091–1111, November 2005.