Proximity and applications in general metrics

Gavrilova, Marina

http://hdl.handle.net/1880/42433
doctoral thesis

University of Calgary graduate students retain copyright ownership and moral rights for their thesis. You may use this material in any way that is permitted by the Copyright Act or through licensing that has been assigned to the document. For uses that are not allowable under copyright legislation or licensing, you are required to seek permission.

Downloaded from PRISM: https://prism.ucalgary.ca
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-38468-3
ABSTRACT

In this thesis data structures that represent the proximity information for a system of geometrical objects are investigated and efficient algorithms for their construction and maintenance are developed.

The focus of the studies is on generalized Voronoi diagrams and Delaunay tessellations for weighted set of sites under Euclidean, power, Manhattan and supremum metrics in 2 and $d$-dimensional space. General properties are developed and a generic incremental algorithm for data structure construction is obtained for generalized Voronoi diagrams and Delaunay tessellations in $d$ dimensions. The method is implemented using exact arithmetic for power metric. A sweep-plane method is presented and tested for the Manhattan and supremum metrics. A method to construct the power diagram from the Voronoi diagram based on a bisector transformation technique is presented and experimental studies are conducted. The space of all triangulations in $d$ dimensions is considered and theorems about the distance between any two elements of this space and the size of this space are proven for 2-dimensional space.

The theory is applied to the collision detection problem that arises during computer simulation of physical systems in the second part of this thesis. Regular spatial subdivision, regular spatial tree and set of segment trees data structures are introduced and investigated. A generalized algorithm for the collision detection optimization in the system of moving objects in 2 and $d$ dimensions is developed. Topological and velocity update events in the dynamic system of moving objects are defined and studied. Algorithms for construction and maintenance of data structures are described and tested.

A comparison of data structures is performed in the terms of their efficiency, space requirements and the complexity of their construction. Applied problems from areas of granular-type material simulations are considered and some recommendations about the choice of appropriate data structure for implementation are presented based on the test results.
ACKNOWLEDGEMENTS

I gratefully acknowledge the support, guidance and encouragement that I received from my supervisor, Dr. Jon Rokne, throughout the course of my studies. I thank him for careful reading, valuable suggestions and helpful discussions of my dissertation. I would like to thank my co-supervisor, Dr. Oleg Vinogradov, for his help, advice and encouragement.

I would like to express my gratitude to Dr. Jack Snoeyink and Dr. Wayne Eberly for valuable comments and suggestions. I am thankful for the financial support in the form of scholarships from the Killam Foundation and from the National Science and Engineering Research Council of Canada.

Finally, my special thanks to my husband, Dmitri Gavrilov, for his love and support. Thanks to my parents Lev and Tatiana Felman for believing in me. And my very special thanks to my son, Andrei, for giving me inspiration and motivation to explore. It is to him that I dedicate my dissertation.
# TABLE OF CONTENTS

**APPROVAL PAGE** ........................................................................................................ ii

**ABSTRACT** .................................................................................................................. iii

**ACKNOWLEDGEMENTS** .............................................................................................. iv

**TABLE OF CONTENTS** ................................................................................................... v

**LIST OF TABLES** .......................................................................................................... xi

**LIST OF FIGURES** ........................................................................................................... xii

**EPIGRAPH** ..................................................................................................................... xvi

## CHAPTER 1: INTRODUCTION ......................................................................................... 1

1.1 Computational Geometry Algorithms in Computer Simulation .................................. 2

1.2 Thesis Outline .............................................................................................................. 3

1.3 Thesis Content ............................................................................................................. 4

## CHAPTER 2: PRELIMINARIES AND GENERALIZED VORONOI DIAGRAMS .......... 6

2.1 Geometrical Preliminaries ............................................................................................. 6

2.2 Definition and Properties of the Voronoi Diagram ....................................................... 8

2.3 Definition and Properties of the Delaunay Tessellation ................................................ 12

   2.3.1 Definition and Properties of the Planar Delaunay Triangulation ......................... 13

   2.3.2 Definition and Properties of the $d$-dimensional Delaunay Tessellation ............... 16

2.4 Methods for the Voronoi Diagram and the Delaunay Tessellation Construction .......... 17

   2.4.1 The Worst-Case Time Complexity for the VD Construction ............................... 18

   2.4.2 The Straightforward Method ................................................................................ 18

   2.4.3 The Incremental Method ....................................................................................... 19

   2.4.4 The Edge Flipping Method .................................................................................... 20

   2.4.5 The Divide-and-Conquer Method ....................................................................... 20
2.4.6 The Sweep-Plane Method .............................................................. 21
2.4.7 The Convex Hull Method .............................................................. 21
2.5 The Generalized Voronoi Diagram ................................................. 22
2.6 The Generalized Delaunay Tessellation ......................................... 26
2.7 Methods for Construction of the Generalized Voronoi Diagram ........ 26
2.7.1 The Worst-Case Time Complexity for the Generalized VD
Construction .......................................................................................... 27
2.7.2 The Incremental Method .............................................................. 27
2.7.3 The Divide-and-Conquer Method ................................................ 27
2.7.4 The Sweep-Plane Method ........................................................... 28
2.7.5 The Convex Hull Method ........................................................... 29

CHAPTER 3: PROPERTIES OF THE GENERALIZED VORONOI DIAGRAM ....... 30
3.1 The Power Diagram ........................................................................... 31
3.1.1 Definition of the Power Diagram and Preliminaries ..................... 31
3.1.2 Properties of the Power Diagram in $d$ Dimensions ..................... 35
3.1.3 The Delaunay Tessellation in the Power Metric ......................... 45
3.2 The Weighted Voronoi Diagram in the Euclidean Metric ................ 46
3.2.1 Definition of the Euclidean Weighted Voronoi Diagram and
Preliminaries ......................................................................................... 46
3.2.2 Properties of the Euclidean Weighted Voronoi Diagram in $d$
Dimensions ......................................................................................... 49
3.2.3 The Euclidean Delaunay Tessellation ......................................... 60
3.3 The Weighted Voronoi Diagram in the Manhattan and the Supremum
Metrics .................................................................................................. 62
3.3.1 Definition of the Weighted Voronoi Diagram in the Manhattan and
the Supremum Metric and Preliminaries ........................................... 62
3.3.2 Properties of the RWVD in $d$ Dimensions .................................. 66
3.3.3 The Delaunay Tessellation Corresponding to the Manhattan
(Supremum) VD ............................................................................... 76
3.4 Concluding Remarks .......................................................................... 78
5.6 The Complexity of the Algorithm ......................................................... 137
5.7 Implementation and Experimental Results ............................................. 138
5.8 The Sweep-Plane Algorithm for Spheres in the Power Metric ................. 140
5.9 Concluding Remarks ............................................................................. 142

CHAPTER 6: CONSTRUCTION OF THE POWER DIAGRAM FROM THE
VORONOI DIAGRAM .................................................................................. 143

6.1 Literature Review .................................................................................. 143
6.2 Chapter Overview .................................................................................. 145
6.3 Relationships between the Power diagram and the Voronoi diagram ........... 145
6.4 Basic Idea of the Algorithm .................................................................... 147
6.5 Main Scheme of the Algorithm ............................................................... 149
6.6 Implementation of the Algorithm ............................................................ 154
6.7 Experimental Results ............................................................................ 155
6.8 Concluding Remarks ............................................................................. 157

CHAPTER 7: SPACE OF TESSELLATIONS .................................................. 158

7.1 Chapter Overview .................................................................................. 159
7.2 The Abstract Voronoi Diagram ............................................................... 160
7.3 The Space of Generalized Delaunay Tessellations .................................. 161
7.4 The Space of All Tessellations ............................................................... 169

CHAPTER 8: COLLISION DETECTION OPTIMIZATION ................................ 175

8.1 Chapter Overview .................................................................................. 176
8.2 Collision Detection between Two Arbitrary Shaped Objects ..................... 177
8.3 Collision Detection between Two Spherical Objects ................................ 178
8.4 Collision Detection for a Set of Objects ................................................ 180
8.5 Computer Simulation ............................................................................. 181
8.6 Data Structure Maintenance ................................................................... 182
8.7 Algorithm Description ........................................................................... 183
8.8 Implementation Details .......................................................................... 185
8.9 Time-Step Oriented Simulation ............................................................. 186
CHAPTER 9: DYNAMIC DATA STRUCTURES

9.1 Dynamic Computational Geometry ................................................................. 190
9.2 Dynamic Data Structures ............................................................................. 191
9.3 The Dynamic Generalized Voronoi Diagram .............................................. 194
  9.3.1 Algorithms for the Dynamic Voronoi Diagram Construction ............. 194
  9.3.2 Algorithms for the Dynamic Generalized Voronoi Diagram
      Construction .............................................................................................. 196
  9.3.3 Dynamic Generalized Delaunay Triangulation Algorithm ............... 197
  9.3.4 The Topological Event ............................................................................ 198
  9.3.5 Determination of the Topological Event ............................................. 201
  9.3.6 The Algorithm for the Dynamic Generalized DT Maintenance ....... 202
  9.3.7 Algorithm Complexity .......................................................................... 203
9.4 The Regular Spatial Subdivision ................................................................. 205
  9.4.1 The Topological Event .......................................................................... 206
  9.4.2 The Algorithm for the Regular Spatial Subdivision Maintenance .... 207
  9.4.3 Algorithm Complexity .......................................................................... 208
9.5 The Regular Spatial Tree ............................................................................. 209
  9.5.1 Algorithm Complexity .......................................................................... 209
9.6 The Set of Segment Trees ........................................................................... 210
  9.6.1 The Topological Event .......................................................................... 211
  9.6.2 The Algorithm for the Set of Segment Trees .................................... 212
  9.6.3 Algorithm Complexity .......................................................................... 213
9.7 Performance Analysis .................................................................................. 215

CHAPTER 10: APPLICATIONS TO GRANULAR-TYPE MATERIAL

SIMULATION ......................................................................................................... 218

10.1 Definition of Granular-Type Materials ...................................................... 218
10.2 Shaker Mill Model ...................................................................................... 219
10.3 Shaker Mill Model as a Multibody Dynamics Model ............................... 221
10.4 Collision Detection Optimization for Shaker Mill Simulation Model ...... 222
LIST OF TABLES

9.7.1 Algorithm efficiency comparison table for planar case ........................................... 215
9.7.2 Algorithm efficiency comparison table for $d$ dimensions ........................................ 216
10.4.1 Monosized data sets .................................................................................................. 223
10.4.2 Experimental results for monosized data sets .......................................................... 224
10.4.3 Polysized data sets ...................................................................................................... 226
10.4.4 Experimental results for polysized data sets .............................................................. 226
10.4.5 Sample shaker mill data sets ....................................................................................... 227
LIST OF FIGURES

2.2.1 The planar Voronoi diagram.................................................................9
2.2.2 Bounded (a) and unbounded (b) Voronoi regions in the plane ...............10
2.3.1 The Delaunay triangulation in the plane .............................................13
2.3.2 Two possible triangulations of a quadrilateral ....................................15
3.1.1 Geometrical interpretation of the power distance and the power bisector ....32
3.1.2 The power diagram for a set of circles in the plane ................................33
3.1.3 Ray intersects with the boundary of the convex hull .........................38
3.1.4 Angle between spheres ......................................................................40
3.1.5 Inscribed sphere $C$ intersects sphere $P_i$ ..........................................41
3.1.5 Positions of center of sphere $Q$ relative to inscribed sphere $C$ ............38
3.1.6 Point $y$ is the closest point of the bisector to both spheres $P$ and $Q$ ......43
3.1.7 Proof of the fact that $Q$ is the nearest-neighbor of $P$ .......................44
3.1.8 The power Delaunay triangulation.......................................................45
3.2.1 The distance and bisector in the Euclidean metric ...............................47
3.2.2 Euclidean weighted Voronoi diagram .................................................48
3.2.3 An unbounded ray $r$ inside $EVar(P)$ .................................................51
3.2.4 Illustration for the proof of Theorem 3.2.1 .........................................52
3.2.5 Assumption 3.2.4 is violated for $m = 4$ ............................................54
3.2.6 Assumption 3.2.4 for $m = 3$ .............................................................54
3.2.7 Illustration for the proof of Theorem 3.2.3 ........................................58
3.2.8 The Euclidean distance between two circles ......................................58
3.2.9 Illustration for the proof of Theorem 3.2.4 .........................................59
3.2.10 Euclidean weighted Voronoi diagram and Delaunay triangulation .......60
3.2.11 The EWVD and the corresponding Delaunay tessellation .................61
3.3.1 Spheres in the Manhattan and the supremum metrics in $R^3$ ...............63
3.3.2 Distance in the Manhattan and the supremum metric .........................63
3.3.3 The RWVD in the Manhattan (a) and the supremum (b) metrics in the plane ......64
3.3.4 Distance between two circles in the supremum metric ..........................64
3.3.5 $L_\infty$ bisector is a region in the plane ......................................................... 64
3.3.6 Two pairs of $L_\infty$ spheres are equidistant ................................................. 65
3.3.7 Bisectors in $L_1$ in the plane ........................................................................... 68
3.3.8 Bisectors in $L_\infty$ in the plane ........................................................................ 68
3.3.9 Two quadrants with axes for spheres $P$ and $Q$ ............................................ 70
3.3.10 Rectangular hull for a set of spheres in the supremum metric ....................... 71
3.3.11 Proof of Lemma 3.3.2 ..................................................................................... 73
3.3.12 Illustration for the proof of Theorem 3.3.3 ..................................................... 75
3.3.13 Illustration for proof of Theorem 3.3.4 .......................................................... 75
3.3.14 Weighted VD and DT in the Manhattan (a) and the supremum (b) metrics .... 77
3.3.15 The RWVD and the corresponding Delaunay tessellation ............................ 77
4.4.1 Four circles are cocircular in the power metric ............................................... 84
4.5.1 Four circles are cocircular in the Euclidean metric ......................................... 91
4.5.2 Circles with reduced radii ................................................................................ 92
4.5.3 Complex transformation ................................................................................. 93
4.5.4 Infinite number of inscribed spheres ............................................................... 97
4.5.5 Linearly dependent spheres ............................................................................. 98
4.6.1 Inscribed sphere in 2D .................................................................................... 102
4.6.2 Families of inscribed spheres for sphere $P_1$ ................................................. 103
4.6.3 Families of inscribed spheres for spheres $P_1$ and $P_2$ ................................. 104
4.6.4 One, two or zero inscribed spheres ................................................................. 106
4.7.1 Sample output for a set of 100 sites ............................................................... 114
4.7.2 The time ratio vs. the number of sites ............................................................ 116
4.7.3 The percentage of wrong edges vs. the number of sites ................................. 117
4.7.4 The time ratio vs. the perturbation .................................................................. 118
4.7.5 The percentage of wrong edges vs. the perturbation ..................................... 118
5.2.1 Pyramids in 3D in Manhattan metric ............................................................ 122
5.2.2 Illustration for the proof of Lemma 5.2.1 ....................................................... 124
5.3.1 The parabola in $L_1$ ...................................................................................... 125
5.3.2 The intersection between the sweep plane and a pyramid ............................. 126
5.3.3 The parabola widens as the sweep plane moves through the pyramid ............ 127
5.3.4 The collision point between two parabolas and the corresponding wave front .... 128
5.3.5 Illustration for proof of Lemma 5.3.3 ................................................................ 130
5.3.6 The portion of a parabola in the wave front .............................................. 131
5.3.7 The site event in the wave front .................................................................... 131
5.3.8 The circle event in the wave front ............................................................... 132
5.4.1 Illustration of the definition of the site event ................................................ 134
5.7.1 Program interface ....................................................................................... 139
5.7.2 The number of sites vs. time ....................................................................... 140
5.8.1 Waves in the power metric ........................................................................... 141
6.3.1 The transformation of the power bisector ...................................................... 146
6.4.1 The transformation of Voronoi edges ........................................................... 147
6.4.2 The modification of the Voronoi diagram .................................................... 148
6.4.3 The modification of the quadrilateral ........................................................... 148
6.5.1 The quadrilateral \( p_i p_{i+1} p_{i+2} p_{i+3} \) ...................................................... 150
6.5.2 Non-convex quadrilateral ............................................................................. 150
6.5.3 Two possible configurations for the center of the inscribed circle ............... 152
6.5.4 Proof of Theorem 6.5.2 ................................................................................ 153
6.6.1 Program interface ......................................................................................... 155
6.7.1 The number of swaps required vs. the number of sites ................................ 156
6.7.2 The elapsed time vs. the number of sites ..................................................... 157
7.3.1 The curvilinear Delaunay edge ..................................................................... 165
7.3.2 Curvilinear Delaunay (a) and Euclidean (b) tessellations ............................ 166
7.3.3 Extended curvilinear Delaunay (a) and Euclidean (b) tessellations .......... 166
7.3.4 Embedding an edge of Delaunay tessellation into the plane ...................... 167
7.3.5 Embedding infinite edges into the plane ...................................................... 168
7.4.1 Swap operation in the plane .................................................................... 170
7.4.2 The canonical subgraph ............................................................................. 171
7.4.3 Edge transformations .................................................................................. 172
7.4.4 The configuration after the first step ......................................................... 172
Inspiration is needed in geometry, just as much as in poetry.

A. S. Pushkin

As long as algebra and geometry proceeded along separate paths, their advance was slow and their applications limited. But when these sciences joined company, they drew from each other fresh vitality and thenceforward marched on a rapid pace towards perfection.

J. L. Lagrange
CHAPTER 1: INTRODUCTION

Simple computations involving geometric objects were done as far back as records exist. The advent of computers opened up the possibilities for more complex geometric computations on larger numbers of objects. This gave rise to a new field called computational geometry, which generally deals with algorithms for solving geometric problems suitable for computer implementation.

The term *computational geometry* was originally introduced by Forrest to describe computational techniques in computer-aided design [Forrest 71]. Later, Shamos used this term to refer to the design and analysis of algorithms for different geometric problems in his Ph.D. thesis at Yale University [Shamos 77]. His work dealt with traditional geometry problems on point sets in Euclidean space and algorithms for their implementation. Shamos was one of the first to define computational geometry as an individual branch of applied mathematics.

Generally, computational geometry studies a system of geometrical objects, such as sets of points, lines, planes, polygons and polyhedra, and their properties and related data structures. Computational geometry also focuses on the design of efficient algorithms when the number of objects in the input set is large. It investigates algorithms for performing different tasks on these geometrical sets, for example, finding the closest point to a given query point or the minimum enclosing circle for a set of points. It is also concerned with the optimality of geometric algorithms in terms of space and computation time required to perform the task, dependent on the number of objects.

The problems that are investigated vary from purely mathematical ones, such as the design and analysis of algorithms in the field of combinatorics, searching, convex sets, spline curves and surfaces, arrangements of lines and so on, to computer-oriented practical problems. Examples of the latter are problems from computer graphics, computer-aided design, pattern recognition and geographical information systems (GIS-systems) [de Berg et. al. 97, Okabe et. al. 92].
More and more applications can be found in astronomy, biology, cartography, chemistry, geography, mechanics, metallurgy, physics, statistics and urban planning as well as in other disciplines.

1.1 Computational Geometry Algorithms in Computer Simulation

Many problems encountered in everyday life require the symbiosis of mathematical methods and computer science approaches for their successful solution. One of the applied fields, where efficient data structures and where optimal algorithms are crucial and the application of computational geometry methods is necessary, is the computer simulation of physical systems.

When such a system is being simulated, it is represented by a set of objects, which can interact with each other according to physical laws. Many physical systems are simulated that way, from robots and manufacturing conveyors to molecular structures and gases, from biological systems of growing plants to avalanches and flowing ice. Generally, the dynamics of these systems cannot be studied analytically because of the large number of bodies in the system. Computer simulation is therefore almost the only way for investigating properties of these physical systems. The computational time and resources required for simulation grow very quickly as the number of objects in the system increases. The use of efficient computational geometry algorithms for optimization of the simulation tasks in such systems becomes crucial for the success of the simulation.

The first problem that must be solved is how to find an appropriate data structure that enables the efficient representation of the topological information and numerical parameters of the simulated system. The data structure should allow easy modification and efficient queries on the data. Efficient algorithms for its construction must be well defined and estimates on lower and upper bounds should be obtained.

The second problem that has to be solved during the simulation of a dynamic system is the collision detection problem. When simulated objects move in space they can collide and these collisions must be detected and handled in an appropriate manner. In a straightforward approach, each pair of objects must be checked to predict the next
collision. When the number of objects is very large collision detection consumes the bulk of the computational effort. This slows down the simulation so that the number of objects that can be simulated is severely restricted. Because of this real time implementations are only feasible for a small number of objects. More sophisticated approaches, taking advantage of computational geometry tools, are essential to speed up the solution and hence improve the simulation.

In this thesis some solutions for these problems are provided.

1.2 Thesis Outline

The main purpose of this research is to develop and analyze computational geometry algorithms and data structures and to consider their application to simulation of physical systems. When such systems are simulated, spheres are often used as a simple and effective model for a variety of objects, such as grain and ice particles, balls, beads, atoms and molecules [Krantz 96, Lubachevsky 91, Marin et. al. 93, Szeliski and Tonnesen 92, Vinogradov 92, Reinolds 87]. Some complex polyhedral objects can also be approximated by bounding spheres for easier visualization and faster collision detection [Hubbard 95 (Ph.D. Thesis), Hubbard 95, O'Rourke and Badler 79]. Therefore, data structures for spherical objects can be studied and used as efficient data representation for many applied problems.

In the first part of this thesis, properties of the Voronoi diagram and its straight-line dual, Delaunay tessellation, are investigated. The Voronoi diagram is a fundamental data structure for computational geometry that can be used to store proximity information for sets of objects. This geometric structure was first introduced almost a century ago [Voronoi 08]. More recently, a generalized Voronoi diagram was defined (see, for example, [Okabe et. al. 92]). This thesis focuses on the investigation of properties, algorithms and applications of Voronoi diagrams and Delaunay tessellations for sets of circles and spheres under different metrics (power, Manhattan, supremum and Euclidean) in 2 and d-dimensional space. Complexities of the algorithms are estimated, and the relationships among Delaunay tessellations in different metrics are examined. The
general property of closeness of objects, which is an invariant of the metric, is also considered. The space of all tessellations for a given set of sites is introduced and an estimate of the upper bound on the total number of tessellations is obtained.

In the second part of the thesis the optimization of data structures for collision detection for a set of moving objects in 2 and $d$-dimensional space is investigated. Such data structures as dynamic generalized Voronoi diagrams and dynamic generalized Delaunay triangulations, regular spatial subdivision, regular spatial trees and sets of segment trees are considered. Comparison of the data structures is performed in terms of time and space complexity of algorithms.

Collision detection algorithms for all of the considered data structures are implemented and tested for the applied problem from the area of computer simulation of the granular-type material system. Some recommendations about the choice of appropriate data structure for implementation are presented based on the test results.

1.3 Thesis Content

Chapter 1 discusses computational geometry algorithms in computer simulation and outlines the thesis.

Chapter 2 gives a literature review that emphasizes recent developments in the area of computational geometry data structures and algorithms and discusses concepts of generalized Voronoi diagrams and generalized Delaunay tessellations for objects other than points in different metrics.

Chapter 3 defines properties of the generalized weighted Voronoi diagram and Delaunay tessellation in Euclidean, power, Manhattan and supremum metrics in 2 and $d$-dimensional space.

Chapter 4 describes an incremental method for construction of Delaunay tessellations in power, Euclidean, Manhattan and supremum metrics in $d$ dimensions and provides formulas for the $\text{INCIRCLE}$ test computation. It also presents a method for exact
computations of the power diagram and compares its efficiency to the direct implementation of the incremental method.

Chapter 5 presents a sweep-plane method for Delaunay triangulation construction in Manhattan and supremum metrics in the plane. This method has optimal worst-case time complexity. This chapter also includes a discussion of a sweep-plane technique for the power metric. An implementation of the sweep-plane method and test results are described at the end of this chapter.

Chapter 6 provides a swapping technique for the construction of a power diagram from a Voronoi diagram in the plane. This technique was implemented. The experimental results from the implementation are also presented.

Chapter 7 discusses the extended curvilinear Delaunay tessellation and the space of all tessellations.

Chapter 8 investigates the application of the developed methods to collision detection. It presents a generalized algorithm for collision optimization and dynamic data maintenance in computer simulation systems.

Chapter 9 considers various data structures for dynamic collision detection, including dynamic generalized Voronoi diagrams and Delaunay triangulations, regular spatial subdivisions, regular spatial trees and sets of segment trees. These data structures are compared in terms of their efficiency and memory requirements.

Chapter 10 illustrates some practical applications of the proposed algorithms to applied problem from the field of mechanics. The implementation of the methods for collision detection optimization and comparison analysis of these methods based on the test results are presented in this chapter.

Chapter 11 provides conclusions and outlines some topics for further research.
CHAPTER 2: PRELIMINARIES AND GENERALIZED VORONOI DIAGRAMS

This chapter introduces some geometrical preliminaries that will be used in the rest of the thesis. It describes the definitions and properties of the Voronoi diagram and the Delaunay tessellation for a set of points in \( d \)-dimensional space and lists methods for their construction. Then the generalized Voronoi diagram and Delaunay tessellation in \( d \)-dimensional space under different distance functions are introduced. Some methods developed for these data structures are discussed.

2.1 Geometrical Preliminaries

Consider the \( d \)-dimensional real Cartesian space \( \mathbb{R}^d \) containing \( d \)-tuples of real numbers \( (x_1, x_2, \ldots, x_d) \) called points (see of [Okabe et. al. 92] for precise definitions).

Euclidean space \( E^d \) can be considered as an instance of \( \mathbb{R}^d \), where the Euclidean distance between two points \( x = (x_1, x_2, \ldots, x_d) \) and \( y = (y_1, y_2, \ldots, y_d) \) from \( E^d \) is computed as \( d(x, y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2} \). In the sequel 2D is used to denote 2-dimensional space and 3D is used to describe 3-dimensional space.

Geometrical objects in \( \mathbb{R}^d \) can be characterized in terms of properties of sets of points in \( \mathbb{R}^d \). Thus, the concepts of open, closed, bounded, unbounded, connected, simple-connected, disconnected, convex and star-shaped sets can be introduced. A full and precise description of the set properties used in this thesis can be found in [Okabe et. al. 92].

Some geometrical concepts are now defined.

---

1 The definitions given in this section can be found in Chapter 1, Section 1.3 of the book [Okabe et. al. 92].
Definition 2.1.1 The convex hull of a set of points \( S \) in \( R^d \) is the boundary of the smallest convex set in \( R^d \) containing \( S \) [Preparata and Shamos 85].

Definition 2.1.2 A simplex in \( R^d \) is the convex hull of any set \( p_1, \ldots, p_{d+1} \) of \( d + 1 \) points that do not all lie on the same hyperplane in \( R^d \). The simplex is denoted by \( (p_1, \ldots, p_{d+1}) \) [Okabe et. al. 92].

Definition 2.1.3 If a set in \( R^2 \) constructed from a finite number of applications of intersection and union to a finite number of half planes is a non-empty connected bounded set, then this set is called a polygon [Okabe et. al. 92].

The boundary of the polygon consists of straight-line segments called edges and their endpoints, called vertices.

Definition 2.1.4 If a set in \( R^d \) constructed from a finite number of applications of intersection and union to a finite number of half spaces is a non-empty connected bounded set, then this region is called a polyhedron [Okabe et. al. 92].

The boundary of a polyhedron consists of \( (d-1) \)-dimensional polyhedra called \( (d-1) \)-faces, the boundary of a \( (d-1) \)-dimensional polyhedron consists of \( (d-2) \)-dimensional polyhedra called \( (d-2) \)-faces, and so on. Note that \( (d-1) \)-faces are also called facets, 1-faces are called edges and 0-faces are called vertices. A convex polyhedron is often called a polytope.

Definition 2.1.5 A graph \( G = (V, E) \) (vertex set \( V \), edge set \( E \)) is planar if it can be embedded in the plane without edge crossings [Preparata and Shamos 85].

Definition 2.1.6 A straight-line planar embedding of a planar graph determines a partitioning of the plane called a planar subdivision [Preparata and Shamos 85].

Definition 2.1.7 A planar subdivision is a triangulation if all its bounded regions are triangles [Preparata and Shamos 85].

A triangulation of a finite set of points \( S \) is a planar graph with maximum number of edges and with nodes located at the points of \( S \) [Preparata and Shamos 85].
2.2 Definition and Properties of the Voronoi Diagram

The Voronoi diagram for a set of points in Euclidean space is one of the fundamental data structures of computational geometry and its properties have been studied extensively [de Berg et. al. 97, Boissonnat and Yvinec 98, Okabe et. al. 92, Preparata and Shamos 85]. The generalization of the Voronoi diagram, called the generalized Voronoi diagram, has been studied to a lesser extent.

The Voronoi diagram of a set of points \( S \) can be informally defined as a division of the space according to the nearest-neighbor rule, where each point from \( S \) is associated with a region of the Euclidean space closest to a given point from \( S \). Such diagrams were named after the Russian scientist Voronoi, who lived at the beginning of this century and who first seriously investigated the properties of a planar diagram defined according to this rule as part of his investigations in number theory [Voronoi 1908].

Another mathematician, Dirichlet, had mentioned the diagram earlier [Dirichlet 1850]. Accordingly the Voronoi diagram is also called a Dirichlet tessellation. There are other names for the Voronoi diagram, such as Thiessen polygons, which are rarely used in the literature [Okabe et. al. 92]. Shamos and Hoey were the first to conduct thorough studies of the planar Voronoi diagrams for sets of points [Shamos and Hoey 75]. They also developed a worst-case optimal method for the Voronoi diagram construction. As the varieties of VD applications and its importance to various fields became recognized, deeper studies of the VD properties were conducted.

A definition of the Voronoi diagram in \( d \) -dimensional Euclidean space \( E^d \) can be given as follows:

**Definition 2.2.1** For a finite set \( S \subseteq E^d \), a Voronoi diagram (VD) of \( S \) associates to each \( p \in S \) a Voronoi region \( Vor(p) \) such that

\[
Vor(p) = \{ x \in E^d \mid d(x, p) \leq d(x, q), \forall q \in S - \{p\} \},
\]
where $d(\cdot,\cdot)$ denotes the Euclidean distance function [Okabe et. al. 92].

Note. Strictly speaking, according to this definition $VD$ is not a partitioning of space since the Voronoi regions are closed sets (they include their boundaries) and, therefore, they can share facets. A more strict approach that defines different elements of $VD$ (vertices, edges and faces) separately could be used instead, however this would lead to a more complex definition.

An example of a Voronoi diagram for a set of points in the plane is given in Figure 2.2.1.

Consider the bisector $B(p, q) = \{ \mathbf{x} | d(p, \mathbf{x}) = d(q, \mathbf{x}) \}$ between two sites $p, q \in S$ of a Voronoi diagram in $E^d$. This bisector is a line perpendicular to the line segment $pq$ that bisects this segment in the Euclidean metric in 2D. It is the plane (hyperplane) perpendicular to the segment $\overline{pq}$ that bisects this segment in 3D ($d$ dimensions). The bisector divides the space into two halfspaces. Denote the halfspace that contains the point $p$ as $H(p,q) = \{ \mathbf{x} | d(p, \mathbf{x}) \leq d(q, \mathbf{x}) \}$. By definition this halfspace also contains the bisector $B(p,q)$ [Okabe et. al. 92].

An equivalent definition of the Voronoi diagram can be given:

---

1 Note that although the letter $d$ both describes a $d$-dimensional space (as in $R^d$) and a distance function of two arguments $d(\cdot,\cdot)$ (as in $d(p,q)$) it is clear from the context for each usage what is meant.
**Definition 2.2.2** For a finite set $S \in E^d$, a Voronoi diagram of $S$ is the collection of all Voronoi regions $\text{Vor}(p), p \in S$ such that $\text{Vor}(p) = \bigcap_{q \in S \setminus \{p\}} H(p, q)$ [Okabe et. al. 92].

A proof for the statement that Definition 2.2.1 and Definition 2.2.2 are equivalent is found in [Okabe et. al. 92].

A Voronoi region of a Voronoi diagram of $n$ sites is thus obtained as the intersection of $n-1$ half-spaces, and it is represented by a convex polyhedron, which can be unbounded. The boundary of a Voronoi region consists of up to $n-1$ facets, where each facet is a $d-1$ dimensional face; the boundary of the $d-1$ dimensional face consists of $d-2$ dimensional faces and so on. A 1-dimensional face is called a Voronoi edge, and a 0-dimensional face is called a Voronoi vertex. In the 2-dimensional case the Voronoi diagram consists of planar Voronoi regions, whose boundaries are straight-line segments or rays. Examples of bounded Voronoi region $\text{Vor}(p)$ and unbounded Voronoi region $\text{Vor}(q)$ in the plane are given in Figure 2.2.2.

![Figure 2.2.2 Bounded (a) and unbounded (b) Voronoi regions in the plane](image)

Most Voronoi diagram properties are proven for the planar case. A detailed list of these properties can be found in [Okabe et. al. 92]. Generalizations of these properties for $d$ dimensions are found in [Boissonnat and Yvinec 98].

First, note that the properties of the Voronoi diagram listed in this section are obtained under the non-cosphericity assumption for the set $S$ of $n$ points in $E^d$:

---

1 The proof is found in Chapter 2, Section 2.1 of the [Okabe et. al. 92].
Assumption 2.2.1 No $d + 2$ points of the set $S$ in $E^d$ are cospherical [Boissonnat and Yvinec 98].

One of the important properties of the Voronoi diagram in $E^d$ is now outlined (for more details see [Boissonnat and Yvinec 98]). A transformation $\phi$ of the sites of the Voronoi diagram in $E^d$ is performed so that the sites are lifted vertically onto a paraboloid $P$ in $E^{d+1}$. The set of hyperplanes tangent to the paraboloid $P$ at the points obtained by the transformation $\phi$ is constructed. The intersection of the $n$ halfspaces lying above the constructed hyperplanes is an unbounded polytope $V(S)$ that contains the paraboloid $P$. This unbounded polytope is called a Voronoi polytope $V(S)$.

Property 2.2.1 The Voronoi diagram can be obtained by projecting the faces of the Voronoi polytope $V(S)$ onto $E^d$ [Boissonnat and Yvinec 98].

The following properties of the Voronoi diagrams can be proven under Assumption 2.2.1:

Property 2.2.2 Each Voronoi region is a non-empty convex polyhedron in $E^d$ [Boissonnat and Yvinec 98].

Property 2.2.3 Each Voronoi vertex is the common intersection of exactly $d + 1$ Voronoi regions in $E^d$ [Boissonnat and Yvinec 98].

Property 2.2.4 (empty-sphere property) Each Voronoi vertex $v$ is the common point of $d + 1$ Voronoi regions $Vor(p_i)$, $i = 1..d + 1$. Then $v$ is a center of a sphere $C$ inscribed between the $d + 1$ distinct points $p_1, p_2, ..., p_{d+1}$ from $S$, such that there is no point $q \in S$ distinct from $p_1, p_2, ..., p_{d+1}$ inside $C$ [Boissonnat and Yvinec 98].

Property 2.2.5 (nearest-neighbor property) If $q \in S$ is the nearest-neighbor of $p \in S$ then Voronoi regions $Vor(p)$ and $Vor(q)$ have a common facet [Boissonnat and Yvinec 98].

Property 2.2.6 The Voronoi region $Vor(p)$ is unbounded if and only if the point $p \in S$ belongs to the convex hull of $S$ [Boissonnat and Yvinec 98].
The number of elements of the Voronoi diagram in the plane and in higher dimensions has been estimated as follows:

**Property 2.2.7** The number of Voronoi diagram vertices for a set of \( n \) sites in the plane is at most \( 2n - 5 \) and the number of edges is at most \( 3n - 6 \) when \( n \geq 3 \) [Preparata and Shamos 85].

**Property 2.2.8** The maximum number of Voronoi vertices for a set of \( n \) sites in \( E^d \) for \( n \geq d + 1 \) can be estimated\(^1\) as:

\[
v_{\text{max}} < 2 \binom{n-1-r}{r}, \text{ for even dimensions } d = 2r;
\]

\[
v_{\text{max}} < \frac{n}{n-r} 2 \binom{n-r}{r}, \text{ for odd dimensions } d = 2r - 1 \text{ [Klee 80].}
\]

The listed properties of the VD can be used to solve many mathematical problems, for example to construct the minimum spanning tree or relative neighborhood graph or to determine the nearest-neighbor [O'Rourke 94]. An extensive review of the numerous applications of VD's in archeology, biology, cartography, mechanics, geography, statistics and urban planning can be found in [Okabe et. al. 92]. An application to the simulation of granular-type materials can be found in [Chang 92], where the VD is used to represent the microstructure of the material.

### 2.3 Definition and Properties of the Delaunay Tessellation

Another fundamental computational geometry structure is the Delaunay tessellation (the Delaunay triangulation in the plane). The definition and properties of the planar Delaunay triangulation are first discussed. The definition of the \( d \)-dimensional Delaunay tessellation in Euclidean space is then introduced.

\(^1\) The formulas for the number of faces in higher-dimensional Voronoi diagrams can be found in [Seidel 91].
2.3.1 Definition and Properties of the Planar Delaunay Triangulation

The planar Delaunay triangulation is often considered under the non-collinearity assumption:

**Assumption 2.3.1** No three points from the set $S$ lie on the same line in the plane [Delaunay 34].

In 1934, Delaunay proved an important theorem that introduced the Delaunay triangulation:

**Theorem 2.3.1** The straight-line dual of the Voronoi diagram for a set of sites $S$ in $E^2$ is a triangulation of $S$ [Delaunay 34].

**Definition 2.3.1** A *Delaunay triangulation* (DT) is the straight-line dual of the Voronoi diagram obtained by joining all pairs of sites whose Voronoi regions share a common Voronoi edge [Delaunay 34].

The Delaunay triangulation corresponding to the Voronoi diagram from Figure 2.2.1 is shown in Figure 2.3.1.

![Figure 2.3.1 The Delaunay triangulation in the plane](image)

The DT has been extensively studied in the literature and following is a list of important properties of the DT obtained under Assumption 2.3.1.
Property 2.3.1 For the Voronoi diagram of \( n \) points on the plane and its dual Delaunay triangulation the following relationships are true when \( n \geq 3 \):

\[
\begin{align*}
v_{DT} &= n, \\
e_{DT} &= e_{VD} \leq 3n - 6, \\
f_{DT} &= v_{VD} \leq 2n - 5,
\end{align*}
\]

where \( v_{VD} \) and \( v_{DT} \) represent the number of vertices in \( VD \) and \( DT \), correspondingly, \( e_{VD} \) and \( e_{DT} \) represent the number of edges in \( VD \) and \( DT \) and \( f_{DT} \) is the number of triangles in \( DT \) [Lawson 77].

Property 2.3.2 If the number of vertices on the boundary of the convex hull of \( S \) is equal to \( B \) then the following equations are true:

\[
\begin{align*}
e_{DT} &= 3n - B - 3, \\
f_{DT} &= 2n - B - 2 \quad [\text{Yvinec 88}].
\end{align*}
\]

Property 2.3.3 (empty-circle property) The circumcircle of any Delaunay triangle does not contain any points of \( S \) in its interior [Lawson 77].

Property 2.3.4 If each triangle of a triangulation of the convex hull of \( S \) satisfies the empty circle property, then this triangulation is the Delaunay triangulation of \( S \) [Lawson 77].

Another important property is the min-max property [Lawson 77]. Consider a convex quadrilateral \( Q_t \) constructed on the points \( a, b, c \) and \( d \) in the plane. It can be triangulated by constructing either edge \( \overline{ac} \) or edge \( \overline{bd} \). Thus two different triangulations are possible (see Figure 2.3.2). Denote the six angles in the first triangulation as \( \alpha_i, i = 1 \ldots 6 \) and the six angles in the second triangulation as \( \beta_i, i = 1 \ldots 6 \). It is said that the edge \( \overline{ac} \) satisfies the local min-max criterion if the following equation holds:
\[
\min_i (\alpha_i) \geq \min_i (\beta_i) \text{ [Lawson 77].}
\]
In the example presented in Figure 2.3.2 the edge \( \overline{ac} \) satisfies the local \( \min \)-\( \max \) criterion.

![Figure 2.3.2 Two possible triangulations of a quadrilateral](image)

A triangulation satisfies the **global \( \min \)-\( \max \) criterion** if every internal edge of a convex quadrilateral in the triangulation satisfies the local \( \min \)-\( \max \) criterion [Lawson 77].

**Property 2.3.5** The Delaunay triangulation satisfies the global \( \min \)-\( \max \) criterion [Lawson 77].

**Property 2.3.6** If a triangulation of the convex hull of \( S \) satisfies the global \( \min \)-\( \max \) criterion then it is the Delaunay triangulation of \( S \) [Lawson 77].

Some other properties of the Delaunay triangulation are that the DT selects the shortest diagonal in the quadrilateral and satisfies the **equiangularity property** that states that the triangulation maximizes lexicographically the increasing sequence of angles that appear in the triangles of a triangulation [Edelsbrunner 87]. DT also minimizes the radius of the maximum circumscribed circle of all circumscribed circles constructed for all triangles of the DT [Dazevo and Simpson 89].

A **swap operation** can be introduced on a triangulation. Consider the convex quadrilateral \( Q_1 \) with internal edge \( \overline{ac} \). If the edge \( \overline{ac} \) is deleted and internal edge \( \overline{bd} \) is constructed, a different triangulation is obtained. This operation is called a **swap operation** or a **flip operation** (see Figure 2.3.2) [Lambert 93].

Now, the **swapping procedure** can be defined on a triangulation [Lambert 93]. Let \( Q \) be a set of quadrilaterals in a triangulation.
Swapping procedure:

1. If all of the quadrilaterals in $Q$ satisfy some given local optimization criterion, stop.

2. Choose a quadrilateral that does not satisfy a local optimization criterion, perform a swap operation on this quadrilateral and go to 1.

Given some flip rule (called local optimization criterion), the quadrilateral that satisfies this local optimization criterion is called optimally triangulated. A locally optimal triangulation is one where each quadrilateral formed by adjacent triangles is optimally triangulated. It can be constructed by the flip algorithm, which repeatedly flips the diagonals of non-optimal quadrilaterals [Lambert 93]. Any planar triangulation of the interior of the convex hull can be swapped to a triangulation where every quadrilateral satisfies some local optimization criterion in $O(n^2)$ time and the swaps can be performed in any order [Lambert 94].

Local optimization criteria for DT construction are often discussed in GIS-systems, terrain modeling, cartography, surface modeling, computer graphics and computer-aided geometric design [Okabe et. al. 92].

2.3.2 Definition and Properties of the $d$-dimensional Delaunay Tessellation

The diagram dual to the Voronoi diagram in $d$-dimensional space when $d \geq 3$ is called a Delaunay tessellation [Okabe et. al. 92].

The Delaunay tessellation is often considered under the non-collinearity assumption:

**Assumption 2.3.2** No $d+1$ points from the set $S$ lie on the same hyperplane in $E^d$ [Okabe et. al. 92].

The Delaunay tessellation corresponding to the Voronoi diagram for a set of points $S$ can be defined as following:

**Definition 2.3.2** A *Delaunay tessellation* corresponding to the Voronoi diagram for a set of points $S$ in $E^d$, $d \geq 3$ is a collection of $d$-dimensional simplices such that for each
Voronoi vertex \( v = \text{Vor}(p_1) \cap \text{Vor}(p_2) \cap \ldots \cap \text{Vor}(p_{d+1}) \) there is a simplex \((p_1, p_2, \ldots, p_{d+1})\) in the Delaunay tessellation [Okabe et. al. 92].

**Property 2.3.7** (empty-sphere property) A sphere circumscribed to any simplex in the Delaunay tessellation contains no point from \( S \) in its interior [Lawson 86].

**Property 2.3.8** A tessellation where every simplex satisfies the empty-sphere property is a Delaunay tessellation [Boissonnat and Yvinec 98].

Some properties of the planar Delaunay triangulation have been extended to \( d \) dimensions in [Rajan 91]. The generalization of planar equiangularity property to \( d \) dimensions has been studied in [Schmitt and Spehner 93]. Some additional properties of the Delaunay tessellation can be found in [Boissonnat and Yvinec 98].

### 2.4 Methods for the Voronoi Diagram and the Delaunay Tessellation Construction

Methods for the construction of Voronoi diagrams and Delaunay tessellations are now classified. These methods are usually described in terms of either the Voronoi diagram or the Delaunay tessellation (triangulation). Note that since the Voronoi diagram and the Delaunay tessellation are dual to each other, each can be transformed into the other. Therefore, it is sufficient to construct either of those diagrams to obtain the other [Okabe et. al. 92]. Also, note that most of the algorithms are developed for the planar Voronoi diagram and the Delaunay triangulation.

The deterministic algorithms that will be discussed are as follows:

- Straightforward method
- Incremental method
- Edge flipping method
- Divide-and-conquer method
- Sweep-plane method
- Convex hull method
A comparison of the Delaunay triangulation algorithms based on the above techniques can be found in [Su and Drysdale 95]. An extensive review of Voronoi diagram methods is given in [Okabe et. al. 92]. Recently, randomized algorithms became the focus of extensive investigation. Randomization yields simplicity and efficiency of the algorithm at the cost of losing determinism. An extensive review of randomized algorithms for the construction of VD's can be found in the book by Mulmuley [Mulmuley 94]. This research is restricted to studying deterministic algorithms, however.

2.4.1 The Worst-Case Time Complexity for the VD Construction

The worst-case time complexity for the construction of the VD and the DT of \( n \) points in the plane is \( \Omega(n \log n) \) according to [Preparata and Shamos 85]. The worst-case time complexity for construction of the \( d \)-dimensional Voronoi diagram and the Delaunay tessellation of \( n \) points is \( \Omega(n^{\lceil d/2 \rceil}) \) [Klee 80, Seidel 87].

2.4.2 The Straightforward Method

A straightforward method for the Voronoi diagram construction is based on Definition 2.2.2 of the VD. It works by intersecting \( n - 1 \) halfplanes for each of the \( n \) points in the input and reporting the resulting Voronoi regions as output. This algorithm has \( \Theta(n^3) \)

\(^1\) The complexity of an algorithm concerns number of operations used by an algorithm on any instance of the problem during algorithm execution. It is normally not calculated explicitly, only estimates are given using a common notation summarized in [Preparata and Shamos 85]. In the following it is assumed that operation counts are made for operations of comparison of real numbers and that the number of points (sites) of the input set \( S \) is \( n \). Expressions time complexity of the algorithm or an algorithm runs in time will be sometimes used. They refer to asymptotically worst-case time complexities of the algorithm.
time complexity. The straightforward method has the worst time efficiency among all of the methods for VD construction typically considered [Preparata and Shamos 85].

Another straightforward method described in [Lawson 77] constructs the Voronoi vertices and edges one by one in an order in which a traveler walks along the edges of the diagram. The paper did not consider the time complexity for the method.

2.4.3 The Incremental Method

The *incremental method* is a simple but powerful method for the Voronoi diagram construction. It has been one of the most popular methods for studying the diagram and for its practical implementation. The method starts with a simple Voronoi diagram for three sites, and modifies the diagram by adding one new site at a time. In the worst case, the addition of a site requires time proportional to the number of sites added so far, and consequently the worst-case time complexity for the incremental method is $\Theta(n^2)$ (see, for example, [Green and Sibson 77]).

Among the incremental algorithms for the Delaunay triangulation construction, methods based on incremental construction and on incremental search can be distinguished.

The *incremental construction* algorithms add sites to the diagram one by one, maintaining the Delaunay triangulation [Guibas 92]. They first locate the triangle containing the new point by applying the *CCW (Counter Clock Wise)* test. Then they update the diagram by performing diagonal swaps on edges, based on the results of the *INCIRCLE* test.

The *incremental search* algorithms [Dwyer 91, Li and Milenkovic 90] grow the diagram one valid triangle at a time, starting with one Delaunay triangle and then performing a diagonal swap of an edge, if necessary, when a new triangle has to be considered. The worst-case time complexity of these methods is $\Theta(n^2)$.

A method slightly different from the incremental method for the Delaunay triangulation was proposed by Watson [Watson 81]. The author does not keep track of triangle adjacencies and simply deletes all triangles whose circumcircle contains the new site and
then re-triangulates the resulting star-shaped polygon. The worst-case time complexity of this method is \( \Theta(n^2) \).

The incremental method for the Delaunay triangulation in the plane was successfully implemented by Lischinski [Lischinski 94]. The incremental Voronoi diagram construction in 3-dimensional space was performed by Inagaki [Inagaki et. al. 92].

2.4.4 The Edge Flipping Method

The so-called edge flipping method is often considered together with the incremental method for the Delaunay triangulation construction. It was introduced by Sibson in 1978 for triangulating the convex hull [Sibson 78]. The author showed that any convex hull triangulation can be converted to the Delaunay triangulation by flipping diagonals in the quadrilateral according to the min-max criterion in \( \Theta(n^2) \) time in the plane. Lawson showed that application of min-max or empty circle criteria in the plane are equivalent for DT construction [Lawson 77]. Joe presented a method for constructing the 3D Delaunay tesselation using a local transformation technique [Joe 89, Joe 91]. The algorithm runs in optimal worst-case time \( \Theta(n^2) \). Edelsbrunner and Shah studied the flipping technique to construct Delaunay tesselation in \( d \) dimensions [Edelsbrunner and Shah 96]. The worst-case time complexity of the algorithm is \( O(n^{\lceil(d+1)/2\rceil}) \) for \( d \geq 3 \).

2.4.5 The Divide-and-Conquer Method

Another popular method for the Voronoi diagram construction uses the divide-and-conquer paradigm. The set of Voronoi sites is recursively divided into smaller subsets, and the Voronoi diagrams for these are constructed recursively. Then the obtained Voronoi diagrams are merged into the resulting diagram. Since the merge step can be performed in \( \Theta(n) \) time, the resulting complexity of the algorithm is \( \Theta(n \log n) \). Two basic geometric primitives, the CCW (Counter Clock Wise) orientation test and the INCIRCLE test, are used for the diagram construction.
The method was first proposed for the Voronoi diagram construction by Shamos and Hoey [Shamos and Hoey 75], and then for the Delaunay triangulation by Guibas and Stolfi [Guibas and Stolfi 85]. The algorithm runs in optimal worst-case time $\theta(n \log n)$.

A discussion of problems in the generalization of divide-and-conquer methods to higher dimensions is presented in [Yvinec 88]. An investigation of the Delaunay tessellation properties in connection with the divide-and-conquer technique in $d$ dimensions can be found in [Hazelwood 88].

2.4.6 The Sweep-Plane Method

In 1987, Fortune suggested to use the sweep-plane technique to construct the Voronoi diagram in the plane [Fortune 87]. A special line called the sweepline moves from left to right and sweeps the sites one by one. Every time a site is swept, a part of the resulting Voronoi diagram is constructed. The worst-case time complexity of the method is $\theta(n \log n)$. This method has become one of the popular methods for the planar Voronoi diagram construction. It was used to find a closest pair among convex planar objects [Bartling and Hinrichs 92], to solve the all-nearest-neighbors problem [Graf and Hinrichs 93] and to construct the minimum enclosing circle [Dehne and Klein 87].

2.4.7 The Convex Hull Method

This method is used for the $d$-dimensional Voronoi diagram construction. It utilizes the relationship between Voronoi diagrams in $E^d$ and convex hulls in $E^{d+1}$ (see Property 2.2.1, Section 2.2 of this thesis for more details).

The method transforms the sites of a Voronoi diagram into certain points in the higher-dimensional space. Their convex hull is generated and transformed back to the original space to obtain the Voronoi diagram. The algorithm that constructs the planar Voronoi diagram from the convex hull in 3D was first presented in [Brown 79]. The relationship between Voronoi diagrams in $E^d$ and arrangements of hyperplanes were discussed in [Edelsbrunner and Seidel 86].
The problem of constructing convex hull in $d$ dimensions was extensively studied by Seidel [Seidel 81, Seidel 87 and Seidel 97]. An average-case analysis\(^1\) of algorithms for convex hulls and Voronoi diagrams was performed by Dwyer [Dwyer 88, Dwyer 91, Dwyer 93]. An algorithm for the convex hull construction in $d$ dimensions which runs in time proportional to the size of the resulting convex hull was developed in [Chan et. al. 95].

The space complexity of the convex hull of $n$ points in $d$-dimensional space is $\Theta(n^{\lfloor d/2 \rfloor})$ for $d \geq 3$ [Edelsbrunner 87]. Chazelle was first to develop a worst-case optimal $\Theta(n^{\lfloor d/2 \rfloor})$ algorithm to construct convex hull in higher dimensions [Chazelle 93]. This allows worst-case optimal $\Theta(n^{\lceil d/2 \rceil})$ algorithm for the construction of VD in $d$ dimensions [de Berg et. al. 97]. The implementation of the convex hull method for 3 and 4 dimensions and the construction of the corresponding Voronoi diagram in 2 and 3 dimensions can be found, for example, in [Barber et. al. 96].

### 2.5 The Generalized Voronoi Diagram

As the variety of applications of the Voronoi diagram was recognized, it became clear that many practical situations required modification of the original diagram. Such fields as biology, gas dynamics, chemistry, nuclear physics and mechanics often consider the behavior of 3-dimensional objects. In these applications a sphere is widely used as a simple and effective model for a variety of objects. Therefore Voronoi diagrams for circles or spheres can be studied and used as efficient data structures for such applied problems [Okabe et. al. 92]. Some other computational geometry problems, such as packing and covering spheres, illuminating balls and finding the intersection, union, and closest-neighbors for sets of disks and spheres are also solved with the use of generalized VD [Edelsbrunner 86, Imai et. al. 85, Okabe et. al. 92, Sharir 85].

\(^1\) See [Dwyer 88] for more information on average-case analysis.
The studies in these directions have been performed since the early 70's when the concept of an ordinary Voronoi diagram was extended and generalized. The definition of the Voronoi diagram allows easy generalization. The concept of the Voronoi diagram can be extended not only to higher dimensions but also to differently shaped objects and to different metrics. The next part of this thesis discusses the properties of the generalized Voronoi diagram and presents algorithms for its construction for some of the generalizations.

The generalized Voronoi diagram can be informally defined as a partitioning of space into regions, each of which is the locus of points of $R^d$ not further from a given object $P \in S$ than from any other object $Q \in S, Q \neq P$.

The more formal definition can be given as follows:

Consider a set of objects $S$ in $R^d$, where $|S| = n$.

**Definition 2.5.1** A generalized Voronoi diagram (GVD) for a set of objects $S$ in $R^d$ is the collection of generalized Voronoi regions

$$GVor(P) = \{x \mid d(x, P) \leq d(x, Q), \forall Q \in S - \{P\} \},$$

where $d(x, P)$ is the distance function in $R^d$ between a point $x \in R^d$ and an object $P \in S$ [Okabe et. al. 92].

As in the case of ordinary Voronoi diagram, an equivalent definition of the generalized Voronoi diagram can be given [Okabe et. al. 92]. The generalized bisector $B(P, Q) = \{x \mid d(x, P) = d(x, Q)\}$ is the geometrical locus of points that are equidistant to two distinct objects $P$ and $Q$ from the set $S$. Note that the form of the bisector depends of the distance function $d(x, P)$ being used. The generalized bisector divides the space into two halfspaces. Denote the halfspace that contains the object $P$ as $H(P, Q) = \{x \mid d(P, x) \leq d(Q, x)\}$. By definition this halfspace also contains the generalized bisector $B(P, Q)$. 
Definition 2.5.2 A generalized Voronoi diagram for a set of objects \( S \) is the collection of generalized Voronoi regions \( GVor(P), P \in S \) where

\[
GVor(P) = \bigcap_{Q \in S \setminus \{P\}} H(P, Q) \quad \text{[Okabe et. al. 92].}
\]

Thus, a generalized Voronoi region of a generalized Voronoi diagram of \( n \) sites is obtained as the intersection of \( n-1 \) half-spaces. The boundary of the generalized Voronoi region consists of up to \( n-1 \) facets, where each facet is a \( d-1 \) dimensional face; the boundary of the \( d-1 \) dimensional face consists of \( d-2 \) dimensional faces and so on. A 1-dimensional face is called a generalized Voronoi edge, and a 0-dimensional face is called a generalized Voronoi vertex.

One of the generalizations can be obtained by modifying the rules according to which the Voronoi region can be computed. Thus, the Voronoi diagram for circles or spheres can be introduced.

The concept of weighting a site \( P \) can be introduced so that each site \( P \in S \) is assigned a real number \( w(P) \), the so-called weight of \( P \). The distance from \( P \) to a point \( x \in R^d \) is then measured as a function of the weight \( w(P) \) and distance \( d(x, p) \), where point \( p \) is a fixed reference point, which belongs to \( P \). If \( P \) is a sphere, then its center is usually selected as the reference point \( p \).

There are four general formulas often considered in the literature for weighted distance computations [Okabe et. al. 92]. They are:

- multiplicatively weighted distance function \( d_m(x, P) = \frac{1}{w(P)} d(x, p) \);
- additively weighted distance function \( d_a(x, P) = d(x, p) - w(P) \);
- compoundly weighted distance function \( d_c(x, P) = \frac{1}{w_1(P)} d(x, p) - w_2(P) \)
power distance function\(^1\) \(d_p(x, P) = \text{pow}(x, P) = d(x, p)^2 - w(P)\).

The Voronoi diagrams obtained by using these distance functions are called \textit{multiplicatively weighted VD}, \textit{additively weighted VD}, \textit{compoundly weighted VD} and \textit{power diagram}, respectively. The power diagram is sometimes called the \textit{Laguerre} diagram in the literature. It can be also referred to as the Voronoi diagram in the \textit{power metric} [Boissonnat and Yvinec 98, Okabe et. al. 92].

The additively weighted VD and the power diagram can be interpreted as a Voronoi diagram for circles and spheres. These diagrams are often considered in the literature, mainly because they possess important properties that make the algorithms for their construction and modification easy to develop and apply to particular problems [Ash and Bolker 86, Aurenhammer 86, Lee and Wong 80].

Other generalizations can be obtained by modifying the rules to compute the distance \(d(x, p)\) between two points \(x, p \in R^d\) thus obtaining the Voronoi diagram in different metrics. In particular, the distance \(d(x, p)\) between points \(x(x_1, x_2, ..., x_d)\) and \(p(p_1, p_2, ..., p_d)\) is computed as \(d(x, p) = \left(\sum_{i=1}^{d}|x_i - p_i|^p\right)^{1/p}\) in the Minkowski metric \(L_p\).

This distance is computed as \(d(x, p) = \sum_{i=1}^{d}|x_i - p_i|\) in the \textit{Manhattan} (\(L_1\)) metric, as \(d(x, p) = \sqrt{\sum_{i=1}^{d}(x_i - p_i)^2}\) in the \textit{Euclidean} metric and as \(d(x, p) = \max_{i=1, d}|x_i - p_i|\) in the \textit{supremum} (\(L_\infty\)) metric.

Generalizations of the additively weighted Voronoi diagram in Manhattan, supremum and Euclidean metrics will be considered in detail in the following chapters.

\(^1\)The power function was first mentioned as a generalized distance function by Laguerre and Voronoi at the beginning of the century [Voronoi 1908].
2.6 The Generalized Delaunay Tessellation

The definition of a generalized Delaunay triangulation in the plane can be given as:

**Definition 2.6.1** A *generalized Delaunay triangulation* (GDT) is the dual of the generalized Voronoi diagram obtained by joining all pairs of sites whose generalized Voronoi regions share a common generalized Voronoi edge [Okabe et. al. 92].

A definition of a generalized Delaunay tessellation can be given similarly to the definition of a Delaunay tessellation given in [Okabe et. al. 92] (see section 2.3.2).

**Definition 2.6.2** A *generalized Delaunay tessellation* corresponding to the generalized Voronoi diagram for a set of objects $S$ in $\mathbb{R}^d$, $d \geq 3$ is a collection of $d$-dimensional simplices such that for each generalized Voronoi vertex

$$v = \text{Vor}(p_1) \cap \text{Vor}(p_2) \cap \ldots \cap \text{Vor}(p_{d+1})$$

there is a simplex $(p_1, p_2, \ldots, p_{d+1})$ in the generalized Delaunay tessellation [Okabe et. al. 92].

2.7 Methods for Construction of the Generalized Voronoi Diagram

The methods for generalized Voronoi diagram construction are similar to methods for ordinary Voronoi diagram construction. They are:

- Incremental method
- Divide-and-conquer method
- Sweep-plane method
- Convex hull method

Among all of the above generalizations of the Voronoi diagram, the power diagram is the best known. This can be explained by the simplicity of the bisector in the power metric and by the similarities with the Voronoi diagram. The largest number of references in the following section is devoted to power diagram algorithms.
2.7.1 The Worst-Case Time Complexity for the Construction of the Generalized VD

The worst-case time complexity for the construction of the power diagram and additively weighted Voronoi diagram of \( n \) objects is \( \Omega(n \log n) \) in the plane and \( \Omega(n^{\lceil d/2 \rceil}) \) in \( d \) dimensions [Aurenhammer 91, Sharir 95, Boissonnat and Yvinec 98].

2.7.2 The Incremental Method

Aurenhammer and Edelsbrunner suggested an incremental insertion method for power diagram construction that runs in \( \theta(n \log n) \) time in the plane [Aurenhammer and Edelsbrunner 84]. The incremental method was also successfully employed to construct planar Voronoi diagrams for line segments [Kokubo 85], polygons [Srinivasan and Nackman 87] and Voronoi diagrams in 3-dimensional space [Inagaki et. al. 92] and constrained Delaunay triangulations [Inagaki and Sugihara 94]. Schaudt and Drysdale suggested a generalized scheme of the incremental algorithm to compute the Manhattan and the supremum Delaunay triangulations in \( \theta(n^{\lceil (d+1)/2 \rceil}) \) time for \( d \geq 3 \). [Schaudt and Drysdale 92]. Edelsbrunner and Shah suggested incremental flipping algorithm to construct the power Delaunay tessellation in \( d \) dimensions [Edelsbrunner and Shah 96]. The worst-case time complexity of this algorithm is \( \theta(n^{\lceil (d+1)/2 \rceil}) \) for \( d \geq 3 \). Finally, Choset suggested an extension of the incremental technique for VD for arbitrary shaped objects in \( d \)-dimensional Euclidean space [Choset 97].

2.7.3 The Divide-and-Conquer Method

A number of algorithms for constructing power diagrams for circles in the plane are based on the divide-and-conquer technique. Lee and Drysdale were among the first who investigated the construction of Voronoi diagrams for circles in the Euclidean plane. They suggested an \( \theta(n(\log n)^2) \) algorithm that worked under the assumption that the circles do not intersect [Lee and Drysdale 81]. Sharir [Sharir 85] presented an
\( \theta(n(\log n)^2) \) algorithm for construction of the generalized Voronoi diagram for a set of disks, polygons and line segments in the Euclidean metric in the plane.

Imai, Iri and Murota [Imai et. al. 85] developed an optimal worst-case time \( \theta(n \log n) \) algorithm for planar power diagram construction. Algorithms based on the divide-and-conquer technique for constructing Voronoi diagrams for point sets in the Manhattan and the supremum metric have also been developed in [Klein and Wood 88, Lee and Wong 80]. Drysdale presented a divide-and-conquer algorithm for convex distance functions\(^1\) [Drysdale 90].

Klein provided a divide-and-conquer algorithm to compute the abstract Voronoi diagram. It works in \( \theta(n(\log n)^2) \) time under the assumption that any set \( S \) can be split into two approximately equal subsets in \( O(n) \) time and that the bisecting curve is acyclic (see [Klein 89] for more details). Mehlhorn [Mehlhorn et. al. 90] extended the algorithm to work without the requirement of acyclicility of the bisector.

### 2.7.4 The Sweep-Plane Method

The introduction of the sweep-plane algorithm for constructing the Voronoi diagrams for sets of points by Fortune provided a new approach for construction of power diagrams for circles [Fortune 87]. Rosenberger [Rosenberger 91] modified Fortune’s sweep-plane algorithm to construct Voronoi diagrams for additively weighted sites (circles) in Euclidean space. Shute [Shute et. al. 91] suggested a sweep-plane algorithm for the \( L_1 \) and \( L_\infty \) Delaunay triangulations of point sites and Skyum [Skyum 91] studied the application of the sweep-plane algorithm for the generalized Delaunay triangulation. Held discusses the sweep-plane method for planar areas bounded by straight-lines and circular arcs [Held 94]. McAllister [McAllister et. al. 93, McAllister et. al. 96] studied the

\(^1\) See [Drysdale 90] for definition of convex distance function.
sweep-plane algorithm for any polygonal distance function\(^1\) and for sites that are disjoint convex polygons. These methods have worst-case optimal time complexity \(O(n \log n)\).

### 2.7.5 The Convex Hull Method

Aurenhammer adopted an approach based on the duality of convex hulls and Voronoi diagrams of all orders in higher dimensions. He proved that the power diagram in \(R^d\) can be constructed via convex hulls\(^2\) in \(R^{d+1}\) [Aurenhammer 87, Aurenhammer 91]\(^3\). This relationship has also been studied in [Devillers et. al. 92]. This transformation was extended for the additively weighted Voronoi diagram construction in [Boissonnat and Yvinec 98]. These methods construct the generalized Voronoi diagram in \(O(n^{\lceil d/2 \rceil})\) time in the worst case [Boissonnat and Yvinec 98].

In the following chapters some properties of Voronoi diagrams and Delaunay tessellations in the weighted Euclidean, Manhattan, supremum and power metrics in 2 and \(d\)-dimensional space will be presented. New algorithms to compute generalized Voronoi diagrams in different metrics and in 2 and \(d\) dimensions will be described. Some general properties of the set of all tessellations in 2-dimensional space will be also considered.

---

\(^1\) See [McAllister et. al. 96] for definition of a polygonal distance function.

\(^2\) The method is similar to one described in the Preliminaries, Section 2.4.7.

\(^3\) A relationship between power and additively weighted Voronoi diagrams in higher dimensions was also established. See [Aurenhammer 87, Aurenhammer 91] for more details.
CHAPTER 3: PROPERTIES OF THE GENERALIZED VORONOI DIAGRAM

The goal of this chapter is to generalize the properties of the Voronoi diagram and the Delaunay tessellation to different metrics in $d$-dimensional space and thus to demonstrate the similarities between the weighted Voronoi diagrams in $R^d$. So far, these generalizations do not seem to have been developed in the literature surveyed. The properties established for the weighted VD are generalizations of properties of the ordinary Voronoi diagram for a set of points in $d$ dimensions. The proofs of these properties constitute the main contribution of this chapter.

The diagrams are considered for the weighted Euclidean, Manhattan, supremum and power metrics in $d$-dimensional space under the assumption that the spheres of the original set $S$ do not intersect. This is a reasonable assumption in the practical applications considered, such as computer simulation of granular-type materials and biological systems of growing plants.

There are many other fields where the generalized Voronoi diagrams of spherical non-intersecting objects can be applied. For example, power diagrams are used in metallurgy [Connoily 83], engineering [Ghosh and Mukhopadhyay 91], chemistry [Lieb 82] and biology [Richards 77]. Additively weighted Voronoi diagrams are employed in urban planning and natural sciences [Okabe et. al. 92].

The main contribution of this chapter is in the extension of the properties of the ordinary Voronoi diagram to weighted Voronoi diagrams for the power, Euclidean and Manhattan (supremum) metrics in $R^d$. Important results include the proof of the following properties of the weighted generalized VD in $d$ dimensions:

- the property that states that the generalized Voronoi vertex belongs to the intersection of $d+1$ Voronoi regions;
- the definitions of the inscribed sphere and the empty-sphere property for different metrics;
• the definitions of the nearest-neighbor for different metrics and the proofs of the nearest-neighbor properties;

• the definitions of the convex hull for different metrics and the condition for establishing whether a Voronoi region is bounded or unbounded.

The classification of these properties along with their proofs was not previously published for any generalized Voronoi diagram in $d$ dimensions. These results will be used in the rest of the thesis in the development of algorithms for the generalized VD and DT construction.

### 3.1 The Power Diagram

The properties of the weighted Voronoi diagram in the power metric are studied in this section. First, the definitions and known properties of the power diagram are summarized. Next, new properties of the power diagram for a set of spheres in $d$ dimensions are introduced with their proofs.

The important results presented in this chapter include a criterion for establishing whether the power region is bounded or unbounded; the property that a vertex $v$ of the power diagram is the common intersection of exactly $d+1$ power regions; the empty-sphere property and the nearest-neighbor property for the power diagram in $R^d$. The power Delaunay tessellation is discussed at the end of this section.

#### 3.1.1 Definition of the Power Diagram and Preliminaries

Consider a point $x = (x_1, x_2, ..., x_d)$ and a sphere $P$ with center $p = (p_1, p_2, ..., p_d)$ and radius $r_p$ in $R^d$. The power distance between a sphere and a point is defined as $d(x, P) = d(x, p)^2 - r_p^2$, where $d(x, p)$ is the Euclidean distance between points in $R^d$. 
that can be computed as $d(x, p) = \|x - p\| = \sqrt{\sum_{i=1}^{d} (x_i - p_i)^2}$ [Okabe et. al. 92]¹. The geometric interpretation of the power distance is that it is the square of the distance between a point $x$ and a sphere $P$ computed along a line tangent to the sphere $P$ (see Figure 3.3.1).

Figure 3.1.1 Geometrical interpretation of the power distance and the power bisector

A point $x$ lies inside the sphere $P$ or belongs to its boundary if and only if the power distance $d(x, P)$ is non-positive, i.e. negative or zero. The following notation for the sphere $P$ with center in point $p$ and radius $r_p$ will be used in the sequel: $P = \{p, r_p\}$.

The power diagram of a set of spheres $S$ is studied under the following assumption:

**Assumption 3.1.1** The spheres of the set $S$ do not intersect, i.e. $\|p - q\| \geq r_p + r_q$ for any two distinct spheres $P = \{p, r_p\}$ and $Q = \{q, r_q\}$ from $S$.

**Definition 3.1.1** A *power region* $PVor(P)$ of the sphere $P \in S$ in $R^d$ is the set of points $x \in R^d$ that are not further under the power metric from the sphere $P$ than from any other sphere $Q \in S$:

$$PVor(P) = \{x \in R^d | d(x, P) \leq d(x, Q), \forall Q \in S - \{P\}\}$$ [Okabe et. al. 92].

---

¹ The definitions and some properties referenced in this chapter can be found in Chapter 3, Section 3.1 of the book [Okabe et. al. 92].
A *power bisector* $B(P,Q)$ in $R^d$ is the geometrical locus of points from $R^d$ that are equidistant to two distinct sites $P$ and $Q$ in the power metric:

$$B(P,Q) = \{ x \mid d(x, P) = d(x, Q) \}.$$

The power bisector $B(P,Q)$ is a line perpendicular to the segment with endpoints in $P$ and $Q$ for circles $P$ and $Q$ in 2D (see Figure 3.3.1). In higher dimensions, the bisector is a hyperplane perpendicular to the segment $\overline{PQ}$. The power bisector divides the space into two halfspaces. A *halfspace* that contains the sphere $P$ is defined as $H(P,Q) = \{ x \mid d(x, P) \leq d(x, Q) \}$. By definition this halfspace also contains bisector $B(P,Q)$ and thus the bisector divides the space into two connected halfspaces $H(P,Q)$ and $H(Q,P)$.

An equivalent definition of a power region $PVor(P)$ can now be written as follows:

**Definition 3.1.2** A *power region* $PVor(P)$ of the sphere $P \in S$ in $R^d$ is the intersection of all halfspaces $H(P,Q)$:

$$PVor(P) = \bigcap_{Q \neq P} H(P,Q) \text{ [Okabe et. al. 92].}$$

**Definition 3.1.3** A *power diagram* $(PD)$ for the set of spheres $S$ in $R^d$ is the collection of all power regions $PD(S) = \{ PVor(P_1), \ldots, PVor(P_n) \}$ [Okabe et. al. 92].

Figure 3.1.2 shows the power diagram for a set of circles in the plane.

![Figure 3.1.2 The power diagram for a set of circles in the plane](image)
The known properties of the power diagram are now presented. The properties of the planar power diagram were first investigated in [Imai et. al. 85], where the form of the power bisector, power region and convex hull for the planar power diagram were introduced. The property that the power region can be limited or unlimited in $R^2$, when the centers of the circles from the set $S$ are on the boundary of the convex hull was proven. Empty planar power regions were considered and the property that the generator of power region might not be inside the region itself when generators intersect was established [Imai et. al. 85].

Aurenhammer extended some of these results to higher dimensions. He proved that any bisector in the power diagram is a hyperplane and that each region of the power diagram is always convex in $R^d$ [Aurenhammer 87]. He also established an important relationship between power diagrams in $R^d$ and convex hulls in $R^{d+1}$, similar to one described in [Brown 79] for the ordinary Voronoi diagrams. This relationship is also described in [Boissonnat and Yvinec 98]. The closest-pair property (which uses a different definition of the distance between two spheres than the one introduced in this chapter) for the power metric in $R^d$ was recently established in [Guibas and Zhang 98]. The empty-sphere property was established in [Edelsbrunner and Shah 96]. It was also proven that the tessellation satisfying the empty-sphere criterion is Delaunay tessellation in power metric [Edelsbrunner and Shah 96].

The relationships between power diagrams and power Delaunay tessellations have been studied in [Edelsbrunner and Shah 96]. It was proven that power Delaunay tessellation is

---

1 The centers of the circles from the set $S$ are sometimes referred to as generators of the power regions.

2 The case when a generator has an empty Voronoi region is not considered in this thesis.

3 See Preliminaries, Section 2.2 and Section 2.4 for more details.

4 See Chapter 18, Section 18.3 of the [Boissonnat and Yvinec 98].
a regular triangulation in $R^d$. It was also shown that the power Delaunay tessellation is a simplicial complex and a triangulation of $S$ (see [Edelsbrunner and Shah 96] for definitions). The proof is based on the relationship between power diagrams in $R^d$ and convex hulls in $R^{d+1}$ [Aurenhammer 91].

Other properties of the power diagram in $R^d$ were not considered in the literature surveyed. They are now proven here.

### 3.1.2 Properties of the Power Diagram in d Dimensions

This section presents the generalization of the Voronoi diagram properties to the power diagram in $d$ dimensions. Let $S$ be a set of spheres in $R^d$.

First, the equation of the power bisector in $R^d$ is given.

**Property 3.1.1** The bisector $B(P, Q)$ in the power metric is a hyperplane defined by the equation $(x, p - q) = \frac{1}{2} (\|p\|^2 - \|q\|^2 - r_p^2 + r_q^2)$ for sites $P$ and $Q$ where $P, Q \in S, P \neq Q$ and $x, p, q \in R^d$, where $(\cdot, \cdot)$ denotes a scalar product.

**Proof** The bisector in the power metric is defined by the equation

$$\|x - p\|^2 - r_p^2 = \|x - q\|^2 - r_q^2.$$ 

The bisector never intersects either sphere under the assumption of non-intersecting spheres. Thus the equation of the bisector can be written as

---

1 Note that the proof of these properties could be also carried out by using the relationship between power diagrams and convex hulls established in [Aurenhammer 87].

2 This trivial fact has been previously known: the equation for power bisector has been presented in [Aurenhammer 89] in a different form and in [Okabe et. al. 92]) for the planar case in the form similar to one given in Property 3.1.1.
\[(x-p, x-p)-r_p^2 = (x-q, x-q)-r_q^2.\]

Since the second-degree terms can be canceled this can be rewritten as

\[-2(x,p)+\|p\|^2 - r_p^2 = -2(x,q)+\|q\|^2 - r_q^2.\]

An equation for the bisector in \(d\)-dimensional space is therefore found as

\[(x,p-q) = \frac{1}{2} \left( \|p\|^2 - \|q\|^2 - r_p^2 + r_q^2 \right).\]

Let 2-dimensional vectors \(x = (x, y), p = (p_x, p_y)\) and \(q = (q_x, q_y)\) be given. Following Property 3.1.1, the equation of the power bisector in the plane can be written as

\[x(p_x - q_x) + y(p_y - q_y) = \frac{1}{2} \left( p_x^2 + p_y^2 - q_x^2 - q_y^2 - r_p^2 + r_q^2 \right).\]

**Property 3.1.2** Each power region \(P\text{Vor}(P)\) is a convex region in \(R^d\) space, containing the whole sphere \(P\).

**Proof** To prove that the sphere \(P\) lies entirely inside its power region \(P\text{Vor}(P)\), first note that the distance between the sphere \(P\) and any point \(x\) that belongs to the interior of the sphere or to its boundary is zero or negative. Furthermore, for any point lying outside the sphere the distance is positive. Therefore, any point \(x\) lying on or inside the sphere \(P\) is closer to \(P\) than to any other sphere \(Q\), since \(x\) cannot belong to both \(P\) and \(Q\) under Assumption 3.1.1. Consequently, the sphere \(P\) lies completely inside its power region \(P\text{Vor}(P)\).
To prove that the power region is convex\(^1\), recall that the power region \(PVor(P)\) can be defined as the intersection of halfspaces \(H(P,Q)\). Since each of the halfspaces is a convex region relative to \(P\), their intersection is also a convex region.

\[\begin{align*}
\end{align*}\]

\textit{Note.} Since the power region \(PVor(P)\) is convex, it follows that it is also star-shaped relative to the center of the sphere \(P\). This obvious fact will be used in the Chapter 7.

Some less trivial properties of the power diagram in \(\mathbb{R}^d\) are now proven under the assumptions\(^2\) given below:

**Assumption 3.1.2** No \(d+2\) spheres \(P_1, P_2, \ldots, P_{d+2}\) are cospherical, i.e. there is no point \(x \in \mathbb{R}^d\) such that \(d(x, P_1) = d(x, P_2) = \ldots = d(x, P_{d+2})\) in the power metric.

**Assumption 3.1.3** No \(d+1\) centers of the spheres \(P_1, P_2, \ldots, P_{d+1}\) lie on the same hyperplane in \(\mathbb{R}^d\).

---

\(^1\) The trivial fact that the power region is convex was also proven in [Imai et. al. 85, Aurenhammer 87].

\(^2\) Generalized Voronoi diagrams are often considered in the literature under those assumptions to simplify the proof of some properties (see, for example, [Okabe et. al. 92, Boissonnat and Yvinec 98]). The algorithms for generalized Voronoi diagram construction are also usually studied under these standard assumptions (see, for example, [Edelsbrunner and Shah 96, Schaudt and Drysdale 92]). If these assumptions are not satisfied, a perturbation of the input can be performed (see, for example, [Edelsbrunner and Mücke 88, Yap 88]). In practical implementations, the assumptions can be violated due to the round-up errors. To avoid these situations exact arithmetic can be employed [Fortune and Wyk 93, Karasick et. al. 91, Gavrilova et. al. 96].
Definition 3.1.4 A convex hull $CH(S)$ of a set of spheres $S$ in the power metric is the convex hull of the set $S_p$ of the centers of the spheres from the set $S$.

Theorem 3.1.1 The power region $PVor(P)$ is unbounded if and only if the center $p$ of the sphere $P \in S$ lies on the boundary of the convex hull $CH(S)$.

Proof First, it can be shown that if $p$ lies on the boundary of the convex hull of the set of centers of spheres from $S$, then the power region $PVor(P)$ is unbounded. By contradiction, assume that $PVor(P)$ is bounded and let $f_1, f_2, \ldots, f_k, k \geq 3$ be an ordered sequence of its facets. Each facet $f_i, i = 1..k$ belongs to a bisector $B(P, Q_i)$ between sphere $P$ and sphere $Q_i$ with the center $q_i$. Consequently, $p$ is internal to the polyhedron $q_1q_2\ldots q_k$ and therefore cannot belong to the boundary of the convex hull (under Assumption 3.1.3).

Now it will be shown that if power region $PVor(P)$ is unbounded then $p$ lies on the boundary of the convex hull $CH(S)$.

The claim is also proven by contradiction. Assume that $PVor(P)$ is unbounded and that $p = (p_1, p_2, \ldots, p_d)$ does not touch the boundary of the convex hull. Since $PVor(P)$ is unbounded and star-shaped (see Property 3.1.2), an unbounded ray $r$ starting from point $P$ can be selected so that it lies completely inside the power region $PVor(P)$. Since $p$ is not on the boundary of the convex hull of $S$, the ray $r$ will intersect this boundary at some point. Assume that it intersects with a facet $F$ of the boundary (see Figure 3.1.3 for illustration in the planar case).

Define a coordinate system with the center in $p$ and with the $x$ axis along the ray $r$ from the point $p$. Note, that at least one of the sites belonging to the facet $F$ has a positive $x$ coordinate, since otherwise $p$ would lie outside the convex hull. Assume without loss of generalization that this is a site $Q$ with center $q = (q_1, q_2, \ldots, q_d)$, i.e. $q_1 > 0$. 
Figure 3.1.3 Ray intersects with the boundary of the convex hull

Consider the distance between the circle \( P = \{ p, r \} \) and an arbitrary point \( a = (a_1, 0, \ldots, 0) \) outside of circle. Clearly, \( d(a, P) = a_1^2 - r_p^2 \) and \( d(a, Q) = (a_1 - q_1)^2 + q_2^2 + \cdots + q_d^2 - r_q^2 \).

Taking the limit as \( a_1 \to +\infty \), it follows that \( d(a, P) - d(a, Q) \to +\infty \), i.e. starting from some positive \( a_1' \) when \( a_1 > a_1' \) the point \( a \) becomes closer to \( Q \) than to \( P \). Consequently, \( a \) cannot belong to \( PVor(P) \), which contradicts the fact that the ray \( r \) lies completely inside \( PVor(P) \).

\[ \square \]

**Theorem 3.1.2** A vertex \( v \) of the power diagram is a common intersection of exactly \( d + 1 \) power regions.

**Proof** Assume that the vertex \( v \) is the intersection of more than \( d + 1 \) power regions. Then \( v \) is equidistant from more than \( d + 1 \) spheres, which are the generators of the power regions. This contradicts the non-degeneracy Assumption 3.1.2.

Now, assume that the vertex \( v \) is the intersection of \( k \) power regions, where \( k < d + 1 \). Assume \( k = d \); the proof for the remaining cases when \( k < d \) is similar. Since \( v \) is a vertex of the power diagram, it must be equidistant from some set of \( d \) spheres \( P_1, P_2, \ldots, P_d \). The coordinates of \( v \) must satisfy the following system of equations:

\[ d(v, P_1) = d(v, P_2) = \cdots = d(v, P_d). \]

The higher order terms in this system can be cancelled to produce an equivalent linear system that consists of \( d - 1 \) equations in \( d \) variables (the coordinates of \( v \)).
Write this linear system as $Ax = b$. Since this system has at least one solution (because $v$ exists), it can be concluded that $\text{rank}(A) = \text{rank}(A | b)$. Also, $\text{rank}(A) \leq d - 1$. Consequently, the system has infinitely many solutions, i.e. it describes a line or a hyperplane. Therefore, $v$ is on an edge (or a face) of the power diagram, i.e. $v$ cannot be a vertex of the power diagram, which contradicts the statement of the theorem.

The empty-sphere property in the power metric will now be proven. Consider two intersecting spheres $A = (a, r_a)$ and $B = (b, r_b)$, $r_a > 0$, $r_b > 0$ (see Figure 3.1.4). Assume that one sphere does not lie completely inside the other.

Denote one of the points of intersection between the spheres by $c$. Then the angle $\varphi$ between the spheres $A$ and $B$ can be defined as the angle $acb$.

**Lemma 3.1.1** The angle $\varphi$ between two intersecting spheres $A$ and $B$ is acute if and only if $r_b^2 > d(b, A)$, $\varphi$ is right if and only if $r_b^2 = d(b, A)$, and $\varphi$ is obtuse if and only if $r_b^2 < d(b, A)$.

---

1 Note that this property was established independently from the similar result presented in [Edelsbrunner and Shah 96]. It is formulated and proven in this chapter in terms of the angle between spheres in the power metric.
Proof Note that \( \cos \varphi = \frac{r_a^2 + r_b^2 - |ab|^2}{2 r_a r_b} = \frac{r_b^2 - d(b, A)}{2 r_a r_b} \) from the cosine theorem. The denominator of the formula is always positive. Consequently, the sign of the expression is determined by the sign of the numerator.

A sphere \( C \) with center \( \xi \) and radius \( \rho \), inscribed among spheres \( P_1, P_2, \ldots, P_{d+1} \) in the power metric is now introduced. The center of the inscribed sphere is equidistant from each of the \( d+1 \) given spheres. Thus the following definition can be given:

**Definition 3.1.5** A sphere \( C = \{\xi, \rho\} \) inscribed among \( d+1 \) spheres \( P_1, P_2, \ldots, P_{d+1} \) in the power metric is a sphere with center \( \xi = (\xi_1, \xi_2, \ldots, \xi_d) \) and radius \( \rho \), such that

\[
\rho^2 = d(\xi, P_1) = d(\xi, P_2) = \ldots = d(\xi, P_{d+1}).
\]

*Note.* The fact that the inscribed sphere \( C = \{\xi, \rho\} \) exists for any set of \( d+1 \) spheres satisfying Assumption 3.1.3 is proven in Chapter 4, Theorem 4.4.2 for \( R^d \).

The inscribed sphere \( C \) intersects each of the spheres \( P_1, P_2, \ldots, P_{d+1} \) at right angles since \( \rho^2 = d(\xi, P_i) \). The illustration for the 2-dimensional case is presented in Figure 3.1.5.

![Inscribed sphere](image)

*Figure 3.1.5 Inscribed sphere \( C \) intersects sphere \( P_i \)*

Note that the spheres \( P_1, P_2, \ldots, P_{d+1} \) will be called spheres *cospherical* to inscribed sphere \( C \) in the sequel.
Definition 3.1.6 The inscribed sphere $C$ is called an empty sphere\(^1\) in the power metric if no sphere from $S$ intersects $C$ at an acute angle and if no sphere from $S$ completely belongs to $C$.

Theorem 3.1.3 (the empty-sphere property) Consider a vertex $v$ of the power diagram $v = PVor(P_1) \cap PVor(P_2) \cap \ldots \cap PVor(P_{d+1})$. Then $v$ is the center of a sphere $C = \{v, \rho\}$ inscribed between $P_1, P_2, \ldots, P_{d+1}$ in the power metric. Any sphere $Q \in S$ different from $P_1, P_2, \ldots, P_{d+1}$ either does not intersect $C$ or intersects $C$ at an obtuse angle (i.e. $C$ is empty in power metric).

Proof First, consider the case when $r_q = 0$, i.e. $Q$ is a point site. Assume that it intersects $C$. If $q$ lies on the boundary of $C$, then $\|v - q\| = \rho$ and $d(v, Q) = \rho^2$, and Assumption 3.1.2 (no $d+2$ spheres are cospherical) is violated. If $q$ lies inside $C$, then $\|v - q\| < \rho$ and $d(v, Q) < \rho^2 = d(v, P_1)$, which contradicts the fact that $v$ is a power vertex.

Now, assume that $r_q > 0$ and that $Q$ intersects $C$ at right or acute angles. If $Q$ intersects $C$ at a right angle, then it follows from Lemma 3.1.1 that $d(v, Q) = \rho^2$ and this contradicts Assumption 3.1.2. If $Q$ intersects $C$ at an acute angle then it follows from Lemma 3.1.1 that $d(v, Q) < \rho^2$ and this contradicts the fact that $v$ is a power vertex.

$\blacksquare$

Note. The inscribed sphere $C$ cannot contain the centers of any spheres in its interior, since $\|v - p\|^2 = d(v, P) + r_p^2 \geq d(v, P) \geq \rho^2$.

A symmetric distance between two spheres in the power metric is now introduced. The nearest-neighbor property will be proven with the use of this definition. An important lemma is proven first.

\(^1\) This is a common term used in the literature (see, for example, [Okabe et. al. 92]).
Consider the bisector $B(P,S)$ of two spheres $P = \{p, r_p\}$ and $S = \{s, r_s\}$. Denote the point of intersection of the bisector $B(P,S)$ and the segment $\overline{ps}$ by $y$.

**Lemma 3.1.2** The point of intersection of the bisector $B(P,Q)$ and the segment $\overline{pq}$ is the closest point on the bisector $B(P,Q)$ to both spheres $P$ and $Q$.

**Proof** Consider an arbitrary point $x \in B(P,S)$ (see Figure 3.1.6). Let a coordinate system with axes along $B(P,Q)$ and $\overline{pq}$ be given. Then, $d(x,Q) = |xq|^2 - r_q^2$ and $d(y,Q) = |yq|^2 - r_q^2$. Since $|xq| \geq |yq|$, it is follows that $d(x,Q) \geq d(y,Q)$.

![Figure 3.1.6](image)

Figure 3.1.6 Point $y$ is the closest point of the bisector to both spheres $P$ and $Q$.

**Definition 3.1.7** The distance between two spheres in the power metric in $R^d$ is defined as $d(P,Q) = 2d(y,P) = 2d(y,Q)$, where $y$ is the point of intersection of the bisector $B(P,Q)$ and the segment $\overline{pq}$.

**Theorem 3.1.4** (the nearest-neighbor property) If $Q \in S$ is the nearest-neighbor of $P \in S$ then power regions $PVor(Q)$ and $PVor(P)$ have a common facet.

**Proof** Assume that the sphere $Q$ is the nearest neighbor of $P$ and assume that $PVor(P)$ and $PVor(Q)$ do not share a facet. Consider the segment $\overline{pq}$. The endpoint $p$ belongs to $PVor(P)$ and the endpoint $q$ belongs to $PVor(Q)$. Since $PVor(P)$ and $PVor(Q)$ do not intersect, the segment $\overline{pq}$ must cross some power region $PVor(S)$ (see Figure 3.1.7).
Figure 3.1.7 Proof of the fact that \( Q \) is the nearest-neighbor of \( P \)

Denote the point of intersection of \( \overline{pq} \) with the bisector \( B(P,S) \) by \( x \) and denote the point of intersection of \( \overline{pq} \) with the bisector \( B(P,Q) \) by \( z \). First, note that the point \( z \) lies outside the power region \( PVor(P) \). It cannot lie in the interior of the power region \( PVor(P) \) since it's equidistant from two sites \( P \) and \( Q \). Also, it cannot lie on the boundary of \( PVor(P) \) since power regions \( PVor(P) \) and \( PVor(Q) \) do not share a facet. Thus, the point \( z \) lies outside the power region \( PVor(P) \) and, therefore,

\[
\|x - p\|^2 - r_P^2 < \|z - p\|^2 - r_P^2.
\]

Denote the point of intersection of the segment \( \overline{ps} \) and \( B(P,S) \) by \( y \). According to Lemma 3.1.2 \( y \) is the closest point of the bisector to the point \( P \), i.e.

\[
\|y - p\|^2 - r_P^2 \leq \|x - p\|^2 - r_P^2.
\]

From these inequalities it follows that

\[
\|y - p\|^2 - r_P^2 < \|z - p\|^2 - r_P^2, \text{ i.e. } d(y, P) < d(z, P).
\]

But according to Definition 3.1.7 \( d(P,S) = 2d(y, P) \) and \( d(P,Q) = 2d(z, P) \). Thus it can be concluded that \( d(P,S) < d(P,Q) \), which contradicts the fact that \( Q \) is the nearest neighbor of \( P \).

\[\square\]
3.1.3 The Delaunay Tessellation in the Power Metric

The definition of the straight-line dual of the power diagram in $\mathbb{R}^d$ is now introduced.

**Definition 3.1.8** A *power Delaunay tessellation (PDT)* is a diagram such that for each power vertex $v = PVor(P_1) \cap PVor(P_2) \cap ... \cap PVor(P_{d+1})$ there is a simplex $(p_1, p_2, ..., p_{d+1})$ in the tessellation [Okabe et al. 92].

The power Delaunay tessellation corresponding to the power diagram in 2D from the Figure 3.1.2 is presented in the Figure 3.1.8.

![Figure 3.1.8 The power Delaunay triangulation](image)

The following property follows directly from Theorem 3.1.3 (the empty-sphere property).

It will be used in the following chapters for power diagram construction.

**Property 3.1.3** The sphere inscribed between the sites that form a simplex in the power Delaunay tessellation is an empty sphere (in terms of Definition 3.1.6).

**Proof** Consider a vertex $v$ of the power diagram

$$v = PVor(P_1) \cap PVor(P_2) \cap ... \cap PVor(P_{d+1}).$$

According to Theorem 3.1.2 it exists and is uniquely defined. Then $v$ is the center of a sphere $C = \{v, p\}$ inscribed between $P_1, P_2, ..., P_{d+1}$ comprising a simplex $(p_1, p_2, ..., p_{d+1})$ of the power DT. Then, according to Theorem 3.1.3 (the empty-sphere property), the inscribed sphere $C$ is empty.

■
3.2 The Weighted Voronoi Diagram in the Euclidean Metric

The Euclidean weighted Voronoi diagram is the first additively weighted Voronoi diagram considered in this chapter. Up till now, only a few properties of such diagrams were listed in the literature.

In this section the Euclidean weighted VD is studied for a set of spheres in $d$ dimensions under the assumption that the spheres do not intersect. The contribution of this section is in the establishing the following facts for the Euclidean weighted VD in $R^d$. The convex hull of the set of spheres in the Euclidean metric is introduced. A criterion to determine whether the Euclidean Voronoi region is bounded or unbounded is presented. It is proven that a vertex $v$ of the diagram is the common intersection of exactly $d+1$ Euclidean Voronoi regions. The empty-sphere and the nearest-neighbor properties are formulated and proven. Finally, the dual of the Euclidean weighted Voronoi diagram, the Euclidean weighted Delaunay tessellation, is discussed.

3.2.1 Definition of the Euclidean Weighted Voronoi Diagram and Preliminaries

The weighted Voronoi diagram in the Euclidean metric belongs to the class of additively weighted Voronoi diagrams. The distance $d(x,p)$ between two points $x = (x_1, x_2, ..., x_d)$ and $p = (p_1, p_2, ..., p_d)$ in $R^d$ in the Euclidean metric is calculated as

$$
\|x - p\| = \sqrt{\sum_{i=1}^{d} (x_i - p_i)^2}.
$$

The additively weighted distance function between a point $x$ and an object $P$ in $R^d$ is defined as $d(x,P) = d(x,p) - w(P)$, where $p$ is the reference point of the object $P$ and $w(P)$ is the weight [Okabe et. al. 92].

One of the common ways to define the additively weighted distance between a point $x$ and a sphere $P$ is to select $p$ as the center of the sphere and $w(P)$ as the radius $r_p$ of the sphere. The resulting formula $d(x,P) = d(x,p) - r_p$ has an obvious geometric interpretation as the shortest distance between the point $x$ and the sphere $P$ (see Figure 3.2.1).
Figure 3.2.1 The distance and bisector in the Euclidean metric

Note. A point $x$ lies on or inside the sphere $P$ if and only if the distance $d(x, P)$ is non-positive.

The properties of the Euclidean diagram will be studied under the following assumption:

**Assumption 3.2.1** The spheres of the set $S$ do not intersect, i.e. $\|p - q\| \geq r_p + r_q$ for any two distinct spheres $P = \{p, r_p\}$ and $Q = \{q, r_q\}$ from $S$.

**Definition 3.2.1** An *Euclidean region* $\text{EVor}(P)$ of the sphere $P \in S$ in the Euclidean metric in $\mathbb{R}^d$ is the set of points that are closer to $P$ than to any other sphere $Q \in S$:

$$\text{EVor}(P) = \{x \in \mathbb{R}^d | d(x, P) \leq d(x, Q), \forall Q \in S - \{P\}\} \quad \text{[Okabe et. al. 92].}$$

The definition of the Euclidean bisector is given similarly to the power metric definition. Note, however, that the form of the bisector differs. An *Euclidean bisector* is defined as $B(P, Q) = \{x | d(x, P) = d(x, Q)\}$ in $\mathbb{R}^d$. It divides the space into two quasi-halfspaces (see Figure 3.2.1). A *quasi-halfspace* that contains the sphere $P$ is defined as $H(P, Q) = \{x | d(x, P) \leq d(x, Q)\}$. Under Assumption 3.2.1 each sphere completely belongs to one quasi-halfspace and thus the bisector divides the space into two connected quasi-halfspaces $H(P, Q)$ and $H(Q, P)$. The equivalent definition of an Euclidean region $\text{EVor}(P)$ can now be written as:

**Definition 3.2.2** An *Euclidean region* $\text{EVor}(P)$ of the sphere $P \in S$ in $\mathbb{R}^d$ is the intersection of all quasi-halfspaces $H(P, Q)$:
\[ EVor(P) = \bigcap_{Q \in P} H(P, Q). \]

**Definition 3.2.3** An *Euclidean weighted Voronoi diagram (EWVD)* for the set of spheres \( S \) in the Euclidean metric in \( \mathbb{R}^d \) is the collection of all Euclidean regions

\[ EWVD(S) = \{ EVor(P_1), ..., EVor(P_n) \}. \]

The Euclidean weighted VD for four circles in the plane is shown in Figure 3.2.2

![Figure 3.2.2 The Euclidean weighted Voronoi diagram](image)

The known properties of the Euclidean weighted VD are now listed. The properties of the Euclidean weighted VD in 2D were studied in [Sharir 85]. The author proved that the bisector is either a hyperbolic arc or a straight-line segment, the Voronoi region is star-shaped in respect to its generator and that the Voronoi vertex is located on the intersection of 3 Voronoi bisectors. He also proved an empty-circle property and established relationship between unbounded Voronoi faces and convex hull in the plane [Sharir 85]. Besides these properties, Ash and Bolker discussed the generator recognition problem in the additively and multiplicatively weighted planar VD [Ash and Bolker 86].

Two simple properties of the Euclidean weighted VD were extended to higher dimension in [Okabe et. al. 92]. Thus, it was proven that bisector of a Euclidean weighted VD is either a hyperbolic curve or a hyperplane and that the VD region is star-shaped with respect to its generator in \( \mathbb{R}^d \) [Okabe et. al. 92]. The relationship between additively weighted VD and convex hulls in higher dimensions has been established in [Boissonnat and Yvinec 98].
Other properties of the Euclidean weighted VD in $R^d$ were not considered in the literature. They are established in the following section of the thesis. Some differences between previously obtained results and results presented in his section are now outlined. The form of the bisector in the Euclidean metric makes it harder to establish the fact that a vertex $v$ of the diagram is the common intersection of exactly $d + 1$ Euclidean Voronoi regions. The criterion of whether a Voronoi region is bounded or unbounded cannot be based on the convex hull condition, as it is for power diagrams (see Theorem 3.1.1). Thus a different criterion based on the convex hull of a set of spheres is introduced. The definitions of the nearest-neighbor and the empty-sphere, specific for the Euclidean metric, are also given. Interestingly, they make the proof of the nearest-neighbor and empty-sphere properties in the Euclidean metric somewhat easier than the proof for power metric despite the fact that the Euclidean bisector is no longer a straight line.

### 3.2.2 Properties of the Euclidean Weighted Voronoi Diagram in $d$ Dimensions

The basic properties of the EWVD that describe the form of the bisector and the Euclidean region are first given (note that they have also been listed without proof for the planar case in [Okabe et. al. 92]).

**Property 3.2.1** The bisector $B(P,Q)$ of an EWVD is one half of the hyperboloid with poles $p$ and $q$ and parameter $l = |r_p - r_q|$.

**Proof** All points $x$ of the bisector $B(P,Q)$ satisfy the equation $\|x - p\| - r_p = \|x - q\| - r_q$. This equation can be rewritten as $\|x - p\| - \|x - q\| = r_p - r_q$. Assuming without loss of generality that $r_p \geq r_q$, it can be seen that the equation defines one half of a hyperboloid with poles $p$ and $q$ and parameter $l = r_p - r_q$. The other half of the hyperboloid defined by the equation $\|x - p\| - \|x - q\| = -(r_p - r_q)$ does not represent a bisector.
Note. If \( r_p = r_q \) then the bisector is a hyperplane (i.e. a hyperboloid with parameter \( l = 0 \)).

**Property 3.2.2** Each Euclidean region \( EVor(P) \) is a connected region in \( R^d \) space, containing the whole sphere \( P \). The region is star-shaped relative to the center of the sphere \( P \).

**Proof** First, it will be proven that the sphere \( P \) lies entirely inside its Euclidean region \( EVor(P) \). Note that the distance between any point \( x \) lying on or inside the sphere \( P \) is zero or negative. Furthermore, for any point lying outside the sphere the distance is positive. Therefore, any point \( x \) lying on or inside the sphere \( P \) is closer to \( P \) than to any other sphere \( Q \), since \( x \) cannot belong to both \( P \) and \( Q \) under Assumption 3.2.1 (because the spheres do not intersect). Consequently, the sphere \( P \) lies completely inside its Euclidean region \( EVor(P) \).

To prove the second statement, recall that the Euclidean region \( EVor(P) \) can be defined as the intersection of quasi-halfspaces \( H(P,Q) \). Since each of the quasi-halfspaces is a star-shaped region relative to the center of the sphere \( P \), their intersection is also a star-shaped region relative to the center of the sphere \( P \).

\[ \blacksquare \]

Note. The Euclidean region is not necessarily convex. Since the boundary of the Euclidean region is formed by bisectors, which in the general case are hyperboloids, the concave side of such a bisector belongs to a non-convex Euclidean region.

The non-degeneracy assumptions are now introduced:

**Assumption 3.2.2** No \( d + 2 \) spheres \( P_1, P_2, \ldots, P_{d+2} \) are cospherical, i.e. there is no point \( x \in R^d \) such that \( d(x, P_1) = d(x, P_2) = \ldots = d(x, P_{d+2}) \) in the Euclidean metric.

**Assumption 3.2.3** No \( d + 1 \) centers of spheres \( P_1, P_2, \ldots, P_{d+1} \) lie on the same hyperplane in \( R^d \) and no \( d + 1 \) spheres are tangent to the same hyperplane.
A convex hull of a set of spheres $S$ is now introduced.

**Definition 3.2.4** A convex hull $CH(S)$ of a set of spheres $S$ in the Euclidean metric is an intersection of all convex sets containing $S$ in $R^d$.

Note that this definition differs from the definition of the convex hull in the power metric (Definition 3.1.4, page 38) since it represents the smallest convex domain in $R^d$ containing all spheres from $S$. The new properties of the EWVD in $R^d$ are now established under Assumptions 3.2.1, 3.2.2 and 3.2.3.

**Theorem 3.2.1** The sphere $P \in S$ touches the boundary of the convex hull $CH(S)$ if and only if its Euclidean region $EVor(P)$ is unbounded.

**Proof** The following claim will be proven first: if $P$ lies on the boundary of the convex hull $CH(S)$ then its Euclidean region is unbounded.

Assume that sphere $P$ lies on the boundary of the convex hull $CH(S)$. From the definition of convex hull there exists a hyperplane $H$ tangent to $P$ such that all sites of $S$ lie on the same side of $H$ (see Figure 3.2.3).

![Figure 3.2.3 An unbounded ray $r$ inside $EVor(P)$](image)

$H$ splits the space into two halfspaces, and one of them contains all sites of $S$. Denote this halfspace by $H(S)$. Denote the point of intersection between $H$ and $P$ by $x$ and select a ray $r$ outside of $H(S)$ originating at $x$ and orthogonal to $H$. Clearly, for any point $a \in r$ the point $x$ is the closest point of $H(S)$ to $a$. Clearly,
\[ d(a, P) = \|x - a\| = \min_{y \in H(S)} \|y - a\|. \] Also, \[ \min_{y \in H(S)} \|y - a\| \leq \min_{Q \in S} d(a, Q) \] since all sites belong to \( H(S) \). Therefore, the site \( P \) is closer to \( a \) than to any other site. Therefore, \( a \in EVor(P) \) and, consequently, the whole ray \( r \) also belongs to \( EVor(P) \). Therefore, \( EVor(P) \) is unbounded.

Now, it will be proven that if \( EVor(P) \) is unbounded then \( P = \{(p_1, p_2, \ldots, p_d), r_P\} \) must touch the boundary of the convex hull \( CH(S) \). Conversely, assume that \( EVor(P) \) is unbounded and suppose that \( P \) does not touch the boundary of the \( CH(S) \). Since \( EVor(P) \) is unbounded and star-shaped (see Property 3.2.2), an unbounded ray \( r \) can be selected so that it starts from a point \( p \) that lies completely inside \( EVor(P) \). Since \( P \) is not on the boundary of \( CH(S) \), the ray \( r \) will intersect this boundary at some point (see Figure 3.2.4). Assume that it intersects with a facet \( F \) of the boundary between sites \( Q_1 = \{(q_{11}, q_{12}, \ldots, q_{1d}), r_{q_1}\}, Q_2 = \{(q_{21}, q_{22}, \ldots, q_{2d}), r_{q_2}\}, \ldots, Q_d = \{(q_{d1}, q_{d2}, \ldots, q_{dd}), r_{q_d}\}. \)

Note that the case when the ray intersects the site \( Q_i \) that belongs to the convex hull can be proven similarly.

![Figure 3.2.4 Illustration for the proof of Theorem 3.2.1](image)

Define a coordinate system \( q^* \) with center in \( p \) and \( q_1^* \) axis along the ray \( r \). Select a site \( Q_i \) such that the sum \( (q_{i1}^* + r_{q_i}) \) is maximum among all \( Q_i \), \( i = 1..d \). Without loss of generality, denote this site \( Q_1 \). Note that if this maximum sum is less than or equal to the
radius of the site $P$, then site $P$ is on the boundary of the convex hull $CH(S)$ and this contradicts the assumption. Thus, the following inequality is true: $r_p < q_{11}^* + r_{q_1}$.

Next, consider the distance between a point $a = (a_1,0,\ldots,0)$ and site $P$. Clearly, $d(a,P) = a_1 - r_p$, and $d(a,Q) = \sqrt{(a_1 - q_{11}^*)^2 + \cdots + (q_{1d}^*)^2} - r_{q_1}$.

Then $\lim_{a_1 \to +\infty} d(a,P) - d(a,Q) = q_{11}^* + r_{q_1} - r_p$.

But $r_p < q_{11}^* + r_{q_1}$, thus $d(a,P) - d(a,Q) \to q_{11}^* + r_{q_1} - r_p > 0$. This means that starting from some positive $a_1$ when $a_1 > a_1^*$ the point $a$ becomes closer to $Q$ than to $P$. Consequently, $a$ cannot belong to the $EVar(P)$, which contradicts the fact that the ray $r$ lies completely inside $EVar(P)$.

\begin{definition}
A cone in $R^d$ defined by the apex $a \in R^d$, axis $n = (n_1, n_2, \ldots, n_d)$ and aperture $\alpha \in [0..\pi/2]$ is the set of all points $x \in R^d$ such that the angle between vectors $x-a$ and $n$ is $\alpha$. The apex $a$ also belongs to the cone.

\textit{Note} A cone in $R^1$ consists of apex only. A cone in $R^2$ consists of 2 rays originating at the apex. A cone with aperture $\alpha = \pi/2$ is a $(d-1)$-hyperplane.

\begin{assumption}
Let $P_1, P_2, \ldots, P_m$ be $2 \leq m \leq d$ spheres in $R^d$, $d \geq 4$. Construct an $(m-1)$-hyperplane $H$ spanning the centers of spheres $p_1, p_2, \ldots, p_m$. Let $C_H$ be the sphere in $H$ inscribed between $P_1, P_2, \ldots, P_m$ (if it exists). Denote the point where $P_i$ touches $C_H$ by $p_i^C$, $i=1..m$. Then $p_i^C$ can not belong to an $(m-2)$-hyperplane $H_C$ in $H$ (see Fig. 3.2.5 for an example for $m = 4$).
\end{assumption}
Figure 3.2.5. Assumption 3.2.4 is violated for $m = 4$.

*Note 1* Let $v \in H$ be the center of the sphere $C_H$. The Assumption 3.2.4 is not satisfied if and only if $p_1, p_2, \ldots, p_m$ belong to a cone in $H$ with apex at $v$.

*Note 2* For $d \leq 3$ the statement of Assumption 3.2.4 follows from the Assumption 3.2.1 (non-intersecting spheres):

$m = 2$. $H$ is just a line and $H_C$ is a single point. Since the spheres are not allowed to intersect, $p_1^C$ and $p_2^C$ cannot both coincide with $H_C$.

$m = 3$. $H$ is a plane and $H_C$ is a line (see Fig. 3.2.6). $H_C$ intersects $C_H$ at 2 points at most. Then at least 2 of $p_1^C$, $p_2^C$ and $p_3^C$ must coincide, which cannot happen because the spheres are not allowed to intersect.

Figure 3.2.6. Assumption 3.2.4 for $m = 3$. 
Theorem 3.2.2 A vertex $v$ of the EWVD is the common intersection of exactly $d+1$ Euclidean regions under Assumptions 3.2.1, 3.2.2, 3.2.3 and 3.2.4.

Proof Assume that $v$ is the intersection of more than $d+1$ Euclidean regions. Then $v$ is equidistant from more than $d+1$ spheres (these spheres are the generators of the Euclidean regions). But this contradicts Assumption 3.2.2.

Next, assume that the vertex $v$ is the intersection of $k$ Euclidean regions, where $k < d+1$. Assume $k = d$, the proof for the remaining cases when $k < d$ is similar.

Since $v$ is a vertex of the EWVD, it must be equidistant from spheres $P_1, P_2, ..., P_d$. Denote $v = (v_1, v_2, ..., v_d)$ and $r = d(v, P_1)$, then $v$ is the center of a sphere with radius $r$ that is tangent to all spheres $P_1, P_2, ..., P_d$. The coordinates of this sphere satisfy the following equations

$$(x_{i1} - v_1)^2 + (x_{i2} - v_2)^2 + ... + (x_{id} - v_d)^2 - (r_i + r)^2 = 0, \quad i = 1..d,$$

where $p_i = (x_{i1}, x_{i2}, ..., x_{id})$ and $r_i$ are the center and the radius of $P_i$, respectively.

The following system can be written

$$(x_{i1} - (v_1 + \xi_1))^2 + (x_{i2} - (v_2 + \xi_2))^2 + ... + (x_{id} - (v_d + \xi_d))^2 - (r_i + (r + \xi_{d+1}))^2 = 0, \quad i = 1..d.$$

Obviously, this system has a solution $(\xi_1, \xi_2, ..., \xi_d, \xi_{d+1}) = (0,0,...,0)$. Add an additional equation of the form $\xi_k = \epsilon$, where $k \in [1..d+1]$. The LHS of the extended system represents a function $F_k : R^{d+1} \to R^{d+1}$. Obviously, $F_k(0) = 0$.

It is known from the mathematical analysis that if the Jacobean of a smooth function $F(x)$ is not equal to zero at some point $x_0$ then the function has an inverse $F^{-1}(y)$ defined in some neighborhood of $y_0 = F(x_0)$ [Lang 69].

The function $F_k$ is smooth (it is a polynomial) and the Jacobean has the following form
If for some $k$ \( J(F_k)(0) \neq 0 \) then \( F_k \) is invertible in 0, which means that there is some small neighborhood $|\varepsilon| < \varepsilon_1$ such that the system has a solution for every $\varepsilon$ in this neighborhood. In this case, $(v, r)$ is not a unique inscribed sphere.

Assume that $J(F_k)(0) = 0$ for every $k$. Then the first $d$ rows in the determinant do not contain a non-zero $d$-minor, i.e. they are linearly dependent. Since

$$
\begin{vmatrix}
x_{11} - v_1 & x_{12} - v_2 & \ldots & x_{1d} - v_d \\
x_{21} - v_1 & x_{22} - v_2 & \ldots & x_{2d} - v_d \\
\vdots & \vdots & \ddots & \vdots \\
x_{d1} - v_1 & x_{d2} - v_2 & \ldots & x_{dd} - v_d \\
\end{vmatrix}
= 0,
$$

points $p_1, p_2, \ldots, p_d$ and $v$ must lie in the same hyperplane $H$.

Perform a coordinate transformation such that $v$ becomes the center of coordinates and the axis $x_d$ is perpendicular to $H$. Denote the new coordinates of centers of spheres by $y_{ij}, i, j = 1..d, \ y_{id} = 0, i = 1..d$.

Since the first $d$ rows of $J(F_k)(0)$ are linearly dependent, the following determinant is zero:

$$
\begin{vmatrix}
y_{11} & y_{12} & \ldots & y_{1,d-1} & r_1 + r \\
y_{21} & y_{22} & \ldots & y_{2,d-1} & r_2 + r \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
y_{d1} & y_{d2} & \ldots & y_{d,d-1} & r_d + r \\
\end{vmatrix}
= 0,
$$

Note that the $i$-th row in the determinant represents a projection of the point $p_i$ onto the cone $C$ in $R^d$ defined by the equation $y_d = \sqrt{y_1^2 + y_2^2 + \ldots + y_{d-1}^2}$. Denote this projection by $p_i^* = (y_{i1}, y_{i2}, \ldots, y_{i,d-1}, r_i + r), i = 1..d$. Since the above determinant is
zero, all points $p_i^*$ and $v \neq 0$ must lie in a hyperplane $H_1$. Thus, all $p_i^*$ belong to the intersection of the cone $C$ and the hyperplane $H_1$.

It will now be shown that the projection of the intersection of the cone $C$ and the hyperplane $H_1$ onto the hyperplane $H$ is a cone in $H$. Perform another coordinate transformation: rotate the system around vector $(0,0,0,...,1)$ such that the normal $n_1$ to $H_1$ lies in the plane $y_1y_d$. Denote the new coordinates by $z$. Then the equation of $H_1$ transforms to $z_1 = \alpha z_d$, $0 \leq \alpha \leq 1$ (note that the angle between $H_1$ and $H$ is at least $\pi/4$ and at most $\pi/2$). The points $p_i$ are the projections of $p_i^*$ onto $H$. Their new coordinates satisfy the following equations:

$$z_{i1}^2 + z_{i2}^2 + \ldots + z_{i,d-1}^2 = (r_i + r)^2 \text{ and } z_{i1} = \alpha (r_i + r), \; i = 1..d.$$  

$$\left(\frac{1}{\alpha^2} - 1\right) z_{i1}^2 = z_{i2}^2 + \ldots + z_{i,d-1}^2, \text{ when } \alpha > 0, \text{ or } z_{i1} = 0 \text{ when } \alpha = 0.$$  

Thus, it was shown that if $J(F_k)(0) = 0$ for every $k$ then the points $p_i$ must lie on a cone in $H$ with the apex at $v$, which contradicts the Assumption 3.2.4. Therefore, $(v, r)$ is not a unique inscribed sphere.

The definition of the sphere inscribed among $d + 1$ spheres is now presented:

**Definition 3.2.6** A sphere $C = \{\xi, \rho\}$ inscribed among $d + 1$ spheres $P_1, P_2, ..., P_{d+1}$ in the Euclidean metric is a sphere with center $\xi = (\xi_1, \xi_2, ..., \xi_d)$ and radius $\rho$, such that $\rho = d(\xi, P_1) = d(\xi, P_2) = ... = d(\xi, P_{d+1})$.

**Note.** The existence of the inscribed sphere for a set of spheres in the Euclidean metric is considered in Chapter 4, Section 4.5.

Note that spheres $P_1, P_2, ..., P_{d+1}$ will be called spheres *cospherical* to the inscribed sphere $C$ in the sequel.
**Definition 3.2.7** An inscribed sphere $C$ is called *empty* in the Euclidean metric if no sphere from $S$ intersects its interior.

**Theorem 3.2.3** (the empty-sphere property) Consider a vertex $v$ of the EWVD $v = EVor(P_1) \cap EVor(P_2) \cap \ldots \cap EVor(P_{d+1})$. Then $v$ is the center of a sphere $C = \{v, \rho\}$ inscribed between $P_1, P_2, \ldots, P_{d+1}$ in the Euclidean metric and there is no sphere $Q \in S$ different from $P_1, P_2, \ldots, P_{d+1}$ intersecting $C$ (i.e. $C$ is empty in the Euclidean metric).

![Figure 3.2.7 Illustration for the proof of Theorem 3.2.3](image)

**Proof** Assume that there exists such a sphere $Q \in S$, which intersects the inscribed sphere $C$ (see Figure 3.2.7). Then $d(v, Q) \leq \rho = d(v, P_1)$. If $d(v, Q) < \rho$ then $v$ is closer to $Q$ than to any of spheres $P_1, P_2, \ldots, P_{d+1}$. But this contradicts the fact that $v$ belongs to $EVar(P_1)$. If $d(v, Q) = \rho$ then the Assumption 3.2.2 is violated because more than $d + 1$ spheres are cocircular.

The distance between two spheres in the Euclidean metric is now defined (see Figure 3.2.8).

![Figure 3.2.8 The Euclidean distance between two circles](image)
**Definition 3.2.8** The distance between two spheres in the Euclidean metric in $\mathbb{R}^d$ is defined as $d(P, Q) = \|p - q\| - r_p - r_q$.

**Theorem 3.2.4** (the nearest-neighbor property) If $Q \in S$ is the nearest-neighbor of $P \in S$ then the Euclidean regions $EVor(Q)$ and $EVor(P)$ have a common facet.

**Proof** Assume that the sphere $Q$ is the nearest neighbor of $P$ and assume that $EVor(P)$ and $EVor(Q)$ do not share an edge. Consider the segment $pq$. Obviously, the endpoint $p$ belongs to $EVor(P)$ and the endpoint $q$ belongs to $EVor(Q)$. Since $EVor(P)$ and $EVor(Q)$ do not intersect, the segment $pq$ must cross some Euclidean region $EVor(S)$ (see Figure 3.2.9).

Denote the point of intersection of $pq$ with the bisector $B(P, S)$ by $x$. Obviously, $\|x - s\| - r_s < \|x - q\| - r_q$ since $x \in EVor(S)$ and $x \notin EVor(Q)$. Denote the point of intersection of segment $ps$ and $B(P, S)$ by $y$. Clearly, $y$ is the point of the bisector closest to $P$, i.e. $\|y - p\| - r_p \leq \|x - p\| - r_p$.

![Figure 3.2.9 Illustration for the proof of Theorem 3.2.4](image)

Similarly, $\|y - s\| - r_s \leq \|x - s\| - r_s$ and, therefore, $\|y - s\| - r_s < \|x - q\| - r_q$. Consequently, $d(P, S) = (\|y - p\| - r_p) + (\|y - s\| - r_s) < (\|x - p\| - r_p) + (\|x - q\| - r_q) = d(P, Q)$, which contradicts the fact that $Q$ is the nearest neighbor of $P$. 

$\blacksquare$
3.2.3 The Euclidean Delaunay Tessellation

The dual of the EWVD is now introduced.

**Definition 3.2.9** An Euclidean Delaunay tessellation\(^1\) (EDT) is a diagram such that for each Euclidean vertex \(v = EVor(P_1) \cap EVor(P_2) \cap ... \cap EVor(P_{d+1})\) there is a simplex \((p_1, p_2, ..., p_{d+1})\) in the tessellation [Okabe et. al. 92].

The planar Euclidean Delaunay tessellation and corresponding Euclidean weighted Voronoi diagram is shown in Figure 3.2.10. Unlike the Delaunay tessellation of points or the power Delaunay tessellation, the EDT of spheres can contain duplicate and/or intersecting faces. This is due to the fact that the Euclidean regions can be non-convex.

![Figure 3.2.10 The Euclidean weighted Voronoi diagram and the Delaunay tessellation.](image)

A planar example of this phenomenon is presented in the Figure 3.2.11. The Delaunay tessellation (triangulation) of this set of spheres contains the following triangles: \(p_{st}\), \(q_{st}\), \(p_{sq}\) and \(p_{tq}\). Clearly, edges \(pq\) and \(st\) of the triangulation intersect.

---

\(^1\) The term *tessellation* is used in [Okabe et. al 92] to describe the dual of the generalized VD.
Property 3.2.3 The sphere inscribed between the sites that form a simplex in the Euclidean Delaunay tessellation is the empty sphere (in terms of Definition 3.2.7).

Proof Consider a vertex $v$ of the EWVD $v = EVor(P_1) \cap EVor(P_2) \cap ... \cap EVor(P_{d+1})$.

According to Theorem 3.2.2 it exists and is uniquely defined. Then $v$ is the center of a sphere $C = \{v, \rho\}$ inscribed between $P_1, P_2, ..., P_{d+1}$ comprising a simplex $(p_1, p_2, ..., p_{d+1})$ of the EDT. Then, according to Theorem 3.2.3 (the empty-sphere property), the inscribed sphere $C$ is empty.
3.3 The Weighted Voronoi Diagram in the Manhattan and the Supremum Metrics

This section presents properties of the additively weighted Voronoi diagrams in the Manhattan and the supremum metrics for a set of spheres in $d$ dimensions under the assumption that the spheres do not intersect. The following important properties are established. The convex hull of the set of spheres in the Manhattan and the supremum metric is introduced and the criterion for whether the Voronoi region is bounded or unbounded is presented. The fact that a vertex $v$ of diagram is the common intersection of exactly $d+1$ Voronoi regions is proven. The empty-sphere and the nearest-neighbor properties are established. Finally, the Delaunay tessellation dual to the Manhattan and the supremum weighted VD is discussed.

3.3.1 Definition of the Weighted Voronoi Diagram in the Manhattan and the Supremum Metric and Preliminaries

The weighted Voronoi diagrams in the Manhattan ($L_1$) and the supremum ($L_\infty$) metrics also belong to the class of the additively weighted Voronoi diagrams. It will be shown that they are similar in many aspects. The term *rectangular weighted Voronoi diagram* will refer to these diagrams. The distance $d(x,p)$ between two points $x=(x_1,x_2,...,x_d)$ and $p=(p_1,p_2,...,p_d)$ in $\mathbb{R}^d$ is defined as $d(x,p) = \sum_{i=1}^{d} |x_i - p_i|$ in the Manhattan metric and as $d(x,p) = \max_{i=1...d} |x_i - p_i|$ in the supremum ($L_\infty$) metric.

Thus, the *sphere* in the Manhattan metric is a diamond, and the *sphere* in the supremum metric is a cube$^1$ (see Figure 3.3.1).

---

$^1$ In the rest of the chapter the term sphere is used since it is used to represent the topological rather than the geometrical object.
The additively weighted distance between a point $x$ and a sphere $P = \{p, r_p\}$ can be defined as $d(x, P) = d(x, p) - r_p$, where distance is defined in the Manhattan or the supremum metric (see Figure 3.3.2).

**Note.** A point $x$ lies on or inside the sphere $P$ if and only if the distance $d(x, P)$ is non-positive.

**Assumption 3.3.1** The spheres of the set $S$ do not intersect, i.e. $d(p, q) \geq r_p + r_q$ for any two distinct spheres $P = \{p, r_p\}$ and $Q = \{q, r_q\}$ from $S$ in $L_1$ ($L_\infty$) metric in $R^d$.

The weighted Voronoi region in $L_1$ and $L_\infty$ metrics is now defined:

**Assumption 3.3.2** No $d + 2$ spheres $P_1, P_2, \ldots, P_{d+2}$ are cospherical, i.e. there is no point $x \in R^d$ such as $d(x, P_1) = d(x, P_2) = \ldots = d(x, P_{d+2})$ in the Manhattan (supremum) metric.
**Definition 3.3.1** The distance between two spheres \( P = (p, r_p) \) and \( Q = (q, r_q) \) in \( L_1 \) (\( L_\infty \)) in \( R^d \) is defined as \( d(P, Q) = d(p, q) - r_p - r_q \).

An illustration of this definition for the supremum metric in the plane is given in Figure 3.3.3.

![Figure 3.3.3 Distance between two circles in the supremum metric](image1)

**Assumption 3.3.3** No two facets of spheres (cubes in \( L_\infty \) or diamonds in \( L_1 \) metric) belong to the same hyperplane (see Figure 3.3.4). No two pairs of spheres are equidistant (see Figure 3.3.5).

**Note.** When facets of two cubes or diamonds belong to the same hyperplane, their bisector can become a region in \( R^d \) (see Figure 3.3.4).

![Figure 3.3.4 \( L_\infty \) bisector is a region in the plane](image2)

![Figure 3.3.5 Two pairs of \( L_\infty \) spheres are equidistant](image3)
**Definition 3.3.2** A rectangular Voronoi region $RVor(P)$ of the sphere $P \in S$ in the $L_1$ ($L_\infty$) metric in $R^d$ is the set of points that are closer to $P$ than to any other sphere $Q \in S$: $RVor(P) = \{ x \in R^d \mid d(x, P) \leq d(x, Q), \forall Q \in S - \{P\} \}$, where the distance $d(\cdot, \cdot)$ is defined in the corresponding metric [Okabe et al. 92].

Under Assumption 3.3.1 each bisector $B(P, Q) = \{ x \mid d(x, P) = d(x, Q) \}$ between two distinct spheres $P$ and $Q$ divides the space into two quasi-halfspaces $H(P, Q)$ and $H(Q, P)$.

Thus, an equivalent definition of a rectangular Voronoi region $RVor(P)$ can be given as:

**Definition 3.3.3** A rectangular Voronoi region $RVor(P)$ of the sphere $P \in S$ in the $L_1$ ($L_\infty$) metric in $R^d$ is the intersection of all quasi-halfspaces $H(P, Q)$:

$$RVor(P) = \bigcap_{Q \neq P} H(P, Q)$$ [Okabe et al. 92].

**Definition 3.3.4** A rectangular weighted Voronoi diagram in $L_1$ ($L_\infty$) metric (RWVD) for the set of spheres $S$ in $R^d$ is the collection of all Voronoi regions

$$RWVD(S) = \{ RVor(P_1), ..., RVor(P_n) \}$$ [Okabe et al. 92].

Figure 3.3.3 shows the Manhattan Voronoi diagram for a set of sites (a) and the supremum Voronoi diagram for a set of sites (b) in the plane.

![Manhattan Voronoi Diagram](image)

(a) (b)

Figure 3.3.6 The RWVD in the Manhattan (a) and the supremum (b) metrics in the plane
The known properties of Voronoi diagrams in the Manhattan and the supremum metrics for a set of points in the plane follow. The bisector of planar VD consists of at most three straight-line segments and the Voronoi region is star-shaped region with respect to its generator [Lee and Wong 80]. The Voronoi region of a point site located on the boundary of the convex hull is an unbounded region in the Manhattan metric\(^1\) [Lee 80]. The empty-sphere property for Minkowski metrics was mentioned without proof in [Schaudt and Drysdale 92]. The properties about the form of the bisector and the Voronoi region were generalized to higher dimensions in [Okabe et. al 92]. The relationship between RWVD and convex hulls in higher dimensions has been established in [Boissonnat and Yvinec 98].

Other properties of the RWVD in \(R^d\) were not considered in the literature. They are established in the following section of the thesis. A few important observations can be made. First, the definition of the rectangular hull (analogous to the definition of the convex hull of a set of spheres in Euclidean metric) is introduced in order to obtain a criterion for whether a Voronoi region is bounded or unbounded. The rectangular hull is not convex, thus the proof of the criterion differs significantly from the proofs of the similar properties for the previously considered metrics. The proofs of the empty-sphere and nearest-neighbor properties are similar to the proofs presented for Euclidean VD. This can be explained by the fact that the Voronoi diagrams in both metrics belong to the class of additively weighted VD.

3.3.2 Properties of the RWVD in d Dimensions

The properties of the RWVD are similar to those of the Euclidean VD.

All the properties in this section are proven under Assumptions 3.3.1, 3.3.2 and 3.3.3. The basic properties of the EWVD that describe the form of the bisector and the

\(^1\) Note that converse statement was not proven even for 2D.
Euclidean region are first given (note that they have also been listed without proof in [Okabe et. al. 92]).

**Property 3.3.1** The bisector $B(P,Q)$ in $L_1$ ($L_\infty$) consists of a set of hyperplane facets in $R^d$ under Assumption 3.3.3.

**Proof** The proof is presented for the $L_1$ case. The proof for the $L_\infty$ case is similar.

Any point $x \in B(P,Q)$ satisfies the equation $d(x,P) = d(x,Q)$. This equation can be written as

$$
\sum_{i=1}^{d} |x_i - p_i| - \sum_{i=1}^{d} |x_i - q_i| - r_p + r_q = 0.
$$

Note that for any site $P$ the space can be split into $2^d$ unbounded regions (quadrants), each defined by a set of inequalities of the form $(-1)^s_i (x_i - p_i) \geq 0$, $s_i \in \{0,1\}$, $i = 1..d$. Inside each quadrant, the expression $x_i - p_i$ does not change sign from negative to positive or from positive to negative, and, therefore, $|x_i - p_i|$ can be replaced with the expression $(-1)^s_i (x_i - p_i)$.

Consider the $2^d$ regions obtained by intersecting quadrants corresponding to $P$ and $Q$ (note that some of them can be empty). Within each region, the bisector equation can be rewritten without absolute value signs. Under the Assumption 3.3.3 neither of the equations will become degenerate. This can be shown as follows.

Assume that

$$
\sum_{i=1}^{d} (-1)^{s_i} (x_i - p_i) - r_p = \sum_{i=1}^{d} (-1)^{s_i} (x_i - q_i) - r_q
$$

for all points $x$ within some region, i.e. all $x_i$'s can be cancelled from the equation. Then $s_i = t_i$, $i = 1..d$ (i.e. each $x_i$ has same sign in both LHS and RHS) and

$$
\sum_{i=1}^{d} (-1)^{s_i} p_i + r_p = \sum_{i=1}^{d} (-1)^{s_i} q_i + r_q.
$$

Then the two facets of $P$ and $Q$, which are normal to the vector $((-1)^{s_1}, (-1)^{s_2}, \ldots, (-1)^{s_d})$, belong to the same hyperplane, which contradicts the Assumption 3.3.3.
Thus, if the equation has a solution, then it will define a hyperplane. When this hyperplane is intersected with the region where the equation is defined, a \((d - 1)\)-dimensional facet is obtained, which is a part of the bisector inside this region. Thus, the whole bisector is a union of at most \(4^d\) such hyperplane facets.

\[\Box\]

*Note* The forms of the bisectors in the \(L_1\) and the \(L_\infty\) metrics are very similar. In the plane, the diagrams can be transformed into each other by \(45^\circ\) rotation (see Figure 3.3.7 and Figure 3.3.8 for an illustration). However, this transformation would not conserve the radii of spheres, since they would change by a factor of \(\sqrt{2}\). This simple relationship was also established in [Okabe et. al. 92].

![Figure 3.3.7 Bisectors in \(L_1\) in the plane](image)

![Figure 3.3.8 Bisectors in \(L_\infty\) in the plane](image)

The following properties are proven for the supremum metric. The proofs for the Manhattan metric are similar.

**Property 3.3.2** Each rectangular Voronoi region \(RVor(P)\) is a connected region in \(R^d\), containing the whole sphere \(P\). The region is star-shaped relative to the center of the sphere \(P\).
**Proof** To prove that the sphere $P$ lies entirely inside its rectangular Voronoi region $RVor(P)$, recall that the distance between any point $x$ lying on or inside the sphere and the sphere $P$ is zero or negative. Furthermore, for any point lying outside the sphere the distance is positive. Therefore, any point $x$ lying on or inside the sphere $P$ is closer to $P$ than to any other sphere $Q$, since $x$ cannot belong to both $P$ and $Q$ under Assumption 3.3.1. Consequently, the sphere $P$ lies completely inside its rectangular Voronoi region $RVor(P)$.

To prove the second statement, recall that the rectangular Voronoi region $RVor(P)$ can be defined as the intersection of quasi-halfspaces $H(P,Q)$. Since each of the quasi-halfspaces is a star-shaped region relative to $p$, their intersection is also a star-shaped region relative to $p$.

\[\]

*Note.* The rectangular Voronoi region is not necessarily convex, since the boundary of the Voronoi region is formed by bisectors, which in the general case are not hyperplanes.

Important properties of the EWVD are now proven under the assumptions stated in the beginning of this section. The notion of the rectangular hull in the Manhattan and supremum metrics is introduced. It is analogous to the notion of convex hull in Euclidean metric. The definition of a $k$-quadrant is first given.

**Definition 3.3.5** A $k$-quadrant $Q_k(P_1, P_2, ..., P_k)$, $1 \leq k \leq d$, constructed on sites $P_1, P_2, ..., P_k$, is an unbounded region defined by the intersection of $k$ mutually orthogonal halfspaces\(^1\) $H(P_{ij})$, $j = 1..k$, such that $H(P_{ij})$ is tangent to one of the facets of

\[\]

\(^1\) Two halfspaces are said to be orthogonal if the hyperplanes representing their boundaries are orthogonal. The set of halfspaces is mutually orthogonal if any two distinct halfspaces from the set are orthogonal.
the site\(^1\) \(P_i\) and does not contain the site. The axis of a \(k\)-quadrant is the set of points within the quadrant equidistant to all sites \(P_i, P_2, ..., P_k\) and to the boundaries of the halfspaces \(H(P_j), j = 1..k\).

A 1-quadrant is a half-space tangent to a site. An example of two 2-quadrants \(R_1\) and \(R_2\) corresponding to spheres \(P\) and \(Q\) in the supremum metric in the plane is given in Figure 3.3.9. The axis of a \(d\)-quadrant is a ray, equidistant to all coordinate hyperplanes. The axis of a \(k\)-quadrant, \(k \geq 2\) is a \((d-k+1)\)-dimensional half-hyperplane, and the axis of a 1-quadrant is a pyramidal region inside the quadrant.

![Figure 3.3.9 Two quadrants with axes for spheres \(P\) and \(Q\)](image)

**Note.** Because of Assumption 3.3.3 no site \(Q \in S\) distinct from \(P_i, P_2, ..., P_k\) can be tangent to the boundary of the quadrant \(Q_k(P_i, P_2, ..., P_k)\).

**Lemma 3.3.1** The \(k\)-quadrant \(Q_k(P_i, P_2, ..., P_k)\) is empty (i.e. it does not intersect any other sites from \(S\)) if and only if its axis is a part of an unbounded Voronoi facet \(RVor(P_i) \cap RVor(P_2) \cap ... \cap RVor(P_k)\).

\(^1\) Site means sphere.
Proof If \( Q_k(P_1, P_2, ..., P_k) \) does not contain any sites then it is claimed that its axis is a part of a Voronoi facet because the points of the axis are equidistant from \( P_1, P_2, ..., P_k \) and they are further from all other sites which lie outside the quadrant. To prove this claim, assume that there is a site \( Q \in S \) distinct from \( P_1, P_2, ..., P_k \) and a point \( a \) on the axis of the quadrant such that \( d(a, P) = \rho < d(a, P_k) \). Then \( a \) is the center of a sphere \( (a, \rho) \) lying completely inside the quadrant and tangent to \( Q \). Then \( Q \) intersects the quadrant, which contradicts the assumption that the quadrant is empty.

Conversely, assume that a quadrant is a part of an unbounded Voronoi facet. Consider an arbitrary point \( x \) on the axis. Then \( x \) is a center of an empty sphere inscribed between \( P_1, P_2, ..., P_k \). This sphere covers the corner of the quadrant. Since the point \( x \) is selected arbitrarily, the radius of the empty sphere is unlimited, and thus, the whole quadrant is covered by such empty spheres. Therefore, the whole quadrant must also be empty.

\[ \square \]

Definition 3.3.6 A complement of a rectangular hull \( RH'(S) \) of a set of spheres \( S \) is a connected unbounded region in \( R^d \) obtained by taking the union of all empty \( k \)-quadrants \( 1 \leq k \leq d \) for the spheres from the set \( S \).

The definition of the rectangular hull in \( L_1 \) (\( L_\infty \)) metric is given as a domain in \( R^d \) containing all spheres from \( S \) (see Figure 3.3.10).

![Figure 3.3.10 Rectangular hull for a set of spheres in the supremum metric](image)
Definition 3.3.7 A rectangular hull RH(S) of a set of spheres in L₁ (Lₘ) metric is a connected region such that \( RH(S) = R^d \setminus RH'(S) \).

Note 1. Equivalent definitions of the k-quadrant and the rectangular hull RH(S) in L₁ (Lₘ) metric can be given in terms of the point dominance relationship [Preparata and Shamos 86].

Note 2. A site \( P \in S \) touches the boundary of the RH(S) if and only if it touches some empty k-quadrant.

Theorem 3.3.1 A sphere \( P \in S \) touches the boundary of the rectangular hull RH(S) if and only if its Voronoi region RVor(P) is unbounded.

Proof A site \( P \in S \) touches the boundary of the RH(S) if and only if it touches some empty k-quadrant \( Q(P, P_2, \ldots, P_k) \). By Lemma 3.3.1, this quadrant is empty if and only if its axis is a part of an unbounded Voronoi face \( RVor(P) \cap RVor(P_2) \cap \ldots \cap RVor(P_k) \). Therefore, RVor(P) is also unbounded.

The bisector \( B(P, Q) \) in L₁ (Lₘ) consists of a set of hyperplane facets (see Property 3.3.1): \( B(P, Q) = \bigcup_{i=1}^{m} F_i \).

Lemma 3.3.2 If two bisectors \( B(P, Q) = \bigcup_{i=1}^{m} F_i \) and \( B(S, T) = \bigcup_{i=1}^{l} G_i \) (where \( P \neq S \) and/or \( Q \neq T \)) intersect at a point \( v \in F_k \), then \( v \) can not lie on the boundary of \( F_k \).

Proof Assume that \( v \) lies on the boundary of \( F_k \). Note that \( d(v, P) = d(v, Q) = d(v, S) \). Assume that \( P \neq S \) (\( Q \) might be equal to \( T \)). Since \( v \) lies on the boundary of \( F_k \), it also belongs to the boundary of some other facet \( F_{k1} \). Therefore, \( v \) is equidistant to two
distinct facets of either $P$ or $Q$. Assume it is equidistant to two distinct facets of $P$ $f_{P_1}$ and $f_{P_2}$ (see Fig. 3.3.11).

![Figure 3.3.11 Proof of Lemma 3.3.2](image)

$v$ is also equidistant to the facet $f_{Q_1}$ of $Q$ and the facet $f_{S_2}$ of $S$. Consequently, two pairs of facets are equidistant: $f_{P_1}$ and $f_{Q_1}$, and $f_{P_2}$ and $f_{S_2}$. Thus, two pairs of sites are equidistant, i.e. $d(P,Q) = d(S,Q)$, which contradicts Assumption 3.3.3.

**Theorem 3.3.2** A vertex $v$ of a RWVD is the common intersection of exactly $d+1$ rectangular Voronoi regions.

**Proof** Assume that $v$ is the intersection of more than $d+1$ rectangular Voronoi regions. Then $v$ is equidistant from more than $d+1$ spheres (which are the generators of the Voronoi regions). This contradicts the assumption that $d+2$ spheres cannot be cospherical (Assumption 3.3.2).

Next, assume that the vertex $v$ is the intersection of $k$ rectangular Voronoi regions, where $k < d+1$. Assume $k = d$, the proof for the remaining cases when $k < d$ is similar. Since $v$ is a vertex of the Voronoi diagram, it must belong to the intersection of $d-1$ bisectors $B_i = B(P_i, P), i = 2, d$. Recall that each bisector is a union of several hyperplanar facets $B_i = \bigcup_{j=1}^{m_i} F_{ij}$. $v$ belongs to the intersection of $(d-1)$ bisectors:
\( v \in \bigcap_{i=2}^{d} \left( \bigcup_{j=1}^{m_i} F_{ij} \right) = \bigcup_{k=1}^{d} \left( \bigcap_{i=2}^{F_{ij_k}} \right) \). According to the Lemma 3.3.2, two hyperplanar facets can not intersect by their edges. Then the intersection of \((d-1)\) hyperplanar facets defines a part of an edge or a facet of RWVD (the proof is similar to the proof of Theorem 3.1.2 for the power diagrams). Therefore, \( v \) must lie on an edge or a facet, which contradicts the fact that \( v \) is a vertex of RWVD.

**Definition 3.3.8** A sphere \( C = \{x, \rho\} \) inscribed among \( d+1 \) spheres \( P_1, P_2, ..., P_{d+1} \) in \( L_1 \) (\( L_\infty \)) metric is a sphere with center \( \xi = (\xi_1, \xi_2, ..., \xi_d) \) and radius \( \rho \), such that \( \rho = d(\xi, P_1) = d(\xi, P_2) = ... = d(\xi, P_{d+1}) \), where the distance is defined in the corresponding metric.

*Note.* The existence of the inscribed sphere for a set of spheres in the \( L_1 \) (\( L_\infty \)) metric is considered in Chapter 4, Section 4.6.

The spheres \( P_1, P_2, ..., P_{d+1} \) will be called spheres *cospherical* to the inscribed sphere \( C \) in the sequel.

**Definition 3.3.9** An inscribed sphere \( C \) is called *empty* in the \( L_1 \) (\( L_\infty \)) metric if no sphere from \( S \) intersects its interior.

**Theorem 3.3.3** (the empty-sphere property) Consider a vertex \( v \) of the RWVD \( v = RVor(P_1) \cap RVor(P_2) \cap ... \cap RVor(P_{d+1}) \). Then \( v \) is the center of a sphere \( C = \{v, \rho\} \) inscribed between \( P_1, P_2, ..., P_{d+1} \) in \( L_1 \) (\( L_\infty \)) and there is no sphere \( Q \in S \) different from \( P_1, P_2, ..., P_{d+1} \) intersecting \( C \) (i.e. \( C \) is empty).

**Proof** Assume that there exists a sphere \( Q \in S \) that intersects the inscribed sphere \( C \) (see Figure 3.3.12). Then, clearly, \( d(v, Q) \) is less or equal to \( d(v, P_1) \), which is equal to the radius of \( C \). Therefore, \( v \) is either closer to \( Q \) than to any of \( P_1, P_2, ..., P_{d+1} \) (which
contradicts the fact that $v$ belongs to $RVor(P_1)$ or $Q$ and all $P_1, P_2, ..., P_{d+1}$ are equidistant to $v$ (which contradicts Assumption 3.3.2).

\[ \]

**Figure 3.3.12 Illustration for the proof of Theorem 3.3.3**

**Theorem 3.3.4** (the nearest-neighbor property) If $Q \in S$ is the nearest neighbor of $P \in S$ then the rectangular Voronoi regions $RVor(Q)$ and $RVor(P)$ have a common facet.

\[ \]

**Figure 3.3.13 Illustration for proof of Theorem 3.3.4**

**Proof** Assume that the sphere $Q$ is the nearest neighbor of $P$ and that the rectangular Voronoi regions $RVor(P)$ and $RVor(Q)$ do not share an edge. Consider the segment $\overline{pq}$. Obviously, the endpoint $p$ belongs to $RVor(P)$ and the endpoint $q$ belongs to $RVor(Q)$. Since $RVor(P)$ and $RVor(Q)$ do not intersect, the segment $\overline{pq}$ must cross some Voronoi region $RVor(S)$ (Figure 3.3.13).
Denote a point of intersection of \( \overline{pq} \) with the bisector \( B(P, S) \) by \( x \). Obviously, since \( x \in RVor(S) \) and \( x \notin RVor(Q) \) then \( d(x, s) - r_s < d(x, q) - r_q \). Denote the point of intersection of segment \( \overline{ps} \) and \( B(P, S) \) by \( y \). It is easy to show that \( d(y, p) \leq d(x, p) \), i.e. that \( y \) is a closest to point \( p \) of the bisector. Then, \( d(y, p) - r_p \leq d(x, p) - r_p \). Similarly, \( d(y, s) - r_s \leq d(x, s) - r_s \) and, therefore, \( d(y, s) - r_s < d(x, q) - r_q \). Therefore, \( d(P, S) = (d(y, p) - r_p) + (d(y, s) - r_s) < (d(x, p) - r_p) + (d(x, q) - r_q) = d(P, Q) \), which contradicts the fact that \( Q \) is the nearest neighbor of \( P \).

\[ \]

### 3.3.3 The Delaunay Tessellation Corresponding to the Manhattan (Supremum) VD

The dual of the RWVD is now introduced.

**Definition 3.3.9** A Delaunay tessellation corresponding to rectangular weighted Voronoi diagram (RDT) is a diagram such that for each Voronoi vertex

\[
v = RVor(P_1) \cap RVor(P_2) \cap ... \cap RVor(P_{d+1})
\]

there is a simplex \( \{p_1, p_2, ..., p_{d+1}\} \) in the tessellation [Okabe et al. 92].

---

1 It can be shown that the segment \( \overline{pq} \) intersects the bisector \( B(P, S) \) at a single point only. However, this fact is not required for the proof of Theorem 3.3.4 and thus its proof is omitted.
Figure 3.3.14 The weighted VD and DT in the Manhattan (a) and the supremum (b) metrics

Figure 3.3.14 displays the Delaunay tessellation in the Manhattan (a) and the supremum (b) metrics in the plane. Note that the RDT of spheres can contain duplicate and/or intersecting faces (see Figure 3.3.15).

Figure 3.3.15 RWVD and corresponding Delaunay tessellation

The following property follows from Theorem 3.3.3 (the empty-sphere property).

**Property 3.3.3** The sphere inscribed between the sites that form a simplex in RDT is the empty sphere (in terms of Definition 3.3.9).

**Proof** Consider a vertex \( v \) of the RWVD \( v = RVor(P_1) \cap RVor(P_2) \cap \ldots \cap RVor(P_{d+1}) \).

According to Theorem 3.3.2 it exists and is uniquely defined. Then \( v \) is the center of a sphere \( C = \{v, \rho\} \) inscribed between \( P_1, P_2, \ldots, P_{d+1} \) comprising a simplex \( (P_1, P_2, \ldots, P_{d+1}) \) of the RDT. Then, according to Theorem 3.3.3 (the empty-sphere property), the inscribed sphere \( C \) is empty.

\[ \blacksquare \]
3.4 Concluding Remarks

The main results obtained in this chapter are now summarized. The concept of Voronoi region and inscribed sphere in $R^d$ was unified for different metrics. Important properties of the diagrams, including the nearest-neighbor and the empty-sphere properties, were proven for $d$ dimensions. Generalized Delaunay tessellations were also discussed.

The properties established in this chapter (in particularly, the empty-sphere property) are used in Chapter 4 for incremental construction algorithms and to obtain formulas for the INCIRCLE test. They are also used in Chapter 5 and Chapter 6 for other techniques for generalized VD construction. The methods for collision optimization presented in Chapter 8 and Chapter 9 rely on the nearest-neighbor property established in this chapter.

Based on the results obtained, the conclusion that properties of the generalized VD's in considered metrics are similar under different definitions of distance function can be made. An interesting topic to discuss is the relationships between Delaunay tessellations in different metrics. It can be concluded that these triangulations are also similar, since the Delaunay tessellations represent a metric-independent closeness relationship among the spheres. A more detailed consideration of relationships between Delaunay tessellations in different metrics will be provided in subsequent chapters.
CHAPTER 4: THE INCREMENTAL CONSTRUCTION OF THE GENERALIZED DELAUNAY TESSELLATION

The properties of generalized Voronoi diagrams and Delaunay tessellations, obtained in the previous chapter, are now applied to the development of methods for the VD and DT construction. This chapter outlines an incremental algorithm for the generalized Delaunay tessellation construction and presents the formulas for the INCIRCLE test computation for the Euclidean, power, Manhattan and supremum metrics in \( d \) dimensions [Gavrilova and Rokne 96, Gavrilova and Rokne 98]. The practical implementation of the incremental construction method for the power diagram using exact computation method concludes this chapter.

The generalized scheme of the incremental construction method is first given for the \( d \)-dimensional case. Note that this scheme combines results obtained in [Schaudt and Drysdale 92] for the Minkowski metrics and in [Edelsbrunner and Shah 96] for power metric in \( R^d \).

The main contribution of this chapter, however, is in presenting the explicit formulas for computation of the INCIRCLE test for the power, Manhattan, supremum and Euclidean metrics for the 2 and \( d \)-dimensional cases, which is the basic primitive for the incremental method. Note that the divide-and-conquer algorithm [Yvinec 88] is also based on the INCIRCLE test, thus the developed theory is applicable to the divide-and-conquer method in higher dimensions. The INCIRCLE test is also used in dynamic VD maintenance algorithms (see Chapter 9 of this thesis).

None of the papers surveyed have presented the explicit formulas for INCIRCLE test computation or even mentioned that the implementation of the algorithm requires these formulas. The formulas for INCIRCLE test for the ordinary VD in the plane were published in [Roos and Noltemeier 92] and for \( d \) dimensions in [Roos 97].

There are only a few papers devoted to the incremental construction method in higher dimensions. Inagaki successfully implemented the incremental method for Voronoi
diagram construction in 3D [Inagaki et. al. 92]. Choset considered a modification of the method for arbitrary shaped objects in \(d\)-dimensional Euclidean space as part of the motion planning problem [Choset 97]. Schaudt and Drysdale outlined a scheme that modifies Seidel's convex hull algorithm to compute Minkowski Delaunay tessellation in \(O(n^{\lceil(d+1)/2\rceil})\) time [Schaudt and Drysdale 92]. Edelsbrunner and Shah [Edelsbrunner and Shah 96] presented an incremental algorithm for all regular triangulations including the power DT. The worst-case time complexity of their algorithm is \(O(n^{\lceil(d+1)/2\rceil})\).

The divide-and-conquer technique in \(d\) dimensions remains practically unexplored, mostly due to the complexity of the merge step. Hazelwood studied this problem in [Hazelwood 88] and Gonzalez and Zheng discussed the divide-and-conquer approximation algorithm for space partitioning [Gonzalez and Zheng 93].

4.1 Chapter Overview

The scheme of the incremental algorithm for \(d\) dimensions is briefly outlined in the beginning of the chapter. The formulas for INCIRCLE test for the power, Manhattan, supremum and Euclidean metrics for the 2 and \(d\)-dimensional cases are then obtained.

The conditions for the INCIRCLE test for the power metric are obtained based on the system of quadratic equations that define the coordinates and the radius of the sphere inscribed among \(d+2\) spheres. The INCIRCLE function is written in a determinant form, which is represented as a 4th order polynomial of time in the 2-dimensional case [Gavrilova and Rokne 96]. The equations for the CCW (Counter-Clock Wise)\(^1\) orientation test for the power metric are also provided.

An approach based on the standard fractional transformation in the complex plane, used to obtain the INCIRCLE function for the Euclidean metric in 2D, is discussed next. The formulas for an inscribed sphere in \(d\) dimensions are provided. The important property that for any given \(d+1\) spheres in \(d\)-dimensional space, in general positions in the

\(^1\) For definitions of the CCW see, for example, [Guibas 91, Dwyer 92].
Euclidean metric, the number of inscribed spheres can either be two, one or zero is proven. The **INCIRCLE** function is obtained based on a set of quadratic equations. The resulting condition for the **INCIRCLE** test is represented as a 8th order polynomial of time in 2D [Gavrilova and Rokne 98].

The **INCIRCLE** test is obtained for the Manhattan and the supremum metrics for \( d \) dimensions. The formula for the radius of the inscribed sphere is first obtained. Then an incremental method for inscribed sphere construction based on the intersection of families of inscribed spheres is presented. The solution is written as a system of linear equations and inequalities. The formulas for the **INCIRCLE** test are also given.

The algorithm for the exact computation of the power DT in 2D using the fixed-precision floating-point arithmetic is discussed next. The practical implementation of the incremental construction method for power diagram using the developed algorithm concludes this chapter.

### 4.2 The Incremental Algorithm

This section briefly describes the general scheme for the incremental construction of the generalized DT suggested in [Schaudt and Drysdale 92] and [Edelsbrunner and Shah 96]. However, the algorithm description is not simply copied from these papers, but an attempt to emphasize the similarities in application of the incremental technique to generalized DT construction is made.

The input for the algorithm is a set of spheres in \( R^d \). The algorithm adds spheres to the diagram one by one. The correct Delaunay tessellation is maintained by performing an **INCIRCLE** test on sets of \( d + 2 \) neighboring spheres called \( d \)-dimensional quadrilaterals. The main scheme of the algorithm can be outlined as follows.
Algorithm 4.2.1

Preprocessing
1. Create a simplex $S_0$ large enough to contain all the sites of the input.

Main loop
2. For each site $P$ of the input set:
   
   2.1 Locate a simplex $S_i$ in the current diagram containing the center of $P$ by applying a point-location algorithm in a spatial subdivision;
   
   2.2 Insert site $P$ into the diagram by connecting it to all vertices of the simplex $S_i$;
   
   2.3 Inspect the newly created $d$-dimensional quadrilaterals to verify if they satisfy the empty-sphere condition by applying the INCIRCLE test;
   
   2.4 Perform swap operation on the quadrilaterals that do not satisfy the empty-sphere condition;
   
   2.5 Repeat steps 2.3-2.5 for all newly created $d$-dimensional quadrilaterals.

3. Delete the initial simplex $S_0$ together with edges connected to it.

4. Report the constructed diagram.

The worst-case time complexity of the algorithm is $O(n^{[d+1/2]})$ and the worst-case space complexity of the algorithm is optimal $O(n^{[d/2]})$. In the planar case, the algorithm runs in $O(n^2)$ time [Schaudt and Drysdale 92, Edelsbrunner and Shah 96].

It is important to note that some algorithms based on flipping technique can come to obstruction when the $d$-dimensional quadrilateral does not satisfy the empty-sphere criteria but no flip is possible [Joe 89, Edelsbrunner and Shah 96]. However, it has been established that when the sites are added one by one (i.e. in the incremental order) this situation can not take place and the above algorithm always works correctly under the
assumption of non-intersecting spheres [Schaudt and Drysdale 92, Edelsbrunner and Shah 96]."

4.3 The INCIRCLE Test

The incremental algorithm uses the metric-specific INCIRCLE test to correct the structure of the diagram after inserting a new site (Step 2.3 of the algorithm). The INCIRCLE test takes coordinates and radii of \(d+2\) spheres belonging to a \(d\)-dimensional quadrilateral and checks if the empty-sphere condition is satisfied.

Definitions of the empty sphere for all considered metrics are found in Chapter 3 (Definitions 3.1.6, 3.2.6, 3.3.9). The empty-sphere condition for \(d+1\) spheres states that a sphere \(C = \{\xi, \rho\}\) inscribed among \(d+1\) spheres which centers comprise a Delaunay tetrahedron is an empty sphere with the center at the corresponding Voronoi vertex. It has been proven that empty-sphere condition is satisfied for each simplex of the Delaunay tessellation for all considered metrics (see Chapter 3, Properties 3.1.3, 3.2.3, 3.3.3). Thus, to check if the empty-sphere condition is satisfied for a \(d\)-dimensional quadrilateral, the inscribed sphere is built on the first \(d+1\) spheres that belong to this quadrilateral and then the distance from its center to the \((d+2)\)-nd sphere is calculated\(^1\). If this distance is less than the radius of the inscribed sphere, then the empty-sphere condition is violated, and a swap operation must be performed on the quadrilateral. Note that the distance from the center of the inscribed sphere to the \((d+2)\)-nd sphere cannot be equal to the radius of the inscribed sphere since no \(d+2\) spheres are cospherical in any metric (due to Assumptions 3.1.2, 3.2.2. and 3.3.2).

The INCIRCLE function is then written as \(INCIRCLE(P_1, \ldots, P_{d+2}) = \rho - d(\xi, P_{d+2})\). The INCIRCLE test computes the value of the INCIRCLE function and compares it to zero. If the value of the function is positive then the empty-sphere condition is satisfied; if the

\(^1\) The construction of the inscribed sphere for each of the considered metrics will be discussed in details later in this chapter.
value is negative then the empty-sphere condition is violated. If the value of the \textit{INCIRCLE} function is equal to zero, then this is the degenerate case (when \( d + 2 \) spheres are cospherical in terms of the metric being used).

Formulas for the \textit{INCIRCLE} test for power, Euclidean and Manhattan (supremum) metrics are now obtained.

4.4 The \textbf{INCIRCLE} Test for the Power Metric

The formulas for the \textit{INCIRCLE} and the \textit{CCW} tests for the power metric are obtained based on a system of quadratic equations that define the coordinates and the radius of the sphere inscribed among \( d + 2 \) spheres. First, a formula for the \textit{INCIRCLE} function for the power metric in \( R^2 \) is obtained [Gavrilova and Rokne 96]. The formula for \( R^d \) is proven later.

4.4.1 The \textbf{INCIRCLE} Test for the Power Metric in the Plane

An example of four circles cocircular in the power metric is given in Figure 4.4.1.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure4.4.1.png}
\caption{Four circles are cocircular in the power metric}
\end{figure}

\textbf{Theorem 4.4.1} The \textit{INCIRCLE} function for a Delaunay quadrilateral of four circles \( P_i = \{(x_i, y_i), r_i\}, \ i = 1..4 \) in the power metric in the plane can be written as

\[
\text{INCIRCLE}(P_1, P_2, P_3, P_4) = \begin{vmatrix}
    x_1 & y_1 & w_1 & 1 \\
    x_2 & y_2 & w_2 & 1 \\
    x_3 & y_3 & w_3 & 1 \\
    x_4 & y_4 & w_4 & 1 \\
\end{vmatrix},
\]
where \( w_i = x_i^2 + y_i^2 - r_i^2, \ i = 1..4 \) if the following condition is satisfied\(^1\)

\[
CCW(P_1, P_2, P_3) = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} > 0.
\]

**Proof** The system describing the inscribed circle \( C \) with center \( \xi = (x_\xi, y_\xi) \) and radius \( \rho \) is given as follows

\[
\begin{cases}
(x_1 - x_\xi)^2 + (y_1 - y_\xi)^2 - r_1^2 = \rho^2 \\
(x_2 - x_\xi)^2 + (y_2 - y_\xi)^2 - r_2^2 = \rho^2 \\
(x_3 - x_\xi)^2 + (y_3 - y_\xi)^2 - r_3^2 = \rho^2
\end{cases}
\]

After a substitution \( \alpha = \rho^2 - x_\xi^2 - y_\xi^2 \), the system is transformed to a linear system in \( \{x_\xi, y_\xi, \alpha\} \). The following solution is obtained for the center of the inscribed circle using Cramer's rule

\[
x_\xi = \frac{1}{2} \begin{vmatrix} w_1 & y_1 & 1 \\ w_2 & y_2 & 1 \\ w_3 & y_3 & 1 \end{vmatrix}, \quad y_\xi = \frac{1}{2} \begin{vmatrix} x_1 & w_1 & 1 \\ x_2 & w_2 & 1 \\ x_3 & w_3 & 1 \end{vmatrix},
\]

where \( w_i = x_i^2 + y_i^2 - r_i^2, \ i = 1..4 \).

Then, \( \text{INCIRCLE}(P_1, P_2, P_3, P_4) \) can be written as

\[\text{INCIRCLE}(P_1, P_2, P_3, P_4) \]

---

\(^1\) If \( CCW(P_1, P_2, P_3) < 0 \) then the sites can be renumbered so that \( CCW(P_1, P_3, P_2) > 0 \).

The case \( CCW(P_1, P_2, P_3) = 0 \) is the degenerate case and is impossible due to Assumption 3.1.3.
The denominator of the resulting formula is the value of the CCW (Counter-Clock Wise) test on the circles $P_1, P_2, P_3$. Imposing the requirement on the numbering of these circles in counter-clockwise order guarantees that the denominator will be positive.

Thus, the $\text{INCIRCLE}$ function can be reduced to a single determinant
The theorem is proven.

4.4.2 The INCIRCLE Test for the Power Metric in d Dimensions

The proof presented in the previous section is now extended to the $R^d$ case.

**Theorem 4.4.2** The INCIRCLE function for a Delaunay quadrilateral of $d+2$ spheres $P_i = \{(x_i, y_i), r_i\}, i = 1..d+2$ in the power metric in $R^d$ can be written as

\[
\text{INCIRCLE}(P_1, P_2, \ldots, P_{d+1}, P_{d+2}) = \begin{vmatrix}
    x_1 & y_1 & w_1 & 1 \\
    x_2 & y_2 & w_2 & 1 \\
    x_3 & y_3 & w_3 & 1 \\
    x_4 & y_4 & w_4 & 1 \\
\end{vmatrix}.
\]

where $w_i = x_{i,1}^2 + x_{i,2}^2 + \ldots + x_{i,d}^2 - r_i^2, i = 1..d+2$ if the following condition is satisfied:

\[
CCW(P_1, P_2, \ldots, P_{d+1}) = \begin{vmatrix}
    x_{11} & x_{12} & \ldots & x_{1,d} & 1 \\
    x_{21} & x_{22} & \ldots & x_{2,d} & 1 \\
    \ldots & \ldots & \ldots & \ldots & \ldots \\
    x_{d+1,1} & x_{d+1,2} & \ldots & x_{d+1,d} & 1 \\
\end{vmatrix} > 0.
\]

**Proof** The system describing the inscribed sphere $C = \{\xi = (\xi_1, \xi_2, \ldots, \xi_d), \rho\}$ in $R^d$ is given as

---

1 Similarly to the planar case, the numbering on spheres can be introduced so that this condition is satisfied.
After a substitution $a = p - 5$, the system is transformed to a linear system of \( \{\xi_1, \xi_2, \ldots, \xi_d, \alpha\} \). Then the solution of the system can be written as

\[
\xi_i = \frac{1}{2} A_i, \quad i = 1, d,
\]

where

\[
A_i = \begin{bmatrix}
  x_{1,i} & x_{1,i+1} & \ldots & x_{1,d} \\
  x_{2,i} & x_{2,i+1} & \ldots & x_{2,d} \\
  \vdots & \vdots & \ddots & \vdots \\
  x_{d,i} & x_{d,i+1} & \ldots & x_{d,d}
\end{bmatrix}
\]

and

\[
w_i = x_{i1}^2 + x_{i2}^2 + \ldots + x_{id}^2 - r_i^2, \quad i = 1, d + 1.
\]

By moving the $w$ column in $A_i$ $(d - i)$ places to the right, the following representation is obtained: $A_i = (-1)^{d-i} B_i$, where

\[
B_i = \begin{bmatrix}
  x_{1,i} & x_{1,i+1} & \ldots & x_{1,d} & w_1 \\
  x_{2,i} & x_{2,i+1} & \ldots & x_{2,d} & w_2 \\
  \vdots & \vdots & \ddots & \vdots & \vdots \\
  x_{d,i} & x_{d,i+1} & \ldots & x_{d,d} & w_{d+1}
\end{bmatrix}, \quad i = 1, d
\]

Then, INCIRCLE\((P_1, P_2, \ldots, P_{d+1}, P_{d+2})\) can be written as
The numerator of the obtained formula can be written in the determinant form (note that it is the expansion of the following determinant by the first row).

\[
\begin{vmatrix}
    x_{d+2,1} - x_{11} & x_{d+2,2} - x_{12} & \cdots & x_{d+2,d} - x_{1d} & w_{d+2} - w_1 \\
    x_{11} & x_{12} & \cdots & x_{1d} & w_1 \\
    \vdots & \vdots & \ddots & \vdots & \vdots \\
    x_{d1} & x_{d2} & \cdots & x_{dd} & w_d \\
    x_{d+1,1} & x_{d+1,2} & \cdots & x_{d+1,d} & w_{d+1}
\end{vmatrix}
\]

By moving the first row \( d + 1 \) places down, the final form of the numerator is obtained as

\[
\begin{vmatrix}
    x_{11} & x_{12} & \cdots & x_{1d} & w_1 \\
    x_{21} & x_{22} & \cdots & x_{2d} & w_2 \\
    \vdots & \vdots & \ddots & \vdots & \vdots \\
    x_{d+1,1} & x_{d+1,2} & \cdots & x_{d+1,d} & w_{d+1} \\
    x_{d+2,1} & x_{d+2,2} & \cdots & x_{d+2,d} & w_{d+2}
\end{vmatrix}
\]
Note that the denominator $A$ of the obtained formula is the value of the $CCW$ test on the first $d+1$ spheres. The denominator $A$ is positive when the $d+1$ spheres are numbered in counter-clockwise order. In this case, the $INCIRCLE$ function can be reduced to a single determinant

$$INCIRCLE(P_1, P_2, \ldots, P_{d+1}, P_{d+2}) = \begin{vmatrix} x_{11} & x_{12} & \cdots & x_{1d} & w_1 & 1 \\ x_{21} & x_{22} & \cdots & x_{2d} & w_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ x_{d+1,1} & x_{d+1,2} & \cdots & x_{d+1,d} & w_{d+1} & 1 \\ x_{d+2,1} & x_{d+2,2} & \cdots & x_{d+2,d} & w_{d+2} & 1 \end{vmatrix}.$$ 

The theorem is now proven.

$\blacksquare$
4.5 The INCIRCLE Test for the Euclidean Metric

The INCIRCLE test for a set of spheres in Euclidean metric is now obtained. The important property that for any given \( d + 1 \) spheres in \( d \)-dimensional space in general positions in Euclidean metric the number of inscribed spheres can either be two, one or zero is also proven [Gavrilova and Rokne 98].

4.5.1 The INCIRCLE Test for the Euclidean Metric in the Plane

An interesting approach based on a transformation in a complex plane is used to obtain the INCIRCLE function for the Euclidean metric in the plane.

Figure 4.5.1 Four circles are cocircular in the Euclidean metric

The condition to determine if four circles are cocircular is now developed (see Figure 4.5.1).

Theorem 4.5.1 The INCIRCLE function for a Delaunay quadrilateral of four circles \( P_i = \{(x_i, y_i), r_i\}, i = 1..4 \) in Euclidean metric in the plane can be written as

\[
\text{INCIRCLE}(P_1, P_2, P_3, P_4) = \\
\left| \begin{array}{cccc}
    x_1 - x_4 & r_1 - r_4 & w_1 \\
    x_2 - x_4 & r_2 - r_4 & w_2 \\
    x_3 - x_4 & r_3 - r_4 & w_3 \\
\end{array} \right|^2 + \left| \begin{array}{cccc}
    y_1 - y_4 & r_1 - r_4 & w_1 \\
    y_2 - y_4 & r_2 - r_4 & w_2 \\
    y_3 - y_4 & r_3 - r_4 & w_3 \\
\end{array} \right|^2 - \\
\left| \begin{array}{cccc}
    x_1 - x_4 & y_1 - y_4 & w_1 \\
    x_2 - x_4 & y_2 - y_4 & w_2 \\
    x_3 - x_4 & y_3 - y_4 & w_3 \\
\end{array} \right|^2,
\]

when the following conditions are satisfied
Proof Assume that \( r_4 \) is the smallest radius of four circles \( P_1, P_2, P_3, P_4 \). Reduce the radii of all circles by value \( r_4' = r_i - r_4, i = 1..3 \). The resulting circles \( P_i^* = \{(x_i, y_i), r_i^*\} \), \( i = 1..3 \) and circle \( P_4^* = \{(x_4, y_4), 0\} \) are cocircular if and only if the original four circles were cocircular (see Figure 4.5.2).

The standard fractional transformation in the complex plane is now used (see [Marsden 73] for more information on complex transformations). A point in the plane with coordinates \((x, y)\) is associated with a complex quantity \( z = x + iy \). A circle \( P_j^* \) is associated with a circle \( Z_j = \{z_j = x_j + iy_j, r_j^*\}, j = 1..3 \). If there exists a circle \((z, r)\) that passes through point \( z_4 = x_4 + iy_4 \) and is tangent to \( (z_i, r_i^*), i = 1..3 \), then the original four circles are cocircular.

\[
\begin{vmatrix}
    y_1 - y_4 & r_1 - r_4 & w_1 \\
    y_2 - y_4 & r_2 - r_4 & w_2 \\
    y_3 - y_4 & r_3 - r_4 & w_3 \\
\end{vmatrix} = \begin{vmatrix}
    x_1 - x_4 & r_1 - r_4 & w_1 \\
    x_2 - x_4 & r_2 - r_4 & w_2 \\
    x_3 - x_4 & r_3 - r_4 & w_3 \\
\end{vmatrix} < w_i, i = 1..3,
\]

where \( w_i = (x_i - x_4)^2 + (y_i - y_4)^2 - (r_i - r_4)^2, i = 1..3 \).

Figure 4.5.2 Circles with reduced radii

The transformation \( W = W(Z) = \frac{1}{Z - z_4} \) maps the \( Z \) plane into a \( W \) plane [Grassmann and Rokne 78]. This transformation maps the circle \((z, r)\) onto a straight line, and the
three other circles \( Z_i = \{z_i, r_i^*\}, i = 1..3 \) into circles \( W_i = \{w_i, \rho_i\} \) (see Figure 4.5.3). Formulas for the centers and radii of the new circles are found as

\[
w_i = \frac{(z_i - z_4^*)}{k_i}, \rho_i = \frac{r_i^*}{k_i}, i = 1..3,
\]

where \( k_i = |z_i - z_4| - (r_i^*)^2 \) [Grassmann and Rokne 78].

Figure 4.5.3 Complex transformation

Returning to the original \( Z \) plane coordinates, the new circles can be represented as

\[
W_i = \{w_i = (u_i, v_i), \rho_i\} = \left\{ \left( \frac{x_i - x_4}{m_i}, \frac{y_i - y_4}{m_i} \right), \frac{r_i - r_4}{m_i} \right\},
\]

where \( m_i = (x_i - x_4)^2 + (y_i - y_4)^2 - (r_i - r_4)^2, i = 1..3 \).

Note that all \( m_i > 0 \), since the original circles do not intersect. If there exists a line

\[
l = \{au + bv + c = 0, \ a^2 + b^2 = 1\}
\]

that is tangent to circles \( W_i, i = 1..3 \), then original circles \( Z_i \) are cocircular. The condition that \( l \) is tangent to circles \( W_i, i = 1..3 \) can be written as

\[
a u_i + b v_i + c = \pm \rho_i, i = 1..3.
\]

The point in infinity is mapped into zero by transformation \( W = W(Z) \). The infinite point and all \( Z_i, i = 1..3 \) lie outside of the circle \( (z, r) \) in \( Z \) plane (since the circle \( (z, r) \) is inscribed between \( Z_i, i = 1..3 \)). Thus circles \( W_i, i = 1..3 \) and origin of the \( W \) plane must also lie on the same side of line \( l \). This can be formulated as a requirement that
expressions \( au_i + bv_i + c, i = 1..3 \) and \( a \cdot 0 + b \cdot 0 + c \) have the same sign, i.e.

\[
\frac{au_i + bv_i + c}{c} > 0, i = 1..3.
\]

This condition can be written as \( \frac{au_i + bv_i}{c} > -1, i = 1..3. \)

Thus, line \( l \) exists if a system of linear equalities and inequalities

\[
\begin{align*}
au_i + bv_i + c &= \pm \rho_i, \\
\frac{au_i + bv_i}{c} &> -1, i = 1..3
\end{align*}
\]

has a solution \((a, b, c)\) and \( a^2 + b^2 = 1. \)

The system of equations can be rewritten in matrix form as \( Ax = b \) where

\[
A = \begin{bmatrix}
    u_1 & v_1 & 1 \\
    u_2 & v_2 & 1 \\
    u_3 & v_3 & 1
\end{bmatrix}, \quad x = \begin{bmatrix}
    a \\
    b \\
    c
\end{bmatrix}, \quad b = \pm \begin{bmatrix}
    \rho_1 \\
    \rho_2 \\
    \rho_3
\end{bmatrix}
\]

From this linear system, a solution column \( x \) is found as \( x = A^{-1}b. \) By applying a Cramer's rule the solution for the linear system is found as

\[
a = \pm \frac{1}{|A|} \begin{vmatrix}
    \rho_1 & v_1 & 1 \\
    \rho_2 & v_2 & 1 \\
    \rho_3 & v_3 & 1
\end{vmatrix}, \quad b = \pm \frac{1}{|A|} \begin{vmatrix}
    u_1 & \rho_1 & 1 \\
    u_2 & \rho_2 & 1 \\
    u_3 & \rho_3 & 1
\end{vmatrix}, \quad c = \pm \frac{1}{|A|} \begin{vmatrix}
    u_1 & v_1 & \rho_1 \\
    u_2 & v_2 & \rho_2 \\
    u_3 & v_3 & \rho_3
\end{vmatrix}
\]

Note that \( |A| \neq 0 \) because it is assumed that the four original circles are cocircular and, consequently, the system \( Ax = b \) has a solution.

The solution that satisfies the system of inequalities and condition \( a^2 + b^2 = 1 \) is now selected. Substituting the formulas obtained for \( a \) and \( b \) in this equation the following formula is obtained

\[
\begin{vmatrix}
    \rho_1 & v_1 & 1 \\
    \rho_2 & v_2 & 1 \\
    \rho_3 & v_3 & 1
\end{vmatrix}^2 + \begin{vmatrix}
    u_1 & \rho_1 & 1 \\
    u_2 & \rho_2 & 1 \\
    u_3 & \rho_3 & 1
\end{vmatrix}^2 = \begin{vmatrix}
    u_1 & v_1 & 1 \\
    u_2 & v_2 & 1 \\
    u_3 & v_3 & 1
\end{vmatrix}.
\]

Returning to initial variables, it can be written as
The system of inequalities \( \frac{au_i + bv_i}{c} > -1, i = 1..3 \) is written as

\[
\begin{vmatrix}
  x_1 - x_4 & r_1 - r_4 & w_1 \\
  x_2 - x_4 & r_2 - r_4 & w_2 \\
  x_3 - x_4 & r_3 - r_4 & w_3 \\
\end{vmatrix}
+ \begin{vmatrix}
  y_1 - y_4 & r_1 - r_4 & w_1 \\
  y_2 - y_4 & r_2 - r_4 & w_2 \\
  y_3 - y_4 & r_3 - r_4 & w_3 \\
\end{vmatrix}
= \begin{vmatrix}
  x_1 - x_4 & y_1 - y_4 & w_1 \\
  x_2 - x_4 & y_2 - y_4 & w_2 \\
  x_3 - x_4 & y_3 - y_4 & w_3 \\
\end{vmatrix}.
\]

Thus, the condition of cocircularity for four circles is proven.

\[\blacksquare\]

### 4.5.2 The Inscribed Sphere for the d-Dimensional Case in the Euclidean Metric

A similar formula is now obtained for the \( d \)-dimensional case.

First formulas for a sphere inscribed between \( d + 1 \) spheres in \( d \)-dimensional space are obtained and a theorem giving the number of such inscribed spheres is proven.

Let \( P_i = \{p_i = (x_{i1}, x_{i2}, \ldots, x_{id}), r_i \}, i = 1..d+1 \) be \( d + 1 \) spheres in \( d \)-dimensional space. It will be shown how the coordinates and the radius of an inscribed sphere \( C = \{\xi = (\xi_1, \xi_2, \ldots, \xi_d), r \} \) can be obtained.

First, the radii of the \( d + 1 \) spheres are reduced by the radius of the smallest sphere. Without loss of generality assume that the smallest sphere has index \( d + 1 \). Define a coordinate system with the center of coordinates at point \( p_{d+1} \). Then the transformed coordinates of the given spheres are \( P_i^* = \{p_i^* = (x_{i1}^*, x_{i2}^*, \ldots, x_{id}^*), r_i^* \}, i = 1..d \), where \( x_{ij}^* = x_{ij} - x_{d+1,j}, i, j = 1..d \), and \( r_i^* = r_i - r_{d+1} \). The last sphere is transformed into a point
at the center of coordinates, i.e. \( P_{d+1}^{-*} = \{(0,0,\ldots,0)\} \). The unknown inscribed sphere coordinates will change to \( \xi_j^* = \xi_j - x_{d+1,j}, \) \( j = 1..d \), and \( \rho^* = \rho + r_{d+1} \).

The coordinates of the inscribed sphere satisfy the equations

\[
d^{(\xi^*,p_i^*)} = \rho^* + r_i^*, i = 1..d + 1.
\]

Expanding the distance function, the following system results

\[
\begin{align*}
\left( x_1^* - \xi_1^* \right)^2 + \left( x_2^* - \xi_2^* \right)^2 + \cdots + \left( x_d^* - \xi_d^* \right)^2 &= \left( \rho^* + r_i^* \right)^2, i = 1..d \\
\left( \xi_1^* \right)^2 + \left( \xi_2^* \right)^2 + \cdots + \left( \xi_d^* \right)^2 &= \left( \rho^* \right)^2
\end{align*}
\]

The last equation can be subtracted from the remaining equations to cancel the quadratic terms

\[
\begin{align*}
2x_{i1}^* \xi_1^* + 2x_{i2}^* \xi_2^* + \cdots + 2x_{id}^* \xi_d^* + 2\rho^* r_i^* &= w_i^*, i = 1..d \\
\left( \xi_1^* \right)^2 + \left( \xi_2^* \right)^2 + \cdots + \left( \xi_d^* \right)^2 &= \left( \rho^* \right)^2
\end{align*}
\]

where \( w_i^* = \left( x_{i1}^* \right)^2 + \left( x_{i2}^* \right)^2 + \cdots + \left( x_{id}^* \right)^2 - \left( r_i^* \right)^2, i = 1..d \).

The solution for the system can be obtained by the following steps. The first \( d \) equations are linear in \( (\xi_1, \xi_2, \ldots, \xi_d, \rho) \) by inspection.

This linear system has \( d \) equations and \( d+1 \) variables. Denote the matrix of the system by \( A \), the column of unknowns by \( x \) and the right-hand side column by \( b \).

The following 3 cases are possible:

**Case 1.** \( \text{rank}(A) = \text{rank}(A \mid b) = d \).

This is the general case. The linear system can be resolved for \( d \) of the variables leaving one of the variables as a free parameter. Assume that \( \xi_k^* \) is left as a free variable. To determine which variable is left free, a variable that can be moved into the right-hand side of the system must be found so that the determinant of the remaining system is non-zero. Thus, a non-zero \( [d \times d] \) minor of matrix \( A \) must be found.
The remaining unknowns will all be linear functions of $\xi_k^*$. They can be substituted into the last equation, which turns into a quadratic equation for $\xi_k^*$. It can have two, one or no real solutions. Consequently, the following statement is true:

**Property 4.5.1** The number of inscribed spheres in the Euclidean metric for a given $d+1$ spheres in $d$-dimensional space in general positions (i.e. $\text{rank}(A) = d$) can be either two, one or zero.

Note that even though the system can have up to two solutions, only those where the radius of the inscribed sphere is positive must be selected.

*Note.* Each of the inscribed spheres corresponds to a distinct Delaunay tetrahedron in the Delaunay tesselation.

*Case 2.* $\text{rank}(A) = \text{rank}(A \mid b) < d$.

In this case, the linear system has an infinite number of solutions, and, consequently, infinitely many inscribed spheres. An example of such a system is given in Figure 4.5.4.

![Figure 4.5.4](image-url)

*Figure 4.5.4* Infinite number of inscribed spheres

*Case 3.* $\text{rank}(A) < \text{rank}(A \mid b) \leq d$.

In this case, the linear system has no solutions, and consequently, there are no inscribed spheres. An example of such a system is presented in Figure 4.5.5.
Figure 4.5.5 Linearly dependent spheres

Note that cases 2 and 3 both represent degenerate sphere arrangements, because the spheres are linearly dependent when \( \text{rank}(A) < d \).

4.5.3 The INCIRCLE Test for the d-Dimensional Case in the Euclidean Metric

The INCIRCLE function for \( d+2 \) spheres can be obtained by determining the coordinates of the inscribed sphere(s) for the first \( d+1 \) spheres, and then computing the distance from the last sphere to the inscribed sphere(s):

\[
\text{INCIRCLE}(P_1, P_2, \ldots, P_{d+2}) = r - \left( \sqrt{\sum_{i=1}^{d} (\xi_i - x_{d+2,i})^2} - r_{d+2} \right),
\]

where \((\xi_1, \xi_2, \ldots, \xi_d, \rho)\) are the coordinates of the sphere inscribed among the spheres \(P_1, P_2, \ldots, P_{d+1}\).

The INCIRCLE function is often used as a part of the condition \(\text{INCIRCLE}(\ldots) = 0\). In particular, this condition is used extensively for dynamic maintenance of Voronoi diagrams of moving objects. This condition can be obtained in the explicit form as follows.

The condition can be rewritten as the following system of equations

\[
d(p_i, \xi) = r_i + \rho, i = 1..d + 2.
\]

If this system has a solution, then the condition is satisfied. Performing transformations similar to those described above reduces the radii of all spheres by the radius of the smallest sphere (assume that this is the \((d+2)\)-nd sphere). The center of the coordinates
is defined as the center of the \((d+2)\)-nd sphere. Then the second-degree terms in the first \(d+1\) equations can be cancelled

\[
\begin{align*}
\left(2x_{i1}^*\xi_1^* + 2x_{i2}^*\xi_2^* + \ldots + 2x_{id}^*\xi_d^* + 2\rho_i^* r_i^* \right) &= w_i^*, \quad i = 1..d+1 \\
\left(\xi_1^* \right)^2 + \left(\xi_2^* \right)^2 + \ldots + \left(\xi_d^* \right)^2 &= \left(\rho^* \right)^2
\end{align*}
\]

where \(w_i^* = \left(x_{i1}^* \right)^2 + \left(x_{i2}^* \right)^2 + \ldots + \left(x_{id}^* \right)^2 - \left(r_i^* \right)^2, \quad i = 1..d+1\).

The first \(d+1\) equations represent a linear system

\[
2\begin{bmatrix}
x_{11}^* & x_{12}^* & \ldots & x_{1d}^* & r_1^* \\
x_{21}^* & x_{22}^* & \ldots & x_{2d}^* & r_2^* \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
x_{d+1,1}^* & x_{d+1,2}^* & \ldots & x_{d+1,d}^* & r_{d+1}^*
\end{bmatrix}
\begin{bmatrix}
\xi_1^* \\
\xi_2^* \\
\vdots \\
\xi_d^* \\
\rho^*
\end{bmatrix}
= \begin{bmatrix}
w_1^* \\
w_2^* \\
\vdots \\
w_d^* \\
\rho^*
\end{bmatrix}.
\]

Assuming that the determinant of the linear system is non-zero\(^1\), the system will always have a unique solution. It can be explicitly written using Cramer's rule:

\[
\xi_i^* = \frac{1}{2} \frac{A_i}{A}, \quad i = 1..d; \quad \rho^* = \frac{1}{2} \frac{A_{d+1}}{A},
\]

where, returning to the original coordinates,

\[
A = \begin{bmatrix}
x_{11} - x_{d+2,1} & x_{12} - x_{d+2,2} & \ldots & x_{1d} - x_{d+2,d} & r_1 - r_{d+2} \\
x_{22} - x_{d+2,1} & x_{22} - x_{d+2,2} & \ldots & x_{2d} - x_{d+2,d} & r_2 - r_{d+2} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
x_{d+1,1} - x_{d+2,1} & x_{d+1,2} - x_{d+2,2} & \ldots & x_{d+1,d} - x_{d+2,d} & r_{d+1} - r_{d+2}
\end{bmatrix},
\]

---

\(^1\) The assumption of non-intersecting spheres guarantees at least that no sphere (except \((d+2)\)-nd sphere) transforms into a point at the center of coordinates (which would yield a zero row in the determinant).
and $A_i$ is obtained by replacing the $i$-th column of $A$ by the right hand side column

$$
\begin{bmatrix}
W_j \\
\end{bmatrix}_{j=1}^{d+1}
$$

$$
A_i = \begin{bmatrix}
x_{11} - x_{d+2,1} & x_{12} - x_{d+2,2} & \cdots & x_{1, i-1} - x_{d+2, i-1} & w_i & x_{1,i+1} - x_{d+2, i+1} & \cdots & x_{1d} - x_{d+2,d} & r_1 - r_{d+2} \\
x_{21} - x_{d+2,1} & x_{22} - x_{d+2,2} & \cdots & x_{2, i-1} - x_{d+2, i-1} & w_2 & x_{2,i+1} - x_{d+2, i+1} & \cdots & x_{2d} - x_{d+2,d} & r_2 - r_{d+2} \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
x_{d+1,1} - x_{d+2,1} & x_{d+1,2} - x_{d+2,2} & \cdots & x_{d+1, i-1} - x_{d+2, i-1} & w_{d+1} & x_{d+1,i+1} - x_{d+2, i+1} & \cdots & x_{d+1, d} - x_{d+2,d} & r_{d+1} - r_{d+2}
\end{bmatrix}
$$

where $w_i = \sum_{j=1}^{d} (x_{ij} - x_{d+2, j})^2 - (r_i - r_{d+2})^2$, $i = 1 \ldots d+1$.

Then the solutions obtained are substituted into the last quadratic equation, arriving at the condition

$$
A_1^2 + A_2^2 + \ldots + A_d^2 = A_{d+1}^2.
$$

This condition is the generalization of the formula obtained for the planar case using the transformation in the complex plane. The additional condition requiring that the radius of the inscribed sphere must be positive must be imposed as well

$$
\rho = \frac{1}{2} \frac{A_{d+1}}{A} - r_{d+2} > 0.
$$

Note. The INCIRCLE function is often used to predict topological events in a dynamic Voronoi diagram. When the spheres move by straight-line trajectories, the condition turns into an equation which is an 8th degree polynomial of time.
4.6 The Inscribed Sphere and the INCIRCLE Test in the Supremum Metric

This section discusses the construction of the inscribed sphere in the supremum ($L_\infty$) metric in $d$ dimensions. Similar considerations apply for the $L_1$ metric. The formula for the radius of the inscribed sphere is obtained first. The method for inscribed sphere construction, based on the intersection of families of inscribed spheres, is presented. The formulas for the INCIRCLE condition are also obtained.

4.6.1 Radius of the Inscribed Sphere

The distance between a point $x = (x_1, x_2, ..., x_d)$ and a sphere $P = \{ p = (p_1, p_2, ..., p_d), r_p \}$ in the supremum metric is defined as $d(x, P) = \max_{i=1..d} |x_i - p_i| - r_p$. Geometrically, a sphere in the supremum metric is a hypercube. A $d$-dimensional hypercube has exactly $2^d (d - 1)$-dimensional facets. Each of the $d$ axes has exactly two facets perpendicular to it. These geometrical properties of the spheres will be used in this section.

The formula for the radius of the inscribed sphere is now obtained. Consider $d + 1$ spheres $P_i = \{ p_i = (x_{i1}, x_{i2}, ..., x_{id}), r_i \}, i = 1..d + 1$. It will be shown how the coordinates and the radius of the inscribed sphere $C = \{ \xi = (\xi_1, \xi_2, ..., \xi_d), r \}$ can be obtained. Denote the distance between two spheres by $d(P_i, P_j) = d(p_i, p_j) - (r_i + r_j)$.

The formula is obtained under the non-degeneracy Assumption 3.3.3.

**Property 4.6.1** The radius of the sphere inscribed among $d + 1$ spheres $P_i, i = 1..d + 1$ can be obtained as $\rho = \frac{1}{2} \max_{i \neq j} d(P_i, P_j)$.

**Proof** By Definition 3.3.8 the inscribed sphere is tangent to each of $d + 1$ given spheres. Therefore, the diameter of the inscribed sphere is equal to the largest gap between any pair of spheres, computed over all pairs of spheres (see Figure 4.6.1).
Note. When the furthest pair of spheres is found (this can be done in $O(d^3)$ time), one of the coordinates of the center of the inscribed sphere is also determined. It is computed as the mid-point between the two furthest spheres.

Because of the non-degeneracy Assumption 3.3.3, no two spheres can be tangent to the same facet of the inscribed sphere and no two spheres can be tangent to opposite facets of the inscribed sphere with the exception of the furthest pair of spheres. Therefore, each of the remaining $d-1$ spheres is tangent to a facet, which is perpendicular to one of the $d-1$ remaining dimensions.

The following section presents the algorithm that can be used to determine the inscribed sphere for $d+1$ spheres in $d$ dimensions.

4.6.2 Determination of the Inscribed Sphere

The incremental method for inscribed sphere construction based on the intersection of families of inscribed spheres is presented. The solution is written as a system of linear equations and inequalities.

Renumber the spheres so that the radius-determining pair of spheres (furthest pair of spheres discussed in note after Property 4.6.1) is $(P_1, P_{d+1})$.

---

1 Remember that these spheres are cubes in the supremum metric.
**Definition 4.6.1** A *sphere family* $F^k$, $1 \leq k \leq d$, generated by the first $k$ spheres $P_1, P_2, ..., P_k$ is the set of all spheres $\{r = (\xi_1, \xi_2, ..., \xi_d), \rho\}$, such that radius $\rho > 0$ and each sphere in the family is inscribed between spheres $P_1, P_2, ..., P_k$ and is tangent to some fixed set of orthogonal facets $f_i \in P_1$, $f_i \in P_2$ ..., $f_i \in P_k$.

Consider first the sequence of sets of sphere families $S_k$, $k = 1..d$.

**Definition 4.6.2** A *set of sphere families* $S_k = \{F^k_1, F^k_2, ..., F^k_m\}$ comprises all families of spheres that can be inscribed into the first $k$ spheres $P_1, P_2, ..., P_k$, $1 \leq k \leq d$.

*Note.* A family of spheres contains an infinite number of spheres, while the set of sphere families contains a finite number of sphere families (i.e. it is set of sets).

A planar example is presented in the Figure 4.6.2. The set of sphere families $S_4$ contains four families of spheres growing in four different directions parallel to the coordinate axis. Each of the spheres inside one family touches a facet of the sphere $P_1$.

![Figure 4.6.2 Families of inscribed spheres for sphere $P_1$](image)

The addition of the sphere $P_2$ reduces the number of families of spheres inscribed between $P_1$ and $P_2$ to two families.
Two possible configurations are presented in Figure 4.6.3.

![Figure 4.6.3 Families of inscribed spheres for spheres $P_1$ and $P_2$](image)

Consider now the $d$-dimensional set $S_i$ that contains $2d$ sphere families, one for each of the facets. Each family has $d$ free variables, i.e. it can grow in $d$ independent directions. Each family of spheres $F_j^{(l)} = \{\xi = (\xi_1, \xi_2, ..., \xi_d, \rho)\}, \ j=1..2d$ is described by an equation in the form

$$\xi_l - x_{il} = r_l + \rho \quad \text{or} \quad x_{il} - \xi_l = r_l + \rho, \ l = 1..d,$$

plus a set of $d-1$ double inequalities in the form

$$-(r_l + \rho) < x_{im} - \xi_m < r_l + \rho, \ m \neq l,$$

which constrain the position of the family so that it touches the first sphere. A requirement that the radius $\rho$ is positive must be also added.

Next consider how the set of families $S_k$ changes into $S_{k+1}$ when the next sphere $P_{k+1}$ is added. The families are constrained so that they only contain spheres touching $P_{k+1}$ by imposing a condition on their coordinates. As a result some of the families disappear when the updated system of equations and inequalities does not have a solution. The families where the radius $\rho$ becomes fixed must also be excluded from the consideration. This is because representatives from the last family set $S_d$ with the predetermined radius
\( \rho = \frac{1}{2} d(P_1, P_{d+1}) \) will have to be selected as inscribed spheres (based on Property 4.6.1) \(^1\).

The remaining families \( F_j^{(k+1)} \) are attributed with an additional equation in the form

\[ \xi_{k+1} - x_{k+1} = r_{k+1} + \rho \text{ or } x_{k+1} - \xi_{k+1} = r_{k+1} + \rho, \quad l_{k+1} \not\in \{l_1, \ldots, l_k\}, \]

plus a set of inequalities constraining the position of the family so that it touches the \( P_{k+1} \) sphere. Note that each family is attributed with an equation constraining a new coordinate, i.e. \( l_{k+1} \not\in \{l_1, \ldots, l_k\} \), otherwise it would fix the value of the radius \( \rho \).

Thus, each family loses one free variable since the number of equations in each family increases by one. Therefore, at step \( k \) each family is defined by \( k \) equations and \( 2k(d-1) \) inequalities plus the condition \( \rho > 0 \).

The procedure must be repeated \( d-1 \) times. At each step, each family gets constrained by an additional equation, and a set of \( d-1 \) double inequalities. As a result, the final set \( S_d \) will contain up to \( d! \) families of spheres each with one free variable – the radius of the inscribed circle \( \rho \). Each of the families is described by \( d \) equations and \( 2d(d-1) \) inequalities plus the inequality \( \rho > 0 \). Now, a representative from each of the families with radius \( \rho = \frac{1}{2} d(P_1, P_{d+1}) \) is selected, so that it touches the last sphere \( P_{d+1} \).

*Note.* The number of spheres inscribed between \( d+1 \) sites in \( \mathbb{R}^d \) is bounded by \[ 2 \left\lfloor \frac{d-1}{2} \right\rfloor \]
for \( d \geq 3 \) and by two for \( d = 2 \) [Lambert 94].

An illustration of the fact that in the plane the total number of inscribed spheres can be zero, one or two is given in Figure 4.6.4.

\(^1\) Note that if a family fixes the radius, then this radius can not be equal to \( \rho = \frac{1}{2} d(P_1, P_{d+1}) \) due to the Assumption 3.3.3.
The inscribed spheres can be calculated by a deterministic algorithm. Note that each of the inscribed spheres corresponds to a distinct Delaunay simplex in the Delaunay tessellation. From the algorithm described above the following theorem can be formulated.

Theorem 4.6.1 Each inscribed sphere \( C = \{ \xi = (\xi_1, \xi_2, \ldots, \xi_d), p \} \) that exists for \( d + 1 \) spheres \( P_i = \{ p, (x_{i1}, x_{i2}, \ldots, x_{id}), r_i \}, i = 1..d + 1 \) in \( d \) dimensions can be found as the solution of the system of equations and inequalities of the form

\[
\begin{align*}
(-1)^{s_1} (x_{i1} - \xi_{i1}) &= \eta_1 + \rho \\
|x_{i1} - \xi_{i1}| &< \eta_1 + \rho \\
\vdots \\
|x_{i_d} - \xi_{i_d}| &< \eta_1 + \rho \\
\end{align*}
\]

\[
\begin{align*}
(-1)^{s_2} (x_{i2} - \xi_{i2}) &= \eta_2 + \rho \\
|x_{i2} - \xi_{i2}| &< \eta_2 + \rho \\
\vdots \\
|x_{i_d} - \xi_{i_d}| &< \eta_2 + \rho \\
\end{align*}
\]

\[
\begin{align*}
(-1)^{s_i} (x_{i_d}, i_i - \xi_{i_d}) &= \eta_i + \rho \\
|x_{i_d, i_i} - \xi_{i_d} - i_i| &< \eta_i + \rho \\
\vdots \\
|x_{i_d, i_d} - \xi_{i_d} - i_d| &< \eta_i + \rho \\
\end{align*}
\]

\[
\begin{align*}
\rho &= \frac{1}{2} d(P_1, P_{d+1}), \quad \rho > 0, \text{ where } s_1, s_2, \ldots, s_d \in \{0, 1\}, \text{ and } i_1, i_2, \ldots, i_d \text{ is some permutation of } d \text{ elements } \{1, 2, \ldots, d\}. 
\end{align*}
\]
Note. The system of $d$ linear equations and $2d(d-1)$ inequalities from Theorem 4.6.1 describes a family of spheres. Each equation depends only on one variable $\xi_{ij}$ and this variable does not appear in any other equation, i.e. the equations are linearly independent. Thus, the system can only have one or zero solutions. If it has one solution, then an inscribed sphere $C$ is found.

The $\text{INCIRCLE}$ function is then simply calculated as the distance from the $P_{d+2}$ sphere to the sphere $C = \{ \xi = (\xi_1, \xi_2, \ldots, \xi_d), \rho \}$ inscribed between $d + 1$ spheres $P_1, P_2, \ldots, P_{d+1}$, if such a sphere exists. Then, the $\text{INCIRCLE}$ function is written as

$$\text{INCIRCLE}(P_1, P_2, \ldots, P_{d+2}) = \rho - d(\xi, P_{d+2})^1.$$ 

For the 2-dimensional case Theorem 4.6.1 can be written as

Theorem 4.6.2 The inscribed sphere $C = \{ \xi = (\xi_1, \xi_2), \rho \}$ for 3 spheres $P_i = \{ x_{i} = (x_{i,1}, x_{i,2}), r_{i} \}_{i=1}^{3}$ in 2 dimensions can be found as the solution of the system of equations and inequalities of the form

$$\begin{cases}
(-1)^q (x_{i,j} - \xi_i) = r_1 + \rho \\
|x_{i,j} - \xi_i| < r_1 + \rho
\end{cases} \quad \text{and} \quad \begin{cases}
(-1)^s (x_{2,j} - \xi_j) = r_2 + \rho \\
x_{i,j} - \xi_i < r_2 + \rho
\end{cases},$$

$$\rho = \frac{1}{2} d(P_1, P_3), \quad \rho > 0, \text{ where } q, s \in \{0,1\}, \ i, j \in \{1,2\}, \ i \neq j.$$ 

Then, the $\text{INCIRCLE}$ function in the plane for four spheres $P_i, i = 1..4$ is represented as:

$$\text{INCIRCLE}(P_1, P_2, P_3, P_4) = \rho - d(\xi, P_4), \text{ where } C = \{ \xi = (\xi_1, \xi_2), \rho \} \text{ is the inscribed sphere for 3 spheres } P_1, P_2, P_3.$$ 

---

1 If there is more than one inscribed sphere, the $\text{INCIRCLE}$ function can assume multiple values, each corresponding to a distinct inscribed sphere. Note that if the inscribed sphere is empty then its center is a vertex of Voronoi diagram.
4.7 Exact Computation Method

Algorithms in computational geometry are usually designed under the assumption that numerical computation can be done precisely. When the methods are implemented, numerical computations are usually carried out in fixed length floating-point arithmetic, so that falsification of the computation (numerical error) is inevitable [Hoffmann 89]. This section discusses an example of an exact computation method. The method has been implemented and tested for power DT construction [Gavrilova et. al. 96, Gavrilova et. al. 98].

4.7.1 Literature Review

A number of methods have been proposed for dealing with numerical errors, see for example [Oishi and Sugihara 95]. One of the approaches is to obtain symbolic bounds on numerical errors so that the stability of computation can be guaranteed [Farouki 89]. Epsilon tolerance method employs a similar idea when two elements of the input set are considered to be at the same location when the distance between them is less than a tolerance $\varepsilon$ [Salesin 91]. Topology-oriented method ensures the consistency of the system topology during the process of computation [Sugihara 92, Sugihara and Iri 94]. The degeneracy-oriented method employs the idea of avoiding degenerate special cases that can be achieved, for example, by conceptual perturbation of the input data [Edelsbrunner and Mücke 88, Fortune 95, Yap 88].

The exact computation methods perform exact operations on the data [Jünger et. al. 91, Karasick et.al. 91, Fortune and Wyk 93]. This approach works under the reasonable assumption that data items under consideration are originally represented as fixed-precision floating-point numbers. Then, in each step of the algorithm, the exact values of all the components are calculated, which leads to the correct result [Fortune and Wyk 93,
Jünger et. al. 91]. The only question is how expensive this exact computation can be [Karasick et al. 91].

4.7.2 Main Scheme of the Algorithm

This section presents a new algorithm for the exact construction of the power DT in 2D using fixed-precision floating-point arithmetic. The approach is based on an algorithm for determining the sign of the sum of a finite set of real numbers [Ratscheck and Rokne 98]. The numbers are represented by normalized binary floating-point numbers of fixed mantissa length (fixed precision floating-point numbers) [Knuth 68].

The power diagram is constructed using the incremental method outlined in the beginning of this chapter. The method is based on the exact computation of the INCIRCLE test and the CCW test for power metric (see Section 4.4.1). First, an algorithm for the exact computation of the sign of a finite sum of real numbers is applied to the computation of the INCIRCLE test and the CCW test [Gavrilova et. al. 98]. Since only the fixed precision floating-point arithmetic is used, all of the computations are based on fast hardware operations. The method is modified by applying a floating-point filter based on interval analysis to improve the performance of the algorithm [Gavrilova et. al. 96]. The worst-case performance of the exact algorithm is the same as one of the direct (inexact) algorithm, that is $O(n^2)$. Tests have been conducted for various distributions of sites including some degenerate distributions. Experiments suggest that the expected running time of the algorithm is $O(n)$.

The ESSA algorithm

This algorithm exploits the idea that it is not always necessary to compute the value of a whole expression to get the value of its sign. The summands are represented as normalized single or double precision floating-point numbers with fixed mantissa length [Knuth 68]. The algorithm takes two sorted lists of floating-point numbers as an input: a list of positive
summands \( a_1 \geq a_2 \geq \ldots \geq a_m > 0 \), where \( m \geq 0 \), and a list of absolute values of negative summands \( b_1 \geq b_2 \geq \ldots \geq b_n > 0 \), where \( n \geq 0 \). The sum, the sign of which have to be determined, is \( S = \sum_{i=1}^{l} s_i = \sum_{i=1}^{m} a_i - \sum_{i=1}^{n} b_i \), where \( l = m + n \) is the total number of summands.

The exponential part of \( a_i, i=1..m \) is denoted by \( E_i \) and the exponential part of \( b_j, j=1..n \) is denoted by \( F_j \).

The main idea of the algorithm is based on a fact that it is possible to deduct the same value \( u \) from both \( a_1 \) and \( b_1 \) so that the sum \( S \) remains the same:

\[
S = \sum_{i=1}^{m} a_i - \sum_{i=1}^{n} b_i = (a_1 - u) - (b_1 - u) + \sum_{i=2}^{m} a_i - \sum_{i=2}^{n} b_i .
\]

By selecting the appropriate \( u \) one can decrease the length of the lists. The worst-case number of arithmetic operations required to compute the sum exactly is \( O(l^2 \log l) \), where \( l \) is the size of the list of summands [Ratscheck and Rokne 98]. The algorithm is now applied for exact computation of the \( CCW \) and \( INCIRCLE \) tests.

**Exact computation of the \( CCW \) test**

Let \( P_1, P_2, P_3 \) be three circles in the plane such that the coordinates of their centers \( p_i = (x_i, y_i), i=1..3 \) are represented as single precision floating-point numbers. The \( CCW \) orientation test is calculated as the sign of a 3x3 determinant

\[
CCW(P_1, P_2, P_3) = \begin{vmatrix}
1 & x_1 & y_1 \\
1 & x_2 & y_2 \\
1 & x_3 & y_3
\end{vmatrix} \quad \text{(see Section 4.4.1)}.
\]

Expanding the determinant a sum is obtained as

\[
x_1 y_2 + x_2 y_3 + x_3 y_1 - x_1 y_3 - x_2 y_1 - x_3 y_2 .
\]
Since $p_i = (x_i, y_i), i = 1..3$ are represented in single precision arithmetic, the products appearing in this sum can be computed exactly using double precision arithmetic [Fortune and Wyk 93]. Thus, a double precision version of ESSA when $l = 6$ is applied to the sum to obtain a correct sign.

**Exact computation of the INCIRCLE test**

The INCIRCLE test in power metric for four circles $P_1, P_2, P_3, P_4$ whose centers belong to a quadrilateral $p_1, p_2, p_3, p_4$ is depends on the sign of a 4x4 determinant:

$$INCIRCLE(P_1, P_2, P_3, P_4) = \begin{vmatrix} x_1 & y_1 & x_1^2 + y_1^2 - r_1^2 & 1 \\ x_2 & y_2 & x_2^2 + y_2^2 - r_2^2 & 1 \\ x_3 & y_3 & x_3^2 + y_3^2 - r_3^2 & 1 \\ x_4 & y_4 & x_4^2 + y_4^2 - r_4^2 & 1 \end{vmatrix},$$

where $p_i = (x_i, y_i), i = 1..4$ are the centers of circles, and $r_i, i = 1..4$ are the radii of the circles.

When the determinant is computed then 24 products of the form

$$x_i y_j (x_k^2 + y_k^2 - r_k^2) = x_i y_j x_k^2 + x_i y_j y_k^2 - x_i y_j r_k^2$$

result, where $i \neq j \neq k$ and $i, j, k = 1..4$.

First, consider the product of the form $x_i y_j x_k^2$. It requires quadruple precision representation for getting exact results with ESSA when the coordinates of points $p_i$, $i = 1..4$ are single precision quantities. However, those expressions could also be computed using four double precision quantities. This is done as follows. With $x_i, y_j, x_k$ in single precision compute the products $x_i y_j$ and $x_k^2$ in double precision. Represent each of the double-precision products $a$ as a sum of two single precision numbers by
splitting each of the double-precision numbers into two single-precision halves $(a)_1 + (a)_2$:

$$x_i y_j = (x_i y_j)_1 + (x_i y_j)_2, \quad x_k^2 = (x_k^2)_1 + (x_k^2)_2.$$  

Finally, execute the four products of the form

$$\left(x_i y_j\right)_\nu \left(x_k^2\right)_\mu$$

in double precision, where $\nu, \mu = 1, 2$. Thus the exact sum $x_i y_j x_k^2$ is computed. This required the addition of four summands.

The same operations are performed to compute products $x_i y_j y_k^2$ and $x_i y_j r_k^2$ exactly. The total number of products of the form $x_i y_j x_k^2$, $x_i y_j y_k^2$ and $x_i y_j r_k^2$ is $24 \times 3 = 72$. Thus the total number of double-precision quantities that need to be added to compute the determinant is $l = 72 \times 4 = 288$.

**Interval methods**

In order to achieve better performance, an interval arithmetic method is first applied to the sum as a filter. Interval methods substitute the operands with their inclusions and then perform operations on the intervals [Moore 66]. If $S^R[a,b]$ is an interval then $a$ and $b$ are the bounds of the interval, $a$ is the lower bound and $b$ is the upper bound.

The value of the first element of the sum is represented as an interval on the $x$ axis. The following summands are sequentially added until the resulting interval $S^R$ is obtained. This interval is an inclusion of the sum of the expression $S \in S^R$. The sign of the sum is positive if both bounds of the interval $S^R$ are positive, the sign of the sum is negative if both bounds of the interval $S^R$ are negative and the sum is equal to zero if both of the interval bounds are equal to zero. When $0 \in S^R$ no decision can be made about the sign of
the sum $S$ [Moore 66]. Then the ESSA is applied to compute the sign of the determinant exactly.

4.7.3 Implementation of the Algorithm

The method is now applied to the incremental construction of the power DT. The input for the algorithm is a set of weighted sites on the plane. The coordinates of their centers and their radii are given as double-precision floating-point numbers. The algorithm adds sites to the diagram one by one, maintaining the correct power DT. The worst-case performance of the exact algorithm is the same as for the inexact algorithm, that is $O(n^2)$, since the exact computation of each single primitive (i.e. CCW or INCIRCLE tests) takes $O(1)$ time (since $l$ is a constant). The Quad-edge data structure is used to represent the power DT and requires $O(n)$ space [Lischinski 94].

The algorithm was implemented in Object-Oriented Pascal in the Borland Delphi environment on IBM PC. The program runs under Windows 95. The experiments were conducted on a 486DX2/66 PC with 16 megabytes of RAM.

The program has a user-friendly interface. The coordinates of the centers of the sites and radii of the input set can be obtained directly from a standard input or generated by a special program. The user can set the parameters of this program to create the original distribution of sites. These parameters are the number of the input circles, distribution of their radii and the type of distribution. The first distribution considered is a uniform distribution of circles in the interior of a rectangle of the fixed size of 700 mm by 700 mm (millimeters) (further referenced as Random distribution). The three degenerate distributions were chosen similarly to the degenerate distributions considered in [Fortune and Wyk 93] for exact VD construction. They are Circle distribution when centers of the sites located on a boundary of a circle with 350 mm radius, Square distribution when centers of the sites located on a boundary of a square of 700 mm by 700 mm size and Grid
distribution when the centers of the sites located at nodes of the rectangular grid (its size is determined by the number of sites). To ensure the degeneracy for the three degenerate distributions the radius of one site can be selected by the user and the radii of all other sites are automatically assigned to be equal to the given radius.

For all distributions, except for the random one, the user can set the perturbation $P$ of the coordinates of the centers of the sites from their original positions. The value of $P$ is chosen by user from $10^{-9}\text{ mm}$ to $0.1\text{ mm}$. Then the program generates two random numbers in the range from $0$ to $P$ for each site and adds them to the $x$ and $y$ coordinates of the center of the site.

![Figure 4.7.1 Sample output for a set of 100 sites](image)

When the process of input generation is completed, the exact computation algorithm constructs the correct triangulation and displays it on the screen. The direct algorithm is also implemented and can be applied to the same input set to demonstrate the difference in
the resulting triangulations. The user has an option not to use the inexact algorithm. If triangulations were constructed by both methods, two resulting triangulations are displayed on screen. The wrong edges, produced by the direct algorithm, are highlighted by the thick lines. The exact algorithm always produces the resulting diagram in the thin lines. The example of the program interface for degenerate Grid distribution of 100 circles with radii equal to 1 mm is given in Figure 4.7.1.

The statistics are shown in the interface window. For the exact algorithm the statistics includes the number of CCW orientational tests, the number of INCIRCLE tests, the number of times when ESSA subroutine was called, the number of degenerate cases encountered, and the elapsed time. For the direct algorithm the program counts the number of CCW tests, the number of INCIRCLE tests, the number of degenerate cases encountered, the number of incorrect edges constructed, and the elapsed time.

4.7.4 Test Results

The performance of the exact algorithm is compared with the performance of direct implementation of the incremental method for power DT construction. The number of ESSA tests performed, the time required for the exact computation method in seconds (Texact), the time required for the direct computation algorithm in seconds (Tdirect), and the number of wrong edges produced by direct algorithm were recorded. The time ratio between the direct and exact algorithms was computed as \[ \text{Tratio} = \frac{\text{Texact}}{\text{Tdirect}} \]

The first series of experiments were performed on Random distribution of sites with radii varying from 1 to 10 mm for 100 to 1000 sites. The percentage of wrong edges computed by the direct method was approximately 1% of all edges. The time required for the direct method to construct power DT for 100 sites is 2.97 sec., for 500 sites is 5.98 sec. and for 1000 sites is 11.09 sec. The value of Tratio grows slowly for Random distribution and
tends to 4, which can be justified since the interval filter implementation requires 4 times more multiplication operation than the direct implementation.

The second series of experiments were conducted to investigate how the performance of the exact algorithm on degenerate distributions is affected by the number of sites in the input set (see Figure 7.4.2). The sites of the input set had the same radii equal to 1 mm. The time required for the direct method to construct power DT for 100 sites for all degenerate distributions was approximately 3 sec., for 500 sites was approximately 6 sec. and for 1000 sites was approximately 12 sec. (the actual recorded values differ from the approximate values by no more than 15%). The time ratio for the Circle distribution is much higher that that of the other degenerate distributions since the interval filter almost never gives the conclusive answer for INCIRCLE tests.

Figure 4.7.2 The time ratio vs. the number of sites

The percentage of wrong edges as a function of number of sites also tends to a constant (see Figure 4.7.3). One can note that for the Circle distribution the percentage of wrong edges is much higher than for other distributions.
Figure 4.7.3 The percentage of wrong edges vs. the number of sites

The direct algorithm simply fails to produce a reliable result, generating 30% of wrong edges. Hence, the computational expense of the exact algorithm pays off by correcting more edges in the triangulation.

In the third series of experiments the algorithm was tested on all four distributions for 100 input sites of the same radii equal to 1 mm. The value of perturbation parameter was increased from $10^{-9}$ to 0.1 mm. The time required for direct algorithm for any perturbation value for all distributions was $= 3$ sec. (the results recorded differed from this approximate value by no more than 5%). Figure 4.7.4 demonstrates the dependence of the time ratio on the perturbation. It can be seen from Table 4.7.4 that the ratio practically remains constant after perturbation reaches some value (1E-6). Another observation that can be made is that for Circle distribution the ratio is much higher than that of the other distributions. This can be explained by the fact that any four sites in Circle distribution are “almost” cocircular, hence the interval filter gives the inconclusive result for almost all INCIRCLE tests.
Figure 4.7.4 The time ratio vs. the perturbation

Figure 4.7.5 demonstrates the dependence of the number of wrong edges (computed by the direct algorithm) of the perturbation. The number of wrong edges increases significantly with decreasing of the perturbation parameter (since the direct algorithm encounters more situations when the calculation of the INCIRCLE test returns zero). The number of wrong edges reaches almost 50% for $P = 10^{-9}$ mm in the Circle distribution. Even though this distribution is the worst in the sense of the efficiency, this pays off in correcting of almost half of edges in the triangulation.

Figure 4.7.5 The percentage of wrong edges vs. the perturbation
Thus, the following conclusions can be drawn. The time required to perform the exact computation is proportional to the time of direct algorithm and the factor depends on the data distribution. For random distribution of input data this factor is approximately 4. The exact algorithm corrects on average 1% of edges for the random distribution and up to 50% edges for some degenerate distributions. The most time was required as well as the most wrong edges were corrected for the Circle distribution.

4.8 Concluding Remarks

This chapter presents the formulas obtained for the \textit{INCIRCLE} test. These formulas can be used not only for the incremental method, but also for the divide-and-conquer algorithm. In this thesis, the obtained formulas are used in Chapter 6 for the algorithm that transforms VD to power diagram in the plane. They are also an important part of the dynamic Voronoi diagram maintenance algorithm (see Chapter 8 and Chapter 9).

The last section of this chapter presents an incremental construction method for the planar power diagram using exact arithmetic. This section gives an example of how one can deal with degenerate situations by applying the exact computation method.
CHAPTER 5: THE SWEEP-PLANE ALGORITHM IN THE MANHATTAN AND THE SUPREMUM METRICS

This chapter presents a new algorithm for planar Voronoi diagram construction in the Manhattan and supremum metrics for weighted sites in the plane based on the sweep-plane technique. It is shown that algorithm run in optimal $O(n \log n)$ time. The implementation of the algorithm and test results are presented at the end of this chapter.

The possibility of the extensions of the sweep-plane technique to weighted point sites was originally discussed by Fortune in [Fortune 861]. The distance was defined as the sum of the weight of the site and the Euclidean distance from the point in the plane to the site. This approach results in the construction of an additively weighted Voronoi diagram of circles in the plane, i.e. the Euclidean Voronoi diagram. It was proven that it can be done in optimal $O(n \log n)$ time. Rosenberger [Rosenberger 91] modified Fortune's sweep-plane algorithm to construct Voronoi diagram for additively weighted sites in Euclidean space. Skyum [Skyum 91] studied the application of the sweep-plane algorithm to some generalized Delaunay triangulations.

Shute [Shute et. al. 91] suggested an original sweep-line algorithm that is different from Fortune's sweep-plane method for $L_\infty$ Delaunay triangulations of point sites. The authors introduced a sweep line parallel to the side of a $L_\infty$ square (i.e. parallel to the coordinate axis). It differs from Fortune's sweep-plane method, which introduces the sweep plane that is related to the correspondence between a cone in 3D and its projection onto the plane. Consequently, activation and deactivation events in Shute's method are different from the site and break events of Fortune's method. Finally, the authors of [Shute et. al. 91] redefined the Delaunay triangulation so that the inscribed circle (square in $L_\infty$) is not the maximum inscribed circle, which resulted in some edges missing on the boundary of the triangulation. The claim was made, however, that those edges can be easily added.

The method has been generalized to Voronoi diagrams with any polygonal distance function in [McAllister et. al. 93, McAllister et. al. 96]. The Manhattan and supremum distance functions can be considered as some specific case of polygonal distance
function, however the method requires some modifications to become applicable for the construction of the Manhattan weighted Voronoi diagram. There are some other differences that allow considering the results presented in this chapter as a contribution. Thus, the theorems established for the sweep-plane method in Manhattan metric in this chapter are proven differently from the similar properties established in [McAllister et. al. 96]. The implementation of the method is different: the special type of events – break events – are introduced in this chapter; the wave front is stored as a collection of collision points (opposite to the sweep front represented as a collection of arcs in [McAllister et. al. 96] implementation). This decision also influences the attributes of the events from the event queue (see Section for more details). Also note that results presented in this chapter were obtained independently of the results presented in [McAllister et. al. 96].

5.1 The Sweep-Plane Technique

The method presented in this chapter is based on Fortune’s sweep-plane technique. The basic idea behind the algorithm for point sites can be described by modeling the waves that result from simultaneously dropping $n$ pebbles onto a calm water surface. The position of each pebble at the moment when it hits the water represents a site of the Voronoi diagram. Circular waves originating at each site travel with the same speed and will first meet at a point equidistant from two corresponding sites. Further intersection of corresponding waves forms a line, which represents a bisector between two sites. Therefore, a Voronoi vertex corresponds to a point where three waves meet. The algorithm can more easily be described by adding a third dimension, which represents time, along the $z$ axis, thus transforming the circular waves to convex cones whose axes are parallel to the $z$ axis. The resulting Voronoi diagram is then obtained as the projection of the intersection between the cones and a sweep plane, angled to be parallel with the sides of the cones.

When generalizing this algorithm to $L_1$ and $L_\infty$ metrics the cones are represented by inverted geometric pyramids with square bases. Following is the description of the
sweep-plane algorithm for constructing the Voronoi diagram in the $L_1$ metric. The algorithm for the $L_\infty$ metric is very similar (see Chapter 3, Section 3.3.2).

5.2 The Sweep-Plane Algorithm in the Manhattan Metric for Weighted Sites in the Plane

The sweep-plane algorithm for VD construction in Manhattan metrics for weighted sites in the plane is now presented. The waves are modeled in the $L_1$ metric. The form of a wave in the $L_1$ metric is a diamond (i.e. a square with sides angled at 45° to the coordinate axis). A curve traced by the intersection of two waves represents a bisector between two sites in $L_1$. It consists of two unbounded rays parallel to the coordinate axes and one line segment angled at 45° to the $x$ axis. Each Voronoi vertex corresponds to a point where three waves meet.

The time is represented as a third dimension directed along the $z$ axis (see Figure 5.2.1).

Figure 5.2.1 Pyramids in 3D in Manhattan metric

---

1 Under the non-degeneracy assumption that the sites do not lie on a line slanted at 45° to the coordinate axis.

2 The converse is not true. Three waves can meet at a point not corresponding to a Voronoi vertex (this occurs when another wave has already passed through this point).
The waves are transformed into *pyramids*, identical in size and differing only in their positioning. Each pyramid originates from the \(xy\) plane at one of the sites of the Voronoi diagram if the weight of the site is equal to zero or from below the \(xy\) plane in the positions corresponding to the non-zero weight. The equation of the pyramid corresponding to the wave originating at the site \(P\) is \(z = d((x, y), P)\), where 
\[
d((x, y), P) = |x - x_p| + |y - y_p| - r_p\]
in the Manhattan metric.

The edge of the pyramid is angled at 45° to the plane. The pyramid corresponding to a site \(P = \{(x_p, y_p), r_p\}\) intersects the \(xy\) plane at the boundary of the site \(P\). Thus the vertex of the pyramid is located at the point \((x_p, y_p, -r_p)\). The waves at time \(t\) can be obtained as the intersection of pyramids with the plane \(z = t\).

Place an observer standing infinitely far below the \(xy\) plane. Assume that the pyramids are opaque. Then the observer sees only the lowest surface of the intersection of the pyramids, i.e. the projection of their intersection on the plane. In the \(L_1\) metric the projection of the intersection of two pyramids consists of two unlimited rays parallel to the coordinate axes and one line segment angled at 45° to the axes in the plane. For the two corresponding sites this is just the bisector in the Manhattan metric (see Chapter 3, Section 3.3). Since every Voronoi edge is a segment of such a bisector, every Voronoi edge is contained in the projections of the intersections of the pyramids.

**Lemma 5.2.1** All Voronoi edges can be obtained by projecting the lowest pyramid intersection visible to the observer located below the \(xy\) plane.

**Proof** Consider three pyramids originating at sites \(P = \{p, r_p\}\), \(Q = \{q, r_q\}\) and \(R = \{r, r_r\}\). Consider the point \(w = (x, y, z_p)\) which belongs to the intersection of two pyramids originated by sites \(P\) and \(R\) and such that \(w\) does not belong to the lowest intersection of pyramids (see Figure 5.2.2).

It will be shown that the projection \(x = (x, y)\) of \(w\) on the \(xy\) plane does not belong to any Voronoi edge.
Since $w$ does not belong to the lowest intersection of pyramids, it lies in the interior of some pyramid $Q$. Consider a point $w_1 = (x, y, z_Q)$ of intersection between the line passing through points $w$ and $x$ and the pyramid originated at $Q$. By the definition of pyramid, $z_P = d(x, P) = d(x, R)$ and $z_Q = d(x, Q)$. Since $z_Q < z_P$, point $x$ is closer to $Q$ then to either $P$ or $R$. Thus $x$ lies strictly within the Voronoi region of $Q$ and does not belong to the Voronoi edge. Since every Voronoi edge is contained in the projections of the intersections of pyramids, thus it can be concluded that every Voronoi edge is contained only in the projection of the lowest pyramid intersections.

Now, consider waves originated by the sites $P$, $Q$ and $R$. When two waves from neighboring sites first meet, they start a bisector, i.e. a set of points equidistant from both sites. As the waves progress with time, they trace out an edge of the Voronoi diagram. When two bisectors meet\(^1\), i.e. all three waves from the sites $P$, $Q$ and $R$ meet at some point $v$, then this point becomes equidistant from all three sites. Thus point $v$ represents a vertex of the Voronoi diagram.

---

\(^1\) In some cases two bisectors do not meet, for example if $p$, $q$ and $r$ belong to the convex hull of Voronoi sites and Voronoi edges define the infinite Voronoi regions.
5.3 The Sweep Plane

**Definition 5.3.1** A *sweep plane* $p_s$ is a plane in 3D that intersects the $xy$ plane at a line parallel to the $y$ axis, and is angled at $45^\circ$ to the $xy$ plane.

**Definition 5.3.2** A *sweep line* $l_s$ is the line obtained by the intersection of the sweep plane and the $xy$ plane.

The equation of the sweep line is $x = x_s$, and the equation of the sweep plane is $x + z = x_s$.

**Definition 5.3.3** A *parabola* in the $L_1$ metric defined by a focus $p$ and a directrix $l$ is the locus of points $x$ in the plane equidistant (in the sense of $L_1$ metric) from $p$ and $l$.

The parabola in $L_1$ metric with focus $p$ is shown in Figure 5.3.1.

![Figure 5.3.1 The parabola in $L_1$](image)

**Lemma 5.3.1** The projection of the intersection between the sweep plane and a pyramid corresponding to a wave originating at $P = \{(x_p, y_p), r_p\}$ is a parabola with focus $(x_p, y_p)$ and directrix $l : (x = x_s + r_p)$ (see Figure 5.3.2).

**Proof** The equation of the pyramid corresponding to the wave originating at $P$ is $z = d((x, y), P)$, where $d((x, y), P) = |x - x_p| + |y - y_p| - r_p$ in the Manhattan metric (see Chapter 3).
Thus, any point \((x, y, z)\) on the intersection of the sweep plane and the pyramid satisfies two equations:

\[
\begin{align*}
  z &= d((x, y), P) \\
  x + z &= x_s
\end{align*}
\]

The projection of the point \((x, y, z)\) onto the plane \(xy\) is the point \((x, y)\). From the above equations it follows that \(x + d((x, y), P) = x_s\) and \(d((x, y), (x_p, y_p)) = x_s - x + r_p\).

Note that \(x_s - x + r_p\) represents the distance between the point \((x, y)\) and the line \(l: (x = x_s + r_p)\), which is parallel to the sweep line. Therefore, \(d((x, y), P) = d((x, y), l)\), i.e. the point \((x, y)\) belongs to the parabola with focus \((x_p, y_p)\) and directrix \(l\).

Conversely, any point \((x, y)\) on the parabola in the \(xy\) plane is the projection of the point \((x, y, d((x, y), P))\). This point \((x, y, d((x, y), P))\) belongs to the pyramid and to the sweep-plane\(^1\). Thus, any point \((x, y)\) on the parabola is the projection of the point on the intersection of the pyramid and the sweep-plane.

\[\square\]

\(^1\) This can be easily shown by directly substituting the coordinates of this point into the equation of the pyramid and a sweep-plane.
Consider now how the parabolas change during the sweep. The sweep plane moves from left to right. It starts with its intersection with the xy plane at the minimum y coordinate of a center of the site, and finishes at the maximum y coordinate of a site. When the sweep plane first meets a pyramid, the parabola originates as a ray, i.e. the parabola with the focus located on the directrix.

**Definition 5.3.4** An axis of a parabola is the line through the focus \( p \) and perpendicular to the directrix \( l \).

All of the parabolas have axes parallel to the \( x \) axis. The parabola widens as the sweep plane moves through the pyramid (see Figure 5.3.3\(^1\)). The following observations follow directly from the parabola definition. The tip of the parabola moves with a velocity which is equal to the half of the sweep-plane velocity, since the tip \( a \) of the parabola is located in the middle point of the segment \( pa' \) (orthogonal to the directrix \( l \)) between the focus of the parabola \( p \) and its directrix \( l \). The parabola widens with the velocity equal to the sweep-plane velocity since the distance between the point \( b \) (located on the intersection of the vertical line passing through the focus \( p \) and the parabola) and the focus of the parabola \( p \) is equal to the distance from the point \( b \) to the directrix \( l \).

Figure 5.3.3 The parabola widens as the sweep plane moves through the pyramid

\(^1\) For simplicity, this figure presents two parabolas constructed for sites with zero weight. The same is true for Figures 5.3.4, 5.3.6, 5.3.7 and 5.3.8.
Consider what the observer can see from below the $xy$ plane. First of all, the observer can see only parabolas corresponding to sites already encountered by the sweep plane. He can also see only portions of parabolas corresponding to exposed pyramid surfaces. He cannot see the portions of the parabolas that correspond to intersection points of a pyramid and the sweep plane that are in the interior of some other pyramid. Therefore, only the wave front remains visible to observer.

When two widening parabolas meet, they intersect by a ray. As they continue widening, they will have just one intersection point, which will be called a collision point (see Figure 5.3.4). The arrow on the figure demonstrates the direction in which Voronoi edge is growing during the sweep.

Definition 5.3.5 A wave front that exists at any moment of time during sweep in the $L_1$ metric is defined as the connected portions of the $L_1$ parabolas that

1) correspond to sites already encountered by the sweep plane and

2) do not belong to the interior of any other parabolas.

The wave front consists of portions of parabolas and collision points (see Figure 5.3.4).

Lemma 5.3.2 At any moment during the sweep every collision point belongs to some Voronoi edge.
Proof Consider a point \((x, y)\) that belongs to the intersection of two parabolas originated at sites \(P\) and \(Q\). This point satisfies equations 
\[
d((x, y), (x_P, y_P)) = x - x + r_p \quad \text{and} \quad d((x, y), (x_Q, y_Q)) = x - x + r_Q.
\]

Therefore, 
\[
x - x = d((x, y), (x_P, y_P)) - r_p = d((x, y), (x_Q, y_Q)) - r_Q, \quad \text{i.e.} \quad \text{point } (x, y) \text{ is equidistant from } P \text{ and } Q.
\]

Note that during the sweep \(x_s\) changes in the range \(-\infty < x_s < +\infty\), and, thus, every point \((x, y)\) of the Manhattan bisector \(B(P, Q)\) satisfies the equation for some value of \(x_s\). Consequently, the whole bisector is traced by the intersection of parabolas during the sweep (but not necessarily by the collision point). Since the collision point does not belong to the interior of any other parabola, it is farther from all other sites. Thus, the collision point belongs to Voronoi edge between \(P\) and \(Q\).

\[\blacksquare\]

Lemma 5.3.3 Every Voronoi edge is traced by a collision point during the sweep.

Proof According to Lemma 5.3.2, as the sweep line moves from left to right the collision point between any two neighboring parabolas in the wave front sweeps some Voronoi edge. It must be proven that any point of the Voronoi edge will be a collision point during the sweep.

Consider a point \(a = (x_a, y_a)\) that belongs to the Voronoi edge between sites \(P\) and \(Q\). As two parabolas corresponding to these sites grow, they must intersect at the point \(x\) at some moment of time, because the intersection of parabolas traces the whole bisector \(B(P, Q)\) during the sweep (see proof of Lemma 5.3.2). It can be proven by contradiction, that point \(a\) will be swept by a collision point, i.e. it does not belong to the interior of some other parabola.

Assume that point \(a\) belongs to the interior of the parabola \(p_R\) corresponding to site \(R\). Consider a line through the point \(a\) perpendicular to the sweep line \(l_s\). Denote the point of intersection between this line and the parabola \(p_R\) by \(b = (x_b, y_b)\), and the point of
intersection with \( l_s \) by \( c = (x_c, y_c) \) (see Figure 5.3.5). Note that \( x_c = x_s \) since the point \( c \) is located on the sweep line.

From the definition of the sweep plane, the following equalities are satisfied: 
\[ d(a, c) = x_s - x_a = d(a, P) = d(a, Q). \]

Therefore, \( a \) is the center of a sphere \( C_a \) with radius \( r_a = d(a, c) \) inscribed between \( c, P \) and \( Q \). The point \( b \) that belongs to the parabola \( p_R \) satisfies the equation \( d(b, c) = x_s - x_b = d(b, R) \), i.e. \( b \) is the center of a sphere \( C_b \) with radius \( r_b = d(b, c) \) inscribed between \( c \) and \( R \).

Since the point \( a \) belongs to the interior of the parabola \( p_R \) it follows that \( d(a, c) > d(b, c) \). Thus, the sphere \( C_b \) lies within the sphere \( C_a \). Therefore, \( R \) is closer to the point \( a \) than sites \( P \) and \( Q \). This contradicts the fact that \( a \) belongs to the Voronoi edge between \( P \) and \( Q \).

It will be now shown how the wave front change when a portion of the parabola appears or disappears. The term portion of the parabola refers to the maximum connected part of the same parabola, which entirely belongs to the wave front. The illustration is given in Figure 5.3.6 (bold line illustrates the wave front).
Lemma 5.3.4 A portion of the parabola can appear in the wave front only when the sweep line reaches a site.

Proof A new parabola appears as a ray parallel to the $x$ axis when the sweep line reaches a new site. According to the observation following Definition 5.3.4, the tips of all previously encountered parabolas move along the $x$ axis with the same velocity (which is equal to the half of the sweep-plane velocity) and the parabolas widen with the same velocity (equal to the sweep-plane velocity). Therefore, none of the parabolas can enter the wave front from behind. This means that when the sweep line moves from one site to another, no new portions of parabolas can appear in the wave front.

A moment when a new portion of parabola appears in a wave front is called a site event (see Figure 5.3.7).
Lemma 5.3.5 A portion of parabola can disappear from the wave front only when a Voronoi vertex is encountered by the wave front.

Proof Consider 3 consecutive portions of parabolas $p_r$, $p_q$, and $p_s$ in the wave front. The Voronoi edge $e_1$ is swept by the collision point $x$ between $p_r$ and $p_q$, and the Voronoi edge $e_2$ is swept by the intersection point $y$ of $p_q$ and $p_s$. Only when points $x$ and $y$ collide does the portion of parabola $p_q$ disappear from the wave front. According to Lemma 5.3.3, two Voronoi edges $e_1$ and $e_2$ meet at a point equidistant from three sites\(^1\). By definition, this point is a Voronoi vertex (see Figure 5.3.8).

The only other case when a portion of parabola can disappear from the wave front is when it enters the interior of some other parabola and thus the portion of new parabola will appear in the wave front. It was shown that none of the parabolas can enter the wave front from behind and cover some portion of parabola from the wave front (see Lemma 5.3.4). When the sweep line reaches a new site, a new parabola appears as a ray, which does not have an interior. Thus, neither of these events will lead to the disappearance of an existing portion of parabola from the wave front.

\[\text{Figure 5.3.8 The circle event in the wave front}\]

\(^1\) There are no four cocircular sites.
A moment when a portion of parabola disappears from the wave front is called a \textit{circle event} (see Figure 5.3.8).

\textbf{Lemma 5.3.6} The number of portions of parabolas in the wave front is $O(n)$ where $n$ is the number of sites.

\textbf{Proof} The number of portions of parabolas in the wave front is equal to the number of collision points plus one. At any moment of time each collision point corresponds to a distinct Voronoi edge. Since the number of Voronoi edges in a planar VD is $O(n)$, the number of parabola portions in the wave front is also $O(n)$.

\section*{5.4 Events and Data Structures}

A sweep-plane algorithm for the construction of the Voronoi diagram of a set of weighted sites in $L_1$ metric is now described. First, a classification of the events that can happen during the sweep is now given.

A \textit{circle event} is an event when three portions of parabolas intersect in a common point, corresponding to a Voronoi vertex, and one of the portions of parabolas disappears.

A \textit{site event} occurs when the sweep plane reaches a pyramid and a new portion of parabola is inserted into the wave front. Since pyramids start below the $xy$ plane, the tip of the parabola at the site event will have a larger $x$ coordinate than the sweep line, i.e. the parabola will lie in front of the sweep line (see Figure 5.4.1).

A \textit{break event} is an event when a collision point moving along the Voronoi edge changes the direction of its movement. In this case no new portions of parabolas appear or disappear. A break event is an event specific for Manhattan (supremum) Voronoi diagram. The illustration of two break events during the sweep is presented in Figure 5.3.4.
The following data structures are maintained during the sweep:

**Input:** Array of $n$ Voronoi sites, sorted in ascending order according to the $x$ coordinate of their leftmost point (i.e. by $x_p - r_p$, where $x_p$ is the abscissa of the center of the site, and $r_p$ is the radius of the site).

**Output:** The Voronoi Diagram in a form of doubly-connected-edge-list [Preparata and Shamos, 86].

**Event queue:** Every entry in the event queue consists of event type, time when this event takes place and attributes of the event. The attributes for the events are:

1) For a site event: a pointer to a site, which the sweep plane passes through and where a new parabola is originated.

2) For a circle event: pointers to two collision points that meet at a point corresponding to a Voronoi vertex.

3) For a break event: a pointer to the collision point, moving along the Voronoi edge.

**Wave front:** A balanced tree of collision points during sweep, sorted by $y$ coordinate.

For each collision point $x$ the following attributes are stored:

1) Pointers to two Voronoi sites that originate the bisector along which the collision point $x$ is moving during sweep,

2) A vector of speed of the collision point $x$: $v = (v_x, v_y)$. 
Note 1. Since the collision point moves along the \( L_1 \) bisector, only the direction of the movement matters, therefore it is sufficient to store only the sign of \( v_y \) as positive, negative or zero.

Note 2. Alternatively, pointers to three Voronoi sites in the case of circle event and pointers to two Voronoi sites in the case of break event could be stored in the event queue. However, this approach would require additional computations.

Thus, the coordinates of any collision point at any given moment of time can be computed. Since the value of the time variable gives the position of the sweep plane, only the \( y \) coordinate of the collision point must be computed.

Having a balanced tree as a data structure for wave front and event queue allows performing insertion and deletion in \( O(\log n) \) time.

5.5 The Sweep-Plane Algorithm in the Manhattan Metric

The sweep plane moves with the constant speed in the direction from left to right along the \( x \) axis. In order to make the description of the algorithm consistent, assume also that there is a pseudo Voronoi site at \((-\infty,0)\) that was encountered by the sweep plane at time \( t = -\infty \). This gives a pseudo-parabola with focus at \( -\infty \). Assume that the intersection of this pseudo-parabola with the parabola originated by the first Voronoi site set at a moment of time \( t = 0 \). As a result, two collision points appear at the same location on the wave front.

The sweep-plane algorithm proceeds as following:

Algorithm 5.2.1

Preprocessing

1. Schedule all site events for all Voronoi sites and place them into the event queue \( Q \), sorted by time.

Main loop
2. While the event queue is not empty do:

   2.1 Retrieve the next event from the event queue $Q$.

   2.2 Advance the time to the time of this event $t_e$.

   2.3 Process the event

      If the event is a site event do:

      - Compute the coordinates of the point where the new parabola first penetrates the wave front;

      - Find the portion of existing parabola in the wave front that is split by the new parabola;

      - Insert two collision points with the same coordinates corresponding to the site event that start moving in opposite directions along the bisector;

      - If a circle event was scheduled for the parabola portion that is split, then cancel this circle event;

      - Schedule future break events or circle events for two new collision points.

      If the event is a circle event do:

      - Remove from the wave front two collision points that met on the wave as a result of the circle event;

      - Insert a new collision point;

      - Schedule new break event or circle event for this point;

      - Update the output data structure with the new Voronoi vertex.

      If the event is a break event do:

      - Update the speed of the collision point that changes its direction while moving along bisector;

      - Schedule new circle event for this point;
- Update the output data structure with the new portion of the Voronoi edge.

3. Report the resulting Voronoi diagram.

Note that as sweep-plane algorithm progresses, the following invariants are maintained. The ordered list of collision points is always maintained in the wave front. The queue contains precisely one site event for each site that has not yet been encountered by the sweep-plane, one circle event for every two neighboring collision points in the current wave front if these collision points meet, and not more than one break event for each collision point.

5.6 The Complexity of the Algorithm

Consider first what happens when the site event is scheduled. The site event happens when the sweep plane reaches a Voronoi site at some moment of time \( t_r \). A position where the new parabola (a ray at that moment of time) first penetrates the wave front must be found. Thus, the coordinates of collision points at time \( t_r \) must be computed. This can be done by computing the coordinates of portions of parabolas on the wave front. This procedure is referred to as advancing the wave front. The straightforward way is to advance the whole wave front and then compute the collision point coordinates. However, it is enough to find only a \( y \) coordinate position to insert the point. Thus, a binary search can be performed and only those collision points that are required for the current comparison must be advanced. This allows the procedure to be performed in \( O(\log n) \) instead of \( O(n) \) time.

Now the procedures associated with circle and break events are discussed. First, note that only one break event or circle event is scheduled in advance for each collision point. There is no need to schedule more than one event since the processing of the first event scheduled for a collision point (for example, a break event) will cancel the second event (circle event) scheduled for this collision point.

Circle events happen when two collision points meet on the wave front. Since the coordinates of the points and their speeds are known, this event can be computed in
advance. Break events happen when a point moving along a bisector changes its direction. This moment can be computed from the form of the bisector since the references to two sites, from which the bisector is originated, are known.

Finally, the algorithm complexity is estimated as follows. According to Lemma 5.3.6 the number of portions of parabolas and, consequently, the number of collision points in the wave front is $O(n)$. The main loop takes events from the event queue and processes them until the event queue is empty. The total number of events in the event queue is $O(n)$: $n$ site events, $O(n)$ circle events and $O(n)$ break events. At most $O(\log n)$ time is required to process each event. Therefore, the resulting complexity of the sweep-plane algorithm in the $L_1$ metric in the plane is $O(n \log n)$.

5.7 Implementation and Experimental Results

The initial generalized Delaunay triangulation and the VD are constructed by a sweep-plane algorithm. The algorithm has been implemented in Object-Oriented Pascal in the Borland Delphi environment. The program runs under Windows 95. The experiments were conducted on a Pentium 166 computer with 32 megabytes of RAM.

The user can manually enter the coordinates of the sites and their radii or he can run a program to generate a specified distribution of the input set of sites. The user can specify the parameters of the distribution, such as the number of circles, the distribution of their radii, and the type of the distribution: Random, Square, Circle or Grid (see Section 4.7 for more details on those distributions). The program then constructs the generalized DT by applying the sweep-plane method, displays it on the screen and reports the time required for the construction. Note that the program can construct the Manhattan as well as supremum generalized VD and DT. The user can select an option to view the VD or DT in the selected metric. Note that the running time of the algorithm does not depend on the choice of these options. The interface of the program that constructs the generalized VD in the supremum metric for 10000 sites for the Random distribution with radii of the sites in the 0.01 to 1 mm range is given in Figure 5.7.1.
Experiments were conducted to determine how the increase in the number of sites would influence the time required to construct the diagram for the Random and the three degenerate distributions: Square, Circle and Grid with the radii of the circles selected from 0.01 to 10 mm range. The experiments show that the algorithm's running time
increases from about 1 sec. for 1000 sites for all distributions to about 34 sec. for Random and Grid distributions for 10000 sites and to about 27 sec. for Circle and Square distributions for 10000 sites. It is interesting to note that the tested distributions are split by the two groups with similar performance rates. Random and Grid distributions belong to the same group and Circle and Square to another. The slightly better performance of the algorithm on the Circle and Square distributions can likely be attributed to the fact that in those distributions the initial sites are located on the boundary of the area, while in the other tested distributions they are located inside the rectangular area.

Figure 5.7.2 The number of sites vs. time

5.8 The Sweep-Plane Algorithm for Spheres in the Power Metric

The final section of this chapter addresses the problem of whether it is possible to apply the sweep-plane technique for the construction of the power diagram. It will be proven that an application of the sweep-plane technique will transform the original problem of constructing the power diagram to the problem of constructing the power diagram for a different set of sites. Note, however, that if paraboloids in the power metric are swept by
some surface other than a plane then it might be possible to construct a wave front that will consist of parabolas, and thus the sweeping technique can be applicable to solve the problem [Devillers et. al. 92].

In the power metric, the waves propagate with decreasing speed, thus producing paraboloids rather than cones. Consider the equation of a wave originated at a site \( \{(x_0, y_0), r_0\} \): 
\[
\begin{align*}
  z &= (x - x_0)^2 + (y - y_0)^2 - r_0^2, \\
  z &\geq -r_0.
\end{align*}
\]
This is a paraboloid with the vertex at \((x_0, y_0, -r_0)\) (see Figure 5.8.1).

![Figure 5.8.1 Waves in the power metric](image)

Thus, the power diagram can be obtained by projecting the first intersection of the paraboloids, as seen from below the \(xy\) plane, onto the \(xy\) plane. Assume that the paraboloids are swept by a plane described by the equation 
\[
z = t + 2px + 2qy,
\]
where \(t\) is the time parameter and \(p\) and \(q\) are arbitrary parameters, controlling the slope of the sweep-plane. Then the equation of the intersection between the sweep plane and a paraboloid can be obtained as:
\[
z = (x - x_0)^2 + (y - y_0)^2 - r_0^2 = t + 2px + 2qy.
\]
The projection of this intersection to the \(xy\) plane satisfies the equation
\[
(x - x_0)^2 + (y - y_0)^2 - r_0^2 = t + 2px + 2qy.
\]
This equation can be transformed into the canonical form as
\[(x-(x_0+p))^2 + (y-(y_0+q))^2 = t + \left(\sqrt{r_0^2 + 2px_0 + p^2 + 2qy_0 + q^2}\right),\]
i.e. it is an equation of a circle with center \((x_0+p, y_0+q)\) and radius
\[
\sqrt{t + \left(\sqrt{r_0^2 + 2px_0 + p^2 + 2qy_0 + q^2}\right)}.
\]
Thus the wave front in the sweep-plane algorithm consists of a set of growing circles. The circles grow along paraboloids. The power edges are traced by the intersections of these paraboloids and the first intersection of paraboloids as seen from below the \(xy\) plane must be found. Therefore, a power diagram must be constructed for a set of circles, which originates the paraboloids. This power diagram is exactly the same as the power diagram for the original set of sites.

This result shows that the power diagram is an invariant for a family of input sets
\[
\{(x_i + p, y_i + q), \sqrt{r_i^2 + 2px_i + p^2 + 2qy_i + q^2}\}, \ i = 1..n, \ p, q \in R.
\]
In other words, if the original set of sites is transformed so that each center is translated by \((p,q)\) and each weight is changed by \(2px_i + p^2 + 2qy_i + q^2\), then the power diagram will not change.

5.9 Concluding Remarks

This chapter presented application of the sweep-plane technique for Manhattan weighted Voronoi diagram and discusses the possibility of application of the method for power metric. It was shown that the sweep-plane technique can be successfully applied for the construction of the Voronoi diagram for the weighted set of sites in Manhattan metric. It was also shown that the straight-forward application of this technique to power metric is not recommended since it transforms the original problem of constructing the power diagram to the problem of constructing the power diagram for a different set of sites.
CHAPTER 6: CONSTRUCTION OF THE POWER DIAGRAM FROM THE VORONOI DIAGRAM

In the previous chapters two methods, based on the incremental and the sweep-plane techniques, were presented for the generalized Voronoi diagram construction. A third method based on a swap technique between different metrics is presented in this chapter [Gavrilova and Rokne 96]. An important question is discussed: How close is the Delaunay tessellation to other generalized Delaunay tessellations? Is there is an algorithm that can transform one into another? Can any triangulation be transformed into any other triangulation and what is the complexity of such a process? The first question is addressed in this chapter, preparing the grounds for the generalized Delaunay tessellations and the set of all tessellations discussion in the following chapter.

6.1 Literature Review

So far, only a few deterministic algorithms are known for constructing the DT from some other triangulation. Usually, this other triangulation is very close in a topological sense to the DT: it is either the DT of \( n-1 \) points to which a new site is added during an incremental construction method or it is some approximation of the DT\(^1\). Then, the method to construct DT is based on swap operations, which ensures that every triangle in the diagram satisfies some local optimization criterion.

The local optimization criteria are usually based on DT properties such as empty circle, min-max angle or equiangularity criteria (see Chapter 2). Lawson was one of the first researchers to study this problem. He showed that the min-max, empty circle and

\(^1\) This approximation is often used in terrain models or GIS-systems.
Thiessen criteria\(^1\) in the plane are equivalent for DT construction [Lawson 77]. He also proved that any triangulation of the interior of the convex hull can be swapped to any triangulation in \(O(n^2)\) [Lawson 72, Lawson 77]. Sibson showed that any convex hull triangulation can be converted to a DT by flipping diagonals in the quadrilateral according to the min-max criterion [Sibson 78]. Cherfils and Hermeline proved that any triangulation of the interior of a non-convex polygon can be swapped into a DT in the plane [Cherfils and Hermeline 90].

A swapping procedure based on some local optimization criterion is used as part of an \(O(n^2)\) incremental algorithm for DT construction in the plane [Lischinski 94]. A flipping algorithm based on the empty sphere criterion with worst-case time complexity \(O(n^2)\) for 3D is given in [Joe 91].

The complexity of the flipping method for any local optimization criterion in the plane was analyzed in [Lambert 94]. Lambert proved that any planar triangulation of the interior of the convex hull can be swapped to a triangulation where every quadrilateral satisfies a local optimization criterion\(^2\) in \(O(n^2)\) time and that the swaps can be performed in any order [Lambert 93, Lambert 94]. This result also ensures that no obstruction can appear during swapping in the plane, i.e. it is always possible to construct locally optimum triangulation starting from any triangulation in the plane in \(O(n^2)\) time.

However, it is not enough to know that one triangulation can be transformed into another: the rules for such a transformation must be known, an algorithm should be developed and

\(^1\) This criterion considers closeness of the regions on which triangulation splits the quadrilateral.

\(^2\) See Preliminaries, Section 2.3.1.
tested. In this chapter a new algorithm, which constructs the power diagram from the Voronoi diagram in $R^2$, is presented.

6.2 Chapter Overview

The difference between the location of the power bisector for weighted sites and Voronoi bisector for original point sites is first computed and the resulting properties of the power diagram are discussed.

A new algorithm based on the swapping technique is presented [Gavrilova and Rokne 96]. It constructs the power diagram for a set $S$ of $n$ circles in the plane from the given Voronoi diagram for the set $S_p$ of centers of the circles. The algorithm is based on the idea of inflating the point sites. Each of the edges in the original Voronoi diagram is transformed to its new location. The INCIRCLE test (see Chapter 4, Section 4.4.1) is used as a swapping criterion for the algorithm. A detailed outline of the algorithm is presented along with the implementation. The algorithm is tested and the observed time is proportional to $O(n)$.

6.3 Relationships between the Power Diagram and the Voronoi Diagram

First, the relationship between bisectors of the power diagram and the Voronoi diagram is established. Consider a set of point sites $S_p$ in the plane. A weight $w_p$ is associated to each point $p$ thus defining a sphere $P = \{p, r_p\}$ with radius $r_p = \sqrt{w_p}$. The spheres from the resulting set $S$ do not intersect by Assumption 3.1.1. Assume that the Voronoi diagram for the non-weighted sites in the plane is known. The degenerate cases, when $d+2$ spheres are cospherical in the power metric and when $d+1$ spheres lie on the same hyperplane are excluded from consideration (Assumptions 3.1.2 and 3.1.3). The relationship between the Voronoi diagram bisector and the power bisector can be established as follows.
Theorem 6.3.1 The bisector of weighted sites \( P = (p, w_p) \) and \( Q = (q, w_q) \) in the power diagram can be obtained by moving the bisector for point sites \( p \) and \( q \) in Voronoi diagram by the distance \( \Delta = \frac{w_p - w_q}{2m} \) towards \( q \) in parallel to its previous location, where \( m \) is the distance between points \( p \) and \( q \) in \( R^d \).

Proof To prove the claim, the power bisector between the Voronoi diagram sites \( P \) and \( Q \) is constructed. Denote the power bisector as \( B(P, Q) \). Define a coordinate system with center \( p \) and \( x \) axis directed along the line segment \( pq \). Then \( p = (0,0,0,...,0) \) and \( q = (m,0,0,...,0) \) (see Figure 6.3.1).

![Figure 6.3.1 The transformation of the power bisector](image)

First, note that any point \( y \) on the power bisector by definition satisfies the equation \( d(y, P) = d(y, Q) \). Therefore, it is also true for the point \( x \) on the intersection between segment \( pq \) and the power bisector \( B(P, Q) \), where \( x = (x,0,0,...,0) \). From this condition the following equation is obtained: \( x^2 - w_p = (m - x)^2 - w_q \). It follows that \( x = \frac{m}{2} + \frac{w_p - w_q}{2m} \). If \( b = \left( \frac{m}{2},0,0,...,0 \right) \) is the point of intersection between the segment \( pq \) and the bisector \( B(p, q) \) of the Voronoi diagram, then the distance between the Voronoi bisector \( B(p, q) \) and the power bisector \( B(P, Q) \) is \( \Delta = x - \frac{m}{2} = \frac{w_p - w_q}{2m} \).
Note that the power bisector always intersects the line segment $pq$ under the assumption that the power sites $P$ and $Q$ do not intersect (see Assumption 3.1.1).

### 6.4 Basic Idea of the Algorithm

The algorithm that constructs the power diagram from a given Voronoi diagram is based on the idea of *inflating* the point sites. This idea is explained in this section. Based on this idea, the precise algorithm that works in the terms of Delaunay triangulation and performs swap operation on the quadrilaterals (see Chapter 2, Section 2.3.1) is presented in the next section.

Weights are first assigned to the sites. Then the circles with the centers at each site are simultaneously constructed, or inflated. The edges of the Voronoi diagram are translated to the new positions corresponding to the inflations (see Figure 6.4.1). The edge corresponding to a pair of sites is translated from the initial bisecting position to the new position according to Theorem 6.3.1. The new locations of the vertices are determined by the intersection of the translated edges. Note that as a result of transformation the length of the edge can change, the edge can possibly disappear from the diagram and then the new edge appears.

![Figure 6.4.1 The transformation of Voronoi edges](image)

Consider a Voronoi diagram for four sites $p_1, p_2, p_3$ and $p_4$. Consider an assignment of a non-zero weight to an arbitrary vertex of the quadrilateral (see Figure 6.4.2). First, assign a non-zero weight to Voronoi vertex $p_1$ (see Figure 6.4.2a). Consider a
transformation of the edge $e$ of the Voronoi region $Vor(p_1)$. The edge $e^*$ of the corresponding power region $P\text{Vor}(p_1)$ is parallel to the edge $e$. Thus, there are no changes in the corresponding power quadrilateral $p_1p_2p_3p_4$, i.e. it is the same as the Delaunay quadrilateral for the non-weighted sites $p_1, p_2, p_3$ and $p_4$ (see Figure 6.4.3a).

![Figure 6.4.2 The modification of the Voronoi diagram](image)

Now, assign the non-zero weight to the site $p_4$ of the Voronoi diagram (while keeping all other weights equal to zero) (see Figure 6.4.2b). The edge $e$ disappears from the resulting power triangulation and a new edge $e^*$ (orthogonal to the former edge $e$) appears. The power quadrilateral $p_1p_2p_3p_4$ is now different from the Delaunay quadrilateral (the edge $p_1p_3$ disappeared and an edge $p_2p_4$ appeared (see Figure 6.4.3b).

![Figure 6.4.3 The modification of the quadrilateral](image)

The algorithm starts from the Delaunay triangulation corresponding to the original Voronoi diagram that has already been computed by one of the known methods. Then all the sites are simultaneously inflated and swap operations are sequentially performed on
the Delaunay triangulation. For each internal edge in the triangulation \textit{INCIRCLE} tests are calculated. If the value of the \textit{INCIRCLE} function is positive then the swap operation will be performed on the quadrilateral. If this value is negative then Delaunay quadrilateral should not be changed.

\textbf{6.5 Main Scheme of the Algorithm}

The algorithm that transforms the ordinary Delaunay triangulation into the power Delaunay triangulation in the plane is now formally written. This algorithm sequentially checks that every quadrilateral in the original Delaunay triangulation satisfies the empty circle criterion for the power metric. This can be established by performing the \textit{INCIRCLE} test for the power metric on each quadrilateral of the triangulation and performing swap operation if the value of the \textit{INCIRCLE} function is positive (see Chapter 2, Section 2.3.1 for swap operation definition). The formula for \textit{INCIRCLE} test for power metric in the plane is given in Chapter 4, Section 4.4.1.

\textit{Algorithm 6.5.1}

\textit{Preprocessing}

1. Put all edges of the Delaunay triangulation into the edge queue.

\textit{Main loop}

2. While the edge queue is not empty do:

2.1 Extract an edge $e_i = (p_{i_2}, p_{i_3})$ from the edge queue.

2.2 Calculate the sign of \textit{INCIRCLE} for quadrilateral $p_i p_{i_2} p_{i_3} p_{i_4}$ containing the edge $e_i$ (see Figure 6.5.1).

2.3 If \textit{INCIRCLE} test is positive do

- perform swap operation on the edge $e_i$;
- if any of the edges \((p_i, p_{i2}), (p_{i2}, p_{i4}), (p_{i}, p_{i3})\) and \((p_{i3}, p_{i4})\) are in the edge queue remove them from the edge queue;
- insert edges \((p_i, p_{i2}), (p_{i2}, p_{i4}), (p_{i}, p_{i3})\) and \((p_{i3}, p_{i4})\) into the end of the edge queue.

3. Report the resulting diagram.

![Diagram](image)

Figure 6.5.1 The quadrilateral \(p_i p_{i2} p_{i3} p_{i4}\)

Algorithm 6.5.1 constructs the triangulation where every quadrilateral satisfies an empty-circle criterion for the power metric. The claim that if a triangulation (tessellation) satisfies the empty-sphere criterion for the power metric then it is a power triangulation (a power tessellation) in \(R^d\) was proven in [Edelsbrunner and Shah 96]. Thus, Algorithm 6.5.1 reports the power triangulation. It is essential to show that the algorithm will always arrive to the power triangulation without coming to obstructions.

**Definition 6.5.1** An *obstruction* in a planar geometrical flipping algorithm is a non-convex quadrilateral, which does not satisfy a local optimization criterion (LOC).

![Diagram](image)

Figure 6.5.2. Non-convex quadrilateral
An example of non-convex quadrilateral is presented in the Figure 6.5.2.

*Note.* Obstructions never appear in topological flipping schemes (see Chapter 7) because curvilinear edges are allowed in triangulations. As a result, any quadrilateral can be flipped.

It will be shown that obstructions can never appear in a planar geometrical flipping algorithm leading to the Delaunay triangulation of point sites (based on the Delaunay triangulation LOC). Also, it will be shown that the same is true for a planar geometrical flipping algorithm leading to the power triangulation of circles (based on the power triangulation LOC). Note that obstructions can appear in flipping to a DT for circles under Euclidean, Manhattan and supremum metrics.

It will be shown that any non-convex quadrilateral is optimal, i.e. it satisfies the local optimization criterion (i.e. it need not be flipped).

**Theorem 6.5.1** Any non-convex quadrilateral is optimal under the Delaunay triangulation LOC.

**Proof** Note that the point \( t \) is located inside the triangle \( pqs \) (see Figure 6.5.2). This triangle lies inside of its circumscribed circle. Therefore, the point \( t \) is located inside the circumscribed circle as well. Then \( \text{INCIRCLE}(p,q,s,t) \geq 0 \), i.e. the quadrilateral satisfies the empty-circle criterion, i.e. it is not an obstruction.

Consider the power triangulation LOC. It will be shown that if the center \( t \) of circle \( T = (t,r_t) \) is located inside the triangle \( pqs \) then the condition \( \text{INCIRCLE}(p,q,s,t) \geq 0 \) for the power distance is satisfied as well (under the assumption that the circles do not intersect).
Lemma 6.5.1 Consider three circles \( P = (p, r_p) \), \( Q = (q, r_q) \) and \( S = (s, r_s) \). Let \( O = (o, r_o) \) be the circle inscribed between \( P \), \( Q \) and \( S \) under the power distance function. Then \( o \) does not belong to the interiors of \( P \), \( Q \) or \( S \).

Proof Note that \( o \) lies on the bisector \( B(P, Q) \). Since the circles do not intersect, the bisector does not intersect the interiors of \( P \) and \( Q \). Therefore \( o \) does not belong to either of them as well. The proof for the third circle \( S \) is identical.

Theorem 6.5.2 Any non-convex quadrilateral is optimal under the power triangulation LOC.

Proof Consider the circles \( P \), \( Q \), \( S \) and \( O \). The center of the inscribed circle \( o \) can lie either inside or outside the triangle \( pqs \) (see Figure 6.5.3).

![Figure 6.5.3 Two possible configurations for the center of the inscribed circle](image)

Denote the projections of \( o \) onto segments \( ps \), \( pq \) and \( qs \) by \( c \), \( d \) and \( e \), respectively. Note that in both cases the triangle \( pqs \) is contained within the union of 6 right triangles \( poc \), \( pod \), \( qod \), \( qoe \), \( soc \) and \( seo \). Note that from Lemma 1 \( o \) does not belong to the interiors of \( P \), \( Q \) and \( S \).

Consider a circle \( T = (t, r_t) \) such that \( t \) belongs to the triangle \( pqs \), i.e. the quadrilateral \( PQST \) is non-convex. It will be shown that \( |ot| \leq r_o \) under the assumption that the sites do not intersect.
Obviously, \( t \) must belong to at least one of the triangles \( \text{ poc, pod, qod, qoe, soe and soc} \). Assume that it belongs to the triangle \( \text{poc} \) (see Figure 6.5.4).

![Figure 6.5.4 Proof of Theorem 6.5.2](image)

Note that the angle \( \text{opc} \) is acute. Therefore, it is possible to construct a segment \( oz \) tangent to \( P \) such that it intersects the segment \( \text{pc} \). Denote the point of intersection by \( y \). Note that \( |oz| = r_o, |pz| = r_p \). Denote \( d = |op| \geq 0 \).

Since the sites are not allowed to intersect, the point \( t \) must belong to the shaded area in the Figure 3. The convex hull of this area is the polygon \( oabc \).

The following inequalities are true:

1. \( |oc| \leq |oy| \leq |oz| = r_o \).

2. \( |oa|^2 = (d - r_p)^2, r_o^2 = |oz|^2 = d^2 - r_p^2 = (d - r_p)(d + r_p) \). Note that \( (d - r_p)^2 \leq (d - r_p)(d + r_p) \), since \( d - r_p \geq 0 \) and \( d + r_p \geq 0 \). Therefore,
\[
|oa| \leq |oz| = r_o
\]

3. From the cosine theorem \( |ob|^2 = d^2 + r_p^2 - 2dr_p \cos \text{opc} \),
\[
|oz|^2 = d^2 + r_p^2 - 2dr_p \cos \text{opz} .
\]
Note that \( \cos \text{opc} = \frac{|oc|}{d} \), \( \cos \text{opz} = \frac{|oz|}{d} \) and (see inequality 1) \( |oc| \leq |oz| \). Therefore, \( |ob| \leq |oz| = r_o \).
Thus, it was shown that $|oa| \leq r_o$, $|ob| \leq r_o$ and $|oc| \leq r_o$. Since $t$ belongs to the polygon $oabc$, $|ot| \leq r_o$ as well.

The power distance $d(o,T) = |ot|^2 - r_t^2$ satisfies the following inequality:

$d(o,T) \leq |ot|^2 \leq r_o^2$. Consequently, $INCIRCLE(P,Q,S,T) \geq 0$, i.e. the non-convex quadrilateral $PQST$ satisfies the empty-circle criterion, i.e. it is not an obstruction.

The worst-case running time of the algorithm is $O(n^2)$ since the algorithm based on some local flip rule can require at most $O(n^2)$ flips to construct a locally optimum triangulation regardless of the order in which the flips are done [Lambert 94]. The space complexity of the algorithm is $O(n)$, since the edge queue does not contain duplicate edges at any moment of time and the total number of edges in the diagram is $O(n)$. The Quadedge data structure used to represent the triangulation also requires $O(n)$ space [Lischinski 94].

6.6 Implementation of the Algorithm

The initial Delaunay triangulation is constructed by the incremental algorithm described in Chapter 4, Section 4.2. The algorithm is implemented in Object-Oriented Pascal in the Borland Delphi environment on PC. The program runs under Windows 95. The experiments were conducted on a 486DX2/66 PC with 16 megabytes of RAM.

The program has a user-friendly interface. The user can create a new distribution of sites manually by using an integrated editor. He can also use a program for automated generation of the distribution, where the user can specify the parameters of the distribution, such as the number of circles, the distribution of their radii, and the type of the distribution: Random, Square, Circle or Grid (see Section 4.7 for more details).
After the desired model is created, the Delaunay triangulation for point sites and the power Delaunay triangulation for the circles are constructed (see Figure 6.6.1). The number of swaps that occurred during the transformation and the elapsed time are reported. The edges of the original DT, where the swaps appeared, are highlighted by a different color. The distribution can be saved for the following tests or modifications.

6.7 Experimental Results

Experiments were conducted for various distributions of sites (Random, Square, Circle and Grid), for different number of sites (from 100 to 3000) where the radii of circles were chosen from 0.01 to 10 mm range.
Figure 6.7.1 The number of swaps required vs. the number of sites

In the first series of experiments the number of circles in Random distribution was gradually increased from 100 to 3000, with a number of independent experiments conducted on each step. The number of the swap operations and required time for the transformation were measured. The graph in Figure 6.7.1 demonstrates that the number of swaps grows as a linear function of the number of sites, gradually increasing from average of 4 swaps for 100 circles to 150 for 3000 circles.

The time, required for the transformation, also grows as a linear function of the number of sites. Figure 6.7.2 shows the time, obtained as a result of the number of independent experiments for the same number of sites, versus the number of sites. It demonstrates that the algorithm performed the transformations in the time proportional to $O(n)$ with only a constant number of swap operations required for Random distribution.
In the second series of experiments the time and the number of swap operations were measured for other distributions such as Circle, Square and Grid for variable number of sites. The experiments show that the test results are “very similar” to those obtained for Random distribution. Thus, for any fixed number of sites and for any distribution the difference in the recorded time never exceeded 8% and the difference in the required number of swap operations never exceeded 15%.

6.8 Concluding Remarks

This section presented the method for planar transformation of the Voronoi diagram for point sites into the power diagram. The worst-case time complexity of the algorithm is $O(n^2)$ while the running time is proportional to $O(n)$ based on the experimental results. The expected number of swaps is a small constant, which normally does not exceed 5% of the number of the input sites.
CHAPTER 7: SPACE OF TESSELLATIONS

The space of tessellations for a set of sites and its relationship to the generalized Voronoi diagram in different metrics is considered in this chapter. The space of tessellations is the set of all tessellations that can be constructed for the given set of sites. It will be shown that there is a notion of proximity among objects, which is independent of the metric. It will also be proven that the generalized Delaunay tessellation can be constructed from any tessellation in the plane. In other words, the swapping distance between any two generalized tessellations that partition the plane is estimated and the total number of tessellations in the space of tessellations is evaluated. This is a new result achieved by introducing the curvilinear Delaunay tessellations. Another novel idea is that by introducing curvilinear edges the obstructions (i.e. locally non-regular quadrilaterals which cannot be swapped) are completely eliminated.

There are only a few articles related to this subject and they all discuss the straight-line triangulations of the interior of some convex or non-convex polygon. Lawson [Lawson 72] estimates the swapping distance for the specific class of triangulations in the interior of a convex hull. Dey [Dey 93] counts the number of triangulations of points on a sphere. Sleator computes the number of triangulations of a polygon and a maximum swapping distance between two triangulations of the polygon as \( O(n) \) [Sleator et. al. 87]. Sugihara proves that any triangulation of a polygon is a DT of some set of sites [Sugihara 94]. Recently, it was demonstrated that there exists a mapping of edges between any two triangulations in the plane such as the mapped edges either intersect or coincide [Aichholzer et. al. 95]. Cherfils and Hermeline show that any triangulation can be swapped into the Delaunay triangulation in the plane and leave the possibility of doing it in higher dimensional space as an open problem [Cherfils and Hermeline 90]. The only paper that considers the VD from the graph-theoretical point of view is paper by McAllister and Snoeyink, where spoke diagrams, which can be viewed as a graph-theoretical dual of the VD, are introduced and studied [McAllister and Snoeyink 94].
7.1 Chapter Overview

A large class of tessellations is considered in this chapter. The preliminaries on abstract Voronoi diagrams [Klein et. al. 93] and axiomatic VD [Stifter 97] are given. Then a definition of a class of well-behaved Voronoi diagrams is introduced. It is shown that power and Minkowski Voronoi diagrams belong to this class, as well as abstract Voronoi diagrams and axiomatic Voronoi diagrams.

Next, the definition of a $d$-dimensional Delaunay tessellation corresponding to a well-behaved Voronoi diagram is introduced in the graph-theoretical sense. Note that this tessellation represents a partitioning of the whole space, not just of the interior of some convex figure.

Two modifications of the definition are made to study Delaunay tessellations in different metrics in a generalized manner. First, the Delaunay tessellation is extended with an infinite point. Thus a $d$-dimensional extended Delaunay tessellation is obtained. The total number of edges in a regular Delaunay tessellation depends on the number of the edges of the tessellation, which varies for different metrics. Adding an infinite point guarantees that the number of edges on the lowest boundary is always $d + 1$ for any tessellation.

Then the extended Delaunay tessellation is considered from the geometrical point of view. It is represented by a curvilinear Delaunay tessellation, where the sites are connected by curves, rather than by straight lines. The literature on the subject normally considers the standard way of embedding the Delaunay tessellation in the plane by connecting appropriate sites with straight-line segments. As a result, the constructed straight-line Delaunay tessellation can contain double or intersecting edges in some metrics and obstructions during swapping can appear. The introduction of the curvilinear Delaunay tessellation guarantees a planar embedding of the Delaunay tessellation without crossing or intersecting edges and also guarantees that there are no obstruction during swapping. Any extended curvilinear Delaunay tessellation can be transformed to any other extended curvilinear Delaunay tessellation using a finite number of swap
operations. The fact that for any two generalized DT there exists a sequence of at most $O(n^2)$ swaps transforming one into another is established in the end of this chapter.

7.2 The Abstract Voronoi Diagram

The definition of an abstract VD is now given. The concept of an abstract Voronoi diagram was introduced by Klein [Klein 89]. It is based on the concept of bisecting curves. Let $S$ be a finite set in the plane with some total ordering $<$ defined on it. Assume that for each pair $(P, Q)$ of sites from $S$ there exists a bisecting curve $J(P, Q)$ that divides the plane into two unbounded regions $D(P, Q)$ and $D(Q, P)$. The curve itself is adjoined to one of these regions based on the site ordering:

$$R(P, Q) = \begin{cases} \{D(P, Q) \cup J(P, Q), P < Q \} \\ D(P, Q), P > Q \end{cases}.$$ 

**Definition 7.2.1** An abstract Voronoi diagram is defined as a collection of all abstract Voronoi regions $A\text{Vor}(P_i) = \bigcap_{j \neq i} R(P_i, P_j), i, j = 1..n$ [Klein 89].

Now, consider a system of bisecting curves generated in some metric. Suppose that the following properties are satisfied: the intersection of bisecting curves consist of finitely many connected components, the Voronoi region is connected and is not empty, and the Voronoi regions do not intersect and cover all space. Then the abstract Voronoi region contains site $P$ for all $P$, the abstract Voronoi region is simple-connected, and the abstract Voronoi region is star-shaped relative to $P$ [Klein 89].

---

1 This is true only if $P \in D(P, Q), \forall Q$.
Stifter [Stifter 97] introduced the class of *axiomatic VD’s*. A set of axioms is stated for a set of curves and points in 3D so that the resulting VD possesses some good properties. It is also proven that properties of axiomatic VD’s are close to those of abstract VD’s.¹

7.3 The Space of Generalized Delaunay Tessellations

The class of well-behaved Voronoi diagrams is now defined:

**Definition 7.3.1** The *class of well-behaved Voronoi diagrams* is a sub-class of the class of all Voronoi diagrams such as each Voronoi diagram is a collection of Voronoi regions, where each Voronoi region \(Vor(P)\) satisfies the following properties:

1) each Voronoi region \(Vor(P)\) contains site \(P\);

2) each Voronoi region \(Vor(P)\) is simple-connected;

3) each Voronoi region \(Vor(P)\) is star-shaped relative to the center of \(P\).

According to the definition, a Voronoi region of a well-behaved Voronoi diagram is non-empty, simple-connected and does not contain holes, i.e. it cannot encircle another Voronoi region.²

**Definition 7.3.2** The *class of well-behaved metrics* \(L\) is the sub-class of all metrics, such that the Voronoi diagram in this metric is a well-behaved Voronoi diagram.

In the previous chapters, the properties of the generalized Voronoi diagrams in Euclidean, power, Manhattan and supremum metrics were investigated. It can be concluded that:

---

¹ The VD with convex distance functions and polygonal distance functions (Minkowski distance functions among others belong to this class, but the power distance function does not) are also considered in the literature [Chew and Drysdale 85, Schaudt and Drysdale 92, McAllister et. al. 96]. They can be classified as abstract VD.

² The fact that the Voronoi region cannot contain holes follows from the fact that it is star-shaped relative to \(p\).
Property 7.3.1 The Euclidean, power, Manhattan and supremum metrics belong to the class of well-behaved metrics $L$.

Proof A generalized Voronoi region in the Euclidean, power, Manhattan or supremum metrics contains the site $P$, is simple-connected and star-shaped relative to the center of $P$ according to Properties 3.1.2, 3.2.2 and 3.3.2 (Chapter 3). Thus the considered generalized VD's belong to the class of well-behaved VD's.

Note. The class of abstract Voronoi diagrams and the class of axiomatic Voronoi diagrams also belong to the class of well-behaved Voronoi diagrams, based on their properties described in [Klein and Wood 88, Klein et. al. 93, Stifter 97].

An example of an ill-behaved Voronoi diagram is the multiplicatively weighted Voronoi diagram, whose regions may contain holes [Okabe et. al. 92].

All of the results presented in this chapter are obtained for the class of well-behaved metrics $L$ under the following non-degeneracy assumption on the set of sites $S$.

Assumption 7.3.1 The sites of a $d$-dimensional Voronoi diagram are located in non-degenerate positions, i.e. no $d + 2$ sites are cospherical (in terms of the specific metric), and the sites do not intersect. No $d + 1$ sites lie on the same hyperplane. For any $k$ sites, where $2 \leq k \leq d$, the set of points of $R^d$ equidistant from these sites is a curve or a hypersurface in $R^d$ (not just a single point)\(^1\).

Assumption 7.3.1 guarantees that any Voronoi vertex is the intersection of exactly $d + 1$ Voronoi regions.

The Delaunay tessellation and the extended Delaunay tessellation in $d$-dimensional space are now introduced. These definitions are given in the graph-theoretical rather than

\[\text{\footnotesize \text{\begin{tabular}{c}
1 This requirement can be formulated in terms of the Jacobean of a vector function not being equal to zero (see the proof of Theorem 3.2.1).
\end{tabular}}}\]
geometrical sense. Thus, the term \textit{k-dimensional face} is used to denote a \((k+1)\)-tuple of sites. The notion of simplex is replaced by \((d+1)\)-tuple.

**Definition 7.3.3** A \textit{well-behaved Delaunay tessellation (WDT)} of a set of sites \(S = \{P_1, P_2, \ldots, P_n\}\) corresponding to a well-behaved Voronoi diagram in \(d\)-dimensional space is a diagram \(DT = (S_0 = V, S_1 = E, S_2 = F, S_3, \ldots, S_d)\) where:

- \(S_0 = V\) is the set of original sites \(S\);
- \(S_1 = E\) is the set of edges, such as the edge \((P_i, P_j) \in S_1\) if the Voronoi diagram contains a \(d-1\) dimensional facet between regions \(\text{Vor}(P_i)\) and \(\text{Vor}(P_j)\);
- \(S_k, k = 2 \ldots d - 1\) is the set of \(k\) -dimensional faces where face \((P_{i_1}, P_{i_2}, \ldots, P_{i_{k+1}}) \in S_k\) if the Voronoi diagram contains a \(d-k\) dimensional face between regions \(\text{Vor}(P_{i_1}), \text{Vor}(P_{i_2}), \ldots, \text{Vor}(P_{i_{k+1}})\);
- \(S_d\) is the set of \((d+1)\)-tuples where a tuple \((P_{i_1}, P_{i_2}, \ldots, P_{i_{d+1}}) \in S_d\) if the Voronoi diagram contains a vertex between regions \(\text{Vor}(P_{i_1}), \text{Vor}(P_{i_2}), \ldots, \text{Vor}(P_{i_{d+1}})\).

Note that in 2D the definition for the \textit{well-behaved planar Delaunay tessellation} is a particular instance of a well-behaved Delaunay tessellation. The definition of the set of edges \(S_1 = E\) states that Voronoi regions share an edge. The definition of \(S_d = S_2\) represents the set of the Delaunay tessellation faces (3-tuples) and states that each of them corresponds to a Voronoi vertex.

Consider the \(d\) -dimensional well-behaved Delaunay tessellation corresponding to a well-behaved Voronoi diagram. It can be extended with the infinite point, thus obtaining the \(d\) -dimensional extended well-behaved Delaunay tessellation. In the rest of the chapter the well-behaved Delaunay tessellation will be referred to as Delaunay tessellation for simplicity.

The total number of edges in the Delaunay tessellation depends on the number of the edges on the outermost boundary of the tessellation, which can be different for different
metrics. Adding the infinite point guarantees that the number of edges on the boundary is always $d + 1$ for any tessellation.

**Definition 7.3.4** An extended Delaunay tessellation (EDT) of a set of sites $S = \{P_1, P_2, \ldots, P_n\}$ corresponding to a well-behaved Voronoi diagram $VD(S)$ is a diagram $EDT = (S_0^e, S_1^e, \ldots, S_d^e)$ which is a superset of Delaunay tessellation, where:

- $S_0^e$ is the set of vertices of WDT extended with the so-called infinite point: $S_0^e = S_0 \cup \{P_\infty\}$, where $P_\infty \not\in S$;
- $S_1^e$ is the union of the set $S_1$ of edges of WDT and the set of infinite edges $(P_i, P_\infty)$ for those $P_i$ whose Voronoi region $Vor(P_i)$ is unlimited;
- $S_k^e, k = 2 \ldots d$ is the union of the set $S_k$ of $k$-dimensional faces of WDT and the set of infinite faces $(P_{i1}, P_{i2}, \ldots, P_{ik}, P_\infty)$, such as there is an unlimited $(d - k + 1)$-dimensional face between regions $Vor(P_{i1}), Vor(P_{i2}), \ldots, Vor(P_{ik})$.

The Delaunay tessellation for a set of sites is the topological dual of the Voronoi diagram for the same set of sites. Any vertex of the Delaunay tessellation corresponds to the Voronoi region of site $P$; an edge of the Delaunay tessellation corresponds to a facet ($(d - 1)$-dimensional edge) of the Voronoi diagram; a $k$-dimensional face of the Delaunay tessellation corresponds to a $(d - k)$-dimensional face of the Voronoi diagram; and a $(d + 1)$-tuple of the Delaunay tessellation (a triangle in 2D) corresponds to a Voronoi vertex.

Consider now the planar extended Delaunay tessellations in well-behaved metrics $L$.

In the previous chapters embedding the Delaunay tessellation in the plane by connecting appropriate sites with straight-line segments was considered. This is a standard way of Delaunay tessellation representation, usually considered in the literature on this subject [Okabe et. al., 92]. As a result, the constructed straight-line Delaunay tessellation can
have double or intersecting edges in some metrics (see example in Chapter 3, Section 3.2.3 for the Euclidean VD).

A **curvilinear Delaunay tessellation** that represents a geometrical interpretation of the extended Delaunay tessellation is now introduced.

**Definition 7.3.5** An edge of the curvilinear Delaunay tessellation is a planar acyclic curve connecting sites \( P \) and \( Q \) and passing through a point on the intersection of two adjacent Voronoi regions \( \text{Vor}(P) \) and \( \text{Vor}(Q) \) (see Figure 7.3.1).

**Definition 7.3.6** A **curvilinear Delaunay tessellation (ECDT)** is the planar embedding of the Delaunay tessellation in which sites are connected by curves according to Definition 7.3.5. An **extended curvilinear Delaunay tessellation (ECDT)** is the planar embedding of the extended Delaunay tessellation in which sites are connected by curves according to Definition 7.3.5.

![Figure 7.3.1 The curvilinear Delaunay edge](image)

This definition allows obtaining a planar embedding of Delaunay tessellation without edge intersections. The introduction of the swap operation on this tessellation leads from the curvilinear Delaunay tessellation to a wider class of curvilinear tessellations. The introduction of the extended curvilinear Delaunay tessellation guarantees that there is a sequence of swap operations that will transform any tessellation from the space of tessellations into any other tessellation from this space.

The following example demonstrates that the curvilinear Delaunay tessellation for points and the curvilinear Delaunay tessellation for circles in the Euclidean metric (the Euclidean tessellation) can contain different numbers of edges. The Delaunay tessellation has five internal edges, while the Euclidean tessellation has seven internal edges, two of them connecting sites \( A \) and \( B \) (see Figure 7.3.2).
Figure 7.3.2 Curvilinear Delaunay (a) and Euclidean (b) tessellations

The introduction of the infinite point makes the total number of edges in both tessellations equal to nine (see Figure 7.3.3)

Figure 7.3.3 Extended curvilinear Delaunay (a) and Euclidean (b) tessellations

In the planar case for the class of well-behaved metrics $L$ the following lemmas are true under Assumption 7.3.1.

Lemma 7.3.1 The planar Delaunay tessellation can be embedded in the plane without edge intersections.

Proof For all well-behaved metrics, the Voronoi region $Vor(P)$ is a simple-connected star-shaped region, containing the site $P$. The boundary of $Vor(P)$ is a sequence of Voronoi edges, by which $Vor(P)$ intersects neighboring Voronoi regions. Thus, if $Vor(P)$ intersects $Vor(Q)$ by a Voronoi edge $e$ (note that because of the Assumption 7.3.1 $e$ cannot be a single point), then a point $x \in e$ can be selected and a path from $P$ to...
$Q$ as two straight-line segments $(P, x)$ and $(x, Q)$ can be constructed (see Figure 7.3.4). This path represents the edge $(P, Q)$ of the Delaunay tessellation.

![Diagram](Image)

Figure 7.3.4 Embedding an edge of Delaunay tessellation into the plane

The path lies completely inside $Vor(P) \cup Vor(Q)$. Obviously, this path does not intersect any other paths inside $Vor(P)$ going from $P$ into other neighboring Voronoi regions. Consequently, no two paths can intersect other than at their endpoints, since no two Voronoi regions can intersect by their interior (based on Definition 7.3.1). Thus, a planar embedding of the Delaunay tessellation has been obtained.

Lemma 7.3.2 The planar extended Delaunay tessellation can be embedded in the plane without edge intersections.

Proof. The extended Delaunay tessellation contains the regular Delaunay tessellation as its subgraph. According to Lemma 7.3.1 the regular Delaunay tessellation is a planar graph. Consequently, a planar embedding for the edges of the EDT, corresponding to the infinite Voronoi regions, must be obtained. Select an infinite ray coming out of the site $P$ inside each unlimited Voronoi region $Vor(P)$. Obviously, none of these rays will intersect with other rays or with edges of the regular Delaunay tessellation.

Now, point $P_\infty$ can be embedded into a point inside one of the unlimited regions. The rays are transformed into curves so that they come into $P_\infty$ (see Figure 7.3.5). This can be done without intersecting the rays.
In the rest of this chapter, the terminology "Delaunay tessellation" and "extended Delaunay tessellation" will refer to the curvilinear Delaunay tessellation and curvilinear extended Delaunay tessellation, respectively, that is, the word "curvilinear" will be omitted for brevity.

Lemma 7.3.3 For the planar EDT of a set of \( n \geq 2 \) sites in the plane, the following expressions for the number of vertices \( v \), the number of edges \( e \) and the number of faces \( f \) are satisfied: \( v = n + 1 \), \( e = 3n - 3 \), and \( f = 2n - 2 \).

Proof Obviously, the number of vertices of the EDT is \( n + 1 \), because they include the original sites as well as the infinite point.

Since EDT is a connected planar graph, Euler's formula is satisfied: \( v - e + f = 2 \) [Behnke et. al. 83]. All of EDT's faces are triangles\(^1\), consequently, \( 3f = 2e \). Therefore, \( f = \frac{2}{3}e \) and \((n + 1) - e + \frac{2}{3}e = 2\). Resolving for \( e \), the equation \( e = 3n - 3 \) is obtained. Substituting into the formula for \( f \) conforms that \( f = 2n - 2 \).

Note 1. A general set of relationships for the number of edges and faces of a planar EDT was obtained. These relationships are true in any metric.

---

\(^1\) This follows from the Assumption 7.3.1.
Note 2. Similar estimates for regular DT can be obtained. However, they will depend on the number of unbounded regions in the corresponding VD.

Note 3. It is conjectured that it is possible to obtain similar strict relationships for the number of faces of the $d$-dimensional EDT, $d > 2$.

7.4 The Space of All Tessellations

The general definitions of the tessellation in the plane are now introduced.

Definition 7.4.1 A *tessellation* for a set of sites $S$ in the plane is a planar, simple-connected diagram $T = (V, E, F)$, which consists of the set of vertices $V = S$, the set of edges $E$ connecting pairs of elements from the set $V$ and the set of faces $F$ such that each face consists of exactly three vertices and three edges from the sets $V$ and $E$.

Similarly to the Delaunay tessellation case, this definition can be extended by adding the infinite point $P_\infty$.

Definition 7.4.2 An *extended tessellation* for the set of sites $S$ in the plane is a planar, simple-connected diagram $ET = (V, E, F)$, which consists of the set of vertices $V = S \cup \{P_\infty\}$ extended with an infinite point, the set of edges $E$ connecting pairs of elements from the set $V$ and the set of faces $F$ such that each face consists of exactly three vertices and three edges from the sets $V$ and $E$.

Such a graph can be represented by a doubly-connected-edge-list data structure [Preparata and Shamos 85].

Definition 7.4.3 The *space of tessellations* $W_2(S)$ is the set of all extended tessellations of the set of sites $S$ in the 2-dimensional space.

It will be shown that the extended Delaunay tessellation in a well-behaved metric belongs to the space of tessellations $W_2$, which means that all of the theorems and properties introduced for the space of tessellations are also applicable to the extended Delaunay tessellation.
Theorem 7.4.1 The extended Delaunay tessellation of a fixed set of sites $S$ in the plane belongs to the space of tessellations $W_2$ for the same set of sites.

Proof To prove the statement of this theorem it must be demonstrated that EDT is a planar, simple-connected graph, such that each face has exactly three edges. According to Lemma 7.3.2, the EDT can be embedded in the plane, i.e. it is a planar graph. For a well-behaved metric $L$ each region of the Voronoi diagram cannot contain holes, which means that the Voronoi diagram covers the whole plane $R^2$. In other words, for any two sites $P$ and $Q$ there is a sequence of neighboring Voronoi regions starting at $Vor(P)$ and ending at $Vor(Q)$. Then there is a path connecting $P$ and $Q$ along the corresponding edges of the Delaunay tessellation. Therefore, EDT is a simple-connected graph. Finally, according to non-degeneracy Assumption 7.3.1, each vertex of the Voronoi diagram is the intersection of exactly three edges, which means that each face of the EDT is bounded by exactly three edges (i.e. it’s a curvilinear triangle).

Consider now a planar tessellation $A \in W_2$ for a set of sites $S$ and take a pair of neighboring faces $s_1 = (P_1, P_3, P_4)$ and $s_2 = (P_1, P_2, P_4)$.

Definition 7.4.4 A swap operation $SWAP(A, s_1, s_2)$ transforms tessellation $A \in W_2$ into tessellation $B \in W_2$ by removing $s_1$, $s_2$ and the edge $e = (P_1, P_2)$ from $A$ and including 2 new faces $s_1^* = (P_1, P_3, P_4)$, $s_2^* = (P_2, P_3, P_4)$ and an edge $e^* = (P_3, P_4)$ (see Figure 7.4.1).

![Figure 7.4.1 Swap operation in the plane](image-url)
Note 1. For each swap operation $B = swap(A, s_1, s_2)$ there is an inverse swap $A = swap(B, s_1^*, s_2^*)$.

Note 2. For a tessellation $A$, there are exactly $3n - 3$ swaps, which can be applied to $A$. This is because each of the $3n - 3$ edges is shared by two neighboring faces, and therefore can be swapped.

The space of planar tessellations $W_2$ of a fixed set of sites $S$ will now be discussed. First, it will be shown that for any two tessellations $A, B \in W_2(S)$ there exists a sequence of at most $O(n^2)$ swaps transforming $A$ into $B$. The definition of the tessellation in canonical form is first introduced.

Definition 7.4.5 A tessellation $C \in W_2(S)$, $|S| \geq 3$ is said to be in canonical form, if it contains the following three faces: $(P_1, P_2, P_n)$, $(P_1, P_\omega, P_n)$ and $(P_2, P_\omega, P_n)$ (Figure 7.4.2).

![Figure 7.4.2 The canonical subgraph](image)

Note. In the canonical tessellation the site $P_\omega$ is connected only to sites $P_1$, $P_2$ and $P_n$.

Lemma 7.4.1 Any tessellation $A \in W_2(S)$ can be transformed into canonical form in at most $O(n)$ swaps.

Proof The tessellation can be transformed to the canonical form in two stages. First, three edges $(P_\omega, P_1)$, $(P_\omega, P_2)$ and $(P_n, P_\omega)$ are constructed as described below. Then all other edges coming from $P_n$ are removed. If the edge $(P_n, P_1)$ does not belong to the original tessellation, then it can be constructed as follows. Construct a curve between two sites $P_1$ and $P_n$, for example, as a straight-line segment (see Fig. 7.4.3a).
The new edge will intersect a number of triangles and edges. Then these edges are swapped one by one, starting from the first one (see Figure 7.4.3b). The edge \((P_n, P_1)\) will be obtained as a result of the last swap (see Figure 7.4.3c). The total number of swaps cannot exceed the number of edges in the tessellation, i.e. \(O(n)\).

The edges \((P_n, P_2)\) and \((P_n, P_{an})\) can be obtained in a similar manner with at most \(O(n)\) swaps. Note that no edges out of \(P_n\) are removed during the construction, therefore, the edge \((P_n, P_1)\) cannot be removed. As the result of the first step, the configuration shown in the Figure 7.4.4 is obtained.
The site $P_n$ is now connected to a sequence of sites $P_1, \ldots, P_{n-1}, P_n$. Now all edges coming from $P_n$ except for $(P_n, P_1)$, $(P_n, P_2)$ and $(P_n, P_\omega)$ are swapped. The resulting tessellation will contain the required canonical subgraph (represented in Figure 7.4.2). The second step requires no more than $O(n)$ swaps, because the total number of swapped edges cannot exceed the total number of edges in the tessellation.

\textbf{Theorem 7.4.2} For any two tessellations $A, B \in W_2(S)$ there exists a sequence of at most $O(n^2)$ swaps transforming $A$ into $B$.

\textbf{Proof} This theorem is proven by induction for the number of sites $n$. First, note that for $n = 2$ only a single tessellation exists (see Figure 7.4.5). Thus, the theorem claim is satisfied for the induction basis.

Assume that the theorem is satisfied for sets $S_i = (P_1, P_2, \ldots, P_i), i = 3 \ldots n - 1$. Now, it can be proven that the theorem claim is satisfied for the set $S_n = (P_1, P_2, \ldots, P_{n-1}, P_n)$.

Because of Lemma 7.4.1, the tessellation $A$ can be transformed into the canonical tessellation $A_c$ in $O(n)$ swaps. Similarly, tessellation $B$ can be transformed into the canonical tessellation $B_c$ in $O(n)$ swaps. Now, the canonical subgraph is removed from the tessellation $A_c$ by removing triangles $(P_1, P_2, P_n), (P_1, P_\omega, P_n)$ and $(P_2, P_\omega, P_n)$ and the site $P_n$. The resulting tessellation $A^*_c$ is a tessellation of the reduced set of sites $S_{n-1}$. Similarly, the canonical subgraph can be removed from $B_c$. Thus the tessellation $B^*_c$ is obtained. Note that the $B^*_c$ is also a tessellation of the reduced set of sites $S_{n-1}$. By
induction, \( A_c^* \) can be transformed into \( B_c^* \) in at most \( O\left((n-1)^2\right) \) swaps. Therefore, the following sequence of transformations can be performed:

\[
A \xrightarrow{O(n)} A_c, \quad A_c^* \xrightarrow{O((n-1)^2)} B_c^* \quad B_c \xrightarrow{O(n)} B.
\]

The last step is the inverse of the sequence of swaps \( B \rightarrow B_c \). The total sequence has a length of \( O(n^2) \).
CHAPTER 8: COLLISION DETECTION OPTIMIZATION

Properties of the fundamental data structures in computational geometry that can be used to efficiently store proximity information for a set of objects were studied in the previous chapters. The algorithms for the construction of generalized Voronoi Diagram for a set of objects under different distance functions were developed. The relationships among DT in different metrics were examined. The general property of closeness of objects, which is an invariant of the metric, was also considered. This knowledge will now be applied to obtain efficient solutions for collision detection problems that occur, in particular, during the simulation of dynamic systems [Sun et. al. 94, Gavrilova et. al. 96].

Collision detection problems arise in many applications, such as motion planning, solid modeling, molecular dynamics, robotics, and computer graphics [Okabe et. al. 92]. When simulated objects move in space they can collide with each other or with the boundaries of the space and these collisions must be detected and handled in an appropriate manner. In a straightforward approach, each pair of moving objects must be checked for collision, which leads to \( O(n^2) \) collision checks to detect the next colliding pair. When the number of objects is very large then collision detection slows down the simulation so that the number of objects that can be simulated becomes restricted. Because of this real time implementations are only feasible for a small number of objects. More sophisticated approaches, taking advantage of computational geometry tools, are essential to speed up the solution and thus to improve such simulation [Hahn 88, Lin 93].

However, only a few papers from the field of computer simulation deal with the optimization issue even though there exist some dynamic data structures and algorithms developed in the field of computational geometry and computer graphics suitable for the collision detection problem [Basch et. al. 97, Held et. al. 96, Lin 93]. Mostly, this is due

---

1 The algorithm was first proposed by the thesis author in [Sun et. al. 94] and then extended in [Gavrilova et. al. 96].
to the fact that during real simulations the physical model of a natural phenomenon is investigated and the collision detection problem becomes only a secondary issue.

The studies conducted in the fields of robotics, computational geometry, computer graphics and computer animation describe various data structures that can be applied to represent a dynamic system. The general property of these structures is that the time required for dynamic updates (insertion/deletion) is $O(\log n)$. Various algorithms to perform such operations are considered in detail in [Lin 93, Paterson and Yao 92, Torraz 90]. Methods for animation of the rigid body simulation are considered in [Baraff 90, Hahn 88].

8.1 Chapter Overview

The contribution of this chapter is a generalized algorithm for collision optimization for a system of moving objects in real-time simulation that can be adapted with some modifications to particular computational geometry data structures [Sun et. al. 94, Gavrilova et. al. 96].

The problem of collision optimization for a set of moving objects is stated and the data structures and the types of the events that can happen during the simulation are described. A general algorithm is developed for collision detection optimization for systems where the trajectories of the objects are given as analytical functions of time. The problems arising during real-time simulation of physical systems, where system of algebraic or differential equations\(^1\) have to be resolved at each time step to predict the state of the system at next time step, are considered next. The types of events are defined and the algorithm for system update in time-step simulation is described. The algorithms developed can be used for collision optimization in $d$ dimensions.

In the following chapters data structures for collision detection for a set of moving objects are investigated. The implementation of the algorithms and experimental results

\(^1\) See, for example, [Kamat 93, Vinogradov 92].
for the applied problem from the field of granular-type material simulation are also presented.

8.2 Collision Detection between Two Arbitrary Shaped Objects

The problem of detection of collisions between two arbitrary shaped objects is defined as follows: given two objects of arbitrary shape, and their spatial coordinates as functions of time, find the time when the objects collide.

For objects of arbitrary shape moving along arbitrary trajectories the calculation of collision time can become a very complicated problem that can only be solved numerically. The general approach used to simulate the behavior of such a system is to recalculate the positions of the objects at each time step. Then the motion of an object between the two time points can be approximated by a spline (in the simplest form, by a straight line).

When the time of collision between two objects is being computed, the two points where objects will collide with each other or with the boundary can be found, for example, by decomposing the object or boundary according to some rule. The grid method, the \( k-d \) tree method, surface decomposition method, sweep-prune method and OBB-tree methods are some of the methods that are used for this purpose [Held et al. 96, Gottschalk et al. 96].

Voronoi diagrams are often featured as one of the data structures used in this process. The application of Voronoi diagrams to detect the collision between two polyhedral objects in 3D is discussed in [Lin 93]. The method is based on decomposing the 3D object into a union of convex polyhedra, constructing the Voronoi diagram of the space surrounding each convex piece and then finding the closest pair among all pieces. Hubbard applies the Voronoi diagram to approximate polygonal objects with spheres and thus to optimize collision detection between these two objects. His method finds application in interactive graphics and computer animation [Hubbard 95]. An incremental algorithm for collision detection between two complex polygons, based on the
construction of the Voronoi diagram for polygons' contour, is described in [Ponamgi et. al. 97].

8.3 Collision Detection between Two Spherical Objects

The problem becomes easier to solve if only specific types of objects, for example disks in 2D or spheres in 3D, are considered. This does not limit the domain of consideration since spheres are often used as simple and effective models to represent physical bodies. For example, spheres are used in the simulation of grain and ice particles, balls and beads, plants and their leaves, atoms and molecules [Krantz 93, Lubachevsky 91, Marin et. al. 93, Mech and Prusinkiewicz 96, Milenkovich 96, Szeliski 92]. Some other objects can also be approximated by bounding spheres [Basch et. al. 97, Hubbard 95 (Ph. D. Thesis), Kim et. al. 97, Reynolds 87]. For example, polyhedral objects can be represented as hierarchies of spheres [Hubbard 95, O'Rourke and Badler 79]. This approximation is often used in the literature because it allows easier manipulation and faster collision detection [Hubbard 95].

When spheres move along trajectories described by some analytical functions, the time of their collision can often be found analytically. The time of collision for any pair of objects can be determined as the minimal positive root of the equation

\[ d(object_a(t), object_b(t)) = 0, \text{ where } d(\cdot, \cdot) \text{ is a distance function.} \]

For example, when spheres move in \( \mathbb{R}^d \) and the distance is defined as a general Minkowski distance function, this equation can be written as:

\[ \sqrt[p]{\sum_{i=1}^{d} (a_i(t) - b_i(t))^p} = r_a + r_b, \]

where \( a_i(t), i = 1..d \) are the coordinates of the center of the \textit{sphere}_a with radius \( r_a \) and \( b_i(t), i = 1..d \) are the coordinates of the center of the \textit{sphere}_b with radius \( r_b \).
For the power distance the distance between two spheres in the power metric is defined as \( D(sphere_a(t), sphere_b(t)) = (a_i(t) - b_i(t))^2 - r_a^2 - r_b^2 \) (see Chapter 3). Thus the time of collision can be determined from the equation

\[
\sum_{i=1}^{d} (a_i(t) - b_i(t))^2 = r_a^2 + r_b^2.
\]

For example, when two disks move in Euclidean space, finding the time of their collision requires the solution of a quadratic equation with real coefficients:

\[
\sqrt{(a_x(t) - b_x(t))^2 + (a_y(t) - b_y(t))^2} = r_a + r_b,
\]

which can be transformed to

\[
(a_x(t) - b_x(t))^2 + (a_y(t) - b_y(t))^2 = (r_a + r_b)^2.
\]

If the disks move along straight-line trajectories, then \( a(t) = a_0 + v_a t, \) \( b(t) = b_0 + v_b t, \) where the velocity \( v_a = (v_{ax}, v_{ay}) \) and initial position \( a_0 = (a_{0x}, a_{0y}) \) are given for the disk \( a \) and the velocity \( v_b = (v_{bx}, v_{by}) \) and initial position \( b_0 = (b_{0x}, b_{0y}) \) are given for the disk \( b. \) Then the equation can be written as

\[
(a_{0x} + v_{ax} t - b_{0x} - v_{bx} t)^2 + (a_{0y} + v_{ay} t - b_{0y} - v_{by} t)^2 = (r_a + r_b)^2.
\]

If this quadratic equation does not have a root, or if all roots are negative, then the two disks will not collide.

Note that a similar equation must be solved for the power metric when disks are moving along straight-line trajectories. The solution for the Manhattan or supremum metrics in the plane can be obtained from a system of linear equations.

When objects move in a bounded domain the time of collision between an object and a boundary is determined by solving a similar equation. If a boundary is represented by a simple curve, the solution can be found analytically in some cases.
In higher dimensions, the degree of the polynomial function of time representing the equation will remain the same, while the formulas will become more complex. The complexity of the computations also increases if the trajectories are represented by higher degree polynomials of time. If the trajectory is not very complex (for example, if it is described by a parabola) then the solution can be found analytically in some cases. Note also that in the Manhattan or supremum metrics the solution is found from a system of linear equations.

When the movement of two spherical objects cannot be described analytically, the position of the object can be recalculated at each time step and the movement can be described by piecewise linear trajectories generated by solving differential equations.

8.4 Collision Detection for a Set of Objects

Consider the problem of optimizing the collision detection for a set of moving objects, generally of arbitrary shape. The collision check is performed to detect the collision time for a pair of objects.

The problem can be stated as follows. A set of $n$ moving objects is given with the equations of their motion in $d$-dimensional space. The simulation space is limited by straight-line boundaries represented by $m$ hyperplanes (straight-line segments in 2D). Assume that $m << n$. It is necessary to detect and handle collisions between objects and between objects and boundaries. All collisions are considered to be instantaneous and one-on-one only. Handling of a collision requires updating of the velocities of the objects participating in the collision and detecting new collisions for the updated objects. In the general case, each time an object’s velocity changes due to the collision or due to the influence of the external forces, collision checks with all other objects should be performed.

A straightforward solution of the collision detection problem can be described as follows. At the start of the simulation, all collisions are detected by checking each pair of disks for collision. This requires $O(n^2)$ collision checks. After each velocity update (due to a
collision) the disk participating in the collision is checked for collisions with each of the other disks. This requires $O(n)$ collision checks. There is no additional time or space overhead associated with the algorithm. There is also no special data structure to maintain. However, for a large system the application of this method is computationally expensive.

The task is to improve this complexity by using computational geometry methods. In order to optimize the collision detection between objects a data structure to store the proximity information for the set of moving objects has to be created. Then the collisions can be computed only for a subset of the objects closest to the given one. The approach to the solution of this problem is based on maintaining a data structure so that it represents the topological information about the set of objects. Changes in the data structure happen only at discrete moments of time. In the following sections the tasks associated with the maintenance of data structures will be considered and the algorithms to decrease the required number of collision checks will be discussed.

8.5 Computer Simulation

The simulation task can be defined precisely. At the time $t_0 = 0$ the state of the system is known. The problem is to simulate the behavior of the system in time $t \in (t_0, t_{end})$, where $t_{end}$ is the time of the end of the simulation run. The simulation is continued as long as the time $t_e$ is smaller then the given time boundary for the end of the simulation $t_{end}$.

The following tasks will be now discussed:

- choosing the appropriate data structure;
- constructing the data structure;
- maintaining the data structure;
- reporting collisions;
- estimating the complexity of the algorithm.
Various data structures that can be used for collision detection optimization are presented in the next chapter. The choice of the data structure depends on different parameters of the simulated system such as number of bodies, density of their packing, physical characteristics of the system etc. The choice of the data structure will be discussed in Chapter 10.

8.6 Data Structure Maintenance

Consider a problem of dynamical maintenance of the data structure associated with a set of \( n \) moving objects. The object trajectories are described by analytical functions of time. Call the simulation of such a system a reduced-model simulation.

As an object moves, the spatial position of the object changes continuously in time. However, the data structure that represents the topological information about objects changes only at discrete moments of time.

**Definition 8.6.1** A *topological event* \( e \) is a specific moment of time when the topology of the data structure changes.

During the motion, objects can collide with each other and with the boundaries. Handling a collision requires updating of the velocities of the objects participating in the collision and detecting new collisions for the updated objects. The velocity update event can be defined as follows.

**Definition 8.6.2** A *velocity update event* \( u \) is some specific moment of time when the velocity of the object changes.

In simulation of physical systems the velocity of an object can change if the trajectory by which the object travels is defined by a piecewise function, e.g. it is obtained by numerical integration of a differential system. In this case the velocity update events are scheduled periodically with some time step in addition to the velocity update events related to collisions. This leads to the time-step simulation scheme described later.

Note that velocity update events are related to the simulation of the physical system and are present in the event queue no matter which collision detection optimization approach
is used (or even if none is used). Topological events are the events related to the data structure and algorithm used for the collision detection optimization. All of these events are stored in the same queue in order of increasing time of event.

**Definition 8.6.3** An *event queue* \( Q \) is a data structure that stores topological events and velocity update events in order of increasing time of event.

The AVL tree is used to represent the queue. Algorithms for updating the event queue in \( O(\log n) \) time can be found in [Cormen et. al. 92].

After the data structure is constructed and methods for detection of topological events are defined, maintenance of the data structure between events is trivial since it is enough to update the corresponding data structure only at the time of the event. The data structure update during the topological event is a very important step since it determines the efficiency of the method. New events should be scheduled. During this step the actual optimization of the collision detection takes place.

### 8.7 Algorithm Description

The general scheme of the algorithm for the dynamic maintenance of the data structure can be characterized as follows:

**Reduced-model algorithm 8.7.1**

**Preprocessing**

1. Construct the topological data structure at the starting position. Schedule velocity update events for all objects at time \( t_0 = 0 \).

**Main loop**

2. While the event queue is not empty do:
   
   2.1 Extract the next event from the event queue \( Q \) and determine the type of the event (topological event, velocity update event);
   
   2.2 Advance the simulation clock to the time of this event \( t_e \);
   
   2.3 Process the event:
- modify the topology of the data structure if the event is a topological event \( e_i \);
- change the velocities of the objects participating in a collision if the event is a velocity update event \( u_i \);

2.4 Update the event queue:
- delete topological and/or velocity update events that become invalid from \( Q \);
- schedule new topological and/or velocity update events;
- insert new events into the event queue.

Note 1. Velocity update events for all objects are initially scheduled for the time \( t = 0 \). By processing each of these events during the algorithm iteration, new topological and velocity update events can be obtained. Thus the need for scheduling any topological events during the preprocessing stage is eliminated.

Note 2. The processing of the velocity update event \( u_i \) always leads to the scheduling of new topological and velocity update events and to deleting previously scheduled events invalidated by the event \( u_i \).

Note 3. The processing of the topological event \( e_i \) often leads to scheduling new velocity update events since the topological structure representing nearest-neighbor relationships is changed. It also leads to scheduling of one or more new topological events. Invalidated topological events may be deleted from the queue. These operations are specific for different data structures.

The complexity of the construction of the topological data structure, time required for its modification and for scheduling new events will be examined in the next chapter. The time required to insert the velocity update events for all objects during the preprocessing step is \( O(n) \), since the events do not need to be sorted. Changing the velocity of the object due to a velocity update event takes constant time. Insertion and deletion from the event queue can be done in \( O(\log n) \) time. The structure of the event queue allows
accessing all the events scheduled for a given object directly. The deletion and insertion events require $O(\log n)$ time. A complexity table for different data structures will be presented in the next chapter.

The algorithm can be used not only to efficiently maintain data structure during the physical system simulation, but also to perform various dynamic queries on the set of moving objects, such as finding a nearest-neighbor, reporting all nearest-neighbors, finding the closest pair or predicting the next collision at any moment of time $t$. It is also possible to gather some information about the system during some period of time. All collisions that took place over some time interval $[t_1, t_2]$ can be stored in the collision list. The list of closest pairs can be created to analyze how the proximity relationships change during simulation. These problems will be discussed in the next chapter.

8.8 Implementation Details

For the above algorithm the following data structures should be maintained:

*Set of objects.* The list of objects with the attributes such as:

1) object identification number;
2) mass, radius;
3) position, velocity, external forces;
4) reference to the element of the topological data structure (for example, a cell in a regular spatial subdivision) to which this object belongs;
5) list of references to the events in the event queue scheduled for the object.

*Topological data structure.* The following data structures are considered: dynamic Delaunay triangulation, regular spatial subdivision, regular spatial tree or set of segment trees. The topological data structure allows storage of the proximity information for a set of moving objects efficiently and optimization of the collision detection. It has references to the objects, which allows accessing any object associated with an element of the topological data structure in constant time.
Event queue. For each event the event queue stores the time of the event, event type, list of elements participating in the event (pointers to two objects for velocity update, and pointers to the elements of the topological structure for topology update).

Cross-references from the topological data structure and the event queue to the objects are used. It is convenient for the algorithm and inexpensive in terms of memory. Note also that if a dynamic Delaunay triangulation is used as a data structure, then objects and their attributes can be stored directly at the DT vertices. Thus the need for the separate list of objects is eliminated.

The number of the objects in the object list is \( O(n) \). The worst case number of the events in the event queue is \( O(n^2) \), since up to \( O(n) \) collision events can be scheduled for each object\(^1\). For all the data structures being considered, only one topological event per object is scheduled. Insertion and deletion operations will still require \( O(\log n) \) time. The size of the topological data structure depends on the choice of data structure and is discussed in the following chapter.

8.9 Time-Step Oriented Simulation

A more complex scheme of time-step oriented simulation that is used to represent complex physical systems is now described.

During the simulation, the state of the system at time \( t \) is “known”. A system of algebraic or differential equations, which describes the evolution of the simulated system, is constructed [Kamat 93, Vinogradov 92]. For example, in mechanical simulation of granular-type materials, these systems of differential equations are based on general laws of rigid body mechanics. Such parameters as positions of the bodies, their velocities, types of contact (for example, plastic or elastic) between bodies and boundaries, groups of bodies (clusters or quasi-rigid bodies), internal forces and external force fields are

\(^1\) Note that the total number of topological events in the queue never exceeds \( O(n) \) (see the following chapter for detailed analysis).
considered to evolve to the next simulation step (see Chapter 10, Section 10.4 for additional information). By solving the differential system numerically it is possible to predict the behavior of the system in a time interval \((t, t + \Delta t)\) for some selected time step \(\Delta t\). The solution of the system describes, in particular, the trajectories of bodies in the time interval \((t, t + \Delta t)\) and the velocities of the bodies in the next time step. The trajectories can be approximated by curves, or, in the simplest case, by straight lines [Vinogradov 92].

The velocity update can happen for two reasons: because bodies collide or because the velocity is changed as a result of the trajectory calculation at the next time step.

**Definition 8.9.1** A *predict trajectory event* \(p\) in a time-step oriented computer simulation happens when the velocity of the body is updated due to the calculation of the system state at the next time step.

**Definition 8.9.2** A *collision event* \(c\) in a time-step oriented computer simulation happens when two bodies collide or a body collides with a boundary.

The velocities of the bodies are updated on each time step. The collision events are scheduled as needed. During time interval \((t, t + \Delta t)\) several topological or collision events can happen. All events are put into the general event queue \(Q\) in sorted order.

**8.10 Algorithm Description**

The general scheme of the algorithm for the dynamic maintenance of the data structure during time-step oriented simulation is follows.

*time-step algorithm 8.10.1*

**Preprocessing**

1. Construct the topological data structure at the starting position. Schedule predict trajectory events for all bodies at time \(t_0 = 0\).

**Main loop**

2. While the event queue is not empty do:
2.1 Extract the next event from the event queue $Q$ and determine the type of the event (topological event, predict trajectory event or collision event);

2.2 Advance the simulation clock to the time of this event $t_e$;

2.3 Process the event and update the event queue:

   if the event is a topological event $e_i$:
   - modify the topology of the data structure;
   - schedule new topological events;
   - check for new collisions;

   if the event is a collision event $c_i$:
   - change the velocities of the bodies participating in a collision;
   - delete events that become invalid from $Q$;
   - schedule new events and insert them into queue in sorted order;

   if the event is a predict trajectory event $p_i$:
   - compute the trajectory of the body;
   - delete events that become invalid from $Q$;
   - schedule the next predict trajectory event at time $t_e + \Delta t$;
   - schedule new topological and collision events and insert them into the queue in sorted order.

This algorithm is similar to Algorithm 8.7.1. The main difference is that this algorithm has an additional event due to the specifics of the time-step simulation. A system of algebraic or differential equations must be solved and the velocities of all bodies must be updated at each time step. This operation can take $\Omega(n)$ time since the changes should be recorded for each body. This overhead is unavoidable in a real system simulation since the trajectories of the bodies and their velocities are computed at each time-step in the numerical integration algorithm anyway. Only one predict trajectory event and one topology update event should be stored for each body. The maximum size of the queue is $O(n^2)$. 
CHAPTER 9: DYNAMIC DATA STRUCTURES

This chapter studies the specifics of the application of some dynamic data structures to the collision detection algorithm described in the previous chapter. Note that properties of the generalized VD and DT as well as the formulas for INCIRCLE test computations, obtained in the earlier sections of the thesis, will be used in the algorithms suggested for dynamic DT construction.

All of the methods presented in this chapter, including the novel idea of application of the dynamic generalized VD for collision detection optimization in the computer simulation of a system of moving objects, were first described in [Gavrilova et. al. 96]. Other computational geometry data structures discussed in this chapter, such as regular spatial subdivision, the regular spatial tree and the set of segment trees, were also considered in dynamic settings and modified specifically for the purpose of collision detection optimization in computer simulation. The comparison analysis of these data structures together with their experimental studies on the example of rigid-body simulation model is the main contribution of this chapter.

The outline of the chapter is now presented. The field of dynamic computational geometry is described and the classification of the dynamic data structures suitable for collision detection is presented. Dynamic generalized Voronoi diagrams and Delaunay tessellations are introduced and their properties are studied. The regular spatial subdivision, the regular spatial tree and the set of segment trees are considered as alternative data structures.

The algorithms for data structure construction, definition of the topological events, updating of the event queue and the data structure are presented for 2 and $d$-dimensional cases. The required number of collision checks is reduced in comparison to the straightforward approach and the complexity of the algorithms is estimated. So far, this is the most comprehensive comparison of collision detection optimization methods that uses various computational geometry data structures and also performs experimental studies of
these data structures. The experiments are conducted on the example of the applied problem of granular-type material simulation.

9.1 Dynamic Computational Geometry

Dynamic data structures, used when a set of objects is assumed to be continuously changing, have been considered in the literature. Atallah introduced the field of dynamic computational geometry by considering problems whole inputs are not static objects but moving objects [Atallah 85]. He defined the term of \( k \)-motion to describe the geometric properties of sets of moving points:

**Definition 9.3.3** A \( k \)-motion in \( d \)-dimensional space is the motion of an object defined by a function \( p(t) = \sum_{i=0}^{k} C_i t^i \), where \( t \) is time and \( C_i \in R^d, i = 0..k \) are constant vectors [Atallah 85].

The motion of the points is therefore a polynomial function of time of maximum degree \( k \).

Davenport-Schinzel sequences, which will be used later in this section, are introduced as follows.

**Definition 9.3.4** Let \( A(n) = \{a_1, a_2, ..., a_n\} \) be an alphabet. A Davenport-Schinzel sequence \( L_{n,s} \) is defined as the set of all strings over the alphabet \( A(n) \) that do not contain any \( a_i a_i \) as a substring and do not contain any \( \xi_{ij}^s (i \neq j) \) as a subsequence, where \( \xi_{ij}^s \) is an alternating string of length \( s + 2 \), defined as follows: \( \xi_{ij}^1 = a_i a_j a_i \),

\[
\xi_{ij}^{2p} = \xi_{ij}^{2p-1} a_j \quad \text{and} \quad \xi_{ij}^{2p+1} = \xi_{ij}^{2p} a_i, (p \geq 1) \quad \text{[Atallah 85]}
\]

Let \( \lambda_s(n) \) be the maximum length that the string \( L_{n,s} \) may have. Then, \( \lambda_s(n) < cn \log n \), where \( c \) is a constant depending on \( s \) [Atallah 85]. This function is used to obtain estimates on the number of intersections of \( n \) real continuous functions of time and to
estimate the number of times the closest (furthest) pair of moving points changes as $t$ increases to infinity (see, for example, [Roos 97]).

These results are used in most papers about Voronoi diagrams for moving points. Atallah, however, did not consider Voronoi diagrams themselves. He obtained results for dynamic convex hull maintenance and some steady-state problems (when $t = \infty$).

A number of other problems in dynamic computational geometry, such as finding minimum spanning trees, maintaining the closest pair, computing minimum diameter, proximity queries and range searching under the $k$-motion model were considered in [Basch et. al. 97, Chan 97, Devillers et. al. 94, Golin et. al. 92, Schwerdt et. al. 97]. An algorithm to maintain a closest pair of polygon edges when the polygon undergoes dynamic transformation was suggested in [Eppstein and Erickson 98]. The possibility of application of this method to pool simulation was also discussed in this paper.

9.2 Dynamic Data Structures

This section briefly discusses the data structures suggested for collision detection optimization in this thesis and gives an extensive literature review of the similar computational geometry data structures that can be applicable to solve various problems that appear in dynamic systems. These data structures were originally considered in [Gavrilova et. al. 96].

The dynamic generalized Voronoi diagram is the first data structure suggested for optimization of collision detection in this chapter. More and more applications of Voronoi diagrams to dynamic problems from various fields can be found in the literature. Voronoi diagrams are used to represent highly detailed geometric models in computer graphics [Hoppe 96] and computer animation [Park and Fusell 97], for surface reconstruction and mesh simplification in geometric modeling [Algorri and Schmitt 96] and computer aided geometric design [Hamann and Tsai 98].

In robotics, the VD is used to decide whether a robot can follow the path without colliding with obstacles [Canny et. al. 91, Dunlaing and Yap 85]. Some applications of
VD to motion planning are found in [Dunlaing and Yap 85, Rohnert 91, Roos and Noltemeier 92].

The regular spatial subdivision\textsuperscript{1} method is the second data structure considered in the thesis. It is based on the partitioning of the whole simulation space into subdomains. In the simplest form, the space is subdivided into axis-parallel rectangles in 2D or cubes in 3D, called cells. Each cell contains references to the objects that intersect it. Each object is associated with the cell where its center currently resides. Collision detection is performed on the objects residing in the neighboring cells.

The space occupancy method for computer animation based on a similar idea is found in [Uchiki et. al. 83]. The cell method which is also based on the same idea is used in molecular dynamics [Marin et. al. 93], computational physics [Rapaport 93], computer simulation of granular systems [Krantz 93] and multibody systems [Lubachevsky 91, Lubachevsky and Stillinger 90]. The problem of choosing the cell size in a molecular gas model for spheres with uniform radius was considered in [Kim et. al. 97].

The regular spatial tree is the third data structure that is proposed for the modification of the regular spatial subdivision method in this chapter. The cells are stored in a balanced binary tree. Only those cells that actually contain objects are stored. Each node of the tree is associated with a cell and implicitly with a set of objects intersecting the cell. The cells are ordered according to their coordinates: by the first coordinate, then by the second coordinate, etc. This method decreases the space requirements at the expense of efficiency.

The regular spatial tree has a structure similar to the binary space partitioning tree (BSP). The BSP tree represents a space, partitioned by hyperplanes that are usually orthogonal to the coordinate axes [Fuchs et. al. 80, Samet 90]. In the simple case the plane is divided by

\textsuperscript{1} The regular space subdivision method was introduced in [Gavrilova et. al. 96] specifically for the purpose of collision detection optimization independently of the similar research presented in [Agarwal et. al. 97, Kim et. al. 97].
hyper-rectangles with their sides parallel to the coordinate axes. Each node of the tree is then associated with a hyper-rectangle. BSP trees are considered for detecting the first collisions between two polygonal objects (not for the collision detection optimization in the system of moving objects) [Lin 93, Held 98], to represent the dynamics of moving segments [Agarwal et. al. 97], for hidden-surface removal in solid modeling [Paterson and Yao 92] and for visualization of dynamic scenes [Torraz 90]. These references are concerned with algorithms for modification of the tree when a node is inserted or deleted due to the movement of the objects and usually do not study the BSP tree as a data structure for collision detection optimization in a multibody dynamic system.

The set of segment trees method is the last method presented in this chapter. In this approach, each of the objects is projected onto each of the coordinate axes. A set of sorted segments is then represented as a set of balanced segment trees, which store the list of segment’s endpoints in the sorted order. The segment endpoints that belong to the same segment are each attributed with a reference to each other and it is easy to determine the next intersection between a pair of segments. Collision between objects is only possible if all of the projections on coordinate axes are intersecting.

Each of the segment trees could be implemented as an interval tree [Cormen et. al. 92]. The interval tree is also mentioned in [Lin 93] as a data structure appropriate to detect a collision between two polygonal objects. Some other data structures, for example, range search trees [Basch et. al. 97], were also suggested for collision detection optimization in a dynamic system. OBB-trees and some other data structures were suggested for efficient collision detection between polygonal objects in [Gottschalk et. al. 96, Lin 93]. A heap, which contains pairs of objects sorted by the distance between objects, is used for the same purpose in [Mirtich and Canny 95].

In this chapter, the methods based on regular spatial subdivision, regular spatial trees and sets of segment trees are discussed with application to the collision detection problem. Note that spherical objects are considered for simplicity, but the methods are in general applicable to arbitrarily shaped objects if these are approximated by spheres. Algorithms
for data structure maintenance are described for reduced model simulation to keep the explanation of topological and velocity update events clear. The general scheme of the time-step algorithm can be used as well, since additional predict trajectory events depend on the simulation system and not on the collision optimization data structure. The complexity of each method is estimated and a comparison of the complexity of the methods is based on their efficiency, memory requirements and extendibility of approach for various inputs is conducted. An experimental comparison of the methods is conducted for the time-step simulation model.

### 9.3 The Dynamic Generalized Voronoi Diagram

The data structure discussed in this section is the dynamic Voronoi diagram and the corresponding Delaunay tessellation in generalized metrics, such as power, Manhattan ($L_1$), supremum ($L_\infty$) and Euclidean in 2 and, in general, $d$-dimensional space.

**Definition 9.3.1** A *dynamic generalized Voronoi diagram* is a generalized Voronoi diagram of a set of sites that can change their spatial positions in time.

**Definition 9.3.2** A *dynamic generalized Delaunay tessellation* is a generalized Delaunay tessellation that corresponds to the dynamic generalized VD.

The problem of maintaining dynamic generalized Voronoi diagram and Delaunay triangulation in 2D is addressed in the following sections. Note that the different problem of dynamic maintenance of static Voronoi diagrams when sites are inserted or deleted over time is sometimes referred to as dynamic Voronoi diagram maintenance in the literature [Doberndt and Yvinec 93].

### 9.3.1 Algorithms for the Construction of Dynamic Voronoi Diagram

Most of the algorithms for construction of the dynamic Voronoi diagrams and corresponding Delaunay triangulations have been developed for point sites in the Euclidean metric in the plane.
A topological-event-oriented maintenance algorithm for Voronoi diagrams was first introduced by Bajaj and Bouma [Bajaj and Bouma 90]. They defined topological events assuming a constant motion of the points in the plane. Their algorithm requires $O(\log n)$ time to process each topological event. They also estimated the maximum number of topological events that can happen during a time step within which points can move into at most a single neighboring Voronoi cell as $O(n)$ topological events.

A more general step-by-step maintenance algorithm for VD diagram of moving points in the plane was developed later by Guibas, Mitchell and Roos [Guibas et. al. 91]. The number of topological events that can happen between two collisions is bounded by $O(n^2\lambda_s(n))$, where $\lambda_s(n)$ is the maximum length of an $(n,s)$-Davenport-Schinzel sequence and $s$ is a constant depending on the motion of the points. The function $\lambda_s(n)$ is slightly superlinear for any constant $s$.

Similar estimates were obtained for $d$ dimensions. The number of topological events in VD of moving points that can happen between two collisions is bounded by $O(n^d\lambda_s(n))$ [Roos 97]. Roos also gave a clear definition of a topological event and studied the topological structure of VD of moving points [Roos 90]. He described an algorithm for dynamic VD maintenance, performing the update of data structure in $O(\log n)$ time per topological event. Voronoi diagrams for moving points have also been studied in [Aunuma et. al. 90, Fu and Lee 91, Huttenlocher et. al. 92].

A worst-case number of topological events that can happen between two collisions is $\Omega(n^2)$ [Roos 90]. This bound is obtained by considering $n/2$ points placed on a straight line that moves parallel to the $x$ coordinate axis and passes by $n/2$ of fixed points located along $x$ axis. The intervals between point are taken so that the new point enters the VD for static points after the previous point already left it [Roos 90].

There is a gap between the best known upper and lower bounds for the maximum number of topological events. Some recent results that narrow this gap have been obtained for the
case when \( k \) out of \( n \) points are moving [Roos 97]. The studies in this direction are continuing.

### 9.3.2 Algorithms for the Dynamic Generalized Voronoi Diagram Construction

The problem of constructing and maintaining a dynamic Voronoi diagram for set of moving objects other than points has rarely been addressed in the literature. However, situations often arise, in computer simulations as well as in other fields, where it is necessary to study and monitor the behavior of moving objects in the plane or in higher dimensions.

Studies seeking efficient algorithms to solve the collision detection problem for moving geometric objects have been performed by Gupta, Janardan and Smid [Gupta et. al. 94]. They studied the dynamic Voronoi diagram for a set of points, lines or hyper-rectangles moving with constant velocities in the plane. They developed efficient algorithms for several problems, such as detecting the nearest collision between two objects and computing the time when such an event occurs. The time complexity of the algorithm for \( n \) points in the plane is \( O(n^2) \).

Abramowski and his colleagues directly addressed the problem of moving circles in the plane. As a solution they suggested that circles be replaced by regular \( k \)-gons moving along piecewise linear trajectories, so that the next collision can be determined in \( O(kn^{2-\varepsilon}) \) time, where \( \varepsilon \) is a positive constant [Abramowski et. al. 88].

A related problem sometimes addressed in the literature deals with planning the motion of a disk between polygonal obstacles. Methods suggested by O'Dunlaing and Yap [O'Dunlaing and Yap 85] as well as by Rohnert [Rohnert 91] give an \( O(n \log n) \) algorithm, where \( n \) is the total complexity of the obstacles. The more complex problem of planning the motion of the disk between moving sites was discussed in Roos and Noltemeier [Roos and Noltemeier 92]. Canny suggested construction of the \( L_\infty \) VD to plan an obstacle avoiding motion of the object [Canny et. al. 91].
Recently, the generalization of dynamic VD to other metrics was considered. Chew considered Voronoi diagram for moving points in $L_1$ and $L_\infty$ metrics and bounded the number of topological events that can happen between two collisions by $O(n^2\lambda_s(n))$ [Chew 93]. Imai K. and Imai H. studied dynamic multiplicatively weighted Voronoi diagrams in Euclidean space and established that the upper bound on the number of topological events is the same as for the ordinary dynamic Voronoi diagram: $O(n^2\lambda_s(n))$ [Imai and Imai 93].

### 9.3.3 Dynamic Generalized Delaunay Triangulation Algorithm

Consider the dynamic generalized VD and the corresponding DT in Euclidean, Manhattan, supremum and power metrics in the plane. When the Delaunay triangulation is maintained for a system of moving disks, it is possible to only check collisions between those disks that are neighbors in the Delaunay triangulation. At the time of collision the distance between two disks becomes zero (in any metric), and therefore the disks constitute a nearest-neighbor pair at this moment in time. Hence, there must be an edge in the Delaunay triangulation connecting these two disks, so the collision will not be missed. Each time a new edge appears in the Delaunay triangulation, the pair of disks connected by the edge must be checked for a collision (see Figure 9.3.1). Thus the collision checks only need to be performed between disks that are neighbors in the dynamic Delaunay triangulation.

![Figure 9.3.1 Neighbors in dynamic Delaunay triangulation](image)

The dynamic DT stores the objects at its vertices. Each object has a list of its neighbors represented as edges of the DT from this vertex. The data structure that is used to
represent a dynamic DT is the Quadedge data structure [Lischinski 94]. Each edge of the DT stores pointers to its vertices, pointers to the next counterclockwise edge around each of the vertices, and to two facets on both sides of the edge. The space required to store the data structure is $O(n)$.

Since at time $t_0 = 0$ the set of sites is static, it is possible to construct the static DT for a set of points in $O(n \log n)$ time in the plane or in $O(n^2)$ time in 3D (see methods described in Chapter 2). The methods to construct the generalized DT in optimal $O(n \log n)$ time in the plane and $O(n^{[d/2]+1})$ in $d$-dimensional space have been presented in Chapter 4.

In the planar case, the modification of Fortune's sweep-plane method for the Manhattan and supremum metrics and the swap method for the power metric can also be used (see Chapters 5 and 6).

**9.3.4 The Topological Event**

Consider the problem of maintaining the dynamic Voronoi diagram over a period of time. When the sites move, a dynamic generalized VD undergoes constant changes. However, most of the time only the location of vertices and the lengths of Voronoi edges change, while the proximity relationship between Voronoi sites remains the same. The topological structure of the VD changes only when a transformation of Voronoi edges takes place. It happens when some edges gradually shrink to zero and disappear and new edges appear and start to grow due to the site movements. The following definitions are generalization of definitions given in [Roos 90] for the ordinary dynamic VD.

**Definition 9.3.3** A topological event in the dynamic generalized Voronoi diagram happens at discrete moments in time when the topological structure of the VD is modified and proximity relationships are altered.

Now, consider the DT corresponding to the VD. One of the properties of the Delaunay triangulation is that each triangle is defined by three sites, such that there exists a circle
tangent to these sites which does not contain any other site (see Chapter 3). The topological event in the DT happens when the corresponding edge of the Voronoi diagram gradually shrinks to zero and then a new edge appears and starts to grow.

**Definition 9.3.4** A topological event in the dynamic generalized Delaunay triangulation occurs when four moving sites in a quadrilateral, comprising two neighboring triangles of the Delaunay triangulation, become cocircular according to the metric being used\(^1\).

The handling of a topological event requires swapping of the diagonal in a quadrilateral consisting of two neighboring Delaunay triangles and the scheduling new topological events for the new quadrilaterals (see Chapter 2 for definition of the swap operation). This takes \(O(\log n)\) since the events are stored in a priority queue represented by an AVL tree.

Consider an example. Figure 9.3.2 represents a topological event in a Voronoi diagram of four circles \(P_1, P_2, P_3,\) and \(P_4\) in the Euclidean metric. For simple geometric illustration, consider the case when the circle \(P_1\) moves towards the other circles (as shown in Figure 9.3.2a). At some moment of time \(t\) four circles become cocircular which results in a degenerate Voronoi diagram when the edge between sites \(P_2\) and \(P_4\) is reduced to zero (Figure 9.3.2b).

\(^1\) There exist degenerate situations when the fourth site just touches the circle inscribed between the first three sites without actually entering it. Such situation corresponds to a double (or quadruple, etc) root of the \textit{INCIRCLE} function (see Section 9.3.5), i.e. when the function reaches zero without actually changing sign. In this case, topological event should not be scheduled.
As the circle $P_1$ continues to move it intersects the circle inscribed between sites $P_2, P_3$ and $P_4$ thus contradicting the empty-circle property. Therefore, the topology of the Voronoi diagram is changing and a new edge between sites $P_1$ and $P_3$ is constructed (Figure 9.3.2c).

Figure 9.3.3 demonstrates the changes in the DT corresponding to the above VD. Figure 9.3.3b represents the swap operation, when the edge of the DT quadrilateral between sites $P_2$ and $P_4$ is deleted and a new edge between sites $P_1$ and $P_3$ is constructed. This example illustrates the topological event for four circles with the same radii; thus the resulting Voronoi diagram has straight-line bisectors in the Euclidean metric. In case when circles with different radii or line segments are considered in the Euclidean or power metrics, the swap operation will remain exactly the same; only the shape of the Voronoi regions will change.
The above results can be formalized as the following statements, similar to ones proven in [Roos 90] for the ordinary Voronoi diagram.

For a finite set of sites $S$ the topological structure of the dynamic generalized VD is locally stable. That is, only continuous deformations take place, under small continuous motions of sites. The topological changes in the structure of the dynamic generalized VD are characterized by swaps of adjacent triangles in the Delaunay triangulation.

Note. Finding an upper bound on the number of topological events is a complex problem. From the known results, a conjecture about the complexity of the VD for moving circles in the plane in the Euclidean, power, Manhattan and supremum metrics can be drawn. It is conjectured that the upper bound on the number of the topological events is equal to $O(n^2\lambda_s(n))$ and that this can be proven by applying an algorithm similar to one described in [Guibas et. al. 91]. A proof of this fact is beyond the scope of this thesis.

9.3.5 Determination of the Topological Event

The determination of the time of a topological event requires finding the minimal positive root $t_0$ of the equation $INCIRCLE(P_1,P_2,P_3,P_4) = 0$, where $P_i = P_i(t), i = 1..4$ are the moving sites and where their coordinates are given as analytic functions of time (the proof can be found in [Roos 90]). This result is directly applicable to any weighted generalized DT, where weights of sites are fixed and the coordinates of centers of sites are analytical functions of time.

Denote the coordinates of the four weighted sites of the DT involved in a swap as $P_i = \{(x_i = x_i(t), y_i = y_i(t)), r_i\}, i = 1..4$. The $INCIRCLE$ function is transformed into a polynomial function of time. Formulas for the $INCIRCLE$ functions for the power, Euclidean and Manhattan Voronoi diagrams are found in Chapter 4 (see Theorems 4.4.1,

---

1 Multiple roots with even multiplicity are excluded from consideration, i.e. the topological event is only scheduled when the $INCIRCLE$ function changes sign in $t_0$. 

---
4.5.1 and 4.6.1). Note that a method for the exact computation of \textit{INCIRCLE} test based on floating-point arithmetic is considered at the end of this chapter.

For straight-line disk trajectories in the power metric the determination of the time of a topological event requires the solution of a 4th order polynomial equation. Although the roots of a 4th order polynomial of time can be computed explicitly they are usually computed numerically using iterative methods such as Newton's method, due to the complexity of the explicit formulas (see [Burden and Faires 97]).

For straight-line disk trajectories in the Euclidean metric the determination of the time of a topological event requires the solution of an 8th order polynomial equation, which requires the application of numerical methods [Burden and Faires 97].

For straight-line disk trajectories in the Manhattan or supremum metric the time of a topological event can be computed by solving a system of linear equations.

\textit{Note.} This approach can be extended to \( d \) dimensions. The topological and velocity update events can be described in a manner similar to the 2-dimensional case. Formulas for the \textit{INCIRCLE} test in \( d \) dimensions in the Euclidean, power and Manhattan metrics are obtained in Chapter 4. A \( d \)-dimensional swap can be defined as by [Roos 97]. It is conjectured that the number of the topological events is \( O(n^d\lambda(n)) \) for all metrics considered. It will be demonstrated that the other data structures for collision detection optimization show much better complexity. Thus the \( d \)-dimensional dynamic VD method is generally not recommended for collision optimization in higher dimensions.

\textbf{9.3.6 The Algorithm for the Dynamic Generalized DT Maintenance}

The algorithm for dynamic generalized DT in the plane for motion described by analytical functions will be described in this section.

\textit{Algorithm 9.3.7}

\textit{Preprocessing}

1. Construct the DT and schedule the velocity update events for each object at \( t_0 = 0 \).
Main loop

2. While the event queue is not empty do:
   2.1 Extract the next event from the event queue $Q$ and determine the type of the event.
   2.2 Advance the simulation clock to the time of this event.
   2.3 Process the event:
      - in case of a topological event $e_i$ perform the necessary swap operation;
      - in case of a velocity update event $u_i$ change the velocities of the objects participating in a collision and record that the actual collision detection took place;
   2.4 Update the event queue:
      if the event is a topological event:
         - cancel swaps (topological events) planned for the four quadrilaterals that disappeared;
         - compute and schedule swaps for the four new quadrilaterals and insert them into the event queue;
         - check for collision between two objects connected by the new edge;
      if the event is a velocity update event:
         - cancel swaps planned for quadrilaterals containing the two objects which are colliding;
         - cancel any velocity update events planned for these two objects;
         - compute new swap events for each quadrilateral containing two objects;
         - check for possible collisions involving all neighbors defined by the DT edges and schedule the velocity update events for detected collisions.

9.3.7 Algorithm Complexity

Estimates are obtained for the 2-dimensional case. During the preprocessing stage the DT can be constructed in $O(n \log n)$ by using one of the known methods (see earlier chapters
for more details). The space required to store the DT is $O(n)$ (when the quad-edge data structure is used [Lischinski 94]).

The swap operation takes $O(1)$ time, insertion and deletion from the sorted queue – $O(\log n)$ (since the queue is implemented as an AVL-tree [Cormen et. al. 92]). When a topological event happens, $O(\log n)$ time is spent for scheduling new topological and velocity update events, i.e. to insert an event into the queue in sorted order. One collision time calculation is performed. When a velocity update event happens, $O(\log n)$ time is needed to schedule new topological and velocity update events, and $O(1)$ time is needed to compute the collision time. The maximum number of topological events in the event queue is $O(n)$, since one topological event is scheduled for every DT edge (and the number of edges is $O(n)$). The maximum size of the event queue is $O(n^2)$, since for each edge of the triangulation there is at most one scheduled topological event at any moment of time and there can be up to $O(n^2)$ collision events in the event queue at any moment of time. The maximum number of events is, however, undetermined, since the trajectories of the bodies are irregular.

The maximum number of collision checks per velocity update can be estimated by the maximum number of neighbors of an object at any moment of time. This number is $O(n)$ in the worst case (when a site in the DT is connected to all other sites). However, for planar DT’s, the average number of neighbors of a site is only a constant [Preparata and Shamos 86, Section 5.5.1]. The total number of collision checks per topological event is equal to the number of new neighbors of an object after the topological event occurs. For the dynamic DT, this number is exactly one (since one new edge appears in the DT due to a topological event).

The number of collisions between objects during the simulation cannot be bounded since the simulation time can be generally unlimited and because the objects move in a bounded space (i.e. unless the objects stop moving, the collisions will keep occurring). Hence, the overhead associated with the use of a particular DT algorithm is usually
estimated by the maximum number of topological events that can occur before a collision happens [Roos 90]. For a planar generalized DT this number can be as large as $O(n^2 \lambda_1(n))$. However, this number can be as low as $O(1)$ for densely packed systems [Chew 93, Imai and Imai 93].

9.4 Regular Spatial Subdivision

This chapter describes the algorithm for optimization of collision detection using regular spatial partitioning. The proposed approach is based on the partitioning of the simulation space into subdomains (see Figure 9.4.1).

For simplicity, the space is subdivided into axis-parallel hypercubes in $d$-dimensional Euclidean space (squares in 2D or cubes in 3D). These are generically called cells in the sequel. The cells are stored in a multi-dimensional array and are accessed directly in constant time. Each sphere $P_i = (p_i, r_i)$ is represented by a pair, including the coordinates of its center $p_i = p_i(t)$ and the radius $r_i$. Assume that the size of the simulation domain is such that there are $k$ cells in each direction.

Consider a $d$-dimensional box with a diameter $l$ as a domain. The size of a cell must not be smaller than the maximum diameter of a sphere. The value $k$ can be determined as the diameter of the simulation space divided by the maximum diameter of sphere $M = \max_{P_i \in S}(2r_i)$, i.e. $k = \left\lceil \frac{l}{M} \right\rceil$. 

![Figure 9.4.1 The regular spatial subdivision](image)
Denote the minimum diameter of a sphere by $m = \min_{R_i \in S}(2r_i)$.

**Assumption 9.4.1** The ratio $\gamma = M/m$ is a constant, which does not depend on $n$.

Under Assumption 9.4.1 the maximum number $n_c$ of spheres within each cell is bounded by a constant (since the volume of a cell is $M^d$ and the volume of a sphere is at least $\alpha m^d$, where $\alpha > 0$ is some constant depending on $d$) [Behnke et. al. 83].

At any discrete moment of time, each object is attached to the cell where its center is located. Each cell contains a list of objects, which currently reside in this cell (note that the length of this list is not greater than a constant due to Assumption 9.4.1). Collision checks need only to be performed between disks that reside in neighboring cells of the regular spatial subdivision.

### 9.4.1 The Topological Event

**Definition 9.4.1** A topological event in a regular spatial subdivision occurs when the center of a sphere moves from one cell to another.

Note that on the first simulation step $t_0$, possible collisions should be scheduled for a given object with all objects from all neighboring cells. A topological event occurs when an object moves into one of these neighboring cells. Collisions with objects from some of these cells were already computed and only those objects in the $3^{d-1}$ new neighboring cells must be checked for collisions. For example, there are only three new neighboring cells in the plane.

The time of a topological event can be determined exactly by computing the time when the center of the sphere passes the boundary of a cell. Handling of the topological event requires a constant number of operations in order to update the data structure: changing the cell coordinates of the object, and moving the object from one cell to another. It is also necessary to schedule the topological event for the object. This takes $O(\log n)$ time.
For each object only one topological event can be scheduled at any particular moment of time.

9.4.2 The Algorithm for the Regular Spatial Subdivision Maintenance

An algorithm for regular spatial subdivision in $d$ dimensions for motion described by analytical functions is presented.

Algorithm 9.4.2

Preprocessing
1. Construct the regular spatial subdivision and schedule velocity update events for time $t = 0$.

Main loop
2. While the event queue is not empty do:
   2.1 Extract the next event from the event queue $Q$ and determine the type of the event.
   2.2 Advance the simulation clock to the time of this event.
   2.3 Process the event:
      - in case of the topological event $e_i$, change the cell coordinates of the disk, and move a disk from one cell to another;
      - in case of the velocity update event $u_i$, change the velocities of the objects participating in a collision;
   2.4 Update the event queue:
      if the event is a topological event:
      - schedule a new topological event when the object will leave the cell that it just entered and insert this event into the queue;
      - check for collision between the object and all objects in the new $3^{d-1}$ neighboring cells;
      if the event is a velocity update event:
      - cancel topological events for each of the colliding objects;
- cancel velocity update events planned for them;
- compute two new topological (cell change) events, one for each object;
- check for possible collisions with all object neighbors residing in the neighboring cells ($3^d$ cells in total);
- schedule the velocity update events for objects that participated in the collision.

### 9.4.3 Algorithm Complexity

Since an object can cross only $O(k)$ cells during motion without collisions (because it cannot leave the simulation domain), the maximum number of topological events before a collision happens is $O(nk)$.

The space requirement for the method is $\Theta(k^d + n)$. The regular spatial subdivision can be constructed by first allocating $k^d$ cells and then placing each of the objects into the appropriate cell based on their coordinates at the moment $t = 0$. The data structure is initialized and constructed in $\Theta(k^d + n)$ time.

Under Assumption 9.4.1 the total number of neighbors for each object is bounded by a constant. Therefore, the total number of collision checks per velocity update is also $O(1)$. The total number of collision checks per topological event is equal to the number of new neighbors of an object after the topological event occurs. Thus, this number is also $O(1)$.

The efficiency of this method strongly depends on the distribution of diameters of objects. For the algorithm to perform well, the maximum number $n_c$ of objects that can fit into a cell must be significantly smaller than the total number of objects $n$. If a system contains one large object and many small objects, then it is possible that a single cell contains all of the objects, and the efficiency of the collision detection is no better than that of the straightforward algorithm (i.e. each pair of objects must be checked for a collision). However, when all of the objects in the system have approximately same size, then each topological event requires only a constant number of collision checks.
9.5 The Regular Spatial Tree

It is possible to improve the space requirement for the previous approach by storing only those cells that contain objects (see Figure 9.5.1).

![Figure 9.5.1 Sectors stored in regular spatial tree](image)

The cells are stored in an AVL tree. Each node of the tree is associated with a cell in \(d\)-dimensional Euclidean space and implicitly with a non-empty set of spheres \(\{P_1, P_2, ..., P_q\}\) whose centers belong to the cell. The ordering is introduced among cells such that they are ordered by their first coordinate (for example, the coordinates of the cell can be taken as coordinates of the left lower corner). The cells with the same first coordinate are ordered by the second coordinate, etc. For example, in the planar case the cells are sorted first by the \(x\) coordinate, and then by the \(y\) coordinate. Then it is possible to store all occupied cells in an AVL tree, sorted according to the proposed ordering.

9.5.1 Algorithm Complexity

The algorithm complexity is estimated for the \(d\)-dimensional case, where \(d\) is a constant. The method reduces the storage to \(\Theta(n)\), since the number of occupied cells cannot exceed the total number of objects in the system. On the other hand, each cell access requires \(O(\log n)\) time in the worst case, since cells are stored in an AVL tree. A query to report all objects residing in a particular cell requires a search in the binary tree.
to be performed. If the requested cell is found, then all of its contents are reported. If the cell is not found in the tree, then an empty set is reported.

The estimates for the number of topological events are the same as for the previous method. However, any operation involving modifications of the data structure (such as moving an object from one cell into another) will require $O(\log n)$ time compared to $O(1)$ direct array access used in the previous method. Hence, each topological event requires $O(\log n)$ operations on the tree. The initial construction of the data structure requires $O(n \log n)$ time, because $n$ objects are inserted into the binary tree. This is usually less than $\theta(n + k^d)$ time required for initialization of a regular subdivision.

Similarly to the previous method, under Assumption 9.4.1 the total number of neighbors for each object is bounded by a constant. Therefore, the total number of collision checks per velocity update and the total number of collision checks per topological event are also $O(1)$.

### 9.6 Sets of Segment Trees

A set of sorted segment trees can be also used for optimization of the collision detection (see Figure 9.6.1).

![Figure 9.6.1 The set of segment trees](image)

The segment tree representing the projections of the disks onto an axis can be represented by a sorted sequence of segment endpoints, which is stored, for example, in a balanced
binary tree. Note that the size of the tree is fixed, since the total number of objects does not change over time.

The data structure used for this method is a set of $d$ balanced binary trees each containing the set of segments along one coordinate axis. The initial data structure construction requires $O(n \log n)$ time, since $2n$ segment endpoints are inserted into each of the binary trees.

9.6.1 The Topological Event

Definition 9.6.1 A topological event occurs when two segment endpoints on one of the axes collide.

When a topological event happens, the bounding boxes (that contain objects in their interior and have size equal to diameter of the object they contain) of the two corresponding objects are tested for intersection. If they intersect, then a collision check is performed between the two objects. Note that a maximum of one collision check is performed per topological event (and none in most cases).

As the objects move, it is necessary to maintain the sequence of segment endpoints in sorted order for each coordinate axis. Therefore, it is required to detect collisions between segment endpoints. When two neighboring endpoints in the sequence collide, they are exchanged in the tree. Note that in the implementation of this method it is not required to rebalance the tree, but just to exchange references to segment endpoints in two consecutive nodes in the tree. When two segment endpoints collide, it is also necessary to schedule new collisions (a maximum of two new collisions can be scheduled). This requires $O(\log n)$ time since it involves insertions into the event queue.

The number of topological events scheduled for an object at any moment of time does not exceed $4d$, since two topological events are scheduled for each segment endpoint on each of the coordinate axis.
9.6.2 The Algorithm for the Set of Segment Trees

An algorithm for segment trees in $d$ dimensions for motion described by analytical functions is presented.

Algorithm 9.6.2

Preprocessing
1. Construct the set of segment trees and schedule the velocity update events for each object at $t_0 = 0$.

Main loop
2. While the event queue is not empty do:
   2.1 Extract the next event from the event queue $Q$ and determine the type of the event.
   2.2 Advance the simulation clock to the time of this event.
   2.3 Process the event:
      - in the case of a topological event swap two neighboring segment endpoints in the tree;
      - in the case of a velocity update event change the velocity of the object;
   2.4 Update the event queue:
      if the event is a topological event:
      - schedule possible collisions for the two swapped segment endpoints and insert the events into the queue;
      - if the bounding boxes of the two corresponding objects intersect then perform a collision check for the two objects.
      if the event is a velocity update event for an object:
      - cancel topological events for the object;
      - schedule new topological events for the object (a maximum of $2d$ events);
      - check for possible collisions based on the intersections between the projections of objects onto one of the coordinate axes.
9.6.3 Algorithm Complexity

Algorithm complexity estimates are given for the $d$-dimensional case, where $d$ is assumed to be a constant. Half of the collisions between segment endpoints, when two segments start to intersect, might result in collision checks between objects. The remaining collisions correspond to topological events, when two segments stop intersecting (no collision checks are required for these events).

The segment tree is constructed initially by sorting the endpoints of the segments in $O(n \log n)$ time. The space requirement is $\theta(n)$ since for each object two segment endpoints are stored in each of the segment trees.

The maximum number of topological events before a collision happens is the number of endpoint collisions in all segment trees. Assuming that the disks move along straight-line trajectories, it can be estimated by $\theta(n^2)$, which is the total number of intersections in a set of $n$ lines. Consider how segment points on one of the coordinate axes move in time. Since the objects move along straight-line trajectories, an endpoint movement can be described by an equation $x_i(t) = x_{i0} + v_i t$, where $x_{i0}$ is an initial position of an endpoint and $v_i$ is its velocity. This is an equation of a line in the plane. The collision between two segment endpoints corresponds to the intersection between two lines. Consequently, the maximum number of endpoint collisions can not exceed the maximum number of intersections between $2n$ lines in the plane, i.e. it is $O(n^2)$ [Roos 90].

The maximum number of collision checks per velocity update for an object is $\theta(n)$, since it is the maximum number of the collision checks that can be performed for an object with all other objects in the system. This bound can be achieved when the bounding boxes of all objects intersect (an illustration is given in Figure 9.6.2).
However, under Assumption 9.4.1 a bounding box of an object (a sphere) can intersect only a constant number of other bounding boxes, i.e. the total number of collision checks per velocity update is $O(1)$. Consider a bounding box $B_1$ of an object $P_1$ with center at $p_1$, which has the maximum diameter $M$. Consider some other bounding box $B_2$ (see Figure 9.6.3).

If $B_1$ and $B_2$ intersect, then $B_2$ must lie within a cube $C$ centered at $p_1$ with diameter $3M$ (see Figure 9.6.3). The volume of $C$ is $(3M)^d$ and under Assumption 9.4.1 it can contain only a constant number of sites in its interior because the volume of each other object is at least $\alpha m^d$, where $\alpha$ is some constant and $m$ is the minimum diameter of an object (see Section 9.4).
Note, however, that the total number of comparisons between bounding boxes performed per velocity update can still be $\Theta(n)$ in the worst case since it is possible that all projections of objects intersect on one of the coordinate axis.

9.7 Performance Analysis

The complexities of algorithms for 2 and $d$ dimensions are now summarized.

The performance of the described algorithms for the planar case under Assumption 9.4.1 is presented in the following table, where $n$ is the number of objects in the simulation space and $k$ is the maximum number of cells along each of the coordinate axes in the regular spatial subdivision and regular spatial tree methods (this applies to Table 9.7.1 and Table 9.7.2).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Straightforward</td>
<td>$\Theta(n)$</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Dynamic DT</td>
<td>$O(n)$</td>
<td>1</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n^2 \lambda_5(n))$</td>
<td>$O(n \log n)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>Subdivision</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$\Theta(nk)$</td>
<td>$\Theta(n+k^2)$</td>
<td>$\Theta(n+k^2)$</td>
</tr>
<tr>
<td>Spatial tree</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$\Theta(nk)$</td>
<td>$O(n \log n)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>Segment trees</td>
<td>$O(1)$</td>
<td>0 or 1</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td>$\Theta(n^2)$</td>
<td>$O(n \log n)$</td>
<td>$\Theta(n)$</td>
</tr>
</tbody>
</table>

A – Collision checks per velocity update  
B – Collision checks per topological event  
C – Time overhead per topological event (excluding collision checks)  
D – Time overhead per velocity update event  
E – Worst case number of topological events that can happen between two collisions

1 This number is only $O(1)$ on average [Preparata and Shamos 86].
F – Initialization
G – Space

Table 9.7.1 Algorithm efficiency comparison table for planar case

Note that the worst-case number of collision checks that can be performed between two velocity update events is $O(n)$ for VD while for other data structures it's $O(1)$. The number of collision checks per topological event is a constant for all methods. The worst-case number of topological events that can happen between two collisions is also the largest for the VD - it's $O(n^2 \lambda_s(n))$. The subdivision method is the most space consuming method.

The implementation of these methods and test results presented in the next chapter allow a detailed comparison of these data structures and also allow a recommendation to be made on the selection of the most appropriate method for some applied problems based on the properties of the simulated system.

The following table displays the performance of the algorithms under Assumption 9.4.1 for $d$-dimensional case, $d \geq 3$, where $d$ is considered to be a constant.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic DT</td>
<td>$O(n)$</td>
<td>1</td>
<td>$O(\log n)$</td>
<td>$O(n^d \lambda_s(n))$</td>
<td>$O(n^{d/2})$</td>
<td>$O(n^{d/2})$</td>
</tr>
<tr>
<td>Subdivision</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$\theta(nk)$</td>
<td>$\theta(n + k^d)$</td>
</tr>
<tr>
<td>Spatial tree</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$\theta(nk)$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Segment trees</td>
<td>$O(1)$</td>
<td>0 or 1</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td>$\theta(n^2)$</td>
<td>$O(n \log n)$</td>
</tr>
</tbody>
</table>

A – Collision checks per velocity update
B – Collision checks per topological event
C – Time overhead per topological event (excluding collision checks)
D – Time overhead per velocity update event
E – Worst case number of topological events that can happen between two collisions
F – Initialization
G – Space

Table 9.7.2 Algorithm efficiency comparison table for \( d \) dimensions

Note that the constant \( d \) participates in some of the estimates. For example, the time required for topological event and for velocity update event in the dynamic VD is

\[
O\left(\log\left[n^{\left\lfloor d/2 \right\rfloor}\right]\right) = O\left(\left\lceil d/2 \right\rceil \log n\right) = O\left(\log n\right).
\]

The complexity of the VD method is the highest among all data structures in \( d \)-dimensional case. Thus, the spatial subdivision, the spatial tree and the set of segment tree are more appropriate data structures for collision optimization in higher dimensions.
CHAPTER 10: APPLICATIONS TO GRANULAR-TYPE MATERIAL SIMULATION

The application of the algorithms developed for collision detection problems in the computer simulation of granular-type material systems is discussed in this chapter.

First, the implementation of the collision detection optimization algorithm (see Chapter 8) for all of the discussed data structures (see Chapter 9) is considered. The experimental studies are conducted for the example of shaker mill simulation model. The efficiency of the methods is compared to that of the straightforward implementation of the collision detection method. The criteria for the choice of the most appropriate data structure are presented based on the experimental results.

10.1 Definition of Granular-Type Materials

Granular-type materials (GTM) are physical systems consisting of a number of interacting rigid bodies moving within fixed or moving boundaries. GTM include grain, sand, broken ice, avalanches, and other physical systems.

A granular-type material is represented by a set of interacting rigid bodies moving within fixed or moving boundaries [Cundall and Strack 79, Vinogradov 92]. The motion of such a system can be defined in terms of rigid body dynamics. The system is simulated in 2D space. Assume that each body moves independently of other bodies. The simulation can be used for such systems as broken ice, billiards, gas, sand or shaker mill. At any particular moment in time the position and velocity of each body, as well as forces acting on the bodies, are calculated.

One of the topics that must be addressed when a GTM system is simulated is the optimization of collision detection. This topic is particularly important for the GTM system due to the large number of objects being simulated and due to an expensive overhead associated with recalculating the collisions for all bodies of the system.
10.2 Shaker Mill Model

The shaker mill simulation is one of the widely used granular-type materials modeled. A shaker mill is a device for producing mechanically alloyed materials [Gavrilov and Vinogradov 98]. It is represented as a cylinder filled with steel balls. The cylinder is oscillated in a vertical direction (see Figure 10.2.1). The balls are moving along parabolic trajectories. The collisions between the balls and between the balls and the cylinder are assumed to be central and frictionless, that is, the tangential velocity component of colliding balls is conserved. The cylinder is assumed to have an infinite mass, so that its velocity is not affected by the collisions.

![Figure 10.2.1 The shaker mill model](image)

The balls move in a gravitational force field. The general equation of motion of a body in a force field is given according to the 2nd Newton's law as

\[ \ddot{x} = \frac{1}{m} F(x, t) \] [Goldsmith 60],

where \( x(t) \) is the position vector of the body, \( m \) is the mass of the body, and \( F(x, t) \) is the force field. For the gravitational field (which is independent of \( x \)), this equation can be solved analytically. The solution is

\[ x(t + \Delta t) = x(t) + v(t) \Delta t + g \frac{\Delta t^2}{2}, \quad \dot{v}(t + \Delta t) = \dot{v}(t) + g \Delta t, \quad \Delta t \geq 0, \]

where \( v(t) \) is the velocity vector of the ball, and \( g \) is the gravitational constant.
The time of collision between two balls can be found by finding the minimal positive root \( t_0 \) of the equation

\[
\|x_1(t) - x_2(t)\| = r_1 + r_2,
\]

where \( x_i(t), i = 1, 2 \) are the positions of the centers of two balls, and \( r_i, i = 1, 2 \) are the radii of the balls. For parabolic motion in the gravitational field, the root is determined exactly by solving a second degree algebraic equation. If the root does not exist, the balls will not collide.

When collision takes place, the velocity of the balls participating in the collision changes instantly. The energy loss due to a collision can be expressed through the coefficient of restitution \( \varepsilon \):

\[
\varepsilon = \frac{|v^\text{after}_1 - v^\text{after}_2|}{|v^\text{before}_1 - v^\text{before}_2|}
\]

where \( v^\text{before}_i, i = 1, 2 \) are the normal velocities before the collision, and \( v^\text{after}_i, i = 1, 2 \) are normal components of velocities after the collision [Johnson 85]. A restitution coefficient \( \varepsilon = 1 \) represents an absolutely elastic collision (no energy loss), and \( \varepsilon = 0 \) represents an absolutely plastic collision (velocities of bodies are the same after collision, with maximum possible energy loss).

The new normal velocities are calculated using the formula

\[
\begin{pmatrix}
v^\text{after}_1 \\
v^\text{after}_2
\end{pmatrix} = \begin{pmatrix}
A - \varepsilon & 1 - A + \varepsilon \\
A & 1 - A
\end{pmatrix} \begin{pmatrix}
v^\text{before}_1 \\
v^\text{before}_2
\end{pmatrix},
\]

\[
A = \frac{(1 + \varepsilon)m_1}{m_1 + m_2},
\]

where \( m_1, m_2 \) are the masses of balls. The velocity change in a ball-to-boundary collision is calculated by similar rules.
10.3 Shaker Mill Model as a Multibody Dynamics Model

A multibody dynamics model represents a GTM as a system of bodies (or balls) moving together in a force field [Gavrilov and Vinogradov 97, Gavrilov and Vinogradov 98, Sun et al. 94] (see Figure 10.3.1). This system is subdivided into a set of clusters of bodies. A cluster is a set of objects, which move in contact with each other [Vinogradov 92]. This simulation technique is applied to the shaker mill simulation [Gavrilov and Vinogradov 98].

Figure 10.3.1 The multibody dynamics model

The motion of a cluster is described by a system of ordinary differential equations, possibly coupled with some algebraic constraints [Vinogradov 92]. The equations for the MBD model can be derived by a variety of techniques. One of these is a modification of the Newton-Euler's method [Sun et al. 94]. This approach is based on the classical laws of rigid body mechanics

\[ \sum F = ma, \]

Newton's second law, where \( \sum F \) is the total force acting on the center of mass of a body, \( a \) is the acceleration of the center of mass, and

\[ \sum M = I\dot{\omega}, \]

where \( \sum M \) is the total linear momentum relative to the center of mass, \( I \) is the moment of inertia and \( \omega \) is the angular velocity of the body. A numerical solution can be obtained by solving a system of second order differential equations. An iterative method can be used for this purpose [Wegert 90].
An explicit generation of equations can be a very computationally expensive operation. In a simulated system, the topology of the cluster may frequently change: links inside the cluster can break and some others may appear. A cluster can split into a couple of new clusters, or two clusters may merge into one. Each of these changes in topology requires the re-generation of the ODE system. An approach, where the governing equations are not generated explicitly, but are embedded into a data structure, representing the GTM system in a natural way, is used [Vinogradov 92].

The time-step collision detection algorithm presented in Chapter 8 is applied for the collision optimization of such a system. The equations are solved numerically in order to determine the state of the system on the next time step. The solution of the differential equations describes, in particular, the trajectories of bodies in the time interval \((t,t + \Delta t]\) [Gavrilov and Vinogradov 98]. These trajectories are approximated by straight lines. Then, the velocity update events can be scheduled in each simulation time step. The topology update event can be computed from the collision optimization data structure. During the time interval \((t,t + \Delta t]\) several events can happen. All events are put into the common event queue \(Q\).

The interface break events are scheduled for a cluster when the breaking of an interface between two balls is detected. This is done by analyzing the reaction forces in links. When the reaction force in a link becomes negative (or less than some small negative tolerance constant), an interface break event must be scheduled [Vinogradov 92]. Note that scheduling of the interface break events does not depend on the data structure used for the collision detection optimization and is relies only onto the properties of the physical system being simulated.

10.4 Collision Detection Optimization for the Shaker Mill Simulation Model

Five algorithms for collision detection were implemented and tested on the example of the shaker mill simulation model. These algorithms are the straightforward method, the dynamic power diagram (PD) method, the subdivision method (the data structure used
was regular spatial subdivision), the *spatial tree method* (regular spatial tree) and the *segment tree method* (based on a set of segment trees).

The algorithms were implemented in Object-Oriented Pascal in the Borland Delphi environment. The program runs under Windows 95. The experiments were conducted on a Pentium 166 computer with 32 megabytes of RAM.

The input data for the programs was generated at the Department of Mechanical Engineering, the University of Calgary. The program used for the input data generation is described in [Gavrilov and Vinogradov 98]. The size of the simulation space (shaker area) was 200 by 200. The radii of the balls was selected in the range from 1 to 10 mm. The *length of the simulation run* was set up to 10 seconds with the time-step equal to 0.005 sec.

### 10.4.1 Monosized Data Sets

The first series of experiments were conducted on the data set containing 100 *monosized* balls (that is, balls with the same radii). The density of the distribution (i.e. the ratio between the combined volume of the balls in the simulation space and the volume of the whole simulation space) was gradually increased from 5% to 70% by varying the radii of the balls (see Table 10.4.1).

<table>
<thead>
<tr>
<th>Packing density</th>
<th>5%</th>
<th>20%</th>
<th>33%</th>
<th>50%</th>
<th>70%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ball radius (mm)</td>
<td>2.5</td>
<td>3</td>
<td>6.5</td>
<td>8</td>
<td>9.5</td>
</tr>
<tr>
<td>Collision events</td>
<td>174</td>
<td>387</td>
<td>661</td>
<td>1143</td>
<td>4170</td>
</tr>
<tr>
<td>Predict trajectory events</td>
<td>199855</td>
<td>199459</td>
<td>198941</td>
<td>198009</td>
<td>192371</td>
</tr>
</tbody>
</table>

**Table 10.4.1 Monosized data sets**

Note that such characteristics of the model as the number of collision events (when an actual collision takes place during simulation) and the number of predict trajectory events (when the state of the system is recalculated on each simulation time-step) do not depend on the collision optimization data structure. Also note that the number of collision events increases as the density increases. The number of the predict trajectory events remains constant, since it does not depend on the density and can be approximately computed as
the number of bodies multiplied by the length of the simulation run and divided by the time-step.

For each of the five algorithms the experiments were conducted for the above data sets with varied densities and the elapsed time, total number of topological events (TE) and the total number of collision checks (CC) performed during simulation run were recorded (see Table 10.4.2).

<table>
<thead>
<tr>
<th>Density</th>
<th>5%</th>
<th>20%</th>
<th>33%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>TE</td>
<td>CC</td>
<td>Time</td>
</tr>
<tr>
<td>Straightforward</td>
<td>1114.61</td>
<td>0</td>
<td>20645172</td>
</tr>
<tr>
<td>Regular subdivision</td>
<td>96.50</td>
<td>1718</td>
<td>93137</td>
</tr>
<tr>
<td>Spatial tree</td>
<td>103.04</td>
<td>1718</td>
<td>93137</td>
</tr>
<tr>
<td>Segment tree</td>
<td>106.94</td>
<td>13038</td>
<td>11542</td>
</tr>
<tr>
<td>Dynamic PD</td>
<td>304.84</td>
<td>778</td>
<td>1926240</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Density</th>
<th>50%</th>
<th>70%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>TE</td>
<td>CC</td>
</tr>
<tr>
<td>Straightforward</td>
<td>1181.12</td>
<td>0</td>
</tr>
<tr>
<td>Regular subdivision</td>
<td>137.04</td>
<td>569</td>
</tr>
<tr>
<td>Spatial tree</td>
<td>149.40</td>
<td>569</td>
</tr>
<tr>
<td>Segment tree</td>
<td>129.68</td>
<td>14466</td>
</tr>
<tr>
<td>Dynamic PD</td>
<td>320.92</td>
<td>482</td>
</tr>
</tbody>
</table>

Table 10.4.2 Experimental results for monosized data sets

First, note that the number of topological events is the maximum for the segment tree method, since the topological event for this data structure happens whenever projections of two balls collide on a coordinate axis. The number of topological events significantly increases for the densely packed system for this method. Observe that the number of topological events is the lowest for the PD.

The number of collision checks for the straightforward method is approximately 10 times larger than the number of the collision checks in the PD, about 20 times larger than that of the regular subdivision and spatial tree methods and about 100 times larger than that of the segment tree method. The smallest number of collision checks for the segment tree method is easily explained by the fact that collision checks are only performed when bounding boxes of two balls intersect, which happens rarely. The number of the collision checks is the largest for the PD method since the number of neighbors of a ball in the PD
is 6 on average. Also note that the PD method is the only method where this number does not depend on the density of the distribution, while for other methods the number of neighbors as well as the number of collision checks increases as the density increases.

A graph that shows the dependence of the elapsed time required to simulate the system of 100 balls on the density of packing is shown in Figure 10.4.1. The first conclusion that can be made is that the use of any collision optimization technique improves the performance of the simulation algorithm by approximately 10 times!

![Graph showing time vs. packing density](image)

**Figure 10.4.1** The time vs. the packing density

The PD method performs approximately 3 times slower than all the other collision optimization methods for low density distributions and approximately 2 times slower for high density distributions. Note that the difference in performance can be partly attributed to the fact that every topological event for the PD requires the solution a polynomial equation of the 4th degree. Also note that the number of collision checks is practically constant for the PD, so that the time almost does not change as the density grows.

The other methods perform better than the PD and the time required by the segment tree method is the smallest, with the regular subdivision method in second place and the spatial tree method in third. Note, however, that the performance of all methods gradually decreases as the density grows. Also note that the time required for the spatial tree method is only 5% more than that of the regular subdivision method, while this method requires significantly less memory (see Table 9.7.1).
10.4.2 Polysized Data Sets

The second series of experiments were conducted for polysized data sets of 100 balls (that is, balls with different radii). The radii were distributed so that the radius of largest ball is approximately 10 times larger than the radius of the smallest ball. Once again, the packing density was increased from 5% to 70% with number of collision events and predict trajectory events measured (see Table 10.4.3).

<table>
<thead>
<tr>
<th>Packing density</th>
<th>5%</th>
<th>20%</th>
<th>33%</th>
<th>50%</th>
<th>70%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ball radius (mm)</td>
<td>0.4-4</td>
<td>1-10</td>
<td>1.8-18</td>
<td>3.2-7</td>
<td>7.5-10</td>
</tr>
<tr>
<td>Collision events</td>
<td>189</td>
<td>375</td>
<td>488</td>
<td>695</td>
<td>3127</td>
</tr>
<tr>
<td>Predict trajectory events</td>
<td>199816</td>
<td>199416</td>
<td>199209</td>
<td>198912</td>
<td>220282</td>
</tr>
</tbody>
</table>

Table 10.4.3 Polysized data sets

The number of topological events, collision checks and the elapsed time were measured (see Table 10.4.4). As the packing density increases, the number of collision checks increases significantly for the regular subdivision and spatial tree methods and their performance becomes even worse than that of the straightforward approach.

<table>
<thead>
<tr>
<th>Density</th>
<th>5%</th>
<th>20%</th>
<th>33%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Time</td>
<td>TE</td>
<td>CC</td>
</tr>
<tr>
<td>Straightforward</td>
<td>1332.66</td>
<td>0</td>
<td>20657627</td>
</tr>
<tr>
<td>Regular subdivision</td>
<td>103.26</td>
<td>1161</td>
<td>298667</td>
</tr>
<tr>
<td>Spatial tree</td>
<td>111.99</td>
<td>1161</td>
<td>298667</td>
</tr>
<tr>
<td>Segment tree</td>
<td>109.68</td>
<td>12432</td>
<td>14154</td>
</tr>
<tr>
<td>Dynamic PD</td>
<td>303.85</td>
<td>881</td>
<td>1934517</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Density</th>
<th>50%</th>
<th>70%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Time</td>
<td>TE</td>
</tr>
<tr>
<td>Straightforward</td>
<td>1250.16</td>
<td>0</td>
</tr>
<tr>
<td>Regular subdivision</td>
<td>461.82</td>
<td>171</td>
</tr>
<tr>
<td>Spatial tree</td>
<td>489.00</td>
<td>171</td>
</tr>
<tr>
<td>Segment tree</td>
<td>226.83</td>
<td>21457</td>
</tr>
<tr>
<td>Dynamic PD</td>
<td>337.68</td>
<td>446</td>
</tr>
</tbody>
</table>

Table 10.4.4 Experimental results for polysized data sets

The segment tree and the PD methods are now compared. The number of collision checks for the PD remains the same and is very close to the number of collision checks obtained for the monosized data set (once again, due to the fact that the number of neighbors of each ball is only 6 on average). The number of collision checks for the segment tree
method is approximately 14 times smaller than that of the PD method for low packing densities and about 4 times smaller for high packing densities. The number of topological events in the PD method is twice as small as that of the segment tree method for low packing densities and almost 30 times smaller for high densities.

A graph that shows the dependence of the elapsed time, required to simulate the system, of the density of packing is shown in Figure 10.4.2. Once again, it can be seen that the PD is very predictable method with the time function behaving almost like a constant function. As the density increases to above 45%, the regular subdivision and the spatial tree method require more time than PD and segment tree method, and as the density approaches 70% these all take more time than the straightforward method. The segment tree method outperforms the PD at low densities but reaches the PD’s level of performance as the packing density becomes higher.

![Graph showing time vs. packing density.](image)

**Figure 10.4.2** The time vs. the packing density

### 10.4.3 Increasing the Number of Particles

In the final series of experiments the number of balls was gradually increased from 10 to 100 with their radii selected in the 5 to 10 mm range. The density was gradually increased from 4.5% to 35% (see Table 10.4.5).

<table>
<thead>
<tr>
<th>Number of balls</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Density</td>
<td>4.5%</td>
<td>8.5%</td>
<td>12.5%</td>
<td>15.0%</td>
<td>20.0%</td>
<td>23.0%</td>
<td>26.5%</td>
<td>31.0%</td>
<td>33.5%</td>
<td>35.0%</td>
</tr>
<tr>
<td>Collision events</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>24</td>
<td>31</td>
<td>52</td>
<td>47</td>
<td>79</td>
<td>121</td>
<td>128</td>
</tr>
<tr>
<td>Predict trajectory</td>
<td>4014</td>
<td>8021</td>
<td>12026</td>
<td>16006</td>
<td>20000</td>
<td>23985</td>
<td>27972</td>
<td>31902</td>
<td>35871</td>
<td>39854</td>
</tr>
</tbody>
</table>

**Table 10.4.5** Sample shaker mill data sets
The graph in Figure 10.4.3 shows the dependance of the elapsed time on the number of particles. It can be seen that the time required for the straightforward algorithm grows as a quadratic function while the time required by all the other methods shows just a linear growth. Once again the segment tree outperforms all other methods while the PD requires the most time among collision detection optimization data structures.

![Graph showing time vs. number of particles]

Figure 10.4.3 The time vs. the number of particles

10.5 Choice of the Appropriate Method for the Collision Detection Optimization

Some general conclusions can be drawn based on the experimental results:

1. The use of any collision optimization technique significantly improves the performance of the simulation algorithm.

2. The segment tree outperformed all of the other collision optimization data structures for all types of the distributions considered. This includes monosized and polysized distributions for high and low packing density.

3. The regular spatial subdivision and the regular spatial tree show similar performance with the regular subdivision method being approximately 5% faster but requiring more space. The space required for the PD, spatial tree and segment tree methods is practically the same (it is $O(n)$).
4. The PD method is the slowest among all optimization methods for monosized distributions and for polysized distributions with low packing density. It performs as well as the segment tree method for polysized high density distributions.

5. The PD method shows a very consistent performance that does not depend very much on the distribution of radii of the particles or their packing density.

6. The regular spatial subdivision and the regular spatial tree method perform almost as badly as the straightforward method for polysized high density distributions.

7. In order to implement the regular subdivision method the size of the simulation space has to be known in advance. This is not required for other optimization data structures.

8. In order to implement the regular spatial subdivision and the regular spatial tree methods the size of the largest object in the system should be known in advance. This information is not required for the segment tree and PD methods.

Based on the results obtained, suggestions can be made about the most appropriate data structure and algorithms for some other simulation problems. Consider some examples of the GTM systems that illustrate which of the above data structures can be appropriate for some particular applications. They are billiard simulation, gas simulation, sand pile simulation and ice flow simulation models.

In all of these tasks there is a set of constraints stated in the original problem formulation. They specify the shape of bodies moving in simulation space, boundaries and collision rules. The above problems are often simulated using a set of spheres or balls in a specified domain. The collisions are instantaneous and one by one. The problem properties, such as the number of objects, their size distribution, density of packing and functions defining their trajectories, vary from problem to problem. The selection of the most appropriate method for collision detection optimization should be based on the properties of the simulated system.
10.5.1 Billiard Simulation

During *billiard* simulation 16 balls of uniform size are moving along straight-line trajectories with constant velocities in a rectangular domain [Lubachevsky 91]. The correspondence between the ball size and the domain size defines a low density of packing. The balls have uniform radii. Any of the methods, including regular subdivision, spatial tree or segment tree, can be used for the optimization of the collision detection.

10.5.2 Gas Simulation

A gas model contains a large number of molecules of equal size. High packing density and large simulation space can characterize the model [Lubachevsky 91]. The molecules can be considered to move along straight-lines trajectories as in the previous model. The segment tree method can be effectively applied for the model simulation.

10.5.3 Ice Flow Simulation

An *ice flow system* can be characterized by a large number of objects of variable size that can be approximated by balls with high packing density, moving along piecewise linear trajectories [Vinogradov 92, Sun et. al. 94]. The dynamic generalized VD or a segment tree can be used for the collision detection optimization.

10.5.4 Sand Pile Simulation

In a *sand pile simulation* the model has large number of particles of approximately equal size [Watson 96]. The movement of each particle can be described by piecewise linear trajectories. The model has a high packing density and the simulation space is small. A simulation approach in which the sand pile is considered as one large cluster can be used. The segment tree method is once again the most suitable collision optimization method.
CHAPTER 11: CONCLUSIONS

This thesis investigated data structures that represent the topology of a system of geometrical objects (circles and spheres) and developed some algorithms for their construction and maintenance.

The focus of the studies was on the Voronoi diagram and Delaunay tessellation for weighted set of sites under the Euclidean, power, Manhattan and supremum metrics in 2 and $d$-dimensional spaces. The thesis attempted to generalize the properties and algorithms for these data structures and to consider each particular Voronoi diagram as an instance of the same class. A set of general properties was developed and a generic incremental algorithm for data structure construction was obtained for generalized Voronoi diagrams and Delaunay tessellations in $d$ dimensions. The method was implemented using exact arithmetic for the power metric in the plane. The sweep-plane method was extended to the Manhattan and supremum metrics and implemented. A new method to construct the power diagram from the Voronoi diagram based on a bisector transformation technique was developed and tested. The space of all triangulations in $d$ dimensions was considered and theorems about the distance between any two elements of this space were proven in 2-dimensional space.

The application of the developed theory to the collision detection problem that arises during computer simulation of granular type materials was considered in the second part of the thesis. Other data structures besides Voronoi diagrams and Delaunay triangulations, such as regular spatial subdivision, regular spatial tree and set of segment trees were introduced and investigated. A generalized algorithm for the collision detection optimization in a system of moving objects in 2 and $d$ dimensions was introduced. Algorithms for the construction and maintenance of data structures along with complexity estimates were described. The developed methods were implemented and tested on an applied problem of a shaker mill simulation. Finally, recommendations about the choice of the appropriate data structure for collision optimization were presented.
BIBLIOGRAPHY


for $L_1$ and $L_\infty$ Delaunay triangulations," *Algorithmica*, vol. 6, pp. 207-221.

243-245.


Sleator, D., Tarjan, R. and Thurston, W. 1986. "Rotation distance, triangulations, and


Stewart, W. 1990. "An algorithm to find the facets of the convex hull in higher


Vasilyev,N. and Gutenmaher, V. 1980. Straight Lines and Curves. Mir Publishers Moscow, Russia


