# New Results on Geometric Spanners and Their Applications

Dissertation

von

**Tamás Lukovszki**

Schriftliche Arbeit zur Erlangung des Grades
eines Doktors der Naturwissenschaften

Fachbereich Mathematik / Informatik und Heinz Nixdorf Institut
Universität-Gesamthochschule Paderborn

Paderborn, 1999

# Acknowledgments

I would like to thank my advisor Prof. Dr. Friedhelm Meyer auf der Heide for his support over the last three years. He gave me the freedom to choose the direction of my research and to my evolvement. Furthermore, I would like to thank the other reviewers of my thesis, Prof. Dr. Endre Szemerédi, who has taught me a lot about how to do research, and Prof. Dr. Susanne Albers. Very special thaks go to my wife Marlies Tenten for reading this thesis and for her psychological support. I am grateful to Matthias Fischer for our joint research and for many personal conversations. Also, I would like to thank Martin Ziegler for our joint research. Furhermore, I thank Silvia Götz, Kay Salzwedel, Christian Sohler, and Stefanie Thies for reading parts of this work and for their helpful comments. Last but not least, I thank all of my colleagues for the inimitable working atmosphere.

This work was supported by the Graduate College of the Heinz Nixdorf Institute "Parallele Rechnernetzwerke in der Produktionstechnik".

# Contents

# Chapter 1

# Introduction

The main goal of this work is to solve a practical problem and develop exact theory which is related but not restricted to the solution of the practical problem. Reaching both of these goals is a great challenge. The problems appearing in practice often have too many parameters to compute with, and some nice theoretical model and its elegant solution often leads to insufficient results in the practice.

Our practical problem is from the area of virtual reality. The aim is to provide methods for a walkthrough system that allow to manage arbitrarily large geometric scenes and that guarantee the navigation and small updates in such scenes in real time. Such systems are inquired in architecture, advertising, tourism, planning of cities, planning of traffic, geographic information systems, medicine, education, etc... In order to achieve our goal we must take a deeper look into the structure of the problems.

It is not surprising that geometry plays an important role in the solution of such problems. The basis for our approach are geometric spanners that are geometric graphs with the property that the length of the shortest path between each pair of vertices is not much larger than the Euclidean distance of the pair. This property can be exploited making the solution of special problems in walkthrough systems possible. Geometric spanners have many other applications in various areas of computer science, for example, in motion planing, VLSI design, they are used for approximating the minimum spanning tree, and for a fully polynomial time approximation scheme for the traveling salesman and related problems. We present older results and we prove new results about geometric spanners that are not restricted to walkthrough systems.

An important generalization of spanners are fault tolerant spanners. Roughly spoken, they are geometric networks that maintain the spanner property even if a certain number of vertices or edges are failing. To calculate with such faults has an extreme

importance if one consider, for example, a network of aircraft corridors, where corridors or airports can fall out because of bad weather or other reasons. In this work we also investigate fault tolerant geometric spanners and we improve many previous results significantly.

## 1.1 Walkthrough systems

The goal of walkthrough systems is a simulation and visualization of a three-dimensional scene, e.g., a city. A scene consists of objects that are usually modeled by triangle meshes. In our example the objects are houses, trees, streetlights, and cars. The visitor of such a scene walks around and her environment is visualized on the screen or on a special output device. For a smooth animation we have hard *real time* requirements. Empirically, the computer has to render at least twenty pictures (frames) per second. If the animation is computed with less than twenty frames per second, navigation in the scenes is hard or impossible.

As mentioned above, in todays computer systems the objects of a scene are modeled by triangle meshes. The triangles of such a mesh contain geometric and object specific information, e.g., three-dimensional coordinates, color, textures, transparency information, etc... For every position of the visitor the computer has to compute a view of the scene. Hidden triangles have to be eliminated (hidden surface removal) and for all visible triangles color and brightness has to be computed. This process is called rendering.

The rendering process is often supported by special hardware, but the time for the rendering of a picture depends on the complexity of the scene, i.e., the number of triangles and the number of pixels that are needed for drawing a triangle. Therefore, if the scene is too large the real time requirement can only be guaranteed if the system does not spend computation time for objects whose influence is not significant for the quality of the picture appearing on the screen. To control this situation real time and approximation algorithms are necessary to reduce the complexity of those parts of the scene that are far away, and thus, have a low influence to the quality of the rendered image.

We begin with a simple model. The scenes that we consider are arbitrarily large and they consist of non overlapping objects placed on the plane. Furthermore, we assume that the objects have roughly the same size. Thus, in our model we replace each object by a unit size ball. We fix an angle $\alpha$ and we say that an object is *important* if it appears from the visitor's position in an angle at least $\alpha$. These objects may appear on the

display greater than a pixel. Representing the objects by points in the plane, we obtain a kind of *circular range searching problem*, because all important objects are within a circle around the visitor's position. The difference between the classical circular range searching problem and our searching problem is that we can exploit the spatial locality of the queries, namely, that the visitor moves continuously and so consecutive query positions are near to one another.

We develop data structures that support the selection of important objects in time nearly linear in the number of selected objects, in particular, independent of the size of the scene. The size of our data structure is linear in the number of objects of the scene. Since we deal with large scenes, for practical purposes the low space complexity is essential.

We also consider *dynamic* scenes, where the visitor is allowed to insert or delete an object at her current position. We present a randomized data structure of Fischer et al. [39] for the *fully dynamic* problem, where both, insertion and deletion are allowed. This data structure supports the updates as fast as the selection of the important objects. Thereafter, we present our own deterministic contribution for the *incremental* problem, where only insertion is allowed. For a discussion about dynamic algorithms we refer to [65] and [37].

Our deterministic solutions have been published in [40].

After this we consider a more general model, where the objects of the scene have different sizes. For the selection of important objects we first present an approximate solution which exploits the fact that the selection problem is a so-called *decomposable searching problem* [15, 65, 35]. Then we show another method which uses geometric transformations to obtain a higher dimensional halfspace range searching problem and solve this problem with known algorithms.

## 1.2 Geometric spanners and weak spanners

Geometric spanners have been studied intensively in the recent years. Let $S$ be a set of $n$ points (also called sites) in $\mathbb{R}^d$, where $d$ is an integer constant. Let $G = (S, E)$ be a graph whose edges are straight line segments between the points of $S$. For two points $p, q \in \mathbb{R}^d$, let $dist_2(p, q)$ be the Euclidean distance between $p$ and $q$. The length $length(e)$ of an edge $e = (u, v) \in E$ is defined as $dist_2(u, v)$. For a path $P$ in $G$ the length $length(P)$ is defined as the sum of the length of the edges of $P$. A path between two points $s, t \in S$ is called an $st$-path. Let $f > 1$ be a real number. The graph $G$ is an *$f$-spanner* for $S$ if for each pair of points $s, t \in S$ there is a path from $s$ to $t$ in $G$

such that the length of the path is at most $f$ times the Euclidean distance $dist_2(s,t)$ between $s$ and $t$. We call such a path an *$f$-spanner path* and $f$ is called the *stretch factor* of the spanner[1]. If $G$ is a directed graph and $G$ contains a directed $f$-spanner path between each pair of points then $G$ is called a directed $f$-spanner. In order to distinguish the edges of a directed from an undirected graph we use $\langle a,b\rangle$ to denote an edge between the vertices $a$ and $b$ in a directed and $(a,b)$ in an undirected graph.

Geometric spanners were introduced in computational geometry by Chew [27]. They have applications in motion planing [28], they were used for approximating the minimum spanning tree [82], and for a fully polynomial time approximation scheme for the traveling salesman and related problems [68, 5]. We present a further application of spanners in walkthrough systems to solve special range searching problems.

The problem of constructing an $f$-spanner for a real constant $f > 1$, that has $O(n)$ edges, has been investigated by many researchers [49, 50, 71, 80, 73, 22, 8, 23, 21, 24, 6, 31]. Keil [49] gave a solution for this problem introducing the $\theta$-graph[2], which was generalized by Ruppert and Seidel [71] and Arya et al. [8] to any fixed dimension $d$. These authors gave also an $O(n\log^{d-1} n)$ time algorithm to construct the $\theta$-graph. Chen et al. [26] proved that the problem of constructing any $f$-spanner for $f > 1$ takes $\Omega(n\log n)$ time in the algebraic computation tree model [13]. Callahan and Kosaraju [22, 23], Salowe [73], and Vaidya [80] gave optimal $O(n\log n)$ time algorithm for constructing $f$-spanners. Several interesting quantities related to spanners were studied by Arya et al. [6]. They gave constructions for bounded degree spanners, spanners with low weight, spanners with low diameter, and for spanners with combinations of these properties. The weight $w(G)$ of a graph $G$ is the sum of the length of its edges.

In walkthrough systems we need a weaker property of a graph. We introduce the notion of weak spanner which express exactly the desired property. A graph $G = (S,E)$ is called a *(directed) $f$-weak spanner* for $S$ if for each pair of points $s,t \in S$ there is a (directed) path from $s$ to $t$ in $G$ such that for each vertex $v$ on this path $dist_2(s,v) \leq f \cdot dist_2(s,t)$ holds.

In this work we focus on an important family of geometric (weak) spanners, on the $\theta$-graphs. Their simple, non hierarchical structure makes them particularly suitable for our purposes in walkthrough systems. Furthermore, their structure builds a good basis for generalizations in order to obtain fault tolerant spanners.

In Chapter 2 we show various old results and prove new results on the $\theta$-graph about the (weak) spanner property. We prove that in the two-dimensional case the $\theta$-graph with outdegree four has the weak spanner property. Then we take a look to the reversed

[1] In the literature the stretch factor is often denoted by $t$.

[2] Yao [82] and Clarkson [28] used a similar construction to solve other problems.

$\theta$-graph and we show that its weak stretch factor is less than in the original graph but we loose the bounded outdegree. After this we generalize the degree four $\theta$-graph using $L_p$-distances to choose the neighbors of the points in the cones. We classify these graphs according to whether they have the weak spanner property or not. Thereafter, we show that four is the lower bound for the outdegree of the $\theta$-graph when we require the weak spanner property. This is the case even if we allow arbitrary convex distance functions to choose the neighbors of a point in the cones.

We also consider higher dimensional $\theta$-graphs. We analyse different methods to compute a $\theta$-frame that have not yet been exactly analyzed, to our knowledge, and we also present an own method for the computation. We show that using our method a $\theta$-frame can be constructed which contains $O(d^{-1/2}(\frac{d^{3/2}}{\theta})^{d-1})$ simplicial cones, where $d \geq 3$ denotes the dimension. This bound is significantly better than the bounds resulting from previous methods.

## 1.3 Fault tolerant spanners

Fault tolerant spanners for a set $S$ of $n$ points in $\mathbb{R}^d$ were introduced by Levcopoulos et al. [54]. Their motivation was the construction of geometric networks that are resilient to edge or vertex faults. Fault tolerant spanners have the property that after removing at most $k$ edges or vertices, $1 \leq k \leq n-2$, the remaining graph still contains a short path between each pair of points. More precisely, after the deletions the shortest path between each pair of points in the remaining graph is at most $f$ times longer than the shortest path in the graph which is obtained from the complete graph after deletion of the same edge/vertex set, where $f > 1$, the stretch factor, is a given real constant. In the case of vertex fault tolerance this definition is equivalent to the requirement that after the deletions the graph remains an $f$-spanner.

In [54] a $k$-vertex and a $k$-edge fault tolerant spanner were constructed, both with $O(k^2 n)$ edges. Furthermore, the authors presented an algorithm which constructs a $k$-edge and $k$-vertex fault tolerant spanner of degree $O(c^k)$ whose weight is bounded by $O(c^k)$ times the weight of the minimum spanning tree of $S$, where $c$ is a constant.

In Chapter 3 we improve most of these bounds significantly. We introduce the notion of higher order $\theta$-graph and we use such a graph to obtain a $k$-edge and $k$-vertex fault tolerant spanner with only $O(kn)$ edges. This bound on the number of edges is asymptotically tight, since each $k$-edge/vertex fault tolerant spanner must be $k$-edge/vertex connected, and hence, contain at least $(k+1)n/2$ edges. Thereafter, we construct an $O(k^2)$ degree $k$-edge and $k$-vertex fault tolerant spanner.

We also study fault tolerant spanners, where the original set of points can be extended by so-called Steiner points, in order to satisfy a stronger but more natural definition of edge fault tolerance, where it is required that after deletion of at most $k$ edges the graph must contain an $f$-spanner path between each pair of original points. We prove $\Theta(kn)$ bounds on the number of edges and on the number of Steiner points in such Steinerized fault tolerant spanners. To our knowledge, Steinerized fault tolerant spanners have not been investigated before.

Our contributions about fault tolerant spanners have been published in [57].

## 1.4 Computational model

The complexity of the problems we study is strongly dependent on the model of computation.

Most algorithms and data structures in computational geometry are described in the *random access machine (RAM)* [4] or the *real RAM* model [66]. In the traditional RAM memory cells can contain $(\log n)$-bit integers that may be compared, added, subtracted, multiplied and divided (with rest). Furthermore, the integers can be also used as pointers to other memory cells (indirect addressing). All these operations take constant time. A variant of the RAM model [81, 45, 44, 51] allows also bitwise logical operations on the integers in constant time.

In the real RAM model memory cells also can store real numbers. Since a real number can contain an infinite amount of information in its binary expansion, the set of valid operations for real numbers must be carefully restricted. It is standard to restrict the set of allowed operations to comparison $<, \leq, =, \geq, >$ and arithmetic operations $+, -, \times, /$, and optional $\sqrt{}$. Conversion between integer and real numbers is not allowed. For some algorithms it is advantageous to extend the set of operations by the non algebraic `floor` operation on real numbers or by certain bit operations in constant time [18]. Many such operations are implemented quite efficiently on existing architectures.

Nevertheless, before using such an augmented model, one should ask whether it is absolutely necessary. Algorithms that depend on additional operations are not really comparable to other algorithms that use only algebraic operations. Furthermore, if we restrict our algorithms to the algebraic model, we often can prove nontrivial lower bounds using the *algebraic decision tree* [13, 77, 34, 69, 67]. Formally, an algebraic decision tree (see [13]) is a binary tree $T$ with a function that assigns

- to any leaf vertex $v$ an output $YES$ or $NO$;
- to any vertex $v$ with exactly one child an operational instruction of the form $f_v := f_{v_1} \circ f_{v_2}$ or $f_v := \sqrt{f_{v_1}}$, where $v_i$ is an ancestor of $v$ in $T$, or $f_{v_i}$ is an input variable or a real constant, and $\circ \in \{+, -, \times, /\}$;
- to any vertex $v$ with two children a test instruction of the form $f_v > 0$ or $f_v = 0$ or $f_v \geq 0$.

For any fixed value $n$ of the input size, one can expand an instance of an algorithm in the algebraic model into an algebraic decision tree. The computation starts at the root and proceeds down to a leaf. The running time of the algorithm is the number of edges on the tree path from the root to a leaf which is in the worst-case the depth of the tree. A decision tree deals with inputs of a particular size. Therefore, an algorithm is represented by a family of decision trees, one for each possible input size.

Using the algebraic decision tree, Ben-Or [13] proved an $\Omega(n \log n)$ lower bound for the time of each algorithm in the algebraic model that solves the *element distinctness problem* in which for given a set of $n$ real numbers it must be decided whether any two are equal. Because this problem can be reduced to many problems in the computational geometry in linear time, this result is very useful for proving further superlinear lower bounds on a variety of geometric problems. For further discussion about this lower bound technique we refer to [66] and [38].

We also apply the above technique and in Chapter 5 we prove an $\Omega(n \log n)$ lower bound on the construction time of weak spanners with Steiner points in the algebraic decision tree model.

Except for Chapter 4 the results presented in this work are obtained by methods that use only operations allowed in the algebraic decision tree model.

## 1.5 Outline

In Chapter 2 we define the $\theta$-graph. We present older results about the spanner property and describe algorithms to compute it. We prove that in the two-dimensional case the $\theta$-graph with outdegree four has the weak spanner property. Then we study the reversed $\theta$-graph. After this we generalize the degree four $\theta$-graph using $L_p$-distances to choose the neighbors of the points in the cones. We classify these graphs according to whether they have the weak spanner property or not. Thereafter, we show that four is the lower bound for the outdegree of the two-dimensional $\theta$-graph when we require the weak

spanner property even if we allow arbitrary convex distance functions to choose the neighbors of a point in the cones.

In Chapter 3 we investigate fault tolerant geometric spanners. We construct a $k$-edge and $k$-vertex fault tolerant spanner with only $O(kn)$ edges. Then we construct an $O(k^2)$ degree $k$-edge and $k$-vertex fault tolerant spanner. Thereafter, we investigate Steinerized fault tolerant spanners, that satisfy a stronger but more natural definition of edge fault tolerance. We prove $\Theta(kn)$ bounds on the number of edges and on the number of Steiner points in such spanners.

In Chapter 4 we present a new application of the $\theta$-graph in walkthrough systems. We describe a randomized data structure of [39] which supports selecting the important objects in $O(k)$ time, where $k$ is the number of vertices of a certain graph in a disc whose radius is $f$ times the radius of the concentric disc containing exactly the important objects, and $f > 1$ is a constant which can be made arbitrary close to 1. This data structure also supports insertion and deletion of an object in the same time as the selection. Then we describe our own deterministic data structure which also supports the selection of important objects and insertion of a new objects at the position of the visitor in $O(k)$ time. Finally, we present two methods for the selection of important objects in a scene which consists of balls with different sizes.

In Chapter 5 we prove an $\Omega(n \log n)$ lower bound on the construction time of weak spanners with Steiner points in the algebraic decision tree model.

# Chapter 2

# The $\theta$-graph

In this chapter we define $\theta$-graph [82, 28, 49, 50, 71, 8] and we investigate some important properties of it. The $\theta$-graph has a rich history in the computational geometry. Yao [82] used this construction to compute the minimum spanning tree connecting $n$ points in the $d$-dimensional space under the $L_1$, $L_2$ and $L_\infty$ metric[1]. Using this graph Clarkson [28] solved the following motion planing problem in two and three-dimensions. Given a set of polyhedral obstacles and points $s$ and $t$. Find a short path from $s$ to $t$ that avoids the obstacles. The notion of $\theta$-graph were introduced by Keil [49]. Keil [49] and Keil and Gutwin [50] studied graphs approximating the complete Euclidean graph in the two dimensional space and they proved a trade off between the number of cones and the stretch factor of the $\theta$-graph. Ruppert and Seidel [71] improved this trade off using some stronger definition of the $\theta$-graph and they generalized it for higher dimensions. Arya et al. [8] studied the problem of maintaining the spanner property of a graph under a sequence of random insertion and deletions [64] of points of the original point set. They obtained a polylogarithmic update time using a $\theta$-graph based data structure.

## 2.1 The two-dimensional $\theta$-graph

In this section we present the definition and the proof of the spanner property of the $\theta$-graph done by Ruppert and Seidel [71]. Then we study the weak spanner property. We show that the $\theta$-graph of degree four has this property and that the $\theta$-graph of degree three has not. Furthermore, we examine $\theta$-graphs of degree four, where the nearest neighbors are chosen using an $L_p$ distance.

[1]For any real $1 \leq p \leq \infty$ the $L_p$ distance of two points in $\mathbb{R}^d$ is defined as $dist_p(s,t) := (\sum_{1 \leq i \leq d} |s_i - t_i|^p)^{1/p}$, where $s_i$ and $t_i$ denote the $i$th coordinate of $s$ and $t$, respectively

Let $S$ be a set of $n$ points in $\mathbb{R}^2$, $k \geq 2$ an integer and $\theta = \frac{2\pi}{k}$ an angle. Rotate the non negative $x$-axis around the origin over angles $i\theta$ for $0 \leq i < k$. This gives $k$ halflines $h_0, ..., h_{k-1}$. Let $c_0, ..., c_{k-1}$ be the cones that are bounded by two successive halflines and let $C := \{c_0, ..., c_{k-1}\}$. Furthermore, for $0 \leq i < k$, let $l_i$ be the halfline obtained by rotation of the $x$-axis over the angle $\frac{2i+1}{2}\theta$ which bisects the cone $c_i$. We call the halfline $l_i$ the *cone axis* of $c_i$. The $\theta$-graph $G_\theta(S)$ of $S$ is defined in the following way. The vertices of $G_\theta(S)$ are the points of $S$. For each point $s \in S$ translate the cones $c_0, ..., c_{k-1}$ and the halflines $l_0, ..., l_{k-1}$ such that the apexes of the cones and endpoints of the halflines become coincident with $s$. Let $c_0(s), ..., c_{k-1}(s)$ and $l_0(s), ..., l_{k-1}(s)$ denote these translated cones and halflines, respectively. For $s, t \in S$ and $0 \leq i < k$ define the distance $dist_{c_i}(s,t)$ as the Euclidean distance between $s$ and the projection of $t$ onto the translated halfline $l_i(s)$ if $t$ is contained in $c_i(s)$, and infinity otherwise. For each $s \in S$ and $0 \leq i < k$, if the cone $c_i(s)$ contains a point of $S$ then add a directed edge from $s$ to one of the points in $c_i(s)$ which is closest to $s$ w.r.t. the distance function $dist_{c_i}$. Figure 2.1 shows an example for the definition of the cones and cone axes.

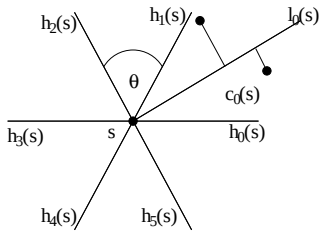


Figure 2.1: The halflines $h_i(s)$ and $l_i(s)$ and the cones $c_i(s)$, $i = 0, ..., k-1$, around a point $s \in S$ in the definition of the $\theta$-graph for $k = 6$.

**Remark:** It may appear circumstantial to define the $\theta$-graph by the angle $\theta$ instead of the number of cones $k$, but the reason will be clear when we generalize it in higher dimensions.

### 2.1.1 The spanner property

Keil [49] proved that the $\theta$-graph for a given set of points is a spanner if $0 < \theta < \frac{\pi}{4}$ and he established an upper bound on the stretch factor of the obtained graph in dependence on the value of $\theta$. Ruppert and Seidel [71] proved the spanner property of the $\theta$-graph for $0 < \theta < \frac{\pi}{3}$ and they improved the bound on the stretch factor. In this subsection we present this result. We modified the original proof in [71] to obtain a shorter one.

**Theorem 2.1** [71] *Let $S \subset \mathbb{R}^2$ be a set of $n$ points. Let $k > 6$ be an integer constant and $\theta = \frac{2\pi}{k}$. Then the $\theta$-graph $G_\theta(S)$ is a spanner for $S$ with stretch factor $\frac{1}{1-2\sin(\theta/2)}$.*

The proof of the stretch factor of $G_\theta(S)$ in Theorem 2.1 is based on the following lemma for which we present an own proof which is shorter than the original one in [71].

**Lemma 2.2** [71] *Let $k \geq 2$, $\theta = \frac{2\pi}{k}$ and $C$ be the set of the corresponding cones. Let $p \in \mathbb{R}^2$ be a point and $c \in C$ be a cone. Furthermore, let $q$ and $r$ be two points in $c(p)$ such that $dist_c(p,q) \leq dist_c(p,r)$. Then $dist_2(q,r) \leq dist_2(p,r) - (1 - 2\sin\frac{\theta}{2})\, dist_2(p,q)$.*

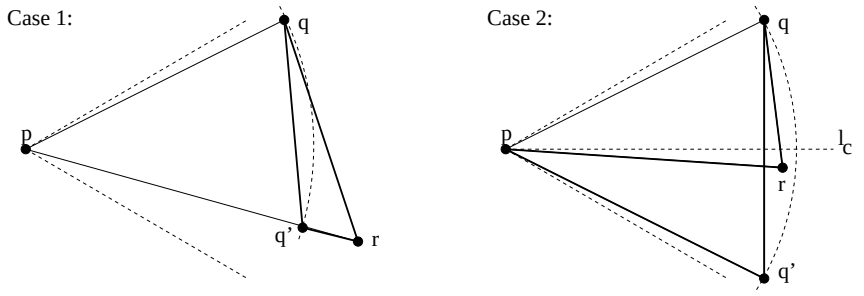


Figure 2.2: The two cases in the proof of Lemma 2.2. Case 1: $dist_2(p,q) \leq dist_2(p,r)$. Case 2: $dist_2(p,q) > dist_2(p,r)$.

**Proof:** (Lemma 2.2) We distinguish two cases (Figure 2.2)

Case 1: $dist_2(p,q) \leq dist_2(p,r)$. Let $q'$ be the point on the line segment $pr$ such that $dist_2(p,q) = dist_2(p,q')$. Applying the triangle inequality for the triangle $qq'r$ we get that $dist_2(q,r) \leq dist_2(q,q') + dist_2(q',r)$. Since $dist_2(q,q') \leq 2\sin\frac{\theta}{2} \cdot dist_2(p,q)$ and $dist_2(q',r) = dist_2(p,r) - dist_2(p,q') = dist_2(p,r) - dist_2(p,q)$, the claim of the lemma follows.

Case 2: $dist_2(p,q) > dist_2(p,r)$. Let $q'$ be the mirror image of $q$ across the cone axis $l_c$. Then $dist_2(p,q') + dist_2(q,r) \leq dist_2(p,r) + dist_2(q,q')$. This follows by application of the triangle inequality two times. Since $dist_2(p,q') = dist_2(p,q)$ and $dist_2(q,q') \leq 2\sin(\theta/2)\, dist_2(p,q)$, we obtain the claim of the lemma. □

**Proof:** (Theorem 2.1) We show that for each two points $s,t \in S$ there is a (directed) $st$-path $P$ in $G_\theta(S)$ such that the length of $P$ is at most $\frac{1}{1-2\sin(\theta/2)}\, dist_2(s,t)$. Consider the path constructed in the following way. Let $s_0 := s$, $i := 0$ and let $P$ contain the single point $s_0$. If the edge $\langle s_i, t\rangle$ is present in the graph $G_\theta(S)$ then add the vertex $t$ to $P$ and stop. Otherwise, let $c(s_i)$ be the cone which contains $t$. Take the point $s_{i+1}$ with $s_{i+1} \in c(s_i)$ and $\langle s_i, s_{i+1}\rangle \in G_\theta(S)$ as the next vertex of the path $P$ and repeat the procedure with $s_{i+1}$.

Consider the $i$th iteration of the above algorithm. If $\langle s_i, t\rangle \in G_\theta(S)$ then the algorithm terminates. Otherwise, if $\langle s_i, t\rangle \notin G_{\theta,k+1}(S)$ then by definition the cone $c(s_i)$ contains at least one point which is not further from $s_i$ than $t$ w.r.t. the distance $dist_c$ and

therefore, $s_i$ has a neighbor $s_{i+1}$ in the graph $G_\theta(S)$ in $c(s_i)$, which is taken as the next vertex of the path $P$. Hence, the algorithm is well defined in each step. Furthermore, Lemma 2.2 implies that $dist_2(s_{i+1}, t) < dist_2(s_i, t)$ and hence, each point is contained in $P$ at most once. Therefore, the algorithm terminates and finds a $st$-path $P$ in $G_\theta(S)$. The bound on the length of $P$ follows by applying Lemma 2.2 iteratively: Let $s_0, ..., s_m$ be the vertices on $P$, $s_0 = s$ and $s_m = t$. Then

$$\sum_{0 \le i < m} dist_2(s_{i+1}, t) \le \sum_{0 \le i < m} \Big( dist_2(s_i, t) - (1 - 2\sin(\theta/2))\ dist_2(s_i, s_{i+1}) \Big).$$

Rearranging the sum we get

$$\begin{aligned} \sum_{0 \le i < m} dist_2(s_i, s_{i+1}) &\le \tfrac{1}{1-2\sin(\theta/2)} \sum_{0 \le i < m} \Big( dist_2(s_i, t) - dist_2(s_{i+1}, t) \Big) \\ &= \tfrac{1}{1-2\sin(\theta/2)}\ dist(s_0, t). \end{aligned}$$

Hence, the graph $G_\theta(S)$ is a spanner for $S$ with stretch factor $\frac{1}{1-2\sin(\theta/2)}$. Clearly, it contains $O(|C|n)$ edges, where $|C| = k = \frac{2\pi}{\theta}$ the number of cones in $C$. □

| $\theta$: | $\frac{2\pi}{7}$ | $\frac{2\pi}{8}$ | $\frac{2\pi}{9}$ | $\frac{2\pi}{10}$ | $\frac{2\pi}{15}$ | $\frac{2\pi}{20}$ |
|---|---|---|---|---|---|---|
| $f$: | 7.56 | 4.26 | 3.17 | 2.62 | 1.71 | 1.46 |

Table 2.1: The stretch factor $f$ for some values of $\theta$.

**Remark:** Table 2.1 shows the bounds on the stretch factor $f$ for some values of $\theta$. Ruppert and Seidel [71] pointed out that in the planar case if $k$ is odd then some improvement can be made on the stretch factor by growing the paths from both ends and taking advantage of the asymmetry of the cones at the end points. They claimed that exploiting this asymmetry a bound near 10 on the stretch factor can be proven even in the case that $k = 5$.

### 2.1.2 Computation

Now we investigate the problem of computing the $\theta$-graph efficiently. In this subsection we describe two algorithms done by Ruppert and Seidel [71] and by Arya et al. [8]. Later in this work we will have similar constructions for which we use modifications of these methods.

**Theorem 2.3** [71, 8] *Let $S \subset \mathbb{R}^2$ be a set of $n$ points. Let $k > 6$ be an integer constant and $\theta = \frac{2\pi}{k}$. Then the $\theta$-graph $G_\theta(S)$ can be constructed in $O(n \log n)$ time using $O(n)$ space.*

**Proof:** We begin with the description of the algorithm of Ruppert and Seidel.

**Algorithm 1.** [71]: The algorithm consists of $k$ phases, one for each cone. In the $i$th phase, $i = 0, .., k-1$, for each point $s \in S$ we compute its neighbor in the $i$th translated cone $c_i(s)$ by performing a plane sweep in the direction $\frac{(2i+1)\pi}{k}$. This is the direction of the $i$th cone axis $l_{c_i}$. We describe the $i$th phase. Let $c_i'$ denote the cone $c_i$ reflected to the origin. We use the fact that a point $q$ is contained in the cone $c_i(p)$ if and only if $p$ is contained in the cone $c_i'(q)$.

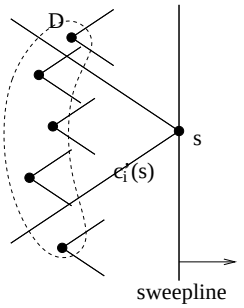


Figure 2.3: Finding the points of $D$ in $c_i'(s)$ when the sweepline reaches the point $s$.

First we initialize a data structure $D = \emptyset$ which allows efficient *orthogonal range queries*. We maintain the invariant that $D$ contains each point of $S$ behind the sweepline whose neighbor has not yet been computed. When the sweepline reaches a point $s \in S$ we perform an orthogonal range query using the data structure $D$ to determine the points in $c_i'(s)$ (Figure 2.3). We join each of these points with $s$ by an edge, then delete them from $D$ and we insert $s$ into $D$. In this way we obviously maintain the invariant. To the orthogonal range queries we must use in a preprocessing an affine transformation on the coordinates such that the transformed boundaries of $c_i$ and $c_i'$ become orthogonal. Then the cone $c_i'(s)$ can be considered as a rectangle with two unbounded sides. Therefore, we can implement $D$ by a *priority search tree*, which was developed by McCreight [60]. A priority search tree needs $O(n)$ space, insertion and deletion of a point can be performed in $O(\log n)$ time, and a query in $O(\log n + k)$ time, where $k$ is the number of reported points. Since each point is inserted and deleted from $D$ exactly once, the total update time of $D$ is $O(n \log n)$. Furthermore, for each point we perform an orthogonal range query. Since each reported point is deleted from $D$, the total query time is $O(n \log n)$, as well, which implies the claim of the theorem.

As we will see, the above algorithm is developed in such a way that it can easily be generalized in higher dimensions substituting the priority search tree by an appropriate higher dimensional orthogonal range searching data structure. We remark that in the two-dimensional case instead of a priority search tree we can use a simple balanced binary search tree for $D$ in which the points are sorted by the signed Euclidean distance

$dist_2^*(s, L_i)$ from a fixed line $L_i$ which is parallel to $l_{c_i}$. Furthermore, for each point in $D$ we maintain a pointer to the next and to the previous point w.r.t. this distance. Then we can report the points of $D$ that are contained in a cone $c_i'(s)$ such that we determine the point $s' \in D$ for which $|dist_2^*(s, L_i) - dist_2^*(s', L_i)|$ is minimal, then using the next-pointers we walk until we find a point which is not contained in $c_i'(s)$. Finally, using the previous-pointers we walk from $s'$ until we find a point which is not contained in $c_i'(s)$. It is easy to verify that in this way we report each point of $D$ contained in $c_i'(s)$.

**Algorithm 2.** [8]: Similar to the to Algorithm 1 we perform $k$ plane sweeps, one for each cone. We describe the $i$th phase. Let $H_i$ and $H_{i+1}$ be the two lines through the origin that are orthogonal to the halflines $h_i$ and $h_{i+1}$, respectively, that are the bounding halflines of cone $c_i$. Furthermore, let $h_i'$ and $h_{i+1}'$ be the lines that contain the halflines $h_i$ and $h_{i+1}$, respectively. We assume that the coordinates are transformed so that $l_{c_i}$ is coincident with the non negative $x$-axis and that $h_i$ and $h_{i+1}$ are the lower and upper bounding halflines of $c_i$, respectively. We assign to the lines $H_i$, $H_{i+1}$ and the $x$-axis directions, as shown in Figure 2.4. Now, for each point $s \in S$ we have to find a point with minimum $x$-coordinate among the points of $S \setminus \{s\}$ that are not below $h_i'(s)$, not above $h_{i+1}'(s)$ and right to $s$. For simplicity of the description we assume that the projections of the points of $S$ to $H_i$ as well as to $H_{i+1}$ are pairwise distinct. Then, for each point $s \in S$ we have to find a point with minimum $x$-coordinate among the points of $S \setminus \{s\}$ that are above $h_i'(s)$ and below $h_{i+1}'(s)$.

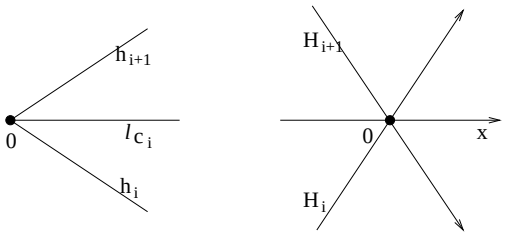


Figure 2.4: The directions assigned to $H_i$, $H_{i+1}$ and the $x$-axis.

We begin with a simpler problem: For each $s \in S$ find a point with minimum $x$-coordinate among the points of $S \setminus \{s\}$ that are above $h_i'(s)$. In order to solve this problem we maintain for the points of $S$ a data structure $T$. $T$ is a binary search tree storing the points of $S$ in its leaves sorted in the direction of $H_1$. At each node $u$ of $T$ we maintain also a point with minimum $x$-coordinate among the points that are stored in the subtree of $u$. We denote the set of points that are stored in the subtree of $u$ by $S_u$.

Let $s \in S$ be a point for which we want to determine the point with minimum $x$-coordinate among the points that are above $h_i'(s)$. Then we initialize a set $M := \emptyset$ and

follow the path from the root to the leftmost leaf which stores a point above $h'_i(s)$, i.e., the first point that is larger than $s$ w.r.t. the direction $H_i$. For each node $u$ on this path, if we move to its left child then we insert its right child into $M$. The final set $M$ consists of $O(\log n)$ nodes such that $\cup_{u \in M} S_u$ contains exactly the points of $S$ that are above $h'_i(s)$. Hence, we have to find a point with minimum $x$-coordinate in $\cup_{u \in M} S_u$. Since at each node $u$ is maintained a point of $S_u$ whose $x$-coordinate is minimal, we can find a point whose $x$-coordinate is minimal in $\cup_{u \in M} S_u$ in $O(\log n)$ time. Note that we can insert a new point in the data structure $T$ in $O(\log n)$ time.

Now we turn to the original problem: For each $s \in S$ find a point with minimum $x$-coordinate among the points of $S \setminus \{s\}$ that are above $h'_i(s)$ and below $h'_{i+1}(s)$. In order to solve this problem we initialize an empty data structure $T$, as described above, and we perform a plane sweep in the opposite direction of line $H_{i+1}$. When we encounter a point $s \in S$ we insert it into the data structure $T$. At this moment $T$ stores exactly the points of $S$ that are below $h'_{i+1}(s)$ and the point $s$. Therefore, by querying $T$ we find a point with minimum $x$-coordinate among the points that are above $h'_i(s)$ and below $h'_{i+1}(s)$. Since each point of $S$ is inserted into $T$ once and for each point we performed one query, the total time of the algorithm is bounded by $O(n \log n)$.

If we renounce the assumption that the projections of the points of $S$ to $H_i$ as well as to $H_{i+1}$ are pairwise distinct and we have to find a point with minimum $x$-coordinate among the points that are not below $h'_i(s)$, not above $h'_{i+1}(s)$ and right to $s$ then the only modification to the above description is the following. The data structure $T$ stores the points of $S$ such that the points that have the same projection to $H_i$ are sorted increasing according to the $x$-coordinate. Then for each point $s \in S$ we can determine in $O(\log n)$ time a point with minimum $x$-coordinate among the points of $S \setminus \{s\}$ that are on $h'_i(s)$ and right to $s$, or above $h'_i(s)$. The second slight modification is that during the plane sweep in the direction of $H_{i+1}$ we process the points with same projection to $H_{i+1}$ right to left. In that way we guarantee that at the moment before a point $s$ is processed the data structure $T$ contains exactly the points that are on $h'_{i+1}(s)$ and right to $s$ or below $h'_{i+1}(s)$. Hence, querying $T$ reports the desired point. Clearly, the running time of the whole algorithm remains $O(n \log n)$. □

Algorithm 2 may appear more complicated than Algorithm 1. We will see the advantage of this approach later when we want to compute the $k$ nearest neighbors in the cones.

### 2.1.3 The weak spanner property for $\theta \leq \frac{\pi}{3}$

In some applications of the $\theta$-graph we need a weaker requirement than the spanner property. In this subsection we investigate the question for which values of $\theta$ we can

guarantee that the $\theta$-graph has the weak spanner property, i.e., the property that for each pair $s, t \in S$ of points the graph contains a path between $s$ and $t$ such that for each point $v$ on this path holds that $dist_2(s, v) \leq f \cdot dist_2(s, t)$, for a real constant $f \geq 1$. This property was first applied by Fischer et al. [39] in order to develop a linear size searching data structure for walkthrough systems. They used a slightly modified definition of the $\theta$-graph. The only difference to the original definition is that they assign the halfline $h_i$ uniquely to the cone $c_i$, $0 \leq i < k$. They claimed the following:

**Theorem 2.4** [39] *Let $S \subset \mathbb{R}^2$ be a set of $n$ points. Let $k \geq 6$ be an integer constant and $\theta = \frac{2\pi}{k}$. Then the $\theta$-graph $G_\theta(S)$ is a weak spanner for $S$ with stretch factor $f = \max(\sqrt{1 + 48\sin^4(\theta/2)}, \sqrt{5 - 4\cos\theta})$. $G_\theta(S)$ can be computed in $O(n \log n)$ time using $O(n)$ space.*

For a sketch of the proof we refer to [39].

| $\theta$: | $\frac{2\pi}{6}$ | $\frac{2\pi}{7}$ | $\frac{2\pi}{8}$ | $\frac{2\pi}{9}$ | $\frac{2\pi}{10}$ | $\frac{2\pi}{15}$ |
|---|---|---|---|---|---|---|
| $f$: | 2 | 1.64 | 1.47 | 1.39 | 1.33 | 1.16 |

Table 2.2: The bounds on the stretch factor $f$ implied by Theorem 2.4 for some values of $\theta$.

### 2.1.4 The weak spanner property for $\theta = \frac{\pi}{2}$

Now we examine the $\theta$-graph for $S$ with four cones $G_{\pi/2}(S)$. We fist give a some stronger definition of $G_{\pi/2}(S)$ than the definition in [39], we prove the weak spanner property of this graph and we show how it can be computed efficiently. The results of this subsection have been published in [40].

Our definition of the $\theta$-graph only differs from the definition in [39] by choosing the nearest neighbor of the points in the cones. We also assume that the boundary halfline $h_i$ is assigned uniquely to the cone $c_i$. For each point $s \in S$ and each cone $c_i$, let $S_{c_i(s)} := c_i(s) \cap S \setminus \{s\}$, i.e., $S_{c_i(s)}$ is the set of points of $S \setminus \{s\}$ that are contained in the cone $c_i(s)$. In [39] the neighbor $s'$ of a point $s \in S$ in $G_\theta(S)$ in the cone $c_i(s)$ is chosen arbitrarily among the points of $S_{c_i(s)}$ with minimum distance from $s$ w.r.t. $dist_{c_i}$. If more than one point has minimum distance w.r.t. $dist_{c_i}$ then we choose as neighbor among them the one whose Euclidean distance from the cone axis $l_{c_i}(s)$ is smallest. With other words we use a *combined distance function* to determine the neighbor of a point $s \in S$ in the cone $c_i(s)$ in $G_\theta(S)$. The first component is $dist_{c_i}(s, s')$, $s' \in c_i(s)$ and the second component – which we denote by $dist^{\perp}_{c_i}(s, s')$ – is the Euclidean distance of $s'$ from

$l_{c_i}(s)$. The neighbor of $s$ in $c_i$ is a point $s' \in S_{c_i(s)}$ for which $(dist_{c_i}(s,s'), dist^{\perp}_{c_i}(s,s'))$ is lexicographically minimal (Figure 2.5). Note that also the neighbor, chosen in this way, is not necessarily unique. For the simple and robust computability of $G_{\pi/2}(S)$ we assume that $h_i$, $0 \leq i < 4$, is the non negative $x$-axis rotated around the origin over the angle $\frac{(2i-1)\pi}{4}$ and $l_{c_i}$ is the non negative $x$-axis rotated around the origin over the angle $\frac{i\pi}{2}$. Note that $dist_{c_i}$ is identical to the $L_\infty$ distance and $dist^{\perp}_{c_i}$ with the $L_1$ distance, $0 \leq i < 4$.

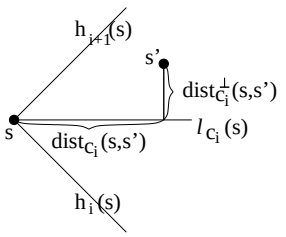


Figure 2.5: The distances $dist_{c_i}$ and $dist^{\perp}_{c_i}$

**Theorem 2.5** *$G_{\pi/2}(S)$ is a weak spanner with stretch factor $\sqrt{3+\sqrt{5}}$. It can be computed in $O(n \log n)$ time using $O(n)$ space.*

**Proof:**
**The weak spanner property:** For the weak spanner property we have to prove the existence of a path $P$ in the graph between each pair $s, t \in S$ of points which contains only points $v \in S$ with the property that $dist_2(s,v) \leq f \cdot dist_2(s,t)$. In order to find such a path we proceed as Ruppert and Seidel [71].

- Let $s_0 := s$, $j := 0$ and let $P$ contain the single point $s_0$. While $s_j \neq t$ do the following.
- Let $c_i(s_j)$ be the cone of $s_j$ which contains $t$. Let $s_{j+1}$ be the neighbor of $s_j$ in $G_{\pi/2}(S)$ in the cone $c_i(s_j)$ and set $j := j + 1$.

This algorithm constructs a path $P$ with the desired property. Unfortunately, Lemma 2.2 does not help us to bound the distance of the furthest point of $P$ from $s$. In order to proof the weak spanner property we define a potential $\Phi$ for each point $s' \in S$ and show that $\Phi$ decreases in "almost" every step of the above algorithm. For each point $s' \in S$, let $\phi_1(s')$ be the $L_1$ distance $dist_1(t,s')$ from $t$ to $s'$ and let $\phi_2(s')$ be the $L_\infty$ distance $dist_\infty(t,s')$ from $t$ to $s'$. We define $\Phi(s')$ to be the pair $(\phi_1(s'), \phi_2(s'))$. We say that $\Phi(s')$ is greater than (resp., greater than or equal to) $\Phi(s'')$ when $(\phi_1(s'), \phi_2(s'))$ is lexicographically greater than (resp., greater than or equal to) $(\phi_1(s''), \phi_2(s''))$. We denote this by $\Phi(s') > \Phi(s'')$ (resp., $\Phi(s') \geq \Phi(s'')$).

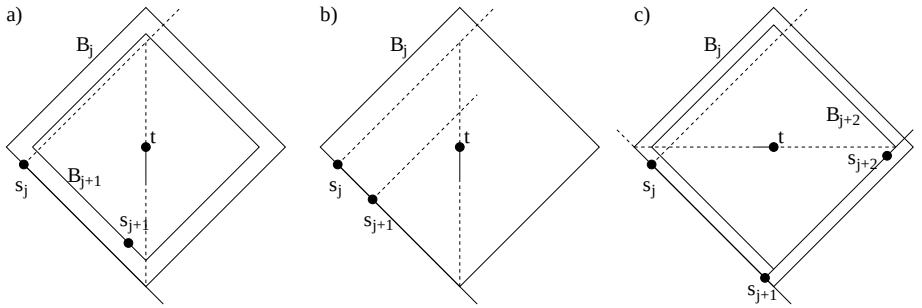


Figure 2.6: The potential $\Phi$ during the construction of $P$.

Consider the $j$th iteration, $0 \leq j \leq l-1$. Since the cone $c_i(s_j)$ contains at least one point (namely $t$), the algorithm is well defined in each step. Let $P = s_0, s_1, ..., s_l$, $s_0 = s$, $s_l = t$, be the path constructed by the algorithm. We show that

1. either $\Phi(s_j) > \Phi(s_{j+1})$, $0 \leq j \leq l-1$,
2. or $\Phi(s_j) > \Phi(s_{j+2})$, $0 \leq j \leq l-2$.

This property implies that the algorithm terminates and finds $t$. Note, if $s_{j+1} = t$ then $\Phi(s_j) > \Phi(t) = (0,0)$. To show the above property, consider the $j$th iteration. By the definition of $G_{\pi/2}(S)$ either there is a directed edge $\langle s_j, t\rangle$ in $G_{\pi/2}(S)$, then the algorithm finds $t$ and terminates; or there exists another point $s_{j+1} \in c_i(s_j)$ such that $\langle s_j, s_{j+1}\rangle \in G_{\pi/2}(S)$ and $dist_{c_i}(s_j, s_{j+1}) \leq dist_{c_i}(s_j, t)$. Let $B_j = \{x \in \mathbb{R}^2 : dist_1(t,x) \leq dist_1(t,s_j)\}$ be the $L_1$ unit disc. The point $s_{j+1}$ is clearly contained in $B_j$ (Figure 2.6) and so $\phi_1(s_j) \geq \phi_{j+1}$. We distinguish three cases.

Case 1: $dist_1(s_j,t) > dist_1(s_{j+1},t)$. Then $\phi_1(s_j) > \phi_1(s_{j+1})$ (Figure 2.6.a). Therefore, property 1 holds.

Case 2: $dist_1(s_j,t) = dist_1(s_{j+1},t)$ and $t \in c_i(s_{j+1})$. Then $\phi_1(s_j) = \phi_1(s_{j+1})$ and $\phi_2(s_j) > \phi_2(s_{j+1})$ (Figure 2.6.b). Therefore, property 1 holds.

Case 3: $dist_1(s_j,t) = dist_1(s_{j+1},t)$ and $t \in c_{i+1 \bmod 4}(s_{j+1})$. Then it can happen that $\phi_1(s_j) = \phi_1(s_{j+1})$ and $\phi_2(s_j) \leq \phi_2(s_{j+1})$. But in this case the next point $s_{j+2}$ can not be on the boundary of $B_j$ (Figure 2.6.c). Therefore, property 2 holds.

Now we prove that for each $v \in P$ the Euclidean distance $dist_2(s,v)$ is at most $\sqrt{3+\sqrt{5}} \cdot dist_2(s,t)$. We have seen that $B = B_0$ contains each point $s_j$ of the path $P$. Consider Figure 2.7. Let $a = dist_2(s,t)$ and $b = dist_2(s,v)$. We have to maximize the quotient $\frac{b}{a}$ such that $t$ is the center of $B$, $s$ is on the boundary of $B$ and $v$ is contained in $B$. For a fixed point $s \in B$ one of the corners $u$ of $B$ satisfy $dist_2(s,u) = \max(dist_2(s,x) : x \in B)$. Hence, we can fix $v$ at a corner of $B$. Using the theorem of Pythagoras and derivation

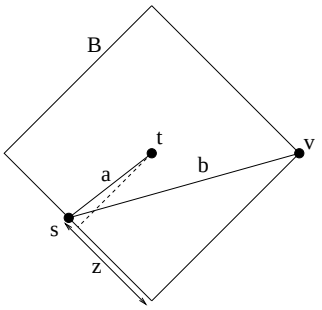


Figure 2.7: Illustration of the definitions for the computation of the stretch factor.

we obtain that $\frac{b}{a}$ $(= \frac{b(z)}{a(z)})$ is maximal when $z = \frac{\sqrt{5}-1}{2}l$, where $l$ is the Euclidean length of the side of $B$, and this maximum is $\sqrt{3+\sqrt{5}}$.

**Computation:** Now we show that $G_{\pi/2}(S)$ can be computed in $O(n \log n)$ time using $O(n)$ space. Our algorithm is a modified version of the algorithm of Ruppert and Seidel [71] which is described in the proof of Theorem 2.3. It consists of four phases. In the $i$th phase we compute for each point $s \in S$ its neighbor in the cone $c_i(s)$. We describe the 0th phase.

At first we sort the points of $S$ non decreasing w.r.t. the $x$-coordinate. The sweepline moves from left to right. We maintain the invariant that for each point $s \in S$ left to the sweepline, its neighbor $t \in S$ in $c_0(s)$ has been computed if $t$ is also left to the sweep line. We initialize a data structure $D := \emptyset$. $D$ will contain the points left to the sweepline whose neighbor has not yet been computed. The points in $D$ are sorted increasing w.r.t. the $y$-coordinate. To handle the distance function $dist_{c_i}$ correctly, when the sweepline reaches a point $t$ we put all points with the same $x$-coordinate as $t$ into a data structure $A$ and we work up these points in the same main step:

1. Let $A$ be the set of points with the same $x$-coordinate as $t$ ordered increasing w.r.t. the $y$-coordinate and let $A' := A$.
2. While $A \neq \emptyset$ we do the following:
   (a) Let $t$ be the point of $A$ with minimum $y$-coordinate. Determine the set of points $B(t) = \{s \in D : t \in c_0(s)\}$. Let the points in $B(t)$ are also ordered increasing w.r.t. the $y$-coordinate. Set $t' := t$. (The variable $t'$ contains at each time the point of $A$ with highest $y$-coordinate such that the points of $A$ with lower $y$-coordinate than $t'$ can not be a neighbor of any point of $D$.)
   (b) For each $s \in B(t)$ (in increasing order w.r.t. the $y$-coordinate) determine the point $t^* \in A$ whose $y$-coordinate is nearest to the $y$-coordinate of $s$ and join $s$ with $t^*$ by a directed edge. Then delete $s$ from $D$ and set $t' = t^*$.

(c) Delete $t$ from $A$. Then delete all points from $A$ having a lower $y$-coordinate than $t'$.

3. Insert the points of $A'$ into $D$.

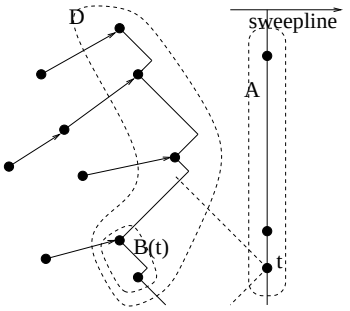


Figure 2.8: Illustration of the main step of the plane sweep.

**Correctness:** We show by induction that the invariant is satisfied after each main step of the plane sweep. In the data structure $D$ we maintain the points left to the sweepline $l$ whose neighbor is not left to $l$. At the beginning of the algorithm no point of $S$ is left to $l$, so by $D = \emptyset$ the invariant holds. In the main step we must determine the points of $D$ whose neighbor is on the sweepline $l$, compute their neighbors, and update $D$, i.e., delete the points from $D$ whose neighbor has been found and insert the points into $D$ that lie on $l$. In $A$ we maintain the points on $l$, for which, it is not yet decided if it is a neighbor of a point of $D$. Therefore, at the beginning $A$ must contain all points of $S$ on $l$. This is satisfied after Step 1. We show that in Step 2, for each point $s \in D$ with $c_0(s) \cap A \neq \emptyset$, the neighbor is computed correctly. We know that for the points $s \in D$, there is no point $s^* \in c_0(s)$ which is left to $l$. On the other hand, if a point $s \in D$ has a point $s^* \in A$ in $c_0(s)$ then $s$ is contained in $B(t)$ for some $t \in A$ in Step 2.a, its neighbor is computed correctly in Step 2.b, and then it is deleted from $D$. In Step 2.c the points are deleted from $A$ that are surely not a neighbor of a point of $D$. Therefore, after Step 2, $D$ contains the points left to $l$ whose neighbor is not left to $l$ and not on $l$. If we want to move the sweepline right to the next point, $D$ must not contain any point whose neighbor is already computed (these points are deleted in Step 2.b), but $D$ must contain the points on $l$, because their neighbor is not yet computed. Since the points on $l$ are inserted into $D$ in Step 3, we can move the sweepline, and the invariant is satisfied.

**Analysis:** To implement $D$, $A$ and $B(t)$ we need data structures that support $(i)$ checking emptiness, $(ii)$ insertion, deletion of a point, $(iii)$ finding the next and the previous element w.r.t. the $y$-coordinate for a given element of the data structure, and $(iv)$ finding the point with nearest $y$-coordinate in the data structure for a real number

(query). Using balanced search trees we obtain an $O(\log n)$ time for insertion, deletion, query and for finding the next and previous element, and $O(1)$ time for checking emptiness. If we link the nodes of the search tree in a doubly linked sorted list then we can also find the next and the previous element in $O(1)$ time. For a point $s \in D$, let $next(s)$ and $prev(s)$ denote the next and the previous element of $s$ in $D$ w.r.t. the $y$-coordinate. In Step 2.a we can determine $B(t)$ in the following way. Find the point $s \in D$ whose $y$-coordinate is nearest to the $y$-coordinate of $t$. If $s$ belongs to $c_0'(t)$ (the reflected cone $c_0(t)$) then we insert $s$ into $B(t)$. Let $s' := s$. While $next(s)$ belongs to $c_0'(t)$, insert $next(s)$ into $B(t)$ and set $s := next(s)$. Than Let $s := s'$. While $prev(s)$ belongs to $c_0'(t)$, insert $prev(s)$ into $B(t)$ and set $s := prev(s)$.

Since each point $s \in S$ is inserted into and deleted from $D$, $A$, and $B(t)$ exactly once and since for each point $s$, the nearest point $t \in A$ is computed once, the whole algorithm takes $O(n \log n)$ time. The size of the data structure is clearly $O(n)$. □

**Remark:** Note that without the second component $dist_{c_i}^{\perp}$ $(= dist_1)$ of the combined distance function, $0 \leq i < 4$, the obtained graph would not necessarily be strongly connected and consequently, it would not necessarily be a weak spanner. To see this consider the following example. Let $S = \{s_1, s_2, s_3, s_4, s_5\}$ and let $(1, 0)$, $(0, 1)$, $(-1, 0)$, $(0, -1)$ and $(0, 0)$ be coordinates of these points, respectively. Then the point $s_5$ in the center is not necessarily reachable from the other points by a directed path, since following holds. For $1 \leq i \leq 4$, the point $s_{(i \bmod 4)+1}$ is contained in the same cone around $s_i$ as $s_5$ and $dist_\infty(s_i, s_{(i \bmod 4)+1}) = dist_\infty(s_i, s_5)$. Therefore, for $1 \leq i \leq 4$, the neighbor of $s_i$ could be $s_{(i \bmod 4)+1}$ in the cone which contains $s_5$, causing that the first four points form a directed cycle around $s_5$. The second component of the combined distance function prohibits this possibility.

### 2.1.5 The reversed $\theta$-graph

The bounded outdegree of the $\theta$-graph is very useful for some applications, for example, for range searching. In this subsection we show that the stretch factor can be improved enormously by reversing the edges in the $\theta$-graph, if we do not require bounded outdegree.

Let $G_\theta^r(S)$ denote the directed graph obtained from $G_\theta(S)$ such that we reverse the orientation of each edge, i.e., for $s, t \in S$ $\langle s, t\rangle$ is an edge in $G_\theta^r(S)$ if and only if $\langle t, s\rangle$ is an edge in $G_\theta(S)$.

**Theorem 2.6** *Let $S \subset \mathbb{R}^2$ be a set of $n$ points.*

*(i) For $\theta \leq \frac{2\pi}{6}$, the graph $G^r_\theta(S)$ is a weak spanner for $S$ with stretch factor 1.*

*(ii) The graph $G^r_{\pi/2}(S)$ is a weak spanner for $S$ with stretch factor $\sqrt{2}$.*

**Proof:**
$(i)$: Consider two arbitrary points $s, t \in S$ and the $st$-path $P = s_0, s_1, ..., s_l$, $s_0 = s$, $s_l = t$, in $G_\theta(S)$ which is constructed such that $s_{i+1}$ is the neighbor of $s_i$ in $G_\theta(S)$ in the cone which contains $t$. Using Lemma 2.2 – and in the case that $\theta = \frac{\pi}{3}$ the fact that the boundary halfline $h_i$ is contained in the cone $c_i$ but $h_{i+1}$ is not – we obtain that $dist_2(s_i, t) > dist_2(s_{i+1}, t)$ for $0 \leq i < l$. This implies that the reversed version $P^r = s_l, ..., s_0$ of $P$ is a $ts$-path in $G^r_\theta(S)$ with the property that $dist_2(t, s_i) < dist_2(t, s)$ for $1 \leq i \leq l$. Hence, the claimed stretch factor follows.

$(ii)$: In the proof of Theorem 2.5 we have seen that for each pair $s, t \in S$ of points there is a directed $st$-path $P$ in $G_{\pi/2}(S)$ which is contained in $B(s) := \{x \in \mathbb{R}^2 : dist_1(t, x) \leq dist_1(t, s)\}$. This implies that the reversed path $P^r$ is a $ts$-path in $G^r_{\pi/2}(S)$ such that $dist_1(t, s_i) \leq dist_1(t, s)$ and hence, $dist_2(t, s_i) \leq \sqrt{2}\, dist_2(t, s)$ for each vertex $s_i \in P^r$. □

### 2.1.6 Generalizations of $G_{\pi/2}(S)$

Now we investigate the question whether the $\theta$-graph $G_{\pi/2}(S)$ becomes a weak spanner, if we choose the nearest neighbors in the cones w.r.t. to an $L_p$ distance and if we rotate the cones around the apex over a fixed angle $\phi$ (Figure 2.9). More precisely, let $1 \leq p \leq \infty$ and $0 \leq \phi < \frac{\pi}{2}$ be real numbers. Let $h_i$ be the non negative $x$-axis rotated around the origin over the angle $\frac{i\pi}{2} - \phi$, $i = 0, ..., 3$, and let $c_i$ be the cone between $h_i$ and $h_{i+1 \bmod 4}$. The halfline $h_i$ is assigned to the cone $c_i$. Let $h_i(q)$ and $c_i(q)$ denote the translated halflines and cones having the end points and apexes, respectively, at the point $q \in \mathbb{R}^2$. For a set $S$ of $n$ points let $G^{\phi,p}_{\pi/2}(S)$ be the graph obtained in the way that for each point $s \in S$ we join $s$ in each translated cone $c_i(s)$ with (one of) its nearest neighbor(s) w.r.t. the $L_p$ distance $dist_p$. In the case that $p = 1$ or $p = \infty$ we use the combined distance measures $(dist_1, dist_\infty)$ or $(dist_\infty, dist_1)$, respectively. Note that the graph $G_{\pi/2}(S)$, as we defined it in Subsection 2.1.4, corresponds to $G^{\pi/4,\infty}_{\pi/2}(S)$.

Unfortunately, the graph $G^{\phi,p}_{\pi/2}(S)$ does not become a weak spanner in general. We first study the graph $G^{\phi,\infty}_{\pi/2}(S)$ and $G^{\phi,1}_{\pi/2}(S)$, $0 \leq \phi < \frac{\pi}{2}$, where in each cone the nearest neighbor is chosen w.r.t. the $(dist_\infty, dist_1)$ and $(dist_1, dist_\infty)$, respectively. Then we apply similar techniques to $G^{\phi,p}_{\pi/2}(S)$, $1 < p < \infty$ and $0 \leq \phi < \frac{\pi}{2}$.

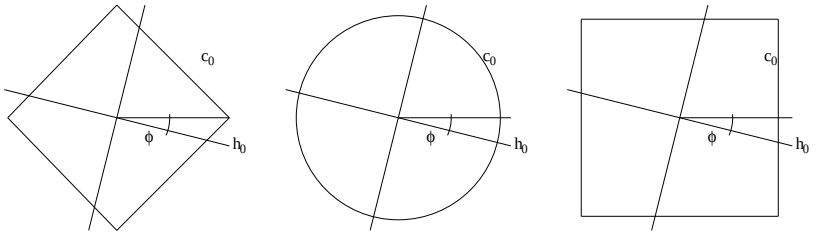


Figure 2.9: The rotated cones and the unit circle w.r.t. the $L_1$, $L_2$ and the $L_\infty$ distance.

**Theorem 2.7**

(*i*) *Let $S \subset \mathbb{R}^2$ be a set of $n$ points and let $\phi \in \{0, \frac{\pi}{4}\}$. Then the graphs $G^{\phi,\infty}_{\pi/2}(S)$ and $G^{\phi,1}_{\pi/2}(S)$ are weak spanners for $S$ with stretch factor $\sqrt{3+\sqrt{5}}$.*

(*ii*) *For each $0 < \phi < \frac{\pi}{2}$, $\phi \neq \frac{\pi}{4}$ and $n \geq 9$, there exist sets $S, S' \subset \mathbb{R}^2$ of $n$ points such that $G^{\phi\infty}_{\pi/2}(S)$ and $G^{\phi,1}_{\pi/2}(S')$ are not weak spanners for $S$ and $S'$, respectively.*

**Proof:**
(*i*): In Theorem 2.5 we proved that $G^{\pi/4,\infty}_{\pi/2}(S)$ is a weak spanner for $S$ with the claimed stretch factor. Clearly, we can compute $G^{0,1}_{\pi/2}(S)$ in the way that we rotate the point set $S$ around the origin by angle $-\frac{\pi}{4}$, then we compute the graph $G^{\pi/4,\infty}_{\pi/2}(S')$ for the rotated point set $S'$ and finally, we rotate the obtained graph back. Therefore, $G^{0,1}_{\pi/2}(S)$ is also a weak spanner with the same stretch factor.

Now we prove the weak spanner property for the graph $G^{0,\infty}_{\pi/2}(S)$. Then by the above arguments we obtain this property also for the graph $G^{\pi/4,1}_{\pi/2}(S)$. The proof is analogous to the proof of Theorem 2.5. We must show that between each two points $s, t \in S$ there is a path $P = s_0, ..., s_l$, $s_0 = s$, $s_l = t$, with the following properties.

1. Each point $s_i \in P$ is contained in the square region $B(s) = \{x \in \mathbb{R}^2 : dist_\infty(t, x) \leq dist_\infty(t, s)\}$.
2. For each $i = 0, ..., l-1$, either ($a$): $\Phi(s_i) > \Phi(s_{i+1})$ or ($b$): $\Phi(s_i) > \Phi(s_{i+2})$, where $\Phi(s_i)$ is the ordered pair $\Big(dist_\infty(t, s_i), dist_1(t, s_i)\Big)$ and the comparison between these pairs has to be made in lexicographical sense.

The first property implies the stretch factor $\sqrt{3+\sqrt{5}}$ and the second property implies that the path $P$ reaches $t$ in $O(n)$ steps. The construction rules of $P$ are the same as in Theorem 2.5: Assume that $P$ is constructed up to the vertex $s_i$ ($\neq t$). Then we take the neighbor of $s_i$ in $G^{0,\infty}_{\pi/2}(S)$ in the cone which contains $t$ as the next vertex $s_{i+1}$ of $P$.

Assume that we have an adversary, who tries to place the points of $S$ so that we do not get to $t$. Assume that $t \in c_0(s)$. Then the adversary must place a point $s_1$ in $c_0(s)$ so

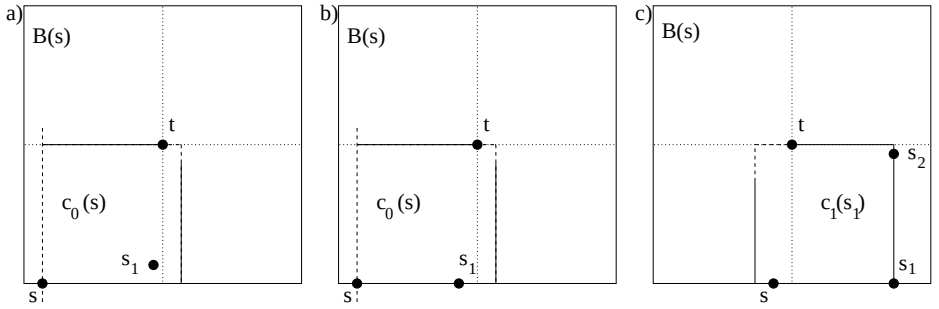


Figure 2.10: The path between $s$ and $t$ is contained in $B(s)$ and it gets strictly nearer to $t$ w.r.$t$ $\Phi$ at least in each second step.

that $(dist_\infty(s,s_1), dist_1(s,s_1)) \leq (dist_\infty(s,t), dist_1(s,t))$ in order to prohibit the edge $\langle s,t\rangle$. If he place $s_1$ so that $dist_\infty(s_1,t) < dist_\infty(s,t)$ then $\Phi(s_1) < \Phi(s)$ (Figure 2.10.a). If $s_1$ is placed on the boundary of $B(s)$ then either $dist_1(s_1,t) < dist_1(s,t)$ and so $\Phi(s_1) < \Phi(s)$ (Figure 2.10.b), or else $s_1$ must be in $c_0(s) \cap c_3(t)$ and so $t \in c_1(s_1)$. Observe that $s_1$ can never be placed in the right bottom corner of $B(s)$, since this corner is further from $s$ than $t$ w.r.t. to $dist_\infty$ in any case. Therefore, $s_2$ can not be placed on the right boundary of $B(s)$. Furthermore, $h_2(s_1)$ is not contained in $c_1(s_1)$ and hence, $s_2$ can not be placed on the bottom boundary of $B(s)$ (Figure 2.10.c). This implies that the adversary must place $s_2$ in such a way that $dist_\infty(t,s_2) < dist_\infty(t,s)$. Consequently, $\Phi(s_2) < \Phi(s)$.

$(ii)$: We construct a set $S$ for which the graph $G^{\phi,\infty}_{\pi/2}(S)$ is not a weak spanner. A set $S'$, for which $G^{\phi,1}_{\pi/2}(S')$ is not a weak spanner, can be obtained from $S$ by rotation around the origin by $-\frac{\pi}{4}$. We distinguish two cases.

Case 1: $0 < \phi < \frac{\pi}{4}$. In order to show that $G^{\phi,\infty}_{\pi/2}(S)$ is not necessarily a weak spanner, we construct a point set $S$ as follows. First we fix the position of $t$ and place a second point $s(= s_0)$ on one of the halflines $h_i(t)$, say on $h_2(t)$. Then $t \in h_0(s)$ and hence, $t \in c_0(s)$. Let $B(s)$ be the square region with center $t$ whose boundaries are parallel to the cone boundaries (i.e., to $h_0$ and $h_1$) and $s$ is on the boundary of $B(s)$. (Figure 2.11) Let $m_0, m_1, m_2, m_3$ be the middle points of the sides of $B(s)$ such that $m_i$ is the intersection of $h_i(t)$ with the boundary of $B(s)$. Note that $s$ is coincident with $m_2$. Let $D(s) := \{x \in \mathbb{R}^2 : dist_\infty(s,x) < dist_\infty(s,t)\}$. Clearly, the region $A(s) := D(s) \cap c_0(s) \setminus B(s)$ is not empty, if $\phi \neq \frac{\pi}{4}$ $(i \in \mathbb{N})$. Placing a point $s_1 \in S$ in the interior of $A(s)$, the point $s_1$ will be the neighbor of $s$ in the cone $c_0(s)$, because it is strictly closer to $s$ than $t$. Note that $s_1$ has an edge to $s$, as well.

Now we must place a point in $c_3(s_1)$ which is closer to $s_1$ than $t$ and further from $s$ than $s_1$, in order to prohibit the edge $\langle s_1,t\rangle$. We place the next point $s_2 \in S$ on $m_1$.

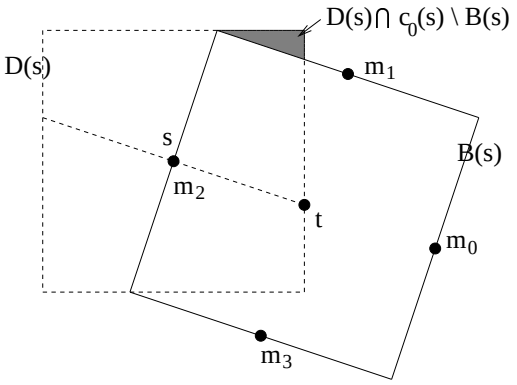


Figure 2.11: Place $s_1$ in $D(s) \cap c_0(s) \setminus B(s)$.

Then $s_1$ has an edge to $s_2$ in $c_3(s_1)$. The point $s_2$ has an edge in $c_1(s_2)$ to $s_1$ and in $c_2(s_2)$ to $s$.

We have the same situation as at the point $s$, just rotated by $-\frac{\pi}{2}$. Hence, we place the next point $s_3$ in the interior of $D(s_2) \cap c_3(s_2) \setminus B(s)$. So, we prohibit the edge from $s_2$ to $t$. Playing this game further, we place the points $s_4$ and $s_6$ on $m_0$ and $m_3$, and $s_5$ and $s_7$ in the interior of $D(s_4) \cap c_2(s_4) \setminus B(s)$ and in the interior of $D(s_6) \cap c_1(s_6) \setminus B(s)$, respectively. In this way we obtain a graph of nine points such that from none of the points $s_0, ..., s_7$ exists a directed path to $t$, because for each point $s_i$, the point $s_{i+1 \bmod 8}$ is in the same cone $c_j(s_i)$ as $t$ and it is slightly closer to $s_i$ than $t$ w.r.t. the distance $dist_\infty$. The remaining $n - 9$ points can easily be placed so that we do not destroy our counterexample (for example, each on the horizontal line incident to $t$, further from $t$ than $\rho / \sin\phi$, where $\rho$ is the Euclidean distance between $t$ and a side of $B(s)$).

Case 2: $\frac{\pi}{4} < \phi < \frac{\pi}{2}$. In this case we must take some care when placing the points. We use the definitions of Case 1. If we put the point $s_0$ on $m_2$ we would have to place a point in the interior of the region $D(s_0) \cap c_0(s_0)$, in order to prohibit the edge $\langle s_0, t \rangle$. But this region is contained entirely in $B(s)$, and so, we could not use the arguments of Case 1.

We move $s_0$ from $m_2$ on the boundary of $B(s)$ in clockwise direction by a Euclidean distance $\epsilon$ (Figure 2.12). The value of $\epsilon$ will be bounded later. Then $t \in c_3(s_0)$. To prohibit the edge $\langle s_0, t \rangle$ and to be able to use the strategy of Case 1, we must place a point $s_1$ in the interior of $A(s_0) := D(s_0) \cap c_3(s_0) \setminus B(s)$. If $\epsilon$ small enough $A(s_0)$ is not empty. Then we place $s_2$ on the boundary of $B(s)$ with Euclidean distance $\epsilon$ in clockwise direction from $m_3$ etc...

Placing the points $s_0, ..., s_7$ in this way, we obtain that for each point $s_i$, the point $s_{i+1 \bmod 8}$ is in the same cone $c_j(s_i)$ as $t$ and it is strictly closer to $s_i$ than $t$ w.r.t. the

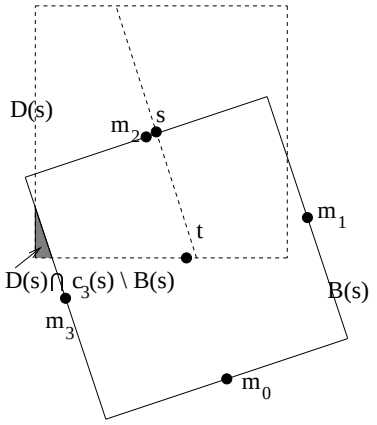


Figure 2.12: Placing the points, when $\frac{\pi}{4} < \phi < \frac{\pi}{2}$.

distance $dist_\infty$. Therefore, from none of the points $s_0, ..., s_7$ exists a path to $t$ in the graph $G^{\phi}_{\pi/2}(\{t, s_0, ..., s_7\})$. The remaining $n-9$ points can easily be placed so that we do not destroy our counterexample.

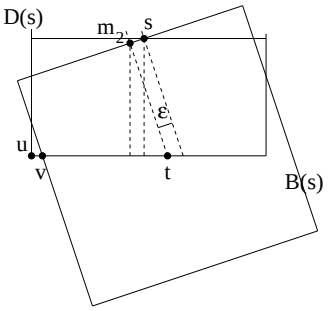


Figure 2.13: Computing the bound on $\epsilon$.

Now we compute the bound on $\epsilon$. Consider Figure 2.13. Let $u$ be the bottom left corner of $D(s)$ and let $v$ be the intersection of the boundary of $B(s)$ with the horizontal halfline which originates at $t$ and incident to $u$. The region $A(s_0)$ is not empty if and only if $dist_\infty(t, v) < dist_\infty(t, u)$. Let $r := dist_\infty(m_2, t)$ and $r' := dist_\infty(s, t)$. Then $r' = r + \epsilon \cos\phi$. We obtain the following two equalities. $dist_\infty(t, u) = r' - \epsilon \sin\phi + r \cot\phi$ and $dist_\infty(t, v) = r/\sin^2\phi$. Hence, $r(1 + \cot\phi) - \epsilon(\sin\phi - \cos\phi) > r/\sin^2\phi$. This gives the bound $\epsilon < r\frac{\cos\phi}{\sin^2\phi}$. □

**Remarks:** The graphs $G^{\phi,\infty}_{\pi/2}(S)$ and $G^{\phi,1}_{\pi/2}(S)$ obviously can be computed in $O(n \log n)$ time and $O(n)$ space using a slight modification of the algorithm described in the proof of Theorem 2.5.

In the proof of Theorem 2.7 $(i)$, similar to the proof of Theorem 2.5, the second component of the combined distance function is necessary. Otherwise, if we had only the first

component, a counterexample could be constructed such that the graph $G^{0,\infty}_{\pi/2}(S)$ does not necessarily become a weak spanner for $S$. For example, let $S := \{s_1, s_2, s_3, s_4, s_5\}$ and let $(-1,0)$, $(0,1)$, $(1,0)$, $(0,-1)$ and $(0,0)$ be the coordinates of these points, respectively. Then by similar arguments as in the remark after Theorem 2.5 we obtain that from non of the points $s_1, s_2, s_3, s_4$ must exist a directed path to $s_5$.

**Theorem 2.8**

(*i*) *Let $S \subset \mathbb{R}^2$ be a set of $n$ points, $p = 2$ and $\phi$ an arbitrary real number. Then the graph $G^{\phi,p}_{\pi/2}(S)$ is a weak spanner for $S$ with stretch factor $\sqrt{3+\sqrt{5}}$.*

(*ii*) *Let $S \subset \mathbb{R}^2$ be a set of $n$ points and let $1 < p < \infty$ and $\phi \in \{0, \frac{\pi}{4}\}$ real numbers. Then $G^{\phi,p}_{\pi/2}(S)$ is a weak spanner for $S$ with stretch factor $\sqrt{3+\sqrt{5}}$.*

(*iii*) *For $p \neq 2$ and $\phi \notin \{0, \frac{\pi}{4}\}$, $0 \leq \phi < \frac{\pi}{2}$, there exists $S$ such that the graph $G^{\phi,p}_{\pi/2}(S)$ is not a weak spanner for $S$.*

**Proof:**
(*i*): It suffices to consider the case that $\phi = 0$, since the Euclidean distance $dist_2$ is invariant for rotation. With other words, the graph $G^{\phi,2}_{\pi/2}(S)$ has the same edges as the graph $G^{0,2}_{\pi/2}(S')$, where $S'$ is the rotated point set $S$ around the origin by the angle $-\phi$. To prove the weak spanner property and the stretch factor of the graph $G^{0,2}_{\pi/2}(S_\phi)$ we can repeat the arguments of the proof of Theorem 2.7 (*i*).

(*ii*): Here we can also repeat the the arguments of the proof of Theorem 2.7 (*i*) and Theorem 2.5, respectively.

(*iii*): We construct the point set $S$ in a similar way as in Theorem 2.7. We adopt the definitions of $B(s)$, $m_i$, $i = 0, ..., 3$, from the proof of Theorem 2.7 (*ii*) and for a point $q \in \mathbb{R}^2$ we define $D(q) := \{x \in \mathbb{R}^2 : dist_p(q,x) < dist_p(q,t)\}$. In order to be able to use the arguments of Theorem 2.7 (*ii*) we only have to show the following.

**Lemma 2.9** *Let $p, \phi \in \mathbb{R}$, $p \geq 1$ and $0 \leq \phi < \frac{\pi}{2}$. The region $A(s) := D(s) \cap c_0(s) \setminus B(s)$ is non empty if and only if either $p > 2$ and $0 < \phi < \frac{\pi}{4}$ or else $1 \leq p < 2$ and $\frac{\pi}{4} < \phi < \frac{\pi}{2}$.*

**Proof:** Assume w.l.o.g. that the point $s$ is coincident with the origin and $D(s)$ is the $L_p$ unit disc. The boundary $\partial D(S)$ of $D(s)$ is described by the implicit function $|y|^p + |x|^p = 1$. Consider the part of $\partial D(s)$ above the $x$-axis and not left to the $y$-axis. In interval $x \in [0,1)$ the function $g(x) = (1+x^p)^{1/p}$ is coincident with this part of $\partial D(s)$. Let $v$ be the intersection point of the halfline $h_1(s)$ and the curve $g(x)$ (Figure 2.14). Note that $v$ is a corner of the square region $B(s)$. Let $l(x)$ be the function describing the line which contains the halfline $h_0(v)$ and so a side of $B(s)$. Let $x(v)$ and $y(v)$ be

the $x$- and the $y$-coordinate of $v$. We examine the derivatives $g'(x) = -(1-x^p)^{\frac{1-p}{p}}x^p$ and $l'(x) = -\tan\phi$ at the point $x(v)$. From the equations

$$\begin{aligned} \frac{y(v)}{x(v)} &= \tan\left(\frac{\pi}{2} - \phi\right) \qquad \text{and} \\ y(v)^p &= 1 - x(v)^p \end{aligned}$$

we obtain that

$$\begin{aligned} x(v) &= \left(\frac{\sin^p\phi}{\sin^p\phi + \cos^p\phi}\right)^{\frac{1}{p}} \qquad \text{and so} \\ g'(x(v)) &= -\tan^{p-1}\phi. \end{aligned}$$

The value of $(g-l)'(x(v)) = -(\tan^{p-1}\phi - \tan\phi)$ is positive in the cases that $p > 2$ and $0 < \phi < \frac{\pi}{4}$ or else $1 \le p < 2$ and $\frac{\pi}{4} < \phi < \frac{\pi}{2}$; zero if $p = 2$ or $\phi \in \{0, \frac{\pi}{4}\}$; and negative if $p > 2$ and $\frac{\pi}{4} < \phi < \frac{\pi}{2}$ or else $1 \le p < 2$ and $0 < \phi < \frac{\pi}{4}$. Consider the cases that $(g-l)'(x(v))$ is positive. From the definition of the derivative function follows that there exists an $\epsilon_0 > 0$ such that for each $\epsilon$, $0 < \epsilon < \epsilon_0$ holds that

$$\frac{(g-l)(x(v)+\epsilon) - (g-l)(x(v))}{\epsilon} > 0.$$

Since $g(x(v)) = l(x(v))$, follows that $(g-l)(x(v)+\epsilon) > 0$. It means that in the non empty interval $(x(v), x(v)+\epsilon_0)$ the function $g$ has strictly greater values than the function $l$. This proves the claim of the Lemma. □

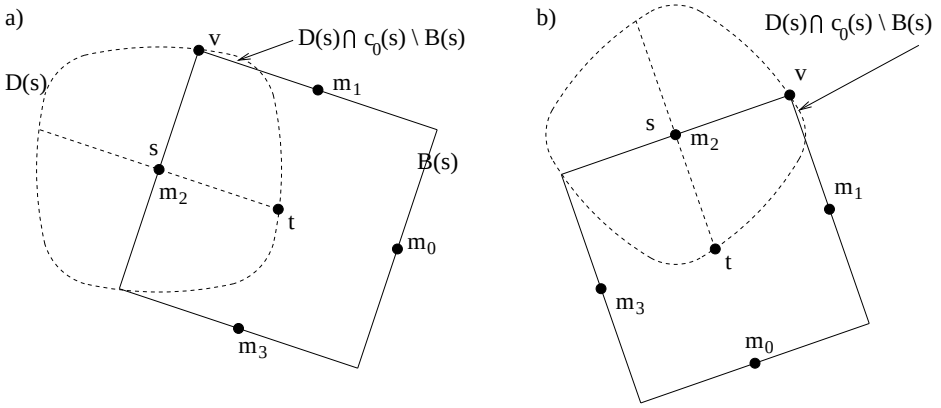


Figure 2.14: The region $D(s) \cap c_0(s) \setminus B(s)$ is non empty if *a)* $p > 2$ and $0 < \phi < \frac{\pi}{4}$, or else *b)* $1 < p < 2$ and $\frac{\pi}{4} < \phi < \frac{\pi}{2}$.

Now we return to the proof of Theorem 2.8 (*iii*). We distinguish two cases.

Case 1: $p > 2$ and $0 < \phi < \frac{\pi}{4}$ or else $1 \le p < 2$ and $\frac{\pi}{4} < \phi < \frac{\pi}{2}$. We fix the point $t$ and place $s$ on the halfline $h_2(t)$. In the same way as in the proof of Theorem 2.7 (*ii*),

we place the points $s_{2i}$ on $m_{2-i \bmod 4}$ and the points $s_{2i+1}$ in the regions $A(s_{2i}) := D(s_{2i}) \cap c_{-i \bmod 4}(s_{2i}) \setminus B(s)$, $0 \leq i < 4$. See Figure 2.14. Lemma 2.9 implies that the regions $A(s_{2i})$ are not empty. Since for $s_i$ the point $s_{i+1 \bmod 8}$ lies in the same cone as the point $t$ and it is strictly closer to $s_i$ than $t$, from none of the points $s_0, ..., s_7$ exists an edge to $t$ in the graph $G^{\phi,p}_{\pi/2}(\{t, s_0, ..., s_7\})$. The remaining $n-9$ points can easily be placed so that we do not destroy our counterexample.

Case 2: $p > 2$ and $\frac{\pi}{4} < \phi < \frac{\pi}{2}$ or else $1 \leq p < 2$ and $0 < \phi < \frac{\pi}{4}$. As in Case 2 of the proof of Theorem 2.7 $(ii)$ we place the points $s_{2i}$ on the boundary of $B(s)$ with Euclidean distance $\epsilon$ in clockwise direction from $m_{i+2 \bmod 4}$, and the points $s_{2i+1}$ in the regions $A(s_{2i}) := D(s_{2i}) \cap c_{i-1 \bmod 4}(s_{2i}) \setminus B(s)$, $0 \leq i < 4$. If $\epsilon$ is small enough the regions $A(s_{2i})$ are not empty. The bound for $\epsilon$ depends on $p$ and $\phi$. □

Table 2.3 summarizes the results of Subsection 2.1.6.

**Remark:** D'Amore et al. [30] showed that the graph $G^{\phi,2}_{\pi/2}(S)$ can be computed in $O(n \log n)$ time and in $O(n)$ space using plane sweeps. For the computation of $G^{\phi,p}_{\pi/2}(S)$, $p \notin \{1, 2, \infty\}$, there is no subquadratic algorithm known.

| $G^{\phi,p}_{\pi/2}$ | $p = 1$ | $1 < p < 2$ | $p = 2$ | $2 < p < \infty$ | $p = \infty$ |
|---|---|---|---|---|---|
| $\phi \in \{0, \frac{\pi}{4}\}$ | yes[2] | yes | yes | yes | yes[3] |
| $0 < \phi < \frac{\pi}{2}$, $\phi \neq \frac{\pi}{4}$ | no | no | yes | no | no |

Table 2.3: Weak spanner property of the graphs $G^{\phi,p}_{\pi/2}(S)$.

### 2.1.7 The graph $G_{2\pi/3}(S)$

In the previous subsections we showed that we can obtain a weak spanner for each set $S$ of $n$ points in the plane by constructing a $\theta$-graph whose outdegree is bounded by four. In this subsection we prove that to obtain a weak spanner for an arbitrary set $S$ of $n$ points, four is a lower bound for the outdegree of the $\theta$-graph when we use *convex distance functions* to choose the nearest neighbor for the points in the cones around them. The lower bound holds even if we allow to use different convex distance functions in different cones. The result of this subsection is also described in [42].

*Convex distance functions* are defined as follows. Let $D$ be a convex, compact set in the plane such that the origin is contained in the interior of $D$, i.e., in $D \setminus \partial D$,

[2]With the combined distance function $(dist_1, dist_\infty)$ we obtain a weak spanner. The second component of the combined distance is necessary.

[3]With the combined distance function $(dist_\infty, dist_1)$ we obtain a weak spanner. The second component of the combined distance is necessary.

where $\partial D$ denotes the boundary of $D$. The origin is called the *center* of $D$. In order to define the distance $dist_D(p,q)$ from a point $p$ to another point $q$ w.r.t. $D$, we first translate $D$ such that the center becomes coincident with $p$. The halfline with endpoint $p$ through $q$ intersects $\partial D$ at a unique point $q'$. Then we define $dist_D(p,q) := \frac{dist_2(p,q)}{dist_2(p,q')}$. Furthermore, we set $dist_D(p,p) := 0$. It can be shown that the triangle inequality $dist_D(p,r) \leq dist_D(p,q) + dist_D(q,r)$, $p,q,r \in \mathbb{R}^2$ holds (see, for example, in [12]).

Let $D_0, D_1$ and $D_2$ be convex shapes such that each contains the origin in its interior and let $D := \cup_{0 \leq i < 3}(D_i \cap c_i)$ the compound shape which is not necessarily convex. We assume that the cone boundary $h_0$ is coincident with the non negative $x$-axis and that $h_i \subset c_i$ and $h_i \not\subset c_{i+1 \bmod 3}$, $0 \leq i < 3$. The shape $D_i$ defines the convex distance function $dist_{D_i}$ and $D$ defines a distance function $dist_D$. Figure 2.15 shows an example for the convex shapes $D_0, D_1$ and $D_2$ and for the compound shape $D$. For a set $S$ of $n$ points in the plane, let $G^D_{2\pi/3}(S)$ be the directed graph obtained such that for each point $s \in S$ and each cone $c_i$, $0 \leq i < 3$, we join $s$ with (one of) its nearest neighbor(s) in $c_i$ w.r.t. $dist_D$.

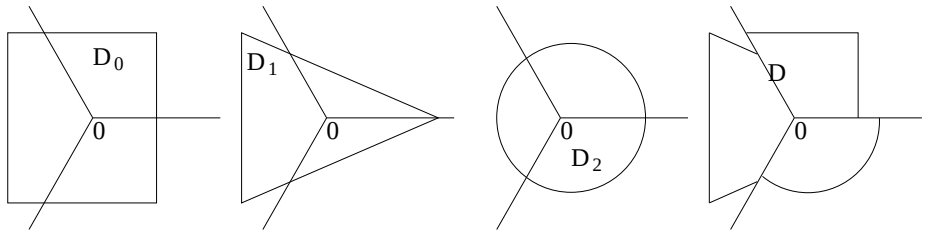


Figure 2.15: Example for the convex shapes $D_0, D_1, D_2$, that contain the origin in their interior, and for the compound shape $D$.

**Theorem 2.10** *Let $dist_D$ be a distance function as defined above. Then there exist a set $S$ of points in the plane such that $G^D_{2\pi/3}(S)$ is not a weak spanner for $S$.*

**Proof:** First we consider the case that $S$ only contains three points. Let $S := \{s, s', t\}$ such that the points have the coordinates $s = (-x, 1)$, $s' = (-x, -1)$ and $t = (0, 0)$, where $x$ is determined as follows. Let $a, b$ be positive real numbers such that for each point $p \in \mathbb{R}^2$ and integer $0 \leq i < 3$ holds that

$$a \cdot dist_\infty(t,p) < dist_{D_i}(t,p) < b \cdot dist_\infty(t,p).$$

Such $a$ and $b$ obviously exist, because the shape $D$ contains the origin in its interior. We set $x \geq \frac{2b}{a}$. Then $s'$ and $t$ are both contained in the same cone $c_2(s)$ around $s$ and

$$dist_D(s,s') < b \cdot dist_\infty(s,s') = 2b \leq a \cdot dist_\infty(s,t) < dist_D(s,t).$$

Therefore, $\langle s, s' \rangle \in G^D_{2\pi/3}(S)$ and $\langle s, t \rangle \notin G^D_{2\pi/3}(S)$. Similarly, $s$ and $t$ are both contained in $c_0(s')$ and $dist_D(s', s) < dist_D(s', t)$. Therefore, $\langle s', s \rangle \in G^D_{2\pi/3}(S)$ and $\langle s', t \rangle \notin G^D_{2\pi/3}(S)$. Consequently, there is no directed path from $s$ or $s'$ to $t$ in $G^D_{2\pi/3}(S)$.

In the case that $S$ contains more than three points then we place the first three points $s, s', t$ as described above and the remaining points on the positive $x$-axis. Then using the above arguments we obtain that there is no directed path from $s$ or $s'$ to any point of $S \setminus \{s, s'\}$ $G^D_{2\pi/3}(S)$. This proves the theorem. □

**Remark:** It can be shown that for each $f \in \mathbb{R}$, a set $S$ can be constructed with the following property. Let $s, t \in S$. Let $P$ be the path constructed such that $s_0 = s$ and $s_{i+1}$ is the neighbor of $s_i$ in $G^D_{2\pi/3}(S)$ in the same cone around $s_i$ which contains $t$. Then $\limsup_{i \to \infty} dist_2(t, s_i) > f \cdot dist_2(t, s)$. Figure 2.16 shows an example for the first steps of the construction of $S$ in the case that the distance $dist_{c_i}(p, q)$, $q \in c_i(p)$, is the Euclidean distance of $p$ and the projection of $q$ to the bisector $l_{c_i}(p)$ of $c_i(p)$. The boundary of the area at $s_i$ which must contain $s_{i+1}$, in order to prohibit the existence of the edge $\langle s_i, t \rangle$ in $G_{2\pi/3}(S)$ and to get further from $t$, is drawn dashed. The general case, if $dist_{c_i}$ is a convex distance function, is described in [42].

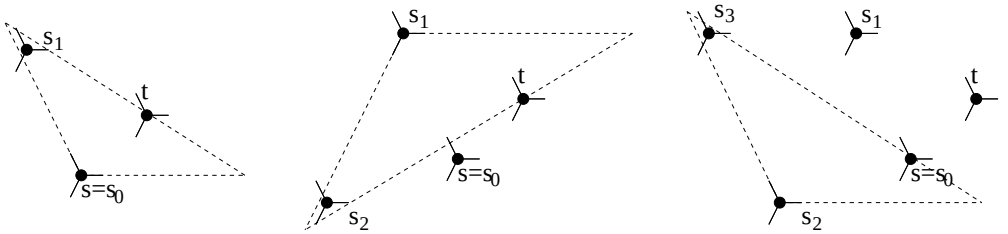


Figure 2.16: Example for the construction of a set $S$ such that $\limsup_{i \to \infty} dist_2(t, s_i) > f \cdot dist_2(t, s)$.

## 2.2 The higher dimensional $\theta$-graph

Now we examine the $d$-dimensional $\theta$-graph, where $d \geq 3$ is an integer constant. For the formal description we need the notion of simplicial cones. We assume that the points of $\mathbb{R}^d$ are represented by coordinate vectors and we do not distinguish between a point and its coordinate vector. Let $p_0, p_1, ..., p_d$ be points in $\mathbb{R}^d$ such that the vectors $(p_i - p_0)$, $1 \leq i \leq d$, are linearly independent. Then the set $\{p_0 + \sum_{i=1}^{d} \lambda_i (p_i - p_0) : \lambda_i \geq 0 \text{ for all } i\}$ is called a *simplicial cone* and $p_0$ is called the *apex* of the cone (see, e.g., in [48]). Sometimes, it is more convenient to use an equivalent definition, where a simplicial cone is defined as the intersection of $d$ halfspaces in $\mathbb{R}^d$ with the property that the

hyperplanes bounding the halfspaces are in general position in the sense that their intersection is a unique point, the apex of the cone. Let $\theta$ be a fixed angle $0 < \theta \leq \pi$ and $C$ be a collection of simplicial cones such that

($i$) each cone $c \in C$ has its apex at the origin,

($ii$) $\bigcup_{c \in C} c = \mathbb{R}^d$,

($iii$) for each cone $c \in C$ there is a fixed halfline $l_c$ having the endpoint at the origin such that for each halfline $l$, which has the endpoint at the origin and is contained in $c$, the angle between $l_c$ and $l$ is at most $\frac{\theta}{2}$.

We call such a collection $C$ of simplicial cones a *$\theta$-frame*. Yao [82] showed a method how a $\theta$-frame $C$ of $(\frac{d}{\theta})^{O(d \log_b d)}$ cones[4] can be constructed, where $b = \frac{d}{d-1}$. Later Ruppert and Seidel [71] suggested another method which results in a $\theta$-frame $C$ of $(\frac{d}{\theta})^{O(d)}$ cones. In the following the number of cones in $C$ is denoted by $|C|$.

**Remark:** We use the $\theta$-frame definition suggested by Ruppert and Seidel [71]. Yao [82] and Arya et al. [8] defined a $\theta$-frame in a slightly different way. They replaced item ($iii$) with item ($iii'$) which says that each cone $c \in C$ has an angular diameter at most $\theta$ and the cone axis $l_c$ is an arbitrary halfline which is contained in $c$ and has the endpoint at the origin. The angular diameter is defined as the maximum angle between two halflines that are contained in $c$ and have their endpoint at the origin. Note that the condition of item ($iii'$) is implied by item ($iii$). Therefore, the techniques that we present in this section can be also applied without change when we use the modified definition of the $\theta$-frame. But we achieve a better bound for the stretch factor of the spanners with the definition in [71].

For a cone $c \in C$ and for a point $a \in \mathbb{R}^d$, we define $c(a) := \{a + x : x \in c\}$, i.e., $c(a)$ is the cone obtained by translating the cone $c$ such that the apex becomes coincident with $a$. Similarly, we define $l_c(a)$ to be the halfline $l_c$ translated such that the endpoint becomes coincident with $a$. For $c \in C$ and $p, q \in \mathbb{R}^d$ such that $q \in c(p)$, let $dist_c(p, q)$ denote the Euclidean distance between $p$ and the orthogonal projection of $q$ to $l_c(p)$.

Now let $S$ be a set of $n$ points in $\mathbb{R}^d$ and let $\theta$ be an angle such that $0 < \theta \leq \pi$. For each point $s \in S$ and each cone $c \in C$, let $S_{c(s)} := c(s) \cap S \setminus \{s\}$, i.e., $S_{c(s)}$ is the set of points of $S \setminus \{s\}$ that are contained in the cone $c(s)$. The $\theta$-graph $G_\theta(S)$ with vertex set $S$ is defined such that for each point $s \in S$ and each cone $c \in C$, $G_\theta(S)$ contains a directed edge from $s$ to (one of) its nearest neighbor(s) in $c(s)$ w.r.t. the distance $dist_c$ if $S_{c(s)} \neq \emptyset$.

[4] In order to enable a better comparison between different methods, we consider $d$ as a variable in the bound on the number of cones of a $\theta$-frame. If $d$ is a constant then $(\frac{d}{\theta})^{O(d \log_b d)} = (\frac{1}{\theta})^{O(d)}$.

This section is organized as follows. In Subsection 2.2.1 we take a closer look at the proof of the spanner property of the $d$-dimensional $\theta$-graph, done by Ruppert and Seidel [71], and we show why it actually works also in $d$-dimensions. After this, in Subsection 2.2.2 we present some methods to compute a $\theta$-frame. First we describe the method of Yao [82] and we give a $(\frac{d}{\theta})^{O(d \log_b d)}$ bound on the number of simplicial cones in the resulting $\theta$-frame, where $b = \frac{d}{d-1}$. Then we present the method of Ruppert and Seidel [71] which constructs a $\theta$-frame of $(\frac{d}{\theta})^{O(d)}$ cones. After this we describe our own method which results in a $\theta$-frame consisting of $O((\frac{d^{3/2}}{\theta})^{d-1})$ cones. Finally, in Subsection 2.2.3 we present two algorithms, done by Ruppert and Seidel [71] and Arya et al. [8], that construct the graph $G_\theta(S)$ for the set $S \subset \mathbb{R}^d$ of $n$ points in $O(n \log^{d-1} n)$ time and in $O(n \log^{d-2} n)$ space.

### 2.2.1 The spanner property

Now we examine the spanner property of the $d$-dimensional $\theta$-graph. Ruppert and Seidel [71] showed that their proof, which we have presented in the proof of Theorem 2.1, also works in the $d$-dimensional case. We give our own proof for the $d$-dimensional version of Lemma 2.2 which is the base of the proof of the spanner property.

**Lemma 2.11** [71] *Let $0 < \theta \leq \pi$ and $C$ be a $\theta$-frame. Let $p \in \mathbb{R}^2$ be a point and $c \in C$ be a cone. Furthermore, let $q$ and $r$ be two points in $c(p)$ such that $dist_c(p,q) \leq dist_c(p,r)$. Then $dist_2(q,r) \leq dist_2(p,r) - (1 - 2\sin\frac{\theta}{2})\, dist_2(p,q)$.*

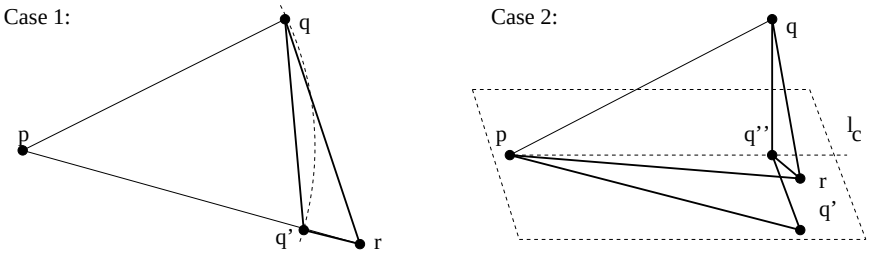


Figure 2.17: The two cases in the proof of Lemma 2.11. Case 1: $dist_2(p,q) \leq dist_2(p,r)$. Case 2: $dist_2(p,q) > dist_2(p,r)$.

**Proof:**
Case 1: $dist_2(p,q) \leq dist_2(p,r)$. For this case we can simply repeat the proof of Case 1 of Lemma 2.2, i.e., we immediately obtain the claim by applying the triangle inequality to the triangle $qq'r$, where $q'$ is the point on the line segment $pr$ such that $dist_2(p,q) = dist_2(p,q')$.

Case 2: $dist_2(p,q) > dist_2(p,r)$. Let $q''$ be the perpendicular projection of $q$ to the cone axis $l_c$ and let $q'$ be the point on the two-dimensional plane containing the triangle $prq''$ such that $dist_2(p,q') = dist_2(p,q)$ and $dist_2(q'',q') = dist_2(q'',q)$. Then we obtain the inequalities

$$dist_2(q,r) \leq dist_2(q,q'') + dist_2(q'',r), \tag{2.1}$$

$$dist_2(q'',r) + dist_2(p,q') \leq dist_2(q'',q') + dist_2(p,r). \tag{2.2}$$

Inequalities (2.1) and (2.2) together with $dist_2(q,q'') = dist_2(q'',q') \leq 2\sin\frac{\theta}{2} \cdot dist_2(p,q)$ imply that

$$\begin{aligned} dist_2(q,r) &\leq dist_2(q,q'') + dist_2(q'',q') + dist_2(p,r) - dist_2(p,q') \\ &\leq dist_2(p,r) - (1 - 2\sin\tfrac{\theta}{2})\ dist_2(p,q). \end{aligned}$$

□

**Theorem 2.12** [71] *Let $S$ be a set of $n$ points in $\mathbb{R}^d$ and let $0 < \theta < \frac{\pi}{3}$. Then the $\theta$-graph $G_\theta(S)$ is a spanner for $S$ with stretch factor $\frac{1}{1-2\sin(\theta/2)}$.*

**Proof:** The proof is exactly the same as in the two-dimensional case. For a pair of points $s, t \in S$ let $P$ be the $st$-path which is constructed so that the start point $s_0$ of $P$ is $s$ and $s_{i+1}$ is the neighbor of $s_i$ in $G_\theta(S)$ in the cone $c(s_i)$ which contains $t$. Then the length of $P$ can be bounded by $\frac{1}{1-2\sin(\theta/2)}$ applying Lemma 2.11 in the same manner as in the proof of Theorem 2.1. □

**Remark:** If one uses the modified definition of the $\theta$-frame, as in [82, 50, 8], i.e., using item (*iii'*) instead of item (*iii*), then Theorem 2.12 can only be proven for an angle $\theta$ in the range $0 < \theta < \frac{\pi}{4}$ and the stretch factor which we achieve is $\frac{1}{\cos\theta - \sin\theta}$ (see [50] for $d = 2$ and [8] for $d \geq 2$).

### 2.2.2 Construction of a $\theta$-frame

In this subsection we examine the problem how to construct a $\theta$-frame in $d \geq 3$ dimensions. In order to be able to compare different methods with one another, we exceptionally consider the dimension $d$ as a variable and not as a constant. Yao [82] was the first who showed how a $\theta$-frame can be constructed in $d \geq 3$ dimensions. First we present Yao's method and we give a $(\frac{d}{\theta})^{O(d\log_b d)}$ upper bound on the number of produced cones, where the basis $b$ of the logarithm is $\frac{d}{d-1}$. Then we briefly describe another construction method done by Ruppert and Seidel [71] which results in a $\theta$-frame of $(\frac{d}{\theta})^{O(d)}$ cones. Finally, we present an own construction which produces a $\theta$-frame of $O((\frac{d^{3/2}}{\theta})^{d-1})$ cones.

#### 2.2.2.1 Yao's method

In order to describe the Yao's method we need some definitions. We begin with the definition of a *j-simplex* in $\mathbb{R}^d$. Let $p_0, ..., p_j$, $0 \leq j \leq d$ be points in $\mathbb{R}^d$ such that the vectors $(p_i - p_0)$, $1 \leq i \leq j$, are linearly independent. Then the set $\Delta(p_0, ..., p_j) := \{\sum_{i=0}^{j} \lambda_i p_i : \lambda_i \geq 0 \text{ for all } i, \text{ and } \sum_{i=0}^{j} \lambda_i = 1\}$ is called a $j$-simplex. The *diameter* of a simplex $\Delta$ is $diam(\Delta) := \sup\{dist_2(p, q) : p, q \in \Delta\}$ which is equal to the length of the longest edge (1-face) of $\Delta$ (see Theorem 5-18 in [48]). The *centroid* of a simplex $\Delta = \Delta(p_0, ..., p_j)$ is the point $centr(\Delta) := \frac{1}{j+1} \sum_{i=0}^{j} p_i$. Finally, we define the radius of a simplex $\Delta = \Delta(p_0, ..., p_j)$ as the radius of the smallest $j$-dimensional ball centered at $centr(\Delta)$ which contains $\Delta$, i.e., $rad(\Delta) := \sup\{dist_2(centr(\Delta), p) : p \in \Delta\}$. For the radius and diameter of any $j$-simplex $\Delta$ holds that $rad(\Delta) \leq \frac{j}{j+1}\, diam(\Delta)$ (see Theorem 5-19 in [48]).

Informally, Yao's method is based on a certain recursive subdivision of the $(d-1)$-simplices on the boundary of the $L_1$ unit sphere until the diameter (and so the radii) of the resulting simplices become appropriately small. Each such simplex defines a simplicial cone of the $\theta$-frame and the centroid of the simplex defines the cone axis, as illustrated in Figure 2.18.

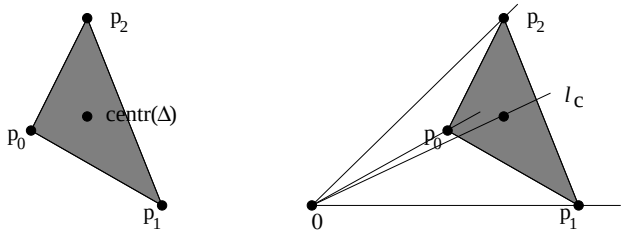


Figure 2.18: Example for a 2-simplex $\Delta$ and for the simplicial cone $c$ defined by them.

Now we describe the method more precisely. The *first barycentric subdivision* of a $j$-simplex $\Delta(p_0, ..., p_j)$ is a family $F$ of $j$-simplices such that the simplices of $F$ are defined as follows. For any $t$ distinct integers $0 \leq i_1, ..., i_t \leq j$, let $p_{i_1 ... i_t} := centr(\Delta(p_{i_1}, ..., p_{i_t}))$. Let $\Pi$ be the set of all permutations of $(0, ..., j)$. The first barycentric subdivision of $\Delta(p_0, ..., p_j)$ is defined as

$$F := \left\{\Delta(p_{i_0}, p_{i_0,i_1}, ..., p_{i_0,i_1,...,i_j}) : (i_0, i_1, ..., i_j) \in \Pi)\right\}.$$

The first barycentric subdivision $F$ of a $j$-simplex $\Delta$ consists of $(j+1)!$ simplices such that

(*i*) $\Delta = \bigcup_{\Delta' \in F} \Delta'$ and

(*ii*) For each $\Delta' \in F$, $diam(\Delta') \leq rad(\Delta) \leq \frac{j}{j+1}\, diam(\Delta)$ (see Theorem 5-19 in [48]).

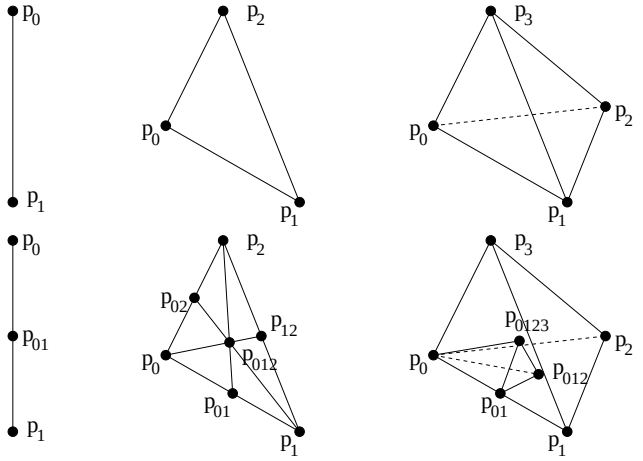


Figure 2.19: Example for a one-, two- and three-dimensional barycentric subdivision. For simplicity, in the three-dimensional case only one of the 4! simplices of the subdivision is shown.

Figure 2.19 shows examples for first barycentric subdivisions.

Let $\epsilon = (\epsilon_1, ..., \epsilon_d)$ be a $d$-tuple such that $\epsilon_i \in \{1, -1\}$. We denote by $H(\epsilon)$ the hyperplane $\{x \in \mathbb{R}^d : \sum_i \epsilon_i x_i = 1\}$ in $\mathbb{R}^d$. For two points $u, v \in H(\epsilon)$ let $\phi(u, v)$ be the angle between the two halflines $\{\lambda u : \lambda \geq 0\}$ and $\{\lambda v : \lambda \geq 0\}$.

**Lemma 2.13** [82] *Let $\Delta = \Delta(p_0, ..., p_{d-1})$ be a $(d-1)$-simplex such that $p_i \in H(\epsilon)$, $0 \leq i < d$. Furthermore, let $u$ and $v$ be two points contained in $\Delta$. Then $\cos(\phi(u, v)) \geq 1 - \frac{d}{2}(dist_2(u, v))^2$.*

**Proof:** For a point $x \in \mathbb{R}^d$ let $\|x\| := (x^T x)^{1/2} = (\sum_i x_i^2)^{1/2}$ be the Euclidean norm of $x$. Using elementary results of the linear algebra we obtain that

$$\begin{aligned}(dist_2(u, v))^2 &= (v-u)^T(v-u) = \|u\|^2 + \|v\|^2 - 2\|u\|\|v\|\cos(\phi(u, v)) \\ &\geq 2\|u\|\|v\|(1 - \cos(\phi(u, v))). \end{aligned} \tag{2.3}$$

Furthermore, $u, v \in \Delta \subset H(\epsilon)$ implies that

$$\|u\|^2 = \sum_{1 \leq i \leq d} u_i^2 \geq \frac{1}{d}\Big(\sum_{1 \leq i \leq d} \epsilon_i u_i\Big)^2 = \frac{1}{d}\,.$$

Therefore, $\|u\| \geq (\frac{1}{d})^{1/2}$, and similarly $\|v\| \geq (\frac{1}{d})^{1/2}$. This together with (2.3) implies that $\cos(\phi(u, v)) \geq 1 - \frac{d}{2}(dist_2(u, v))^2$. □

**Corollary 2.14** *Let $\Delta$ be a $(d-1)$-simplex contained in the hyperplane $H(\epsilon)$. Furthermore, let $0 < \theta \leq \pi$ be a fixed angle. If $rad(\Delta) \leq (\frac{2}{d}(1 - \cos\frac{\theta}{2}))^{1/2}$ then for each point $v \in \Delta$, $\phi(centr(\Delta), v) \leq \frac{\theta}{2}$.*

Now we are ready to describe the whole algorithm which constructs a $\theta$-frame for a given angle $0 < \theta \leq \pi$. Let $e_i$ denote the unit vector in $\mathbb{R}^d$ whose $i$th component is one and all others are zero.

Initialize $C := \emptyset$. For each of the $2^d$ $d$-tuples $\epsilon = (\epsilon_1, ..., \epsilon_d)$, $\epsilon_i \in \{-1, 1\}$, do the following.

1. Let $\Delta(\epsilon) = \Delta(\epsilon_1 e_1, ..., \epsilon_d e_d)$.
2. Compute a family $F$ of simplices such that $\Delta(\epsilon) = \bigcup_{\Delta' \in F} \Delta'$ and $rad(\Delta') \leq (\frac{2}{d}(1-\cos\frac{\theta}{2}))^{1/2}$, for each $\Delta' \in F$, by constructing the first barycentric subdivisions recursively.
3. For each $\Delta' \in F$, construct the simplicial cone $c := \{\lambda p : p \in \Delta',\ \lambda \geq 0\}$ with cone axis $l_c := \{\lambda \cdot centr(\Delta') : \lambda \geq 0\}$ and add $c$ to $C$.

The collection $C$ of simplicial cones constructed in this way is clearly a $\theta$-frame, since $\bigcup_{\epsilon \in \{-1,1\}^d} \Delta(\epsilon) = \bigcup_{\epsilon \in \{-1,1\}^d} \bigcup_{\Delta' \in F} \Delta'$ covers the $L_1$ unit $(d-1)$-sphere and therefore, $\bigcup_{c \in C} c$ covers $\mathbb{R}^d$. Furthermore, using Lemma 2.14 we obtain that each halfline $l$, which is contained in a cone $c \in C$ and has the endpoint at the origin, has an angular distance at most $\frac{\theta}{2}$ from the cone axis $l_c$.

It remains to examine how many cones are created. The diameter of a simplex $\Delta(\epsilon) = \Delta(\epsilon_1 e_1, ..., \epsilon_d e_d)$ is clearly $\sqrt{2}$ and therefore, the radius is $\frac{d-1}{d}\sqrt{2}$. After the $i$th recursive barycentric subdivision we obtain simplices whose radius is at most $(\frac{d-1}{d})^{i+1}\sqrt{2}$. Since the recursion stops when the radius of each simplex becomes less than or equal to $(\frac{2}{d}(1-\cos\frac{\theta}{2}))^{1/2}$, we obtain that the depth $\delta$ of the recursion is at most $-(\frac{1}{2}\log_b(\frac{1}{d}(1-\cos\frac{\theta}{2})) + 1)$, where $b = \frac{d}{d-1}$. Furthermore, a first barycentric subdivision of a simplex contains $d!$ simplices, therefore, a simplex $\Delta(\epsilon)$ is subdivided in at most $(d!)^\delta$ simplices. Using Stirling's formula (see, e.g., in [63]) we obtain that $(d!)^\delta = O((\frac{(2\pi d)^{1/2} d^d}{e^d})^\delta) = (\frac{d}{1-\cos(\theta/2)})^{O(d \log_b d)}$, where $b = \frac{d}{d-1}$. Since the algorithm starts with the $2^d$ simplices $\Delta(\epsilon)$, $\epsilon \in \{-1, 1\}^d$ and since $\lim_{x \to 0}(\frac{x^2/2}{1-\cos x}) = 1$ and $\frac{x^2/2}{1-\cos x} \leq \frac{\pi^2}{8}$, for $0 < x \leq \frac{\pi}{2}$, we obtain that the final number of simplices created by the algorithm is $(\frac{d}{\theta})^{O(d \log_b d)}$. Consequently, also $|C| = (\frac{d}{\theta})^{O(d \log_b d)}$ and $C$ can be obviously constructed in $(\frac{d}{\theta})^{O(d \log_b d)}$ time. We summarize this in the following.

**Theorem 2.15** *Let $0 < \theta \leq \pi$. Then a $\theta$-frame $C$ can be constructed such that the construction time and number of cones in $C$ are both bounded by $(\frac{d}{\theta})^{O(d \log_b d)}$, where $b = \frac{d}{d-1}$.*

#### 2.2.2.2 The Method of Ruppert and Seidel

Ruppert and Seidel [71] suggested the following method. First cover the Euclidean $(d-1)$-sphere by appropriate *spherical caps* and then compute the *Delaunay triangulation* of the centers of the caps. The $(d-1)$-simplices of the Delaunay triangulation together with the center of the sphere define the simplicial cones of the $\theta$-frame. The axis of such a cone is a halfline from the center of the sphere through the Delaunay circumcenter of the simplex. In order to describe this method more precisely, we need some definitions.

Let $S^{d-1} := \{x \in \mathbb{R}^d : \|x\| = 1\}$ denote the $(d-1)$-dimensional Euclidean unit sphere, short the unit $(d-1)$-sphere. Let $0 < \psi \leq \pi$ be an angle and $a$ be a point of $S^{d-1}$. Then the set $\{x \in S^{d-1} : \phi(a,x) \leq \psi\}$ is called a *spherical cap* with center $a$ and angular radius $\psi$. Remember, $\phi(a,x)$ denotes the angle between the halflines $\{\lambda a : \lambda \geq 0\}$ and $\{\lambda x : \lambda \geq 0\}$. In [70] Rogers proved the following.

**Lemma 2.16** [70] *Let $0 < \psi \leq \frac{\pi}{2}$. Then $S^{d-1}$ can be covered by $O(d^{3/2}\log(\frac{d}{\sin\psi})\sin^{-(d-1)}\psi)$ spherical caps with angular radius $\psi$.*

Now we define the *Delaunay triangulation* and the *Voronoi diagram* for a set $P$ of $m$ points in $\mathbb{R}^j$. Then we transfer the definition to a set of points on the unit $j$-sphere $S^j$. For an extensive overview of the numerous variants and numerous applications of the Delaunay triangulation and the Voronoi diagram we refer to the surveys of Aurenhammer [10] and Aurenhammer and Klein [11].

For two distinct points $p, q \in P$ the *dominance* of $p$ over $q$ is defined as the set of points of $\mathbb{R}^j$ being at least as close to $p$ as to $q$. Formally,

$$dom(p,q) := \{x \in \mathbb{R}^j : dist_2(p,x) \leq dist_2(q,x)\}.$$

Clearly, $dom(p,q)$ is a closed halfspace bounded by the $(j-1)$-dimensional hyperplane which is perpendicular to the line segment between $p$ and $q$ and contains the center of this line segment. This hyperplane is called the *separator* of $p$ and $q$. The *region* of a point $p \in P$ is the portion of the space lying in all of the dominances of $p$ over the remaining points in $P$, i.e.,

$$reg(p) := \bigcap_{q \in P \setminus \{p\}} dom(p,q).$$

Since the regions are the intersections of halfspaces, they are convex polyhedrons and they form a polyhedral partition of $\mathbb{R}^j$. This partition is called the *Voronoi diagram* $VD(P)$ of $P$. Note that $VD(P)$ contains exactly $m$ regions. We say that two regions are

*neighbors* of one another if their boundaries share a $(j-1)$-face. The corners (0-faces) of the regions are called *Voronoi vertices.*

Consider a Voronoi vertex $u$ which is on the boundary of the region $reg(p)$ for some $p \in P$. We grow a $j$-ball $B(u)$ centered at $u$ until the boundary of $B(u)$ hits the point $p$. At this moment there are (at least) $j+1$ points on the boundary of $B(u)$, because $u$ is the intersection of (at least) $j$ separator hyperplanes between $p$ and (at least) $j$ other points of $P$, and the separated points have the same distance from the separator. Clearly, $B(u)$ does not contain any point $q \in P$ in its interior, because in that case the separator between $p$ and $q$ would cut $u$ from $reg(p)$, which contradicts the initial assumption that $u$ is a corner of $reg(p)$. If we assume that no $j+2$ points of $P$ are cospherical, i.e., there exists no $(j-1)$-sphere which contains $j+2$ points of $P$, then $B(u)$ contains exactly $j+1$ points of $P$ on its boundary. These points define a $j$-simplex in $\mathbb{R}^j$, called a *Delaunay simplex.* The Delaunay simplices form a simplicial partition of $\mathbb{R}^j$, called the *Delaunay triangulation* $DT(P)$ of $P$. The property that the circumball of a Delaunay simplex does not contain any point in its interior is one of the most important properties of the Delaunay triangulation. It is called the *empty circle (sphere) property.* If $P$ is allowed to contain at least $j+2$ cospherical points then the ball $B(u)$, for some Voronoi vertex $u$, may contain more than $j+1$ points of $P$ on its boundary. In this case we partition the convex hull of this points into simplices arbitrarily.

Klee [52] proved that for $j \geq 2$, the maximum number of vertices that a Voronoi diagram of $m$ points can have – and therefore, also the maximum number of Delaunay simplices – is $\Theta(\lceil \frac{j}{2} \rceil ! m^{\lceil j/2 \rceil})$. Edelsbrunner [35] showed how it can be computed in $O(\lceil \frac{j}{2} \rceil ! m^{\lceil j/2 \rceil})$ worst-case optimal time. The algorithm in [35] maps the $m$ points of $\mathbb{R}^j$ to points in $\mathbb{R}^{j+1}$ and computes the convex hull of the mapped points. Then the simplices of the Delaunay triangulation can be obtained by projecting the faces of the convex hull to $\mathbb{R}^j$.

The definition of the Voronoi diagram and the Delaunay triangulation of a set $P$ of $m$ points in $\mathbb{R}^j$ can be transfered easily to the case that $P$ is a set of points in the unit $j$-sphere $S^j$. Then the dominance of $p$ over $q$ is defined as $dom(p,q) := \{x \in S^j : dist_2(p,x) \leq dist_2(q,x)\}$. The Voronoi region $reg(p) := \bigcap_{q \in P \setminus \{p\}} dom(p,q)$ of a point $p \in P$ becomes a polyhedron on the surface of $S^j$, and the $j$-simplices of the Delaunay triangulation are defined using growing spherical caps centered at Voronoi vertices. The empty circle property is modified to "empty spherical cap property". Then too compute the Delaunay triangulation of $P$ on the surface of $S^j$ is equivalent to the computation of the convex hull $CH(P)$ of $P$. The Delaunay simplices can be obtained immediately from the boundary faces of $CH(P)$. This follows from the next lemma.

**Lemma 2.17** *Let $P$ be a set of $m$ points on the unit $j$-sphere $S^j$. Assume that there is no $(j-1)$-sphere which contains more than $j+1$ points of $P$ on its boundary. Let $p_1, ..., p_{j+1} \in P$. Then the simplex $\Delta(p_1, ..., p_{j+1})$ is a Delaunay simplex if and only if $\Delta(p_1, ..., p_{j+1})$ is a simplex on the boundary of $CH(P)$.*

**Proof:** Let $H$ be the $j$-dimensional hyperplane which contains $\Delta(p_1, ..., p_{j+1})$ and let $H^+$ and $H^-$ be the two open halfspaces in $\mathbb{R}^{j+1}$ separated by $H$. Note that $S^j \cap H$ is a $(j-1)$-sphere and $S^j \cap H^+$ and $S^j \cap H^-$ are open spherical caps (see Figure 2.20). The simplex $\Delta(p_1, ..., p_{j+1})$ is a Delaunay simplex if and only if one of the caps $S^j \cap H^+$ or $S^j \cap H^-$, say $S^j \cap H^+$, does not contain any point of $P$. This is equivalent to the condition that $P \cap H^+ = \emptyset$ and so, $CH(P) \cap H^+ = \emptyset$. But this means that $\Delta(p_1, ..., p_{j+1})$ is a boundary simplex of $CH(P)$. □

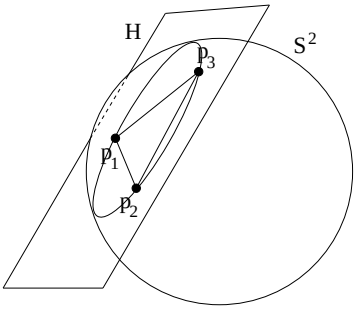


Figure 2.20: Example for a simplex $\Delta(p_1, p_2, p_3)$, $p_1, p_2, p_3 \in S^2$.

If $P$ is allowed to contain more than $j+1$ points incident to a $(j-1)$-sphere then, after the computation of $CH(P)$, we additionally have to triangulate the faces of $CH(P)$ that contain more than $j+1$ vertices.

The simplicial cones of the $\theta$-frame are computed such that we first construct a covering of the sphere $S^{d-1}$ by spherical caps that have an angular radius $\frac{\theta}{2}$, and then we compute the Delaunay triangulation of the centers of the caps. In this way we obtain a family $F$ of $O(\lceil \frac{d-1}{2} \rceil ! m^{\lceil (d-1)/2 \rceil})$ simplices, where $m = O(d^{3/2} \log(\frac{d}{\sin(\theta/2)}) \sin^{-(d-1)} \frac{\theta}{2})$. Hence, the number of simplices is $|F| = (\frac{d}{\sin(\theta/2)})^{O(d)} = (\frac{d}{\theta})^{O(d)}$. For each $\Delta \in F$ let $c(\Delta)$ be the cone $\{\lambda p : p \in \Delta, \lambda \geq 0\}$. Clearly, the cones cover $\mathbb{R}^d$. The axis of a simplicial cone $c(\Delta)$ is defined as the halfline $\{\lambda\, mid(\Delta) : \lambda \geq 0\}$, where $mid(\Delta)$ denotes the center of the circumcircle of $\Delta$, more precisely, $mid(\Delta)$ is the center of the spherical cap $sc(\Delta)$ which contains the vertices of $\Delta$ on its boundary. Since $sc(\Delta)$ contains no vertices in the interior, it must not be greater than the covering caps that are centered at the vertices of $\Delta$, otherwise, $mid(\Delta)$ would not be covered by any cap. But the fact, that $sc(\Delta)$ is not greater than the covering caps, implies that $mid(\Delta)$ is contained in each covering cap centered at a vertex of $\Delta$. Consequently, the angular distance between the

halfline from the origin through $mid(\Delta)$ and a halfline through any vertex of $\Delta$ is not greater than $\frac{\theta}{2}$. Therefore, the simplices $\{c(\Delta) : \Delta \in F\}$ form a $\theta$-frame. The following theorem summarize the above description.

**Theorem 2.18** [71] *Let $0 < \theta \leq \pi$. Then a $\theta$-frame $C$ can be constructed such that the construction time and number of cones in $C$ are both bounded by $(\frac{d}{\theta})^{O(d)}$.*

#### 2.2.2.3 Our method

We subdivide the space $\mathbb{R}^d$ into simplicial cones in two phases. In the first phase we subdivide $\mathbb{R}^d$ into cones with rectangular bases. Then, in the second phase, we subdivide these cones into simplicial cones.

The first phase is based on a subdivision of the surface of the $L_\infty$ unit ball into $(d-1)$-dimensional cubes until the diameter of the cubes become appropriately small. The $L_\infty$ norm $\|u\|_\infty$ of a point $u \in \mathbb{R}^d$ is the maximum of its coordinates. Consider the $d$-dimensional $L_\infty$ unit ball $B^d_\infty := \{x \in \mathbb{R}^d : \|x\|_\infty \leq 1\}$. $B^d_\infty$ is a cube. The *diameter*, the *centroid* and the *radius* of a cube $B \subset \mathbb{R}^d$ is defined similarly as for a simplex: $diam(B) := \sup\{dist_2(p,q) : p,q \in B\}$; $centr(B) := \frac{1}{2^d}\sum_{i=1}^{2^d} p_i$, where $p_1, ..., p_{2^d}$ are the corners (0-faces) of $B$; and $rad(B) := \sup\{dist_2(centr(B), p) : p \in B\}$. For each cube $B$ it holds that $rad(B) = \frac{1}{2}diam(B)$. Clearly, $rad(B^d_\infty) = \sqrt{d}$. Let $H_i := \{x \in \mathbb{R}^d : x_i = 0\}$ be the $(d-1)$-dimensional hyperplane which is perpendicular to the $i$th coordinate axis and contains the origin. Then each $(d-1)$-face of $B^d_\infty$ is contained in exactly one of the $2d$ hyperplanes $H_i \pm e_i$, $1 \leq i \leq d$, where $e_i$ is the $i$th unit vector. Therefore, the number of $(d-1)$-faces of $B^d_\infty$ is $2d$. Considering the hyperplanes $H_i \pm e_i$, $1 \leq i \leq d$, as subspaces of $\mathbb{R}^d$, the contained $(d-1)$-face of $B^d_\infty$ is a $L_\infty$ unit ball in the corresponding subspace. Hence, the radius of the $(d-1)$-faces of $B^d_\infty$ is $\sqrt{d-1}$. In the next lemma we give a bound for $\phi(u,v)$ if the points $u$ and $v$ are contained in the same $(d-1)$-face of $B^d_\infty$.

**Lemma 2.19** *Let $B$ be a $(d-1)$-face of the $L_\infty$ unit ball in $\mathbb{R}^d$ and let $u$ and $v$ be two points contained in $B$. Then $\cos(\phi(u,v)) \geq 1 - \frac{1}{2}(dist_2(u,v))^2$.*

**Proof:** Using inequality (2.3) of Lemma 2.13 and the fact that $\|x\| \geq \|x\|_\infty = 1$, for each point $x \in B$, we obtain that

$$\begin{aligned}(dist_2(u,v))^2 &\geq 2\|u\|\|v\|(1 - \cos(\phi(u,v))) \\ &\geq 2(1 - \cos(\phi(u,v)).\end{aligned}$$

This proves the claim. □

**Corollary 2.20** *Let $B_*$ be a $(d-1)$-dimensional cube contained in a face of $B_\infty^d$ and let $0 < \theta \leq \pi$ be a fixed angle. If $rad(B_*) \leq (2(1-\cos\frac{\theta}{2}))^{1/2}$ then for each point $v \in B_*$, $\phi(centr(B_*), v) \leq \frac{\theta}{2}$.*

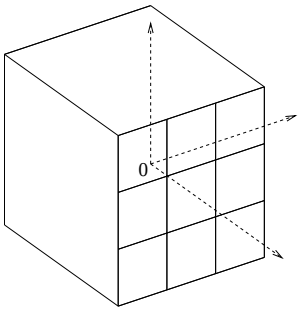


Figure 2.21: Subdivision of a 2-face of $B_\infty^3$. The coordinate axes are drawn dashed.

Now we describe an algorithm which constructs a $\theta$-frame of cones with rectangular bases, for a given angle $0 < \theta \leq \pi$. Corollary 2.20 implies that it is sufficient to subdivide the $(d-1)$-faces of the $L_\infty$ unit ball $B_\infty^d \subset \mathbb{R}^d$ into cubes with radius $(2(1-\cos\frac{\theta}{2}))^{1/2}$. Each such cube $B_*$ defines a cone $c := \{\lambda p \in \mathbb{R}^d : p \in B_*, \lambda \geq 0\}$ of the $\theta$-frame and the centroid $centr(B_*)$ defines the cone axis $l_c := \{\lambda \cdot centr(B_*) \in R^d : \lambda \geq 0\}$ of $c$. Such a subdivision can be easily computed. Since the radius of the $(d-1)$-faces of $B_\infty^d$ is $\sqrt{d-1}$, we have only to subdivide each $(d-1)$-face in each dimension equidistant, into intervals of length $2(\frac{2(1-\cos(\theta/2))}{d-1})^{1/2})$. Then the subdivision of the face is obtained as the Cartesian product of the $d-1$ one-dimensional subdivisions. In this way, each face is subdivided into $(\frac{d-1}{2(1-\cos(\theta/2))})^{(d-1)/2}$ cubes. Since $\lim_{x\to 0}(\frac{x^2/2}{1-\cos x}) = 1$ and $\frac{x^2/2}{1-\cos x} \leq \frac{\pi^2}{8}$, for $0 < x \leq \frac{\pi}{2}$, this is $O((\frac{\kappa \cdot d^{1/2}}{\theta})^{d-1})$, where $\kappa = \frac{\pi}{\sqrt{2}}$. Figure 2.21 shows an example for the subdivision of a 2-face. Using that the number of $(d-1)$-faces of $B_\infty^d$ is $2d$, we obtain that the total number of cubes in the subdivision is $O(d(\frac{\kappa \cdot d^{1/2}}{\theta})^{d-1})$. Note that a cube is fully defined by its centroid, because the radius of each cube is the same. Therefore, the computation time and the space requirement are also $O(d(\frac{\kappa \cdot d^{1/2}}{\theta})^{d-1})$. The following theorem summarize the above description.

**Lemma 2.21** *Let $0 < \theta \leq \pi$. Then a $\theta$-frame $C$ of cones with rectangular bases can be constructed such that the construction time and number of cones in $C$ are both bounded by $O(d(\frac{\kappa \cdot d^{1/2}}{\theta})^{d-1})$, where $\kappa = \frac{\pi}{\sqrt{2}}$.*

Now we turn to the description of the second phase in which we have to subdivide the cones with rectangular bases into simplicial cones, i.e., the $(d-1)$-dimensional cubes into $(d-1)$-simplices. The question of triangulating the $j$-cube $B^j$ has been intensively studied [78, 19, 47]. A simple way of the triangulation is the following. Fix an arbitrary

corner $p_0$ of $B^j$. For each permutation $\sigma$ of $(1, ..., j)$, let $\Delta(\sigma)$ be the simplex whose 0th vertex is $p_{\sigma,0} = p_0$ and the $i$th vertex $p_{\sigma,i}$, $0 < i \leq j$, is the corner of $B^j$ for which the line segment $p_{\sigma,i-1}p_{\sigma,i}$ is parallel to the $\sigma(i)$th coordinate axis[5]. The number of such simplices is clearly $j!$ and it holds that $\bigcup_\sigma \Delta(\sigma) = B^j$. Figure 2.22 shows an example for a simplex in the three-dimensional case. This triangulation is often referred as Kuhn's triangulation. A nice property of this triangulation is that each of the $j!$ simplices contains the centroid of the $j$-cube. Consequently, the corresponding simplicial cones contain the halfline from the origin through the centroid, which we take as the axis of the cones.

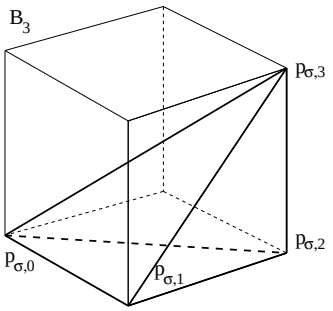


Figure 2.22: Example for the triangulation in the three-dimensional case. The cube $B^3$ and a simplex $\Delta(\sigma)$.

Applying Kuhn's triangulation for each $(d-1)$-cube of the subdivision of the faces of $B^d$ we obtain totally $O((d-1)!\, d(\frac{\kappa \cdot d^{1/2}}{\theta})^{d-1}) = O(d^{3/2}(\frac{d^{3/2}}{\theta})^{d-1})$ simplices, which is already significantly less than the number of cones in a $\theta$-frame constructed with the method of Yao [82] or of Ruppert and Seidel [71].

Note that an explicit representation of the simplices – i.e., storing $d$ vertices for each simplex – in the subdivision would need $O(d^{5/2}(\frac{d^{3/2}}{\theta})^{d-1})$ space. But since the simplices of a cube can be represented by permutations of $1, ..., d-1$, it is sufficient to store only $(i)$ a method which generates all permutations of $1, ..., d-1$ and $(ii)$ the cubes (i.e., the centroids).

Instead of Kuhn's triangulation we can also use a more sophisticated method, for example, the *middle cut triangulation* of Sallee [72]. This method allows us to subdivide a $j$-cube $B^j$ into $O(\frac{j!}{j^2})$ simplices. Applying this triangulation we obtain the following.

**Theorem 2.22** *Let $0 < \theta \leq \pi$. Then a $\theta$-frame $C$ of simplicial cones can be constructed such that the number of cones in $C$ is $O(d^{-1/2}(\frac{d^{3/2}}{\theta})^{d-1})$.*

[5]Bern et al. [16] used the notion of *path simplex* for such simplices

### 2.2.3 Construction of the $\theta$-graph

In Subsection 2.1.2 we have already shown how the $\theta$-graph for a set $S$ of $n$ points in the plane can be computed efficiently. Now we present the generalizations of that algorithm for the case that $S$ is a set of points in the $d$-dimensional Euclidean space $\mathbb{R}^d$, $d \geq 3$. We assume again that $d$ is an integer constant.

#### 2.2.3.1 The method of Ruppert and Seidel

The method of Ruppert and Seidel [71] can be generalized easily to work in $d \geq 3$ dimensions. The algorithm proceeds in $|C|$ phases. In each phase we choose a cone $c \in C$ and for each point $s \in S$ we compute its nearest neighbor in $c(s)$ w.r.t. the distance function $dist_c$, i.e., a point $s' \in c(s)$ whose projection to the cone axis $l_c(s)$ is nearest to $s$ in Euclidean sense.

In order to compute the nearest neighbors we preprocess the point set $S$ into a data structure, which allows orthogonal range queries, and perform a hyperplane sweep in the direction of $l_c$. Before the plane sweep we must apply an appropriate affine transformation $x \to Ax$, $A \in \mathbb{R}^{d \times d}$, to the points $x \in \mathbb{R}^d$ such that the transformed hyperplanes bounding the cone $c$ become orthogonal. Furthermore, the cone axis of the transformed cone, which determines the direction of the plane sweep, must be transformed as follows. Let $h$ be a $(d-1)$-dimensional hyperplane which is perpendicular to the original cone axis $l_c$ and let $h'$ be the hyperplane $h' := \{Ax : x \in h\}$. Then the cone axis of the transformed cone must be perpendicular to $h'$.

During the plane sweep we perform orthogonal range queries using a dynamic $(d-1)$-level data structure which consists of $(d-2)$-levels of *range trees* [56] plus one level of a *priority search tree* [60] to handle the last two dimensions. More precisely, the first level of this data structure is a balanced binary search tree storing the transformed points $s \in S$ in its leaves, sorted by their first coordinates. For each node $v$ of this tree, let $S_v$ be the subset of $S$ stored in the subtree of $v$. Each node $v$ of this tree contains a pointer to the root of a balanced binary search tree – a second level tree – storing the tranformed points $s \in S_v$ in its leaves, sorted by their second coordinates. Each node $w$ of this second level tree contains a pointer to the root of a balanced binary search tree – a third level tree – storing the transformed points $s \in S_w$ in its leaves, sorted by their third coordinates, etc... At the $(d-1)$th level there is a priority search tree which stores a subset of the transformed $S$. Figure 2.23 illustrates the data structure.

The space requirement of this $(d-1)$-level data structure is $O(n \log^{d-2} n)$. We can maintain it in $O(\log^{d-1} n)$ amortized[6] time per insertion and deletion and it supports

[6]The notation of amortized complexity appears often in the analysis of dynamic algorithms. It

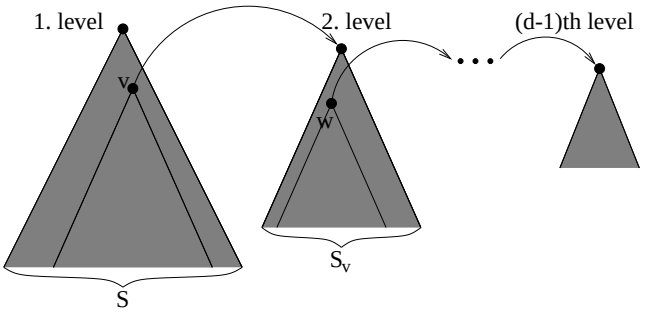


Figure 2.23: The $(d-1)$-level data structure which supports orthogonal range queries.

answering orthogonal range queries (where one side of the $d$th dimension of the range is unbounded, i.e., in the form of $[a, \infty)$) in $O(\log^{d-1} n + k)$ time, where $k$ is the number of reported points. For an exact analysis of this data structure we refer to Luecker [56] and McCraight [60] or to Mehlhorn [61].

Using the above $(d-1)$-level data structure during a plane sweep, we can compute for each point its neighbor in the cone $c$ in $O(n \log^{d-1} n)$ time. Therefore, the total time required to compute the $\theta$-graph is $O(|C| n \log^{d-1} n)$. We conclude.

**Theorem 2.23** [71] *Let $S \subset \mathbb{R}^d$ be a set of $n$ points and let $0 < \theta \leq \pi$ be a real number. Then the $\theta$-graph $G_\theta(S)$ can be constructed in $O((\frac{d^{3/2}}{\theta})^{d-1} n \log^{d-1} n)$ time using $O(n \log^{d-2} n)$ space.*

#### 2.2.3.2 The method of Arya et al.

In order to describe the $d$-dimensional version of the algorithm of Arya et al. [8] for $d \geq 3$, we need some definitions. Let $c$ be a simplicial cone of $C$. Let $h_1, ..., h_d$ be the hyperplanes that bound the halfspaces defining $c$, and let $H_1, ..., H_d$ be the lines such that $H_i$ is orthogonal to $h_i$ and contains the origin, $1 \leq i \leq d$. We assign a direction to each line $H_i$ such that $H_i$ increases in the halfspace bounded by $h_i$ which contains the cone $c$, i.e., $c$ lies in the positive side of $h_i$ w.r.t. to $H_i$. Furthermore, let $L$ be the line which contains the cone axis $l_c$. We assign to $L$ the same direction as to $l_c$. The edges of the $\theta$-graph will be computed by performing plane sweeps with the hyperplanes $h_i$ in the directions of $H_i$.

Let $q$ be any point of $\mathbb{R}^d$. We denote by $q_i$ the $i$th coordinate of $q$, $1 \leq i \leq d$. For $1 \leq i \leq d$, let $q'_i$ be the signed Euclidean distance between the origin and the orthogonal projection of $q$ onto $H_i$, where the sign corresponds to the direction of $H_i$. Finally, we

means that we do not necessarily guarantee that a single update is fast, but that we can bound the average time per operation over the entire sequence of updates and queries.

define $q'_{d+1}$ as the signed Euclidean distance between the origin and the orthogonal projection of $q$ onto $L$.

Using the above terminology, we can write the cone $c$ as

$$c = \{x \in \mathbb{R}^d : x'_i \geq 0,\ 1 \leq i \leq d\},$$

and for a point $q \in \mathbb{R}^d$, the translated cone $c(q)$ as

$$c(q) = \{x \in \mathbb{R}^d : x'_i \geq q'_i,\ 1 \leq i \leq d\}.$$

Let $S$ be a set of $n$ points in $\mathbb{R}^d$ and $0 < \theta \leq \pi$. Computing the outgoing edge for a point $s \in S$ in the $\theta$-graph $G_\theta(S)$ in the cone $c(s)$ is equivalent to finding a point $t$ with minimum $t'_{d+1}$-coordinate among the points $\{t \in S \setminus \{s\} : t'_i \geq s'_i,\ 1 \leq i \leq d\}$. In order to solve this problem for each point $s \in S$, we define a $d$-level data structure which has the form of a range tree [56]. For simplicity of the description we assume that for each two points $s, t \in S$, $s \neq t$ holds that $s'_i \neq t'_i$ for $1 \leq i \leq d$. The modification of the data structure for the case without this assumption is straightforward.

The first level of this data structure is a balanced binary search tree which stores the points $s \in S$ in its leaves, sorted by their $s'_1$-coordinates. For each node $v$ of this tree, let $S_v$ be the subset of $S$ stored in the subtree of $v$. Each node $v$ of this tree contains a pointer to the root of a balanced binary search tree which stores the points $s \in S_v$ in its leaves, sorted by their $s'_2$-coordinates, etc... At the $d$th level there is a balanced binary search tree which stores a subset of $S$, sorted by their $s'_d$-coordinates. At the $d$th level tree of this data structure for each node $v$, we store additionally the point of $S_v$ whose $s'_{d+1}$-coordinate is minimal.

Using this data structure we can compute the edges $\langle s, t\rangle \in G_\theta(S)$ such that $t \in c(s)$ in the following way. We initialize an empty set $M_1$ and follow the path in the first level search tree from the root to the leftmost leaf which stores a point $q \in S$ with $q'_1 \geq s'_1$. For each node $v$ on this path, if we move to the left child, then we insert the right child of $v$ into $M_1$. The final set $M_1$ contains $O(\log n)$ nodes such that

(*i*) $\bigcup_{v \in M_1} S_v = \{q \in S : q'_1 > s'_1\}$ and

(*ii*) $S_v \cap S_w = \emptyset$, for each $v, w \in M_1$, $v \neq w$.

Then we initialize a new empty set $M_2$ and for each node $v \in M_1$, we follow the path in the second level search tree, which stores the points of $S_v$, from the root to the leftmost leaf which stores a point $q \in S$ with $q'_2 \geq s'_2$. For each node $w$ on this path, if we move to the left child, then we insert the right child of $w$ into $M_2$, etc... At the $d$th level we initialize an empty set $M_d$ and for each node $v \in M_{d-1}$, we follow the path in the $d$th

level search tree, which stores the points of $S_v$, from the root to the leftmost leaf which stores a point $q \in S$ with $q'_d \geq s'_d$. For each node $w$ on this path, if we move to the left child, then we insert the right child of $w$ into $M_d$. Then the final set $M_d$ contain $O(\log^d n)$ nodes such that

(*i*) $\bigcup_{v \in M_d} S_v = \{q \in S : q'_i > s'_i,\ 1 \leq i \leq d\}$ $(= c(s) \cap S \setminus \{s\})$ and

(*ii*) $S_v \cap S_w = \emptyset$, for each $v, w \in M_d$, $v \neq w$.

The nodes contained in $M_d$ are called the *canonical nodes* of $s$. With each node $v$, in particular with each canonical node of $s$, we stored a point $t_v$ such that $(t_v)'_{d+1}$ is minimal in the subtree of $v$. Let $t$ be a point such that (*i*) $t = t_v$, for some canonical node $v \in M_d$ and (*ii*) $t'_{d+1} = \min\{(t_v)'_{d+1} : v \in M_d\}$. Then $\langle s, t \rangle$ is an edge in $G_\theta(S)$.

The space requirement of this $d$-level data structure is $O(n \log^{d-1} n)$. It can be maintained in $O(\log^d n)$ amortized time per insertion and deletion and it supports answering orthogonal range queries in $O(\log^d n + k)$ time, where $k$ is the number of reported points. For an exact analysis of this data structure we refer to Luecker [56] or to Mehlhorn [61]. The additional information – i.e., for each node $v$ of each $d$th level tree, the point of $S_v$ whose $s'_{d+1}$-coordinate is minimal – can be computed in $O(n \log^{d-1} n)$ time by a bottom-up procedure and it also can be maintained in $O(\log^d n)$ amortized time per insertion and deletion [8].

**Lemma 2.24** [8] *Let $S$ be a set of $n$ points in $\mathbb{R}^d$ and let $c \in C$. Using the above data structure, for a point $p \in \mathbb{R}^d$, we can compute in $O(\log^d n)$ time a point $q \in c(p) \cap S \setminus \{p\}$ for which $q'_{d+1}$ is minimal, or decide that such a point does not exist. The size of the data structure is $O(n \log^{d-1} n)$ and it can be constructed in $O(n \log^{d-1} n)$ time. Furthermore, it can be maintained in $O(\log^d n)$ amortized time per insertion and deletion.*

Hence, the graph $G_\theta(S)$ can be constructed in $O(|C| n \log^d n)$ time using $O(n \log^{d-1} n)$ space by building the above data structure for each cone $c \in C$ separately and by querying them with each point $s \in S$. We can save a factor of $\log n$ by taking advantage of the fact that all query points are known in advance. We again consider each cone $c \in C$ separately and for each $c$ we perform a plane sweep. We sort the points of $S$ by their $s'_1$-coordinates. Then we process them in decreasing order. All visited points are maintained in the $(d-1)$-dimensional version of the data structure of Lemma 2.24, using only the final $(d-1)$ coordinates $s_2, ..., s_d$ and the value of $s_{d+1}$ for each point $s \in S$. If the sweep plane encounters a new point $s$ then we query the data structure and find a point $t$ such that $t'_i \geq s'_i$ for all $2 \leq i \leq d$, and for which $t'_{d+1}$ is minimal. Since at this moment the data structure contains exactly the points $q$ of $S \setminus \{s\}$ with $q'_1 \geq s'_1$, we know that $t$ is in fact a point of $S \setminus \{s\}$ such that $t'_i \geq s'_i$ for all $1 \leq i \leq d$,

and for which $t'_{d+1}$ is minimal. Therefore, $\langle s,t\rangle$ is an edge in $G_\theta(S)$. After the query we insert $s$ into the data structure and the sweep plane moves to the next point.

Clearly, the above algorithm needs $O(n\log^{d-1} n)$ time in each cone $c \in C$ and it constructs the graph $G_\theta(S)$ correctly. Using the construction of Theorem 2.22 for $C$, we obtain the following.

**Theorem 2.25** [8] *Let $S \subset \mathbb{R}^d$ be a set of $n$ points and let $0 < \theta \leq \pi$ be a real number. Then the graph $G_\theta(S)$ can be constructed in $O((\frac{d^{3/2}}{\theta})^{d-1} n\log^{d-1} n)$ time using $O(n\log^{d-2} n)$ space.*

## 2.3 Conclusions and open problems

We have studied the two- and higher dimensional $\theta$-graph for a given point set. We have introduced the notion of weak spanner and examined the spanner and the weak spanner property of the $\theta$-graphs. We have proven that in the two-dimensional case the $\theta$-graph with outdegree four has the weak spanner property. Then we have generalized the degree four $\theta$-graph using $L_p$-distances to choose the neighbors of the points in the cones and we have classified these graphs according to whether they have the weak spanner property. Then we have shown that four is the lower bound for the outdegree even if we allow arbitrary convex distance functions to choose the neighbors of a point in the cones.

Then we have considered higher dimensional $\theta$-graphs. We have analysed different methods to compute a $\theta$-frame that have not yet been exactly analyzed to our knowledge and we have also presented an own method for the computation. We have shown that in the $d$-dimensional case, using our method, a $\theta$-frame can be constructed which contains $O(d^{-1/2}(\frac{d^{3/2}}{\theta})^{d-1})$ simplicial cones. This bound is significantly better than the bounds resulting from other methods.

Some exciting questions remain open. We are interested on that whether the graph $G_{\pi/2}(S)$ can be generalized in higher dimensions such that the number of cones of the according $\theta$-frame is $2^d$ and we obtain a weak spanner. The following generalization promises a result. For each $\epsilon = (\epsilon_1, ..., \epsilon_d) \in \{1,-1\}^d$, let $c_\epsilon$ be the simplicial cone $\{\sum_{i=1}^d \lambda_i\epsilon_i : \lambda_i \geq 0\}$. Let $C := \{c_\epsilon : \epsilon \in \{1,-1\}^d\}$ be the $\theta$-frame consisting of these $2^d$ cones. For each point $s \in S$ and each cone $c_\epsilon \in C$, the neighbor of $s$ in the cone $c_\epsilon(s)$ is chosen as a nearest point w.r.t. the $L_\infty$-distance. For a pair $s,t \in S$ of points, consider the path $P$ in this graph which is started at $s$ and follows the edge in the cone containing the point $t$. Then we can guarantee that $P$ is contained in the $d$-cube

$B := \{x \in \mathbb{R}^d : dist_\infty(t,x) \leq dist_\infty(t,s)\}$. The question is, whether the common $j$-faces, $1 \leq j < d$, of the cones of $C$ can be assigned to the cones uniquely and a kind of priority over the dimensions can be defined such that we also can guarantee that the path $P$ achieves the point $t$.

We are also interested on algorithms, that compute the $d$-dimensional $\theta$-graph, with lower space and time complexity than the presented algorithms. In particular, since the $\theta$-graph contains only $O(n)$ edges but the computation needs $O(n \log^{d-2} n)$ space, the reduction of the space complexity would be desired. Can be applied for this problem a similar technique which was used by Callahan and Kosaraju [23] to compute the $k$ nearest neighbors for each point $s \in S$ in $\mathbb{R}^d$ in $O(n \log n)$ time and $O(n)$ space?

# Chapter 3

# Fault tolerant spanners

The results of this chapter have been published in [57]. Fault tolerant spanners were introduced by Levcopoulos et al. [54]. Such spanners have the property that after deletion of at most $k$ edges or vertices, each pair of points in the remaining graph is still connected by a short path. For a precise definition we need the following notation. For a set $S \subset \mathbb{R}^d$ of $n$ points let $K_S$ denote the complete Euclidean graph with vertex set $S$. If $G = (S, E)$ is a graph and $E' \subseteq E$ then $G \setminus E'$ denotes the graph $G' = (S, E \setminus E')$. Similarly, if $S' \subseteq S$ then the graph $G \setminus S'$ is the graph with vertex set $S \setminus S'$ and edge set $\{(s_1, s_2) \in E : s_1, s_2 \in S \setminus S'\}$.

Let $f > 1$ be a real number and $k$ be an integer, $1 \leq k \leq n - 2$.

- A graph $G = (S, E)$ is called a *$k$-edge fault tolerant $f$-spanner* for $S$, or $(k, f)$-EFTS, if for each $E' \subset E$, $|E'| \leq k$, and for each pair $s, t$ of points of $S$, the graph $G \setminus E'$ contains an $st$-path whose length is at most $f$ times the length of a shortest $st$-path in the graph $K_S \setminus E'$.
- Similarly, $G = (S, E)$ is called a *$k$-vertex fault tolerant $f$-spanner* for $S$, or $(k, f)$-VFTS, if for each subset $S' \subset S$, $|S'| \leq k$, the graph $G \setminus S'$ is an $f$-spanner for $S \setminus S'$.

Levcopoulos et al. [54] presented an algorithm with running time $O(n \log n + k^2 n)$ which constructs a $(k, f)$-EFTS/VFTS with $O(k^2 n)$ edges for any real constant $f > 1$. The constants hidden in the $O$-notation are $(\frac{d}{f-1})^{O(d)}$ if $f \searrow 1$. They also showed that $\Omega(kn)$ is a lower bound on the number of edges in such spanners. This follows from the obvious fact that each $k$-edge/vertex fault tolerant spanner must be $k$-edge/vertex connected. Furthermore, they gave another algorithm with running time $O(n \log n + c^{k+1} n)$, for some constant $c$, which constructs a $(k, f)$-VFTS whose degree is bounded by $O(c^{k+1})$ and whose weight is bounded by $O(c^{k+1} w(MST))$, where $w(MST)$ is the weight of the minimum spanning tree of the given point set.

## New results

We consider directed and undirected fault tolerant spanners. Our first contribution is a construction of a $(k, f)$-VFTS with $O(kn)$ edges in $O(n \log^{d-1} n + kn \log\log n)$ time. Then we show that the same $k$-vertex fault tolerant spanner is also a $k$-edge fault tolerant spanner. Our bounds on the number of edges in fault tolerant spanners are optimal up to a constant factor and they improve the previous $O(k^2 n)$ bounds significantly. Furthermore, we construct a $k$-vertex fault tolerant spanner with $O(k^2 n)$ edges whose degree is bounded by $O(k^2)$ which also improves the previous $O(c^{k+1})$ bound.

Then we study Steinerized fault tolerant spanners that are motivated by the following. In the definition of $(k, f)$-EFTS we only require that after deletion of $k$ arbitrary edges $E'$ in the remaining graph each pair of points $s, t$ is still connected by a path whose length is at most $f$ times the length of the shortest $st$-path in $K_S \setminus E'$. Such a path can be arbitrarily long, much longer than $dist_2(s, t)$. To see this consider the following example. Let $r > 1$ be an arbitrarily large real number. Let $s, t \in S$ be two points such that $dist_2(s, t) = 1$ and let the remaining $n - 2$ points of $S$ be placed on the ellipsoid $\{x \in \mathbb{R}^d : dist_2(p, x) + dist_2(q, x) = r \cdot f\}$. Clearly, each $f$-spanner $G$ for $S$ contains the edge between $s$ and $t$, because each path which contains any third point $v \in S \setminus \{s, t\}$ has a length at least $r \cdot f$. Therefore, if the edge $(s, t) \in E'$ then the graph $G \setminus E'$ can not be an $f$-spanner for $S$. However, $G \setminus E'$ can contain a path satisfying the definition of the $k$-edge fault tolerance. In some applications one would need a stronger property. After deletion of $k$ edges an $st$-path would be desirable whose length is at most $f$ times $dist_2(s, t)$. In order to solve this problem we extend the original point set $S$ by Steiner points. Then we investigate the question how many Steiner points and how many edges do we need to satisfy the following natural but stronger condition of edge fault tolerance.

Let $f > 1$ be a real number and $k \in \mathbb{N}$.

- A graph $G = (V, E)$ with $S \subseteq V$ is called a *$k$-edge fault tolerant Steiner $f$-spanner* for $S$, or $(k, f)$-EFTSS, if for each $E' \subset E$, $|E'| \leq k$ and for each pair $s, t \in S$ of points, there is an $st$-path $P$ in $G \setminus E'$ such that $length(P) \leq f \cdot dist_2(s, t)$.
- Similarly, $G = (V, E)$ with $S \subseteq V$ is a *$k$-vertex fault tolerant Steiner $f$-spanner* for $S$, or $(k, f)$-VFTSS, if for each $V' \subset V$, $|V'| \leq k$ and for each pair $s, t \in S \setminus V'$ of points, there is an $st$-path $P$ in $G \setminus V'$ such that $length(P) \leq f \cdot dist_2(s, t)$.

To our knowledge, fault tolerant Steiner spanners have not been investigated before. First we show that for each set $S$ of $n$ points, $f > 1$ real constant, and $k \in \mathbb{N}$, a $(k, f)$-EFTSS/VFTSS for $S$ can be constructed which contains $O(kn)$ edges and $O(kn)$

Steiner points. Then we show that there is a set $S$ of $n$ points in $\mathbb{R}^d$, $d \geq 1$, such that for each $f > 1$ and $k \in \mathbb{N}$, each $(k, f)$-EFTSS for $S$ contains $\Omega(kn)$ edges and $\Omega(kn)$ Steiner points.

## 3.1 A $k$-vertex fault tolerant $f$-spanner with $O(kn)$ edges

The construction of a $k$-vertex fault tolerant $f$-spanner for a set $S$ of $n$ points in $\mathbb{R}^d$ with $O(kn)$ edges is based on a generalization of the $\theta$-graph [82, 28, 49, 50, 71, 8, 54]. First we introduce the notion of *ith order $\theta$-graph* of the point set $S$, for $1 \leq i \leq n-1$. Then we prove that for appropriate $\theta$, the $(k+1)$th order $\theta$-graph is a $k$-vertex fault tolerant spanner for the given set of points. Finally, we show how this graph can be computed efficiently.

### 3.1.1 The $i$th order $\theta$-graph

Let $0 < \theta \leq \pi$ be an angle and $C$ be a corresponding $\theta$-frame. We use the notations of the previous chapter. For a simplicial cone $c \in C$ and for a point $p \in \mathbb{R}^d$, $l_c$ denotes the cone axis of $c$, and $c(p)$ and $l_c(p)$ denote the translated cone $\{x+p : x \in c\}$ and the translated cone axis $\{x+p : x \in l_c\}$, respectively. For $p, q \in \mathbb{R}^d$ and $c \in C$ such that $q \in c(p)$, $dist_c(p, q)$ is the Euclidean distance between $p$ and the orthogonal projection of $q$ to $l_c(p)$.

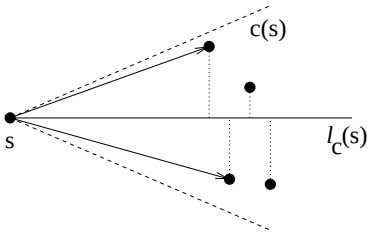


Figure 3.1: The neighbors of a point $s \in S$ in the two-dimensional second order $\theta$-graph, in a cone $c$. The boundary of $c$ is drawn dashed and the projection of the points onto $l_c$ is illustrated by dotted lines.

Now we define the *ith order $\theta$-graph* $G_{\theta,i}(S)$ for a set $S$ of $n$ points in $\mathbb{R}^d$ and for an integer $1 \leq i \leq n-1$ as follows. For each point $s \in S$ and each cone $c \in C$ let $S_{c(s)} := c(s) \cap S \setminus \{s\}$, i.e., $S_{c(s)}$ is the set of points of $S \setminus \{s\}$ that are contained in the cone $c(s)$. Let $N_{i,c}(s) \subseteq S_{c(s)}$ be the set of the $\min(i, |S_{c(s)}|)$-nearest neighbors of $s$ in

the cone $c(s)$ w.r.t. the distance $dist_c$, i.e., for each $s' \in N_{i,c}(s)$ and $s'' \in S_{c(s)} \setminus N_{i,c}(s)$ holds that $dist_c(s,s') \leq dist_c(s,s'')$. Let $G_{\theta,i}(S)$ be the directed graph with vertex set $S$ such that for each point $s \in S$ and each cone $c \in C$ there is a directed edge $\langle s, s' \rangle$ to each point $s' \in N_{i,c}(s)$. Figure 3.1 shows an example for the neighbors of a point $s \in S$ in $G_{\theta,2}(S)$ in a cone $c$. Note that the first order $\theta$-graph corresponds to the $\theta$-graph as defined in the previous chapter.

### 3.1.2 The vertex fault tolerant spanner property

**Theorem 3.1** *Let $S \subset \mathbb{R}^d$ be a set of $n$ points. Let $0 < \theta < \pi/3$ be a real and $1 \leq k \leq n-2$ an integer number. Then the graph $G_{\theta,k+1}(S)$ is a directed $(k, \frac{1}{1-2\sin(\theta/2)})$-VFTS for $S$ and it contains $O(|C|kn)$ edges.*

**Proof:** Let $S' \subset S$ be a set of at most $k$ points. We show that for each pair $s, t \in S \setminus S'$ of points there is a (directed) $st$-path $P$ in $G_{\theta,k+1}(S) \setminus S'$ such that the length of $P$ is at most $\frac{1}{1-2\sin(\theta/2)}\, dist_2(s,t)$. The proof is similar to the proof of Ruppert and Seidel [71]. Consider the path constructed in the following way. Let $s_0 := s$, $i := 0$ and let $P$ contain the single point $s_0$. If the edge $\langle s_i, t \rangle$ is present in the graph $G_{\theta,k+1}(S) \setminus S'$ then add the vertex $t$ to $P$ and stop. Otherwise, let $c(s_i)$ be the cone which contains $t$. Choose an arbitrary point $s_{i+1} \in N_{k+1,c}(s_i)$ as the next vertex of the path $P$ and repeat the procedure with $s_{i+1}$.

Consider the $i$th iteration of the above algorithm. If $\langle s_i, t \rangle \in G_{\theta,k+1}(S)$ then the algorithm terminates. Otherwise, if $\langle s_i, t \rangle \notin G_{\theta,k+1}(S)$ then by definition the cone $c(s_i)$ contains at least $k+1$ points that are not further from $s_i$ than $t$ w.r.t. the distance $dist_c$. Hence, in the graph $G_{\theta,k+1}(S)$ the point $s_i$ has $k+1$ neighbors in $c(s_i)$ and, therefore, in the graph $G_{\theta,k+1}(S) \setminus S'$ it has at least one neighbor in $c(s_i)$. Consequently, the algorithm is well defined in each step. Furthermore, Lemma 2.11 implies that $dist_2(s_{i+1}, t) < dist_2(s_i, t)$ and hence, each point is contained in $P$ at most once. Therefore, the algorithm terminates and finds an $st$-path $P$ in $G_{\theta,k+1}(S) \setminus S'$. The bound on the length of $P$ follows by applying Lemma 2.11 iteratively, in the same way as in [71]: Let $s_0, ..., s_m$ be the vertices on $P$, $s_0 = s$ and $s_m = t$. Then

$$\sum_{0 \leq i < m} dist_2(s_{i+1}, t) \leq \sum_{0 \leq i < m} \Big( dist_2(s_i, t) - (1 - 2\sin(\theta/2))\, dist_2(s_i, s_{i+1}) \Big).$$

Rearranging the sum we get

$$\begin{aligned} \sum_{0 \leq i < m} dist_2(s_i, s_{i+1}) &\leq \frac{1}{1-2\sin(\theta/2)} \sum_{0 \leq i < m} \Big( dist_2(s_i, t) - dist_2(s_{i+1}, t) \Big) \\ &= \frac{1}{1-2\sin(\theta/2)}\, dist_2(s_0, t). \end{aligned}$$

Hence, the graph $G_{\theta,k+1}(S)$ is a $(k, \frac{1}{1-2\sin(\theta/2)})$-VFTS for $S$. Clearly, it contains $O(|C|kn)$ edges, where, by Theorem 2.22, $|C| = O((\frac{d^{3/2}}{\theta})^{d-1})$. □

### 3.1.3 Computation

The graph $G_{\theta,k+1}(S)$ can be constructed in $O(|C|(n\log^{d-1} n + kn\log\log n))$ time using the algorithm of Levcopoulos et al. [54] which we describe in this subsection. The algorithm in [54] was developed to compute the so-called *strong approximate neighbors* and it is a modification of that of Arya et al. [8] which we have explained in Subsection 2.2.3.

**Theorem 3.2** *Let $S \subset \mathbb{R}^d$ be a set of $n$ points. Let $0 < \theta < \pi/3$ be a real and $1 \leq k \leq n-2$ an integer number. Then the graph $G_{\theta,k+1}(S)$ can be computed in $O(|C|(n\log^{d-1} n + kn\log\log n))$ time.*

**Proof:** We use the notation of Subsection 2.2.3. Let $c \in C$ be a cone. Then $h_1, ..., h_d$ denote the hyperplanes that bound the halfspaces defining $c$, and $H_1, ..., H_d$ be the lines such that $H_i$ is orthogonal to $h_i$ and contains the origin, $1 \leq i \leq d$. The line $H_i$ is directed such that it increases in the halfspace bounded by $h_i$ which contains the cone $c$. For $q \in \mathbb{R}^d$, $q'_i$ is defined as the signed Euclidean distance between the origin and the orthogonal projection of $q$ onto $H_i$, where the sign is according to the direction of $H_i$, $1 \leq i \leq d$. Furthermore, let $q'_{d+1}$ be the signed Euclidean distance between the origin and the orthogonal projection of $q$ onto the line which contains $l_c$. The sign is positive if the projection is contained in $l_c$.

Computing the outgoing edges for a point $s \in S$ in the graph $G_{\theta,k+1}(S)$ in the cone $c(s)$ is equivalent to finding the $k^* := \min(k+1, |S_{c(s)}|)$ points $t^1, ..., t^{k^*}$ with smallest $t'_{d+1}$-coordinate among the points $\{t \in S \setminus \{s\} : t'_i \geq s'_i,\ 1 \leq i \leq d\}$. In order to solve this problem we process in the same manner as in Subsection 2.2.3. We first present the modification of the $d$-level data structure of Lemma 2.24 to obtain a data structure which allows us for a point $p \in \mathbb{R}^d$, to compute in $O(\log^d n + k\log\log n)$ time the $k^*$ points $q^1, ..., q^{k^*} \in S_{c(p)}$ for them $q'_{d+1}$ are the smallest.

The only modification of the $d$-level data structure of Lemma 2.24 is that in the $d$th level trees, for each node $v$, we store additionally the $\min(k+1, |S_v|)$ points $q \in S_v$ in a sorted list with smallest $q'_{d+1}$-coordinates. Then for a query point $p \in \mathbb{R}^d$, we first determine the set of $O(\log^d n)$ canonical nodes $M_d$ in the same way as in the data structure of Lemma 2.24. Then we initialize an empty heap $H$ and for each $v \in M_d$, we put the first element of the sorted list stored at $v$ into $H$. Let $q^1$ be the point on the top of the heap $H$. It has a minimal $q'_{d+1}$-coordinate among the points of $S_{c(p)}$. We

remove $q^1$ from $H$ and insert the next point – if any – of the list, which contains $q^1$, into the heap. Let $q^2$ be the point on the top of $H$ after this update. Clearly, $q^2$ has a minimal $q'_{d+1}$-coordinate among the points of $S_{c(p)} \setminus \{q^1\}$. We remove $q^2$ from $H$ and insert the next point – if any – of the list, which contains $q^2$, into the heap. We repeat this until $k+1$ points are removed or the heap $H$ becomes empty. Implementing $H$ by a Fibonacci heap [43], we can build it in $O(|M_d|)$ time, we can insert a new element in $O(1)$ and we can delete the top element in $O(\log |M_d|)$ amortized time. Since the set $M_d$ has a size $O(\log^d n)$, we can find the $k^*$ points with smallest $q'_{d+1}$-coordinates among the points $S_{c(p)}$ in $O(\log^d n + k \log\log n)$ time.

The above data structure can be maintained in $O(\log n)$ time per insertion. This result can be obtained by exploiting its similarity to a $(d+1)$-dimensional range tree and applying the insertion-only algorithm of Mehlhorn and Näher [62] that uses dynamic fractional cascading.

**Lemma 3.3** [54] *Let $S$ be a set of $n$ points in $\mathbb{R}^d$ and let $c \in C$. Using the above data structure, for a point $p \in \mathbb{R}^d$, we can compute in $O(\log^d n + k \log\log n)$ time the $k^* = \min(k+1, |S_{c(p)}|)$ points $q^1, ..., q^{k^*} \in S_{c(p)}$ with smallest $q'_{d+1}$-coordinates. The size of the data structure is $O(kn \log^{d-1} n)$ and it can be maintained in $O(\log^d n)$ time per insertion.*

Hence, the graph $G_{\theta,k+1}(S)$ can be constructed in $O(|C|(n \log^d n + kn \log\log n))$ time using $O(kn \log^{d-1} n)$ space by building the above data structure for each cone $c \in C$ separately and by querying them with each point $s \in S$. We again can save a factor of $\log n$ by performing a plane sweep for each $c \in C$. We sort the points of $S$ by their $s'_1$-coordinates and process them in decreasing order. All visited points are maintained in the $(d-1)$-dimensional version of the data structure of Lemma 3.3, using only the final $(d-1)$ coordinates $s_2, ..., s_d$ and the value of $s_{d+1}$ for each point $s \in S$.

If the sweep plane encounters a new point $s$ then we query the data structure and find $k^*$ points $t^1, ..., t^{k^*}$ such that $t'_i \geq s'_i$ for all $2 \leq i \leq d$, and for them $t'_{d+1}$ are the smallest. Since at this moment the data structure contains exactly the points $q$ of $S \setminus \{s\}$ with $q'_1 \geq s'_1$, we know that $t^1, ..., t^{k^*}$ are in fact points of $S \setminus \{s\}$ such that $t'_i \geq s'_i$ for all $1 \leq i \leq d$, and for them $t'_{d+1}$ are the smallest. Therefore, $\langle s, t^1 \rangle, ..., \langle s, t^{k^*} \rangle$, are edges in $G_{\theta,k+1}(S)$. After the query we insert $s$ into the data structure and the sweep plane moves to the next point. Clearly, the above algorithm needs $O(n \log^{d-1} n + kn \log\log n)$ time in each cone $c \in C$ and it constructs the graph $G_{\theta,k+1}(S)$ correctly. □

**Corollary 3.4** *Let $S$ be a set of $n$ points in $\mathbb{R}^d$, $f > 1$ a real constant, and $k$ an integer, $1 \leq k \leq n-2$. Then there is a $(k,f)$-VFTS for $S$ with $O(kn)$ edges. Such a spanner can be constructed in $O(n \log^{d-1} n + kn \log\log n)$ time.*

**Proof:** We set $\theta$ such that $f \geq \frac{1}{1-2\sin(\theta/2)}$ and $0 < \theta < \pi/3$ and construct $G_{\theta,k+1}(S)$. If $f \searrow 1$ then the constant factors hidden in the $O$-calculus are $(\frac{d^{3/2}}{f-1})^{d-1}$.

## 3.2 $k$-edge fault tolerant $f$-spanners

Levcopoulos et al. [54] claimed that any $(k,f)$-VFTS is also a $(k,f)$-EFTS. We give our own proof of this fact. The proof is simple and holds also for directed spanners.

**Theorem 3.5** *Let $S$ be a set of $n$ points in $\mathbb{R}^d$, $f > 1$ a real constant, and $k$ an integer, $1 \leq k \leq n-2$. Then every (directed) $(k,f)$-VFTS for $S$ is also a (directed) $(k,f)$-EFTS for $S$.*

**Proof:** Let $G = (S, E)$ be a (directed) $(k,f)$-VFTS for $S$. Let $E' \subset E$ be a set of at most $k$ edges. Consider two arbitrary points $s, t \in S$. Let $P^*$ be the shortest (directed) $st$-path in $K_S \setminus E'$. Such a path exists, since the set of $st$-paths in $K_S \setminus E'$ is not empty. It contains, for example, at least one of the $n-2$ paths in $K_S$ of two edges $P_v = s, v, t$, for $v \in S \setminus \{s,t\}$, or the immediate path $P_0 = s, t$, because at least one of them is distinct from $E'$.

We have to show that there is a (directed) $st$-path $P$ in $G \setminus E'$ such that the length of $P$ is at most $f$ times the length of $P^*$. The edges $e$ in $P^*$ that are contained in $G$ will also be contained in $P$. Consider an edge $(u,v)$ ($\langle u,v\rangle$ in the directed case) in $P^*$ which is not contained in $G$. We show that this edge can be substituted by a $uv$-path $P_{uv}$ in $G \setminus E'$ such that $length(P_{uv}) \leq f \cdot dist_2(u,v)$: For each edge $e' \in E'$ (for each $e' \in E' \setminus \{\langle v,u\rangle\}$ in the directed case) we fix one of its endpoints $s_{e'}$ such that $s_{e'} \in S \setminus \{u,v\}$. Let $S'_{uv} := \{s_{e'} : e' \in E'\}$ ($S'_{uv} := \{s_{e'} : e' \in E' \setminus \{\langle v,u\rangle\}$ in the directed case). Note that $S'_{uv} \leq |E'| \leq k$. Since $G$ is a (directed) $(k,f)$-VFTS for $S$, there is a (directed) $uv$-path $P_{uv}$ in $G \setminus S'_{uv}$ such that $P_{uv}$ does not contain any edge of $E'$ and $length(P_{uv}) \leq f \cdot dist_2(u,v)$. The desired $st$-path $P$ is composed of the edges of $P^* \cap G$ and the $uv$-paths for the edges $(u,v) \in P^* \setminus G$ ($\langle u,v\rangle \in P^* \setminus G$ in the directed case). Clearly, $length(P) \leq f \cdot length(P^*)$. □

This, together with Corollary 3.4, leads to

**Corollary 3.6** *Let $S$ be a set of $n$ points in $\mathbb{R}^d$, $f > 1$ a real constant, and $k$ an integer, $1 \leq k \leq n-2$. Then there is a (directed) $(k,f)$-EFTS for $S$ with $O(kn)$ edges. Such a spanner can be constructed in $O(n \log^{d-1} n + kn \log\log n)$ time.*

The proof of Theorem 3.5 implies also the following for directed graphs:

**Theorem 3.7** *Let $S$ be a set of $n$ points in $\mathbb{R}^d$, $f > 1$ a real constant, and $k$ an integer, $1 \leq k \leq n-2$. Let $G = (V, E)$ be a directed $(k, f)$-VFTS for $S$. Let $E' \subset E$ be a set of at most $k$ edges and let $E'' := \{\langle v, u\rangle : \langle u, v\rangle \in E'\}$. Then for each pair $s, t \in S$ of points the graph $G \setminus (E' \cup E'')$ contains an st-path $P$ such that the length of $P$ is at most $f$ times the length of the shortest st-path in $K_S \setminus (E' \cup E'')$.*

## 3.3 A $k$-vertex fault tolerant $f$-spanner with degree $O(k^2)$

We now turn to the problem of constructing fault tolerant spanners with bounded degree. We proceed similar to the method in [6] which constructs a spanner with constant degree. However, we must take much more care, because of the fault tolerant property and the goal of keeping the number of edges small. We have shown that for any real constant $f > 1$ we can construct a directed $(k, f)$-VFTS/EFTS for $S$ whose outdegree is $O(k)$. In this section we give a method to construct a $(k, f)$-VFTS whose degree is $O(k^2)$ from a directed $(k, f^{1/3})$-VFTS whose outdegree is $O(k)$.

### 3.3.1 $k$-vertex fault tolerant single sink spanners

In order to show this construction of a $(k, f)$-VFTS with $O(k^2)$ edges, we need the notion of *$k$-vertex fault tolerant single sink spanner*. This is a generalization of *single sink spanners* introduced in [6]. Let $V$ be a set of $m$ points in $\mathbb{R}^d$, $v \in V$, $\hat{f} > 1$ a real constant, and $k$ an integer, $1 \leq k \leq m-2$. A directed graph $G = (V, E)$ is a *$k$-vertex fault tolerant $v$-single sink $\hat{f}$-spanner*, or $(k, \hat{f}, v)$-VFTssS for $V$ if for each $u \in V \setminus \{v\}$ and each $V' \subseteq V \setminus \{v, u\}$, $|V'| \leq k$, there is an $\hat{f}$-spanner path in $G \setminus V'$ from $u$ to $v$.

Now let $V$ be a set of $m$ points in $\mathbb{R}^d$, $v \in V$ a fixed point, $1 \leq i \leq m-1$ an integer, $\theta$ an angle, $0 < \theta < \pi/3$, and $C$ a $\theta$-frame. We define a directed graph $\hat{G}_{v,\theta,i}(V) = (V, E)$ whose edges are directed straight line segments between points of $V$ as follows. First we partition the set $V$ in clusters such that each cluster contains at most $i$ points. Then we build a tree-like structure based on these clusters. For the clustering we use the cones of $C$. Now we describe this procedure more precisely.

First we create a cluster $cl(\{v\})$ containing the unique point $v$. For each cluster that we create, we choose a point as the *representative* of the cluster. The representative of $cl(\{v\})$ is $v$. The clustering of the set $V \setminus \{v\}$ is recursive. The recursion stops if $V \setminus \{v\}$ is the empty set. Otherwise, we do the following. For each cone $c \in C$ let $V_{c(v)}$ be the set of points of $V \setminus \{v\}$ contained in $c$. If a point is contained in more than one cone then

assign the point only to one of them. If one cone, say $c$, contains more than $m/2$ points, then partition the points of $V_{c(v)}$ arbitrarily into two sets $V^1_{c(v)}$ and $V^2_{c(v)}$ both having at most $m/2$ points. For each nonempty set $V_{c(v)}$, $c \in C$ (or in the case that $V_{c(v)}$ had to be partitioned, for each $V^1_{c(v)}$ and $V^2_{c(v)}$), let $N_{i,c}(v) \subseteq V_{c(s)}$ be the set of the $\min(i, |V_{c(v)}|)$-nearest neighbors of $v$ in $V_{c(v)}$ w.r.t. the distance $dist_c$. The points contained in the same $N_{i,c}(v)$ define a new cluster $cl(N_{i,c}(v))$. Note that in this way we obtain at most $|C|+1$ new clusters. We say that these clusters are the *children* of $cl(\{v\})$ and $cl(\{v\})$ is the *parent* of these clusters. For each new cluster $cl(N_{i,c}(v))$ we choose a representative $u_c \in N_{i,c}(v)$ such that $dist_c(v, u_c) = \max\{dist_c(v, u) : u \in N_{i,c}(v)\}$. Then, for each set $V_{c(v)}$, $c \in C$ (and $V^1_{c(v)}$, $V^2_{c(v)}$ if exist), we recursively cluster $V_{c(v)} \setminus N_{i,c}(v)$ using the cones around $u_c$.

After the clustering is done, for each cluster $cl \neq cl(\{v\})$ we add an edge in $\hat{G}_{v,\theta,i}(V)$ from each point $u \in cl$ to each point $w$ of the parent cluster of $cl$. Figure 3.2 shows an example for $\hat{G}_{v,\theta,i}(V)$. The dotted lines represent the boundaries of the cones at the representatives of the clusters.

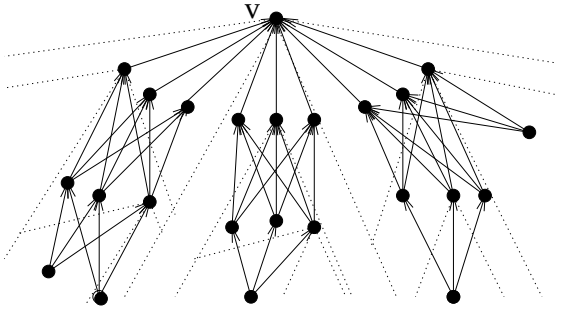


Figure 3.2: The directed graph $\hat{G}_{v,\theta,3}(V)$ for a point set $V$ in $\mathbb{R}^2$.

**Lemma 3.8** *Let $V$ be a set of $m$ points in $\mathbb{R}^d$, $v \in V$ a fixed point, and $1 \leq k \leq m-2$ an integer number. Let $0 < \theta < \pi/3$ be an angle and $C$ be a $\theta$-frame. Then the graph $\hat{G}_{v,\theta,k+1}(V)$ is a $(k, (\frac{1}{1-2\sin(\theta/2)})^2, v)$-VFTssS for $V$. Its degree is bounded by $O(|C|k)$ and it can be computed in $O(|C|(m \log m + km))$ time.*

**Proof:** For each point $u \in V$ let $cl(u)$ denote the cluster containing it. The outdegree of each point $u \in V \setminus \{v\}$ in $\hat{G}_{v,\theta,k+1}(V)$ is bounded by $k+1$, because each point $u$ has only edges to the points contained in the parent cluster of $cl(u)$ and the number of points in each cluster is bounded by $k+1$. (Each internal cluster – i.e., a cluster which is different from $cl(v)$ and has at least one child – contains exactly $k+1$ points). Since each cluster has at most $|C|+1$ children, the indegree of the points is bounded by $(|C|+1)(k+1)$. The bound for the construction time follows from the fact that the recursion has depth $O(\log m)$.

Now we prove the fault tolerant single sink spanner property. Consider an arbitrary point $u \in V \setminus \{v\}$. Let $P_0 := u_0, ..., u_l$, $u_0 = u$ and $u_l = v$, be the unique path from $u$ to $v$ in $\hat{G}_{v,\theta,k+1}(V)$ such that each internal vertex $u_i$, $1 \leq i < l$, is the representative of a cluster. Note that $l = O(\log m)$. The length of $P_0$ is at most $\frac{1}{1-2\sin(\theta/2)}\ dist_2(u,v)$. If the edge $\langle u, v\rangle \in \hat{G}_{v,\theta,k+1}(V)$, this claim holds trivially, otherwise, it follows by applying Lemma 2.11 iteratively to the triples $u_{i+1}, u_i, u_{i-1}$, $i = 1, ..., l-1$, in the same way as in the proof of Theorem 3.1.

Now let $V' \subset V \setminus \{u, v\}$ be a set of at most $k$ points. We show that there is a $uv$-path $P$ in $\hat{G}_{v,\theta,k+1}(V) \setminus V'$ such that $length(P) \leq \frac{1}{1-2\sin(\theta/2)}\ length(P_0)$. This will imply the desired stretch factor $(\frac{1}{1-2\sin(\theta/2)})^2$. Let $P$ be the path constructed as follows. Let $v_0 := u$, $i := 0$ and let $P$ contain the single point $v_0$. If $v_i = v$ then stop. Otherwise, let $v_{i+1}$ be an arbitrary point with $\langle v_i, v_{i+1}\rangle \in \hat{G}_{v,\theta,k+1}(V) \setminus V'$. Add the vertex $v_{i+1}$ to $P$ and repeat the procedure with $v_{i+1}$.

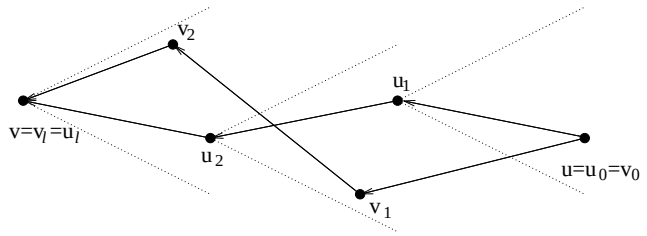


Figure 3.3: The paths $P_0 := u_0, ..., u_l$ and $P := v_0, ..., v_l$. The dotted lines show the cone boundaries.

The above algorithm is well defined in each step. To see this, consider the $i$th iteration. If the cluster $cl(v)$ is the parent of $cl(v_i)$ then the algorithm chooses $v$ as $v_{i+1}$ and terminates. Otherwise, the parent of $cl(v_i)$ contains $k+1$ points and, hence, at least one point disjoint from $V'$. The algorithm chooses such a point as $v_{i+1}$. Clearly, the algorithm terminates after $l = O(\log m)$ steps and constructs a $uv$-path $P = v_0, ..., v_l$ (Figure 3.3) with

$$\begin{aligned}
length(P) &= \sum_{0\leq i<l} dist_2(v_i, v_{i+1}) \\
&\leq \sum_{0\leq i<l} \Big(dist_2(v_i, u_{i+1}) + dist_2(u_{i+1}, v_{i+1})\Big) \qquad (3.1)\\
&= \sum_{0\leq i<l} \Big(dist_2(v_i, u_{i+1}) + dist_2(u_i, v_i)\Big) + \underbrace{dist_2(u_l, v_l)}_{=0} - \underbrace{dist_2(u_0, v_0)}_{=0} \\
&\leq \sum_{0\leq i<l} \tfrac{1}{1-2\sin(\theta/2)}\ dist_2(u_i, u_{i+1}) \qquad (3.2)\\
&= \tfrac{1}{1-2\sin(\theta/2)}\ length(P_0).
\end{aligned}$$

(3.1) holds because of the triangle inequality and (3.2) follows by applying Lemma 2.11 to the triples $u_{i+1}, v_i, u_i$, $i = 0, ..., l-1$:

$$\begin{aligned} dist_2(v_i, u_i) &\leq dist_2(u_{i+1}, u_i) - (1 - 2\sin(\theta/2))\ dist_2(u_{i+1}, v_i) \quad \text{and} \\ dist_2(u_i, u_i) &\leq dist_2(v_i, u_i) - (1 - 2\sin(\theta/2))\ dist_2(v_i, u_i). \end{aligned}$$

By addition and rearrangement of these inequalities we obtain that

$$dist_2(u_{i+1}, v_i) + dist_2(v_i, u_i) \leq \tfrac{1}{1-2\sin(\theta/2)}\ dist_2(u_{i+1}, u_i).$$

Hence, the claimed stretch factor of $\hat{G}_{v,\theta,k+1}(V)$ follows. □

### 3.3.2 A bounded degree $k$-vetrex fault tolerant $f$-spanner

**Theorem 3.9** *Let $S$ be a set of $n$ points in $\mathbb{R}^d$, $f > 1$ a real constant, and $k$ an integer, $1 \leq k \leq n-2$. Then there is a $(k, f)$-VFTS $G$ for $S$ whose degree is bounded by $O(k^2)$. The total number of edges in $G$ is $O(k^2 n)$ and $G$ can be constructed in $O(n \log^{d-1} n + kn \log n + k^2 n)$ time.*

**Proof:** Let $G_0$ be a directed $(k, f^{1/3})$-VFTS for $S$ whose outdegree is $O(k)$, for example, let $G_0$ be the $(k+1)$th order $\theta$-graph $G_{\theta,k+1}(S)$ with $f^{1/3} \geq \frac{1}{1-2\sin(\theta/2)}$. For each point $s \in S$ let $N_{in}(s) := \{u \in S : \langle u, s\rangle \in G_0\}$. Let $G$ be the directed graph with vertex set $S$ which is created such that for each $s \in S$ we construct $\hat{G}_{s,\theta,k+1}(N_{in}(s) \cup \{s\})$ and we add the edges of $\hat{G}_{s,\theta,k+1}(N_{in}(s) \cup \{s\})$ to $G$.

We can bound the degree of $G$ as follows. For each $s \in S$, the graph $G$ contains the edges of $\hat{G}_{s,\theta,k+1}(N_{in}(s) \cup \{s\})$. In this VFTssS each vertex $u$ has an in- and outdegree $O(k)$. Now for each $u \in S$, we have to count the graphs $\hat{G}_{s,\theta,k+1}(N_{in}(s) \cup \{s\})$, $s \in S$, that contain $u$. Clearly, the number of such graphs is equal to one plus the outdegree of $u$ in $G_0$, which is $O(k)$. Therefore, the degree of each $u \in S$ in $G$ is $O(k^2)$.

Now we show that $G$ is a $(k, f)$-VFTS for $S$. Let $S' \subset S$, $|S'| \leq k$. Consider two arbitrary points $s, t \in S \setminus S'$. Since $G_0$ is a $(k, f^{1/3})$-VFTS for $S$, there is an $f^{1/3}$-spanner path $P_0$ in $G_0 \setminus S'$ between $s$ and $t$. Furthermore, for each edge $\langle u, v\rangle \in P_0$, there is an $f^{2/3}$-spanner path $P_{uv}$ in $G \setminus S'$, because $G$ contains all edges of the graph $\hat{G}_{v,\theta,k+1}(N_{in}(v) \cup \{v\})$ which is, by Lemma 3.8, a $(k, f^{2/3}, v)$-VFTssS for $N_{in}(v) \cup \{v\}$. Therefore, the path $P := \cup_{\langle u,v\rangle \in P_0} P_{uv}$ is contained in $G \setminus S'$ and $P$ is a $f$-spanner path between $s$ and $t$. □

## 3.4 Fault tolerant spanners with Steiner points

Now we describe a simple method, which for an arbitrary set $S$ of $n$ points in $\mathbb{R}^d$, $f > 1$, and $k \in \mathbb{N}$, constructs a $(k, f)$-EFTSS and $(k, f)$-VFTSS for $S$ with $O(kn)$ edges and $kn$ Steiner points. Then we prove the surprising fact that these upper bounds on the number of edges and on the number Steiner points in a $(k, f)$-EFTSS are optimal up to constant factors.

### 3.4.1 Upper bounds

**Theorem 3.10** *Let $S \subset \mathbb{R}^d$ be a set of $n$ points, $k \in \mathbb{N}$, and let $f > 1$ a real constant. Then there is a $(k, f)$-EFTSS and a $(k, f)$-VFTSS $G$ for $S$ with $kn$ Steiner points and $O(kn)$ edges.*

**Proof:** Assume that the Euclidean distance between the closest pair of $S$ is one. Otherwise, we scale $S$ accordingly. Let $\epsilon$ be a real number such that $0 < \epsilon \leq (f-1)/3$. Let $f^* = f - 2\epsilon$ and let $G^* = (S, E^*)$ be an $f^*$-spanner for $S$ with $O(n)$ edges. $G^*$ can be computed, for example, using the method described in [23] or in [71]. We construct from $G^*$ a $(k, f)$-EFTSS/VFTSS $G$ for $S$ in the following way. Let $o \in \mathbb{R}^d$ be a fixed point and let $D := \{x \in \mathbb{R}^d : dist_2(o, x) = \epsilon\}$ be the sphere with radius $\epsilon$ whose center is $o$. Let $p^1, ..., p^k$ be $k$ distinct points on $D$. (If $d = 1$, let $p^1, ..., p^k$ be $k$ distinct points such that $0 < dist_2(o, p_i) \leq \epsilon$, $1 \leq i \leq k$.) For each point $s \in S$, translate the sphere $D$ and the points $p^1, ..., p^k$ on it such that $s$ becomes the center of the sphere. Let $s^1, ..., s^k$ denote the translated points around $s$. We construct the graph $G = (V, E)$ such that

$$\begin{aligned} V &:= \{s, s^1, ..., s^k : s \in S\} \qquad \text{and} \\ E &:= \{(s,t) : (s,t) \in E^*\} \cup \{(s, s^i) : s \in S, 1 \leq i \leq k\} \cup \\ &\quad \{(s^i, t^i) : (s,t) \in E^*, 1 \leq i \leq k\}. \end{aligned}$$

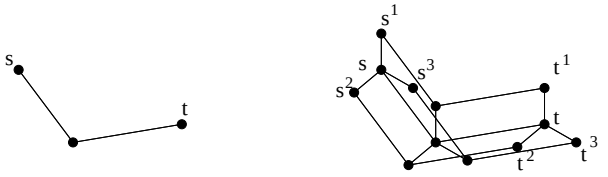


Figure 3.4: Example for the graphs $G^*$ and $G$ for $k = 3, d = 2$.

Clearly, the graph $G$ has $kn$ Steiner points and $O(kn)$ edges. It is obvious that $G$ is a $k$-EFTSS and $k$-VFTSS for $S$ because for each pair of points $s, t \in S$ and for

each $f^*$-spanner path $P^* = s, s_1, ..., s_{l-1}, t$ in $G^*$ between $s$ and $t$, there are $k+1$ edge disjoint and up to the endpoints vertex disjoint $st$-paths $P^0 = s, s_1, ..., s_{l-1}, t$ and $P^i = s, s^i, s^i_1, ..., s^i_{l-1}, t^i, t$, $1 \leq i \leq k$, in $G$ whose length is at most

$$length(P^*) + 2\epsilon \;\leq\; f^* \cdot dist_2(s,t) + 2\epsilon \;\leq\; f \cdot dist_2(s,t).$$

Figure 3.4 shows an example. □

### 3.4.2 Lower bounds

Now we prove that the upper bounds on the number of edges and Steiner points showed in the previus subsection are optimal up to constant factors.

**Theorem 3.11** *For each $k \in \mathbb{N}$, $n \geq 2$, and $f > 1$, there exists a set $S \subset \mathbb{R}^d$ of $n$ points such that each $(k,f)$-EFTSS for $S$ contains at least $k\lfloor n/2 \rfloor = \Omega(kn)$ Steiner points and $(2k+1)\lfloor n/2 \rfloor + (k+1)(\lceil n/2 \rceil - 1) = \Omega(kn)$ edges.*

**Proof:** We give an example for a set $S$ of $n$ points in the plane for which we show that each $(k,f)$-EFTSS for $S$ contains at least $k\lfloor n/2 \rfloor$ Steiner points and $(2k+1)\lfloor n/2 \rfloor + (k+1)(\lceil n/2 \rceil - 1) \approx \frac{3}{2}kn$ edges. For two points $p, q \in \mathbb{R}^d$ let

$$el(p,q) := \{x \in \mathbb{R}^d : dist_2(p,x) + dist_2(q,x) \leq f \cdot dist_2(p,q)\}.$$

If $s, t$ are two points in $S$ and $G$ is a $(k,f)$-EFTSS for $S$ then each $f$-spanner path between $s$ and $t$ must be contained entirely in $el(s,t)$. Clearly, a path which contains a point $r$ outside $el(s,t)$ has a length at least $dist_2(s,r) + dist_2(t,r)$ which is greater than $f \cdot dist_2(s,t)$. Let $G_{st}$ be the smallest subgraph of $G$ which contains all $f$-spanner paths between $s$ and $t$. Since $G$ is a $(k,f)$-EFTSS, $G_{st}$ must be $k$-edge connected. Otherwise, we could separate $s$ from $t$ in $G_{st}$ by deletion of a set $E'$ of $k$ edges, and therefore, we would not have any $f$-spanner path in $G \setminus E'$. Since the graph $G_{st}$ is $k$-edge connected, Menger's Theorem (see, e.g., in [20]) implies that it contains at least $k+1$ edge disjoint $st$-paths. Hence, $G_{st}$ – and, therefore, $el(s,t)$ – contains at least $k$ vertices different from $s$ and $t$ and at least $2k-1$ edges of $G$.

Now we show how to place the points of $S$ in order to get the desired lower bounds. We construct the set $S$ of $n$ points in the plane hierarchically bottom-up. First we assume that $n$ is a power of two. Let $l$ be a horizontal line and let $o$ be any fixed point of $l$. We place the points of $S$ on $l$. We put $s_1 \in S$ to $o$ and $s_2 \in S$ right to $s_1$ such that $dist_2(s_1, s_2) = 1$. Let $\epsilon > 0$ be a fixed real number. We translate $el(s_1, s_2)$ with the points $s_1$ and $s_2$ right on $l$, by a Euclidean distance $f + \epsilon$. This translation

guarantees that $el(s_1, s_2)$ and the translated ellipsoid are distinct. Let $s_3$ and $s_4$ denote the translated points $s_1$ and $s_2$, respectively. In general, in the $i$th step, $1 \leq i < \log n$, we translate the ellipsoid $el(s_1, s_{2^i})$ with the points $s_1, ..., s_{2^i}$ to the right on $l$, by a Euclidean distance $f \cdot dist_2(s_1, s_{2^i}) + \epsilon$. Denote the translated points by $s_{1+2^i}, ..., s_{2^{i+1}}$ (Figure 3.5). Then the ellipsoids $el(s_1, s_{2^i})$ and $el(s_{2^i+1}, s_{2^{i+1}})$ are distinct. We say that the ellipsoid $el(s_1, s_{2^{i+1}})$ is the *parent* of $el(s_1, s_{2^i})$ and $el(s_{1+2^i}, s_{2^{i+1}})$. Furthermore, we call the two children of an ellipsoid *siblings* of one another. We denote by $parent(el(.,.))$ and $sib(el(.,.))$ the parent and the sibling of an ellipsoid $el(.,.)$, respectively.

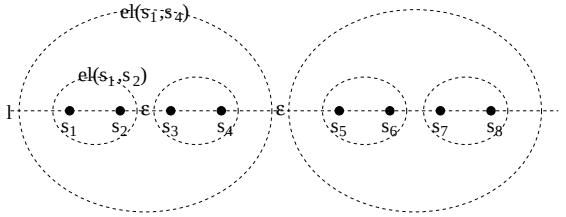


Figure 3.5: Example for a set $S$ of $n$ points for which each $(k, f)$-EFTSS contains at least $k\lfloor n/2 \rfloor$ Steiner points and $\lfloor n/2 \rfloor (2k+1) + (\lceil n/2 \rceil - 1)(k+1)/2$ edges.

Now we count the Steiner points and the edges in an arbitrary $(k, f)$-EFTSS $G$ for the set $S$. Consider a pair of points $s_{2j-1}, s_{2j} \in S$, $1 \leq j \leq n/2$. For this pair, there are at least $k+1$ edge disjoint paths in $G$ contained entirely in $el(s_{2j-1}, s_{2j})$. Since, for $j \neq j'$ the ellipsoids $el(s_{2j-1}, s_{2j})$ and $el(s_{2j'-1}, s_{2j'})$ are disjoint, each $el(s_{2j-1}, s_{2j})$ contains in the interior at least $k$ Steiner points and $2k+1$ edges. Furthermore, $s_{2j-1}$ and $s_{2j}$ must be $(k+1)$-edge connected with the points of $sib(el(s_{2j-1},s_{2j}))$. Therefore, we have at least $k+1$ edges contained entirely in $parent(el(s_{2j-1},s_{2j}))$ that have exactly one endpoint in $el(s_{2j-1},s_{2j})$. We can repeat these arguments at each level of the hierarchy of the ellipsoids. Then we obtain that the number of edges in $G$ is at least $(2k+1)n/2 + (k+1)(n/2-1) = (3k+2)n/2 - k - 1$ and the number of Steiner points is at least $kn/2$.

Now we consider the case that $n$ not a power of two. Let $2^i < n < 2^{i+1}$. We place the points in the same way as in the case when $n$ was a power of two. After we translated $el(s_1, s_{2^i})$ and obtained the points $s_{1+2^i}, ..., s_{2^{i+1}}$ we simply remove $s_{n+1}, ..., s_{2^{i+1}}$. Let $G$ be an arbitrary $k$ edge fault tolerant Steiner $f$-spanner for $S$. The number of Steiner points in $G$ is at least the sum of the Steiner points contained in $el(s_1, s_{2^i})$ and the Steiner points contained in $el(s_{2^i+1,s_n})$. Similarly, the number of edges in $G$ is at least the number of edges contained entirely in the interior of $el(s_1, s_{2^i})$ plus the number of edges contained entirely in the interior of $el(s_{2^i+1,s_n})$ plus the number of edges crossing the boundary of these two ellipsoids. We have seen that $G$ contains in the interior of $el(s_1, s_{2^i})$ at least $k2^{i-1}$ Steiner points and $(3k+1)2^{i-1} - (k+1)$ edges. Furthermore,

there are at least $k+1$ edge disjoint paths between $el(s_1, s_{2^i})$ and $el(s_{2^i+1}, s_n)$, and so the number of edges crossing the boundary of $el(s_1, s_{2^i})$ is at least $k+1$. We count the Steiner points and edges contained in $el(s_{2^i+1}, s_n)$ recursively. Let $n = \sum_{0 \leq i \leq \lfloor \log n \rfloor} b_i 2^i$ with $b_i \in \{0, 1\}$. Then the number of Steiner points in $G$ is at least

$$\sum_{1 \leq i \leq \lfloor \log n \rfloor} b_i k 2^{i-1} \quad = \quad k\lfloor n/2 \rfloor,$$

and the number of edges in $G$ is at least

$$\begin{aligned} \Big( \sum_{1 \leq i \leq \lfloor \log n \rfloor} b_i (3k+2) 2^{i-1} \Big) - (k+1) + b_0(k+1) \\ = (3k+2)\lfloor n/2 \rfloor - (k+1) + b_0(k+1) \\ = (2k+1)\lfloor n/2 \rfloor + (k+1)(\lceil n/2 \rceil - 1). \end{aligned}$$

Hence, the claim of the theorem follows. □

## 3.5 Conclusion and open problems

We have shown that for each set $S \subset \mathbb{R}^d$ of $n$ points, each real constant $f > 1$, and each integer $1 \leq k \leq n-2$, a $(k, f)$-VFTS for $S$ can be constructed with $O(kn)$ edges. This bound is optimal up to a constant factor and it improves the former $O(k^2 n)$ bound of Levcopoulos et al. [54]. Then we have given a short proof of the fact that each $(k, f)$-VFTS is also a $(k, f)$-EFTS even in the directed case. It implies that we can construct a $(k, f)$-EFTS with $O(kn)$ edges which bound is also optimal up to a constant factor and improves the former $O(k^2 n)$ bound of Levcopoulos et al. [54]. After this we have constructed a $(k, f)$-VFTS whose outdegree is bounded by $O(k^2)$. The best known former bound for the degree was $O(c^{k+1})$ for a constant $c$. Finally, we have considered a more natural but stronger definition of edge fault tolerance, which not necessarily can be satisfied using only simple edges between points of $S$ and we have studied fault tolerant Steiner spanners. We have proven that for each $S$, each real constant $f > 1$, and each $k \in \mathbb{N}$ there is a $(k, f)$-EFTSS and a $(k, f)$-VFTSS for $S$ with $O(kn)$ edges and $kn$ Steiner points. Then we have shown that there exist $S$, such that for each $f > 1$ and each $k \in \mathbb{N}$, any $(k, f)$-EFTSS for $S$ has $\Omega(kn)$ edges and Steiner points.

Some interesting problems remain unsolved. Is it possible to construct a $(k, f)$-VFTS whose degree is bounded by $O(k)$? Levcopoulos et al. [54] studied fault tolerant spanners with low weight. Let $w(MST)$ be the weight of the minimum spanning tree of $S$. In [54] it is proven that for each $S$ a $(k, f)$-VFTS can be constructed whose weight is $O(c^{k+1} w(MST))$ for some constant $c$. Can this upper bound be improved? In [54] it

is also proven that $\Omega(k^2 w(MST))$ is a lower bound on the weight. Is it possible to construct a $(k, f)$-VFTS with lower weight using Steiner points? Finally, we do not know any results for fault tolerant spanners with low diameter.

# Chapter 4

# Applications in walkthrough systems

In this chapter we examine some algorithmic aspects of the management of large geometric scenes in interactive walkthrough systems. The goal of walkthrough systems is a simulation and visualization of a three-dimensional scene. A scene consists of objects that are usually modeled by triangle meshes. The visitor of such a scene walks around and her environment is visualized on the screen or on a special output device. For a smooth animation we have hard *real time* requirements. Empirically, the computer has to render at least twenty pictures (frames) per second. If the animation is computed with less than twenty frames per second, navigation in the scenes is hard or impossible.

## 4.1 Geometric search problems in walkthrough systems

In todays computer systems the the objects of a scene are modeled by triangle meshes. The triangles of such a mesh contain geometric and object specific information, e.g., three-dimensional coordinates, color, textures, transparency information, etc... For every position of the visitor the computer has to compute a view of the scene. Hidden triangles have to be eliminated (hidden surface removal) and for all visible triangles color and brightness has to be computed. This process is called rendering.

The rendering process is often supported by special hardware, but the time for the rendering of a picture depends on the complexity of the scene, i.e., the number of triangles and the number of pixels that are needed for drawing a triangle. Therefore, if the scene is too large the real time requirement can only be guaranteed if the system

does not spend computation time for objects whose influence is not significant for the quality of the picture appearing on the screen. To control this situation real time and approximation algorithms are necessary to reduce the complexity of those parts of the scene that are far away, and thus, have a low influence to the quality of the rendered image.

Most of the methods used in computer graphics to reduce the complexity of the scene are highly dependent on its geometry. The change of the scene is joined with much programming work. Our goal is to develop real time algorithms for managing large and dynamic geometric scenes. Our scene is *dynamic* in the sense that a visitor can insert and/or delete objects. The basis for our considerations is an abstract modelling of the problem introduced by Fischer et al. [39] and refined in [41].

One of the most important problems to be solved in walkthrough systems is the *search problem*: In order to guarantee the real time behavior of our algorithms it is important that the time for the search, insertion, and deletion of objects is independent of the scene size. We focus on this problem.

The scenes that we consider are arbitrarily large and they consist of non overlapping unit size balls. Our methods exploit the fact that the visitor can only see a relatively small piece of the scene.

We fix an angle $\alpha$. We say that an object is *important* if it appears from the visitor's position in an angle at least $\alpha$ (Figure 4.1). These objects may be, for example, greater than a pixel on the display. Our goal is to develop data structures that support the selection of the important objects.

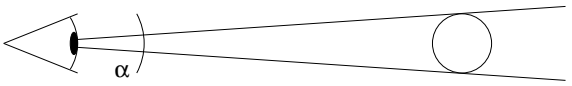


Figure 4.1: The object appears from the visitor's position in angle $\alpha$.

Representing the objects by points in $\mathbb{R}^2$, we obtain a kind of *circular range searching problem*. For a given a set $S \subset \mathbb{R}^2$ of $n$ points. For a query $query(q, r)$ we have to report the points of $S$ in the interior of the circle with center $q$ and radius $r$, where $r$ is determined by the angle $\alpha$. The difference between the classical circular range searching problem and our searching problem is that in the classical range searching problem we know nothing about the structure of the queries. But we can exploit the spatial locality of the queries, namely, that the visitor moves continuously and so consecutive query positions are near to each other. We give efficient solutions for the following problems.

**The moving visitor searching problem:** We say that the visitor moves *slowly* if the quotient $\frac{\delta}{x}$ is a constant, where $\delta$ is the maximum Euclidean distance between two

consecutive query positions and $x$ is the Euclidean distance between a *closest pair* of $S$. This assumption is motivated by the fact that in a scene consisting of non overlapping balls the centers of the balls can not be arbitrarily near to one another. Under the assumption that the visitor moves slowly through the scene, for a query $query(q, r)$ we have to report the points of $S$ in the interior of the circle with center $q$ and radius $r$. We show that under this assumption it is possible to perform a point location for the query position in $O(1)$ time.

**The dynamic searching problem:** In the dynamic case the visitor can insert or delete an object at her current position. Our goal is to develop a linear space data structure with the same query time as in the static case and with an update time similar to the query time.

Our goal is to develop deterministic data structures for the above problems with $O(n)$ space requirement. Since we work with very large data, we want to use as small space as possible. Data structures with superlinear space complexity are not acceptable for our purposes. Moreover, the constants in the $O$-notation are important. Furthermore, we require from our data structures that the reporting and the update time will be nearly linear in the size of the output. More precisely, we require an $O(1 + k)$ query and update time, where $k$ is the number of points of $S$ in a disc whose center is the position $q$ of the visitor and radius is $f$ times the radius $r$ of the query disc, where $f \geq 1$ is a small real constant.

### 4.1.1 Related problems, state of the art

Since the scene which we consider consists of unit size balls, the important objects (the objects that appear from the visitor's position in an angle at least $\alpha$) are contained in a ball whose center is the visitor's position $q$ and radius is $r = \frac{1}{2\sin(\alpha/2)}$. Hence, representing the objects of the scene by points in $\mathbb{R}^2$ ($\mathbb{R}^3$) we can report the important objects by solving a *circular range searching problem.* In the general form of the circular range searching problem we have a set $S$ of $n$ points in $\mathbb{R}^d$. For a query $query(q, r)$ we have to report the points of $S$ that are contained in the ball with center $q$ and radius $r$. This ball is called a *query ball.* In the two-dimensional case a query ball is also called a *query disc.*

#### 4.1.1.1 Circular range searching

For an overview of different kinds of range searching problems we refer to the survey of Matoušek [58] and Agarwal and Erickson [1]. Time optimal solutions for the

two-dimensional circular range searching problem use *higher order Voronoi diagrams*. Informally, the $i$th order Voronoi diagram $VD_i(S)$ of a set $S$ of $n$ sites in the $d$-space, $1 \leq i < n$, partitions the space into regions such that each point within a fixed region has the same $i$ closest sites. The regions of $VD_i(S)$ are convex polyhedra. The number of non empty regions in the two-dimensional $i$th order Voronoi diagram can be bounded by $O(i(n-i))$ [53]. Bentley and Maurer [14] presented a technique which extracts the sites of the Voronoi region containing $q$ in the $i$th order Voronoi diagram of $S$ for $i = 2^0 \log n, 2^1 \log n, 2^2 \log n, ...$ consecutively. It stops, if one of the $i$ sites that are assigned to the Voronoi cell containing $q$ is further from $q$ than the radius $r$ of the query disc. In [14] an $O(\log n \log\log n + k)$ query time is obtained, where $k$ is the number of the sites of $S$ that are contained in the query disc. The space requirement of the data structures is $O(n^3)$. Chazelle et al. [25] improved the query time to $O(\log n + k)$ and the space requirement to $O(n(\log n \log\log n)^2)$ using the algorithmic concept *filtering search*. Aggarwal et al. [3] reduced the space requirement to $O(n \log n)$ applying special *compacting* techniques. They achieve with this method an optimal query time for the circular range searching problem, but the data structure has a superlinear space requirement already in the two-dimensional case. Furthermore, the method in [3] uses the planar separator theorem of Lipton and Tarjan [55] for compacting the Voronoi diagram. We do not know any appropriate counterpart of this theorem which could be used for higher dimensional Voronoi diagrams. To find such a separator theorem seems to be a hard problem.

#### 4.1.1.1 Proximity problems, nearest neighbor queries

In some of the range searching methods one has to solve a *proximity problem*. In the above circular range searching method one have to solve a so called *i-nearest neighbor problem*, i.e., for an integer $i$ the set $S$ of $n$ sites must be preprocessed into a data structure such that for a query point $q$ one can determine fast the $i$ nearest sites of $q$. The case if $i = 1$ is of particular interest. The following results show that it is not possible to get a query time independent of $n$ without any restrictions on this problem.

Finding the *nearest neighbor* of a query point $q$ in $S$ has an $\Omega(\log n)$ time complexity in the algebraic computation tree model [13]. The planar version of this problem has been solved optimally with $O(\log n)$ query time and $O(n)$ space. But in higher dimensions no data structure of size $O(n \log^{O(1)} n)$ is known that answers queries in polylogarithmic time. The *approximate nearest neighbor* problem, i.e., finding a point $p \in S$ to the query point $q$ such that $dist_2(q,p) \leq (1+\epsilon) dist_2(q,q^*)$, where $q^* \in S$ is the exact nearest neighbor of $q$, was solved optimally by Arya et al. [7]. They gave a data structure in dimension $d \geq 2$ of size $O(n)$ that supports answering a query in $O(\log n)$ time. The constant factor in the query time depends on $\epsilon$. This data structure

can be constructed in $O(n \log n)$ time. For proximity problems on point sets in $\mathbb{R}^d$ a comprehensive overview is given by Smid [76].

### 4.1.2 Our approach

Instead of applying sophisticated methods we can use an arbitrary (weak) spanner $G(S)$ for $S$ to answer a circular range query $query(q, r)$ as follows. First we must find a near neighbor $q^* \in S$ – not necessarily the nearest neighbor – of $q$ in the given point set $S$. Then we have to perform a *breadth first search (BFS)* in $G(S)$ which starts at the vertex $q^*$ and visits only points $s \in S$ such that $dist_2(q^*, s) \leq f(r + dist_2(q, q^*))$, where $f$ is the stretch factor of $G(S)$. More precisely, we initialize a list $front := \{q^*\}$ and mark $q^*$ as visited. While $front \neq \emptyset$ we do the following: Remove the first point $p \in front$ from $front$, mark $p$ as visited, and append each non visited neighbor $s \in S$ of $p$ in $G(S)$ with $dist_2(q^*, s) \leq f(r + dist_2(q, q^*))$ to $front$. This implies that the BFS visits each point $s \in S$ with $dist_2(q, s) \leq r$. As we will see, if we use the $\theta$-graph $G_\theta(S)$ as underlying weak spanner for the search, after finding some certain points we must only visit points $s \in S$ with $dist_2(q, s) \leq fr$ in the BFS. The ball with center $q$ and radius $fr$ is called *critical ball* or *critical disc* in the two-dimensional case. Another advantage of using the $\theta$-graph is that in the BFS we must only traverse the outgoing edges at each point. This and the fact that the outdegree of the $\theta$-graph is a constant imply that the search time is bounded by the number of sites $s \in S$ for which $dist_2(q, s) \leq fr$.

The only problem we have is, how to find a point at which we can start the BFS. The results presented in Subsection 4.1.1 show that finding an appropriate near neighbor for a query point is a hard problem in the general case. In order to solve this in constant time we extend the set of $n$ original points by $O(n)$ auxiliary points, so-called *Steiner points* and we exploit the spacial locality of consecutive queries, i.e., we utilize that the visitor moves slowly.

### 4.1.3 Outline

First we describe a randomized solution of Fischer et al. [39] for the moving visitor search problem. We begin with the static problem, where the objects of the scene are static during the walkthrough. Then we describe the fully dynamic version of the data structure, where the visitor is able to insert or delete an object at her current position.

After this we present our own deterministic data structure which has been published in [40]. Here we also begin with the static problem and then we give a solution for the

incremental problem, where the visitor is allowed to insert a new object at her current position.

In both methods the original set $S$ of $n$ points will be extended by Steiner points and the underlying data structures are based on so-called *Steiner weak spanners*. Unfortunately, both solutions – the randomized and the deterministic – use also non algebraic operations for the construction of data structures.

Finally, we investigate the question how we can report the important objects when the scene consists of balls of different sizes. We show a simple technique which allows us to use circular range searching methods to solve this problems. Then we show geometric mappings that reduce this problem to *halfspace range searching problems* in higher dimensions.

## 4.2 A randomized solution

In this section we present a randomized data structure of Fischer et al. [39]. This solution does not take advantage from the assumption that the visitor moves slowly. Let $S$ be a set of $n$ points in the plane, where each point of $S$ represents an object of the scene. The data structures presented in this section are based on the $\theta$-graph $G_\theta(S)$ of the set $S$ as defined in Subsection 2.1.3 for $0 < \theta \leq \frac{\pi}{3}$ and as in Subsection 2.1.4 for $\theta = \frac{\pi}{2}$.

### 4.2.1 The static data structure

For the description of the data structure we need the following definition. Let $a > 0$, $0 < \theta \leq \frac{\pi}{2}$ be real numbers, and $C$ a $\theta$-frame. A set $S$ of $n$ points in the plane is called $(a, \theta)$-*crowded* if for each point $q \in \mathbb{R}^2$ and for each cone $c \in C$ holds that either $S \cap c(q) = \emptyset$ or there is a point $s \in S \cap c(q)$ with $dist_2(q, s) \leq a$. In [39] is stated the following.

**Theorem 4.1** [39] *Let $a > 0$, $0 < \theta \leq \frac{\pi}{3}$ be real numbers, $S$ an $(a, \theta)$-crowded point set in the plane, $G_\theta(S)$ the $\theta$-graph of $S$, and $f$ the weak spanner stretch factor of $G_\theta(S)$. Then for each $s, t \in S$ there is a directed path in $G_\theta(S)$ from $s$ to $t$ such that for each vertex $v$ on this path holds that $dist_2(s, v) \leq dist_2(s, t) + fa$.*

We remark that Theorem 4.1 holds also for the case if $\theta = \frac{\pi}{2}$. This theorem implies that using the $\theta$-graph $G_\theta(S)$ of an $(a, \theta)$-crowded point set $S$ we can perform a query

$query(q, r)$ such that we first find a point $s \in S$ whose distance from $q$ is at most $a$ and then we perform a BFS started at $s$ visiting only vertices contained in the disc $\{x \in \mathbb{R}^2 : dist_2(s, x) \leq r + dist_2(q, s) + fa\}$. We call this disc the critical disc. It remains to solve the problem, how we can make an arbitrary set $S$ of points $(a, \theta)$-crowded such that $a < r$ and how we can find a point $s \in S$ with $dist_2(q, s) \leq a$.

The above problems were solved in [39] by extending the original set $S$ of $n$ points with a set $M$ of Steiner points that are placed on the vertices of a grid which cover the bounding box of $S$. Formally, let $B := [x_1, x_2] \times [y_1, y_2]$ be the bounding box of $S$ (i.e., $B$ is the smallest axis-parallel rectangle which contains $S$) and $l := \frac{a}{1+1/(2\sin(\theta/2))}$. We place a Steiner point $p \in M$ on each position $(x_1 + il, y_1 + jl)$, $i = 0, ..., \lceil \frac{x_2 - x_1}{l} \rceil$, $j = 0, ..., \lceil \frac{y_2 - y_1}{l} \rceil$. With this density of the grid we guarantee that the extended point set becomes $(a, \theta)$-crowded. Furthermore, for each query position we can locate a Steiner point within a distance $a$ in constant time (using the `floor` operation).

Unfortunately, it is not possible to bound the number of Steiner points by any function of $n$, because the bounding box $B$ can be arbitrarily large. If we store all Steiner points, the space complexity could be enormous. Fischer et al. [39] showed the following way to keep the space complexity linear in $n$. For this they introduced the notion of *essential Steiner points*. A Steiner point $p \in M$ is called *essential* if at least one of its neighbors in the undirected $\theta$-graph $G_\theta(S \cup M)$ is an original point of $S$. Let $M'$ be the set of essential Steiner points. In [39] it is proven that number of points in $M'$ is $O(n)$. This follows from the facts that $(i)$ there is no directed edge in the $\theta$-graph which is longer than $a$, because of the $(a, \theta)$-crowdedness and $(ii)$ the number of Steiner points that are not further from an original point than $a$ is bounded by a constant.

The data structure in [39] stores only the original point set $S$, the set $M'$ of the essential Steiner points and the subgraph of $G_\theta(S \cup M)$ induced by $S \cup M'$. In addition, the positions of the essential Steiner points – more precisely, the pairs $(i, j)$ of integers such that $(x_1 + il, y_1 + jl)$ is a position of an essential Steiner point – are maintained in a *perfect hash list* as described in [33, 32]. This randomized data structure supports *lookup* operations in $O(1)$ worst-case time, it needs $O(n)$ space and can be constructed in $O(n)$ time with high probability, i.e., with probability $1 - n^{-\alpha}$, where $\alpha$ can be chosen as an arbitrarily large constant. Using this data structure we achieve the following result.

**Theorem 4.2** [39] *Let $S$ be a set of $n$ points in the plane and $a \in \mathbb{R}$ be a lower bound on the radius of the circular range queries. The set $S$ can be preprocessed into data structure which supports for a point $q \in \mathbb{R}^2$ and radius $r \geq a$ to report the points of $S$ that are not further from $q$ than $r$ in $O(k + (2 + r/a)^2)$ time, where $k$ is the number of points of $S$ in the critical disc. This data structure needs $O(n)$ space.* □

### 4.2.2 The fully dynamic data structure

Now we sketch how a point can be inserted or deleted efficiently. We maintain the set $M'$ of essential Steiner points in a *dynamic perfect hash list* [32]. This randomized data structure supports lookup, insert, and delete operations in $O(1)$ time, where the time bound for the insert and delete operations hold with high probability. The hash list needs linear space.

When a point is inserted or deleted, some of the non essential Steiner points may become essential and vice versa. Since we maintain in our data structure a $(a, \theta)$-crowded point set it is sufficient to check the Steiner points whose Euclidean distance from the position $q$ is at most $a$, whether they change their status from non essential to essential or vice versa. Only these Steiner points can be an undirected neighbor of the inserted or deleted point at the position $q$. Therfore, we have only to check a constant number of Steiner points, and so, we must perform a constant number of operations on the perfect hash list. The constant depends on $\theta$. Furthermore, for updating the adjacencies in the graph we must check also the original points of $S$ within a distance $a$ from $q$. Since these points represent unit size balls, their number is bounded by $O(a^2)$.

**Theorem 4.3** [39] *Let $S$ be a set of $n$ points in the plane and $a \in \mathbb{R}$ be lower bound on the radius of the circular range queries. The set $S$ can be preprocessed into data structure which supports for a point $q \in \mathbb{R}^2$ and radius $r \geq a$ to report the points of $S$ that are not further from $q$ than $r$ in $O(k + (2 + r/a)^2)$ time, where $k$ is the number of points of $S$ in the critical disc. Furthermore, at the positon $q$ a point can be inserted in $O(a^2)$ and deleted in $O(a^2 \log a)$ time with high probability. The data structure needs $O(n)$ space.* □

The advantage of this randomized data structure is that it can be implemented easily. Moreover, the generalization in higher dimensions is straightforward, using a higher dimensional grid and $\theta$-graph. A disadvantage is the large space overhead, because of using hashing.

## 4.3 A deterministic solution

In this section we present a deterministic data structure for the static and for the incremental moving visitor searching problem. To achieve the desired time bounds we need the assumption that the visitor of the scene moves slowly. This data structure was published in [40]. It is based on the graph $G_{\pi/2}(S)$ which was defined in Subsection 2.1.4.

Before we describe the data structure, we prove a useful property of the graph $G_\theta(S)$ for $0 < \theta \leq \frac{\pi}{3}$ and $\theta = \frac{\pi}{2}$. This property implies that we can answer a query $query(q, r)$ such that after the solution of a certain proximity problem, we only have to visit such vertices in the BFS that are not further from $q$ than $fr$, where $f$ is the weak spanner stretch factor of $G_\theta(S)$. Remember, for $\theta = \frac{\pi}{2}$ is $f = \sqrt{3+\sqrt{5}}$ and for $0 < \theta \leq \frac{\pi}{3}$ is $f = \max(\sqrt{1+48\sin^4(\theta/2)}, \sqrt{5-4\cos\theta})$.

**Lemma 4.4** *Let $S$ be a set of $n$ points in $\mathbb{R}^2$ and $\theta$ be an angle such that $0 < \theta \leq \frac{\pi}{3}$ or $\theta = \frac{\pi}{2}$. Furthermore, let $q \in \mathbb{R}^2$ be a query point, $t \in S$ a site, $c(q)$ with apex $q$ which contains $t$, and $s \in S \cap c(q)$ a site for which $dist_c(q,s) = min(dist_c(q,s') : s' \in S \cap c(q))$. Then there is a directed path $P$ in $G_\theta(S)$ from $s$ to $t$ such that for each vertex $v \in P$ holds that $dist_2(q,v) \leq f \cdot dist_2(q,t)$.*

**Proof:** We only prove the lemma for $\theta = \frac{\pi}{2}$. When $0 < \theta \leq \frac{\pi}{3}$, the proof is similar. In the proof of Theorem 2.5 we showed that there is a path $P$ in $G_{\pi/2}(S)$ from $s$ to $t$ which is entirely contained in the square range $B_s = \{x \in \mathbb{R}^2 : dist_1(t,x) \leq dist_1(t,s)\}$. Furthermore, we know that $dist_1(t,s) \leq dist_1(t,q)$. Therefore, the path from $s$ to $t$ is entirely contained in the square $B_q = \{x \in \mathbb{R}^2 : dist_1(t,x) \leq dist_1(t,q)\}$. This implies the claimed bound on $dist_2(q,v)$ for each vertex $v \in P$. □

The above lemma implies the following method to report the points of $S$ that are contained in the query disc. For the center of the query disc $q$, in each cone $c(q)$, $c \in C$, $c(q) \cap S \neq \emptyset$, we have to find a nearest neighbor $nn_c(q)$ w.r.t. $dist_c$. After this, in the BFS procedure we initialize $front$ to be the set $\{nn_c(q) : c \in C, dist_2(q, nn_c(q)) \leq fr\}$. Then we proceed as described in Subsection 4.1.2 visiting only points whose Euclidean distance from $q$ is at most $fr$.

It remains to solve the problem how we can find the nearest neighbors of the query position $q$ in the cones in constant time. For this we exploit that the visitor moves slowly through the scene. We extend the original point set $S$ with $O(n)$ carefully placed Steiner points, then we construct the graph $G_{\pi/2}(S')$ for the extended point set $S'$, and we take advantage of the fact that the nearest neighbors of the query position $q$ in the extended point set $S'$ is close to the nearest neighbors of the previous query position $q_{prev}$. First we present a data structure for the static problem then we describe how we can insert new points into this structure efficiently.

### 4.3.1 The static data structure

We begin with the description of the construction of the data structure. Then we show how we can use it to find the nearest neighbors of the query position in constant time

assuming that the visitor moves slowly. Subsequently, we present algorithmic details of the construction.

#### 4.3.1.1 The construction

We place the $O(n)$ Steiner points similar as in the *mesh generation* method of Bern et al. [17, 18]. First we construct a linear size, balanced quadtree [75, 74] for the point set $S$ and we put the Steiner points to the corners of the boxes of this quadtree. Now we describe this technique briefly.

**Definitions** [17]: A *quadtree* is a recursive subdivision of the plane into square boxes. The nodes of the quadtree are the boxes. Each box is either a *leaf* of the tree or is *split* into four equal-area children. A box has four possible *neighbors* in the four cardinal directions; a neighbor is a box of the same size sharing a side. A *corner* of a box is one of the four vertices of its square. The corners of the quadtree are the points that are corners of its boxes. A side of a box is *split* if one of the neighboring boxes sharing it is split. A quadtree is called *balanced* if each side of an unsplit box has at most one corner in its interior. An *extended neighbor* of a box is another box of the same size sharing a side or a corner with it.

**Building balanced quadtree for** $S$ [17]: We start with a root box $b$ which is concentric with and twice as large as the smallest bounding square of $S$. We recursively split $b$ as long as $b$ has a point of $S$ in its interior and one of the following conditions holds.

($i$) $b$ has at least two points or

($ii$) $b$ has side length $l$ and contains a single point $p \in S$ with nearest neighbor in $S$ closer than $2\sqrt{2}l$ or

($iii$) one of the extended neighbors of $b$ is split.

Then we balance the quadtree. We remark that the balancing increases the space requirement of the quadtree only by a constant factor. After all splits are done, every leaf box, which contains a point of $S$, is surrounded by eight empty leaf boxes of the same size.

**Building linear size balanced quadtree** [17]: The only nonlinear behavior of the above algorithm occurs, when a nonempty box is split without separating points of $S$. If this happens, we need to "shortcut" the quadtree construction to produce small boxes around a "dense" cluster without passing through many intermediate size of boxes. We construct the quadtree for the cluster recursively and we treat the cluster as an individual point. In this way we obtain a linear size quadtree for $S$, i.e., the number

of the boxes is linear in $n$. As mentioned in [17] the linear size quadtree for $S$ can be constructed in $O(n \log n)$ time and $O(n)$ space. We present some algorithmic details later.

Here ends the part borrowed from Bern et al. [17].

We call a dense cluster with the containing box and the eight extended neighbor boxes an *extended cluster*. Within each extended cluster we also maintain the balance property. Hence, each extended cluster contains only a constant number of corners on its boundary. This property will be crucial for the fast navigation.

**Building $G_{\pi/2}$ from the quadtree**: We extend $S$ with the $O(n)$ corners of the linear size quadtree and construct the graph $G_{\pi/2}(S')$ for the extended point set $S'$. If we have the quadtree, we can determine for each point of $S'$ its four neighbors in $G_{\pi/2}(S')$ in constant time. It follows from the fact that the neighbors of a point $p \in S'$ in $G_{\pi/2}(S')$ are in the interior of the leaf box $b(p)$ containing $p$ or they are in the interior of a neighboring box of $b(p)$. Figure 4.2 illustrates the possible cases.

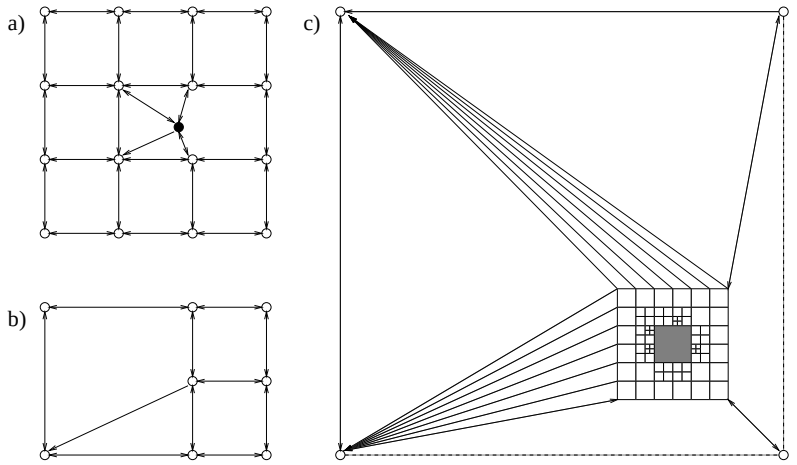


Figure 4.2: The scheme to construct $G_{\pi/2}(S')$ from the quadtree: a) An original point of $S$, the box containing it and the eight extended neighbor boxes. b) A split side. c) A shortcut.

#### 4.3.1.2 Point location in constant time when the visitor moves slowly

If we have $G_{\pi/2}(S')$, we can determine for each point $p'$ of $S'$ the leaf box $b(p')$ of the quadtree containing $p'$ in constant time. Furthermore, if $x$ is the distance between the closest pair of $S$ then the side length of the smallest box of the quadtree is a constant fraction of $x$. These observations imply that for the query position $q$, the box $b(q)$ of the quadtree containing $q$ can be computed in constant time using $G_{\pi/2}(S')$, if we know the box $b(q_{prev})$ containing the previous query position $q_{prev}$, since the line segment

$[q_{prev}, q]$ crosses only a constant number of leaf boxes of the quadtree. The crossings and so the box $b(q)$ can be computed in constant time. Then the nearest neighbor of $q$ in $S'$ in each cone $c(q)$, $c \in C$, also can be determined in constant time. Therefore, in order to answer a query $query(q, r)$, we can start the BFS after we initialized the list $front$ with these nearest neighbors. We conclude:

**Theorem 4.5** *Let $S$ be set of $n$ points in the plane. Assume that the visitor moves slowly. Then there is a data structure which supports for a query $query(q, r)$ to report the points of $S$ within the query disc in $O(1+k)$ time, where $k$ is the number of vertices $v$ of $G_{\pi/2}(S')$ for them $dist_2(q, v) \leq r\sqrt{3+\sqrt{5}}$ holds. The space requirement of the data structure is $O(n)$ and can be constructed in $O(n \log n)$ time.* □

#### 4.3.1.3 Algorithmic details of the quadtree construction

**Computing a quadtree without shortcuts:** If the points of $S$ are distributed such that we never need a shortcut, the quadtree can be constructed in in $O(n \log n)$ time using only algebraic operations. This can be obtained by a method which is very similar to the tree construction algorithm of Vaidya [79] and of Callahan [21]. We sort the points of $S$ by its $x$- and $y$-coordinates obtaining two sorted doubly linked lists $L_x$ and $L_y$. In both lists for each point $p \in S$ we have a cross-pointer to its position in the other list. At each time, when we separate points of $S$ by a vertical and a horizontal line segment, we split out the smaller point set from the lists. We consider a box splitting as a vertical split followed by horizontal split in both sides.

We now describe, how to maintain the sorted lists during these splits. First we create copies $L_x^c$ and $L_y^c$ of the lists $L_x$ and $L_y$. We will need this copies later. In $L_x^c$ we maintain for each item a pointer $p\_orig$ to its position in $L_x$. Similarly, in $L_y^c$ we maintain for each item a pointer $p\_orig$ to its position in $L_y$. Then we perform the following recursive procedure. We create two empty lists $L'_x$ and $L'_y$. They will contain the points of the smaller point set resulting from the split. We search the list $L_x$ simultaneously from left to right and from right to left until we find a point that belongs to the other side of the separating vertical line segment. The search time is linear to the size of the smaller set. W.l.o.g. we assume that the search from left to right was shorter.

Then we begin from left again, walk sequential in $L_x$ up to the first point right to the separating vertical line. During this walk we delete the points from $L_x$ and insert them at the end of $L'_x$. Using the cross-pointers we delete this points from $L_y$, as well, and insert them at the end of $L'_y$. After this vertical splitting the list $L'_x$ is already sorted but the list $L'_y$ is not. It would be inefficient to sort it now, first we leave it unsorted and

split the lists $L_x$ and $L_y$ (the remaining longer lists that are both sorted) recursively horizontal.

At the last recursion level, when $L_x$ (and $L_y$) contains at most one point, we sort all the unsorted lists $L'_x$, $L'_y$, that are created at a higher level of the recursion, as follows. We traverse the two lists $L^c_x$ and $L^c_y$ that are the copies of the original lists $L_x$ and $L_y$ before the deletions. For each encountered item, we remove the corresponding item – which is given by the pointer *p_orig* – from $L'_x$ ($L'_y$, resp.) and insert it again at the end of the same list $L'_x$ ($L'_y$, resp.). In this way we can guarantee that in each list $L'_x$ ($L'_y$, resp.) each two items appear in the same order as in $L^c_x$ ($L^c_y$, resp.). Consequently, all lists $L'_x$ and $L'_y$ are sorted and the total time for the sorting is linear in the length of $L^c_x$.

Finally, we deallocate the copy lists $L^c_x$ and $L^c_y$, make a copy from each list $L'_x$, $L'_y$, and process each $L'_x$, $L'_y$ recursively until each list contains at most one item. Figure 4.3 illustrates the lists appearing in the above description.

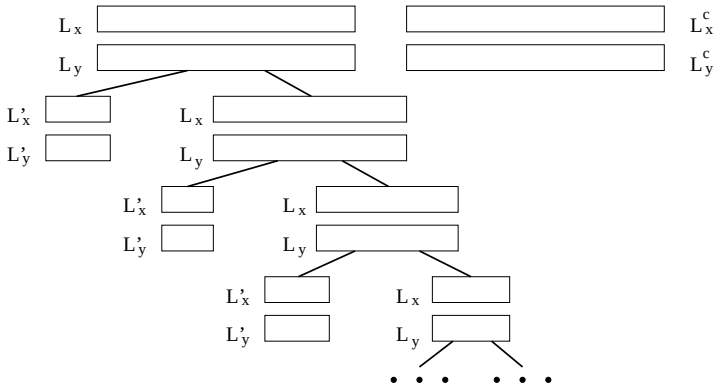


Figure 4.3: The lists $L_x$, $L_y$, $L'_x$, $L'_y$ and $L^c_x$, $L^c_y$ during the splitting algorithm.

Clearly, the running time of above splitting algorithm is $O(n \log n)$, because if an item is splitted out from a list, it will be inserted into a list whose size is at most half the size of its original list. It can happen at most $\log n$ times per item. After the splitting algorithm we have to complete the obtained tree to a balanced quadtree corresponding our definition. This can be done in $O(n)$ time.

**Realize shortcuts:** Now we turn to the problem how we can detect during the above algorithm, whether a shortcut is necessary and how this can be performed in constant time. We describe the solution of Bern et al. [18] for this problem.

In [18] it is assumed that the coordinates of the input points are integer numbers that can be stored in a single computer word. The underlying computation model is a special RAM which is able to perform simple arithmetic, shift and Boolean operations

on words, and to detect the position of the most significant non-zero bit of a computer word in constant time[1]. The sides of all squares in the quadtree have length of the form $2^i$, and for any square of side length $2^i$ the coordinates of all four corners are multiples of $2^i$. For two input points $p = (p_x, p_y)$ and $q = (q_x, q_y)$ we define their *derived square* as the smallest quadtree box containing $p$ and $q$. The side length $l$ of this square can be computed from the position $i$ of the most significant non-zero bit of the maximum of ($p_x$ XOR $q_x$) and ($p_y$ XOR $q_y$), the side length is $l = 2^{i+1}$. The bottom left corner of this square can be found by masking off the last $i$ bits of $p_x$ and $p_y$. The *derived square* of an arbitrary point set is defined analogously. This is the smallest quadtree box containing the point set. It can be computed from the minimum and maximum $x$- and $y$-coordinate in the point set. Since the splitting algorithm maintain the points in sorted lists, the minimum and the maximum coordinate can be determined in constant time. Consequently, we can check at each level of the recursion in constant time, whether a shortcut is necessary by computing the derived box of the current point set and comparing it with the derived box of the point set of its parent in the tree. Hence, the shortcut can be performed in constant time.

### 4.3.2 The incremental data structure, lazy updates

In this subsection we study the incremental version of the search problem, where insertion of a new point into $S$ at the current position of the visitor is allowed. We show how we can insert a point into the graph $G_{\pi/2}(S')$ in the same time as the query time.

When we insert a new point into $G_{\pi/2}(S')$, we first must update the quadtree such that the balance property is maintained. Then we must update the adjacencies at the affected vertices of $G_{\pi/2}(S')$ corresponding to corners of the changed boxes or their neighboring boxes. Updating $G_{\pi/2}(S')$ at the affected vertices costs only a constant time per box even in the case of a changed shortcut. Therefore, the update time is linear in the number of affected boxes. The main problem is to maintain the balance property. Let $l(b)$ be denote the *level* of box $b$, it is the length of the tree path from the root box of the quadtree to the box $b$. If a leaf box $b$ is splitted then the balance condition can be violated in each level higher than $l(b)$ (Figure 4.4). Therefore, the worst-case update time for the rebalancing of the quadtree depends on the depth of the quadtree, and so, on the total scene.

Let $D_r$ be a query disc with radius $r$ and $D_{rf}$ be the cricital disc which is concentric with $D_r$ and has a radius $rf = r\sqrt{3+\sqrt{5}}$. In order to make the rebalancing time independent from the total size of the scene, we perform *lazy updates*, i.e., we split

[1]This operation can also be written as $\lfloor \log_2 a \rfloor$, where $a$ is number stored in a computer word.

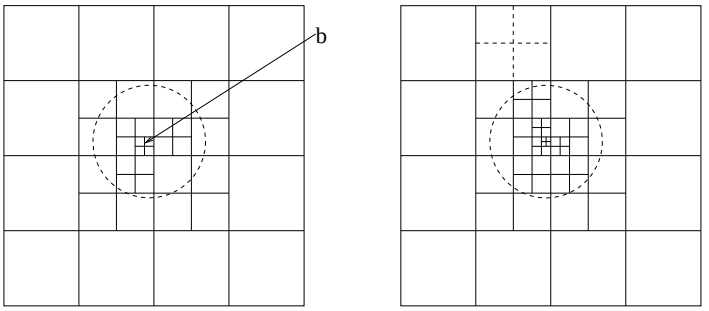


Figure 4.4: Rebalancing of the quadtree

only the boxes that intersect the critical disc $D_{rf}$. After the splittings we update only the vertices of $G_{\pi/2}(S')$ that are contained in $D_{rf}$. The remaining splittings will be performed later, when the current critical disc intersects the according boxes. Therefore, at the beginning of each query we have to check, whether the current critical disc $D_{rf}$ intersects new leaf boxes that violate the balance property. In this case we perform the necessary splittings. In this way we guarantee that the update time is at most linear in the number of vertices of the updated graph $G_{\pi/2}(S')$ in the disc $D_{rf}$. This time is equal to the time of the searching. We summarize:

**Theorem 4.6** *Let $S$ be set of $n$ points in the plane. Assume that the visitor moves slowly. Then there is an incremental data structure for the moving visitor searching problem which supports insertion of a new point at the current position $q$ of the visitor and for query$(q, r)$ it allows reporting of the points of $S$ within the query disc both in $O(1+k)$ time, where $k$ is the number of vertices $v$ of $G_{\pi/2}(S')$ for which $dist_2(q, v) \leq r\sqrt{3+\sqrt{5}}$ holds. The space requirement of the data structure is $O(n)$.* □

## 4.4 Objects with different sizes

In this section we develop data structures for the searching problem in geometric scenes that consist of balls with different sizes. We represent the objects of such a scene by weighted points, such that the weight $w(s)$ of a point $s \in S \subset \mathbb{R}^d$ is the diameter of the ball represented by $s$. Similar to the case of equal size balls, we have to determine the objects appearing from the visitor's position in an angle which is not less than a fixed constant $\alpha$. Using weighted points to represent the objects of the scene the above problem can be formulated as follows: Let $r := \frac{1}{2\sin(\alpha/2)}$. For a query operation query$(q, r)$ we have to report the set $\{s \in S : \frac{1}{w(s)}\ dist_2(q, s) \leq r\}$ of weighted points.

In the first part of this section we develop a general technique, which allows us to

use data structures for circular range searching to report the objects appearing from the visitor's position in an angle at most $\alpha$. This technique is applicable to each decomposable searching problems on weighted points.

In the second part of the section we use geometric transformations to obtain a higher dimensional halfspace range searching problem and solve this problem with known algorithms. This approach was, for example, used in [46, 2, 9, 36, 83] for similar problems.

### 4.4.1 Using circular range searching in size classes

In this subsection we exploit that our search problem is a so-called *decomposable search problem* [15, 65, 35]. We use a special decomposition which allows us to apply circular range searching algorithms for an approximated solution of the original searching problem.

Decomposable search problems are search problems that satisfy the following conditions [35]: Let $S$ be a set of objects and let $S$ be arbitrarily partitioned into two sets $S_1$ and $S_2$. Furthermore, let $Q$ be an arbitrary query object, and $A_1$ and $A_2$ be the answers to the queries for $S_1$ and $Q$ and for $S_2$ and $Q$, respectively. Then the answer $A$ for $S$ and $Q$ can be computed in constant time from $A_1$ and $A_2$. Our searching problem satisfy this condition obviously.

To handle objects with different sizes we partition the points of $S$ into *size classes*. Let $w_{min}$ be the size of the smallest object and $w_{max}$ the size of the largest object of the scene. We define the *aspect ratio* of the scene $A := \frac{w_{max}}{w_{min}}$. Let $l := \lceil \log A \rceil$. We partition the objects into $l$ classes such that in each class the largest object is at most two times larger than the smallest object. More precisely, let $S_i := \{s \in S : 2^{i-1} w_{min} \leq w(s) < 2^i w_{min}\}$ for $i = 1, ..., l$. We call $S_i$ a *size class* of $S$. (If $A$ is a power of two then let $S_l$ also contain the objects with size $w_{max}$.) In each class $S_i$ we treat the points as they had the same size and for each $S_i$ we independently build a circular range searching data structure. Note that if the largest and the smallest object of the scene have a size 1km and 1mm, respectively, we only have twenty classes. We can answer a query $query(q, r)$ for weighted points such that we start a circular range query in each non empty class $S_i$ with center $q$ and radius $r_i = 2^{i-1} w_{min} r$. (This is the distance, from which an object of size $2^i w_{min}$ appears in angle $\alpha$.) Then we concatenate the results and verify for each obtained point $s$, whether $\frac{1}{w(s)}\ dist_2(q, s) \leq r$ holds.

This method works with various circular range searching algorithms. Let $D$ be a data structure used for solving the circular range searching problem in the size classes. Assume that the space requirement of $D$ is $O(f(n))$ and that $D$ supports answering queries

in $O(g(n)+k)$ time, where $k$ depends on the size of the output. Furthermore, assume that $f(n)/n$ is nondecreasing and $g(n)/n$ is nonincreasing. Then the above searching data structure for weighted points has a space complexity $O(f(n)+l)$ and supports answering queries in $O(l \cdot g(n)+k')$ time, where $k' = \sum_{1 \le i \le l} k_i$ and $k_i$ is the output sensitive part of the query time in the circular range searching data structure for the size class $S_i$.

### 4.4.2 Transformation to halfspace range searching

In this subsection we transform the $d$-dimensional problem of reporting the points $\{s \in S : \frac{1}{w(s)}\, dist_2(q,s) \le r\}$ to a halfspace range searching problem in $\mathbb{R}^{d+1}$ if each query has the same radius $r$ (i.e., the angle $\alpha$ is fixed) and to a halfspace range searching problem in $\mathbb{R}^{d+2}$ if $r$ is variable. Then we can solve the halfspace range searching problem by known techniques.

Such geometric transformations were used, for example, in [46, 2, 36, 9, 83]. In particular, we remark that using this method, Gupta et al. [46] presented a solution for the following intersection searching problem: Let $\mathcal{B} := \{B_1, ..., B_n\}$ be a collection of closed $d$-balls in $\mathbb{R}^d$, $d \ge 2$. For a query $d$-ball $Q$ we have to report the balls of $\mathcal{B}$ that are intersected by $Q$. In [46] this problem is transformed to a $(d+2)$-dimensional halfspace range searching problem. This immediately implies also a solution for our $d$-dimensional reporting problem on weighted points if each query disc has the same radius $r$. We map each weighted point $s \in S(\subset \mathbb{R}^d)$ to a $d$-ball $B$ with center $s$ and radius $w(s)r$, and we map the query position $q$ to a ball $Q$ with center $q$ radius zero.

In order to understand the spirit of this kind of geometric mappings, we first transform a $d$-dimensional circular range searching problem (on unweighted points) to a $(d+1)$-dimensional halfspace range searching problem: Let $S \subset \mathbb{R}^d$ be a set of $n$ points. For a query $query(q,r)$ we have to report the points of $S$ whose Euclidean distance from $q$ is at most $r$. For a point $s \in \mathbb{R}^d$ we denote its $i$th coordinate by $s_i$. A point $s \in S$ is contained in the query ball if and only if

$$\sum_{1 \le i \le d} (q_i - s_i)^2 \le r^2$$

holds. Let us define a transform $\gamma$ which maps the points of $S$ to the surface of a $(d+1)$-dimensional paraboloid as follows:

$$\gamma(s) := \Big(s_1, ..., s_d, \sum_{1 \le i \le d} s_i^2\Big).$$

Also, let us define $\beta$ which maps the center $q$ and the radius $r$ of a query to a $(d+1)$-dimensional hyperplane as follows:

$$\beta(q,r) : x_{d+1} = r^2 + \sum_{1 \le i \le d} 2q_i x_i - \sum_{1 \le i \le d} q_i^2.$$

It is easy to show that a point $s \in S$ is contained in the ball with center $q$ and radius $r$ if and only if $\gamma(s) \in \beta(q,r)^-$, where $\beta(q,r)^-$ is the closed halfspace lying below $\beta(q,r)$, i.e., the halfspace $x_{d+1} \le r^2 + \sum_{1 \le i \le d} 2q_i x_i - \sum_{1 \le i \le d} q_i^2$ (Figure 4.5). To see this, it is sufficient to show that

$$dist_2(q,s) \le r \quad \text{if and only if} \quad \sum_{1 \le i \le d} s_i^2 \;\le\; r^2 + \sum_{1 \le i \le d} 2s_i q_i - \sum_{1 \le i \le d} q_i^2.$$

This can be done by simple algebraic manipulation.

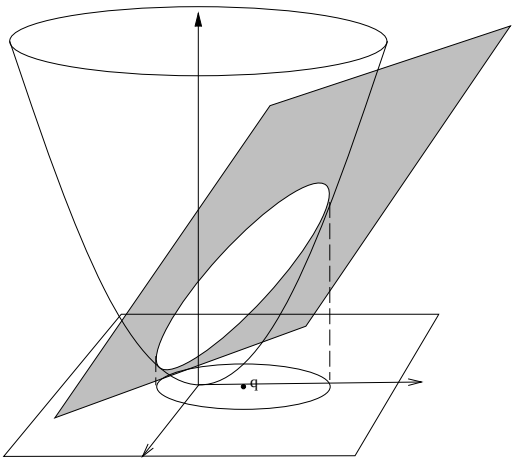


Figure 4.5: $dist_2(q,s) \le r$ if and only if $\gamma(s)$ below $\beta(q,r)$.

Now we consider the following reporting problem. We have a set $S \subset \mathbb{R}^d$ of $n$ weighted points and a radius $r \in \mathbb{R}$. For a query $query(q)$ we must report the points $\{s \in S : \frac{1}{w(s)}\, dist_2(q,s) \le r\}$. Let us define $\gamma$ which maps the points of $S$ to points into $\mathbb{R}^{d+1}$ as follows.

$$\gamma(s) := \Big(s_1, ..., s_d, \sum_{1 \le i \le d} s_i^2 - w(s)^2 r^2\Big).$$

Also, let us define $\beta$ which maps the query point $q$ to a $(d+1)$-dimensional hyperplane as follows:

$$\beta(q) : x_{d+1} = \sum_{1 \le i \le d} 2q_i x_i - \sum_{1 \le i \le d} q_i^2.$$

It is easy to verify that $\frac{1}{w(s)}\, dist_2(q,s) \le r$ holds for $s \in S$ if and only if $\gamma(s) \in \beta(q)^-$. Thus, we reduced the above weighted reporting problem to a $(d+1)$-dimensional halfspace range searching problem.

Finally, we show a transformation in the case that $r$ is also a parameter of the query, i.e., we cannot use $r$ in the mapping of the points of $S$. Let us define $\gamma$ which maps the points of $S$ to points in $\mathbb{R}^{d+2}$ as follows.

$$\gamma(s) := \Big(s_1, ..., s_d, w(s)^2, \sum_{1 \le i \le d} s_i^2\Big).$$

Also, let us define $\beta$ which maps the point $q$ and the radius $r$ to a $(d+2)$-dimensional hyperplane as follows:

$$\beta(q, r) : x_{d+2} = \sum_{1 \le i \le d} 2q_i x_i \; + \; r^2 x_{d+1} \; - \sum_{1 \le i \le d} q_i^2.$$

Also here, it is easy to verify that $\frac{1}{w(s)}\ dist_2(q, s) \le r$ holds for $s \in S$ if and only if $\gamma(s) \in \beta(q)^-$. Thus, we have transformed the above problem to a $(d+2)$-dimensional halfspace range searching problem.

To solve the halfspace range searching problem we can use the data structure of Matoušek [59], which, in $\mathbb{R}^p$, uses $O(n \log\log n)$ space and has $O(n^{1-1/\lfloor p/2 \rfloor} \text{polylog}\ n + k)$ query time, where $k$ is the output size. Alternatively, we can use the data structure of Clarkson and Shor [29] which uses $O(n^{\lfloor p/2 \rfloor + \epsilon})$ space and has $O(\log n + k)$ query time. Substituting $p = d+1$ and $p = d+2$, respectively, we obtain the following:

**Theorem 4.7**

*(i): Let $S$ be a set of $n$ weighted points in $\mathbb{R}^d$ and $r \in \mathbb{R}$. $S$ and $r$ can be processed into a data structure of size $O(n \log\log n)$ such that for a query point $q \in \mathbb{R}^d$ the points $\{s \in S : \frac{1}{w(s)}\ dist_2(q, s) \le r\}$ can be reported in time $O(n^{1-1/\lceil d/2 \rceil} \text{polylog}\ n + k)$, where $k$ is the output size; or $S$ can be processed into a data structure of size $O(n^{\lceil d/2 \rceil + \epsilon})$ and $O(\log n + k)$ query time.*

*(ii): Let $S$ be a set of $n$ weighted points in $\mathbb{R}^d$. $S$ can be processed into a data structure of size $O(n \log\log n)$ such that for a query point $q \in \mathbb{R}^d$ and radius $r \in \mathbb{R}$ the points $\{s \in S : \frac{1}{w(s)}\ dist_2(q, s) \le r\}$ can be reported in time $O(n^{1-1/\lfloor d/2 \rfloor + 1} \text{polylog}\ n + k)$, where $k$ is the output size; or $S$ can be processed into a data structure of size $O(n^{\lfloor d/2 \rfloor + 1 + \epsilon})$ and $O(\log n + k)$ query time.* □

## 4.5 Conclusion and open problems

We have described a randomized solution of Fischer et al. [39] for the static and the fully dynamic moving visitor searching problem and our own deterministic solution for

the static and the incremental moving visitor searching problem in scenes that consist of unit size balls. In the deterministic data structure we have exploited the assumption that the visitor moves slowly. Both solutions support query and update times that are linear in the number of vertices of the underlying data structure in the interior of the critical circle, in particular independent from the total number of objects of the scene. Both data structures have linear size and they are relatively easy to implement. Unfortunately, we were not able to give a deterministic solution for the fully dynamic problem which also supports deletions as fast as insertions. An unpleasant property of both data structures is that we also need non algebraic operations in the construction. The question, whether a deterministic linear size data structure exists, which solves the moving visitor problem and can be constructed in $O(n \log n)$ time using only algebraic operations, remains open.

We have presented solutions for the moving visitor problem in scenes that consist of different size balls. We have seen that this problem can be transformed to a halfspace range searching problem in a higher dimension. Then it can be solved using standard techniques. The question, whether in the halfspace range searching data structures can be exploited that the visitor moves slowly, remains unsolved.

# Chapter 5

# Lower bound on the construction time of weak spanners with Steiner points in the algebraic model

In this chapter we give an $\Omega(n \log n)$ lower bound for the time for each algorithm in the algebraic model [13] that construct a Steiner weak spanner with $o(n \log n)$ edges and with stretch factor $f \geq 1$. The content of this chapter is a slight modification of the result of Chen et al. [26]. They proved such a lower bound on the construction time of Steiner spanners. First we specify more precisely the class of graphs for which we prove the lower bound.

Let $p \in \mathbb{R}$ be a fixed constant such that $1 \leq p \leq \infty$. We measure the distance between points in the $d$-dimensional space $\mathbb{R}^d$ with the $L_p$ metric. Let $S$ be a set of $n$ points in $\mathbb{R}^d$. Let $G = (V, E)$ a (directed) graph such that

(*i*) $V$ is a set of points in $\mathbb{R}^d$,

(*ii*) $S \subseteq V$,

(*iii*) the edges of $G$ are straight line segments in $\mathbb{R}^d$ connecting pairs of points in $V$. The length of an edge is the $L_p$ distance $dist_p$ between the endpoints.

Let $f \geq 1$ be a real number. The graph $G$ is a (directed) *Steiner weak spanner* for $S$ with stretch factor $f$ w.r.t. the $L_p$ metric if for each two points $s, t \in S$, there is a (directed) path $P$ from $s$ to $t$ in $G$ such that for each vertex $v \in P$, $dist_p(s, v) \leq f \cdot dist_p(s, t)$. If $V = S$ then $G$ is a weak spanner for $S$. Note that the definition of a Steiner weak spanner is more general than the definition of a weak spanner for an extended point set $S'$ for $S$ in Chapter 4: for Steiner weak spanners we only require the existence of a $f$-spanner path between the original points of $S$. Therefore, a lower bound on the

running time of algorithms constructing Steiner weak spanners holds also for algorithm constructing weak spanners for $S$ or for an extended set $S'$ of $S$. Clearly, each algorithm that constructs a Steiner weak spanner with $\Omega(n \log n)$ Steiner points or $\Omega(n \log n)$ edges needs $\Omega(n \log n)$ time. Hence, it suffices to prove this lower bound for Steiner weak spanners with $o(n \log n)$ edges and $o(n \log n)$ Steiner points.

We prove the $\Omega(n \log n)$ bound for the one dimensional case when $S \subseteq \mathbb{R}$. This implies the same lower bound for any dimension $d > 1$. We first give a very simple reduction from the well known *element distinctness problem* which has an $\Omega(n \log n)$ bound in the algebraic *decision tree model* [13].

Remember, in the element distinctness problem we have $n$ real numbers $x_1, ..., x_n$, and we must decide whether they are pairwise distinct or not. The main idea for the reduction of this problem to the construction of a Steiner weak $f$-spanner is the following: If $x_i = x_j$ for $i \neq j$ then each Steiner weak $f$-spanner for $S = \{x_1, ..., x_n\}$ contains a path $P$ between $x_i$ and $x_j$ such that for each point $v \in P$, $dist_p(x_i, v) \leq f \cdot dist_p(x_i, x_j) = 0$. It implies that each edge on $P$ has length zero.

Let $A$ be an algorithm that for the input $S$ and $f \geq 1$ constructs a Steiner weak spanner $G$ for $S$ with stretch factor $f$. Assume, that the vertices of $G$ are labeled so that we can distinguish the original points of $S$ from the Steiner points. We construct a graph $G'$ for $G$ such that $G'$ has the same vertex set as $G$ and an edge is in $G'$ if and only if it is an edge in $G$ and has length zero. To solve the element distinctness problem we only have to check for each connected component of $G'$ if it contains two distinct element of $S$ among its vertices. If there is a component with at least two original point of $S$ then output NO; otherwise, output YES. Hence, if we have a Steiner weak spanner $G$ for $S$ with stretch factor $f$, we can solve the element distinctness problem in a time which is linear in the number of edges of $G$, which is $o(n \log n)$. Therefore, algorithm $A$ must have an $\Omega(n \log n)$ running time.

The above lower bound proof is unsatisfying, since in the computational geometry we often assume implicitly that all input elements are distinct. For such inputs we need other arguments. In the case of pairwise distinct real numbers we give a reduction from the *membership problem* in a point set $W$ in $\mathbb{R}^n$. In this problem we have to decide for an input point $x \in \mathbb{R}^n$ if it is contained in $W$. We use the following well known result:

**Theorem 5.1** (Ben-Or [13]) *Any algorithm $C$ that belongs to the algebraic model and solves the membership problem in $W \subseteq \mathbb{R}^n$ need $\Omega(\log N - n)$ worst-case time, where $N = \max(\#W, \#(\mathbb{R}^n \setminus W))$, and $\#W$ and $\#(\mathbb{R}^n \setminus W)$ are the number of disjoint connected components of $W$ and $\mathbb{R}^n \setminus W$.*

Now we prove the following:

**Theorem 5.2** *Let $d \geq 1$ be an integer constant. Any algorithm $A$ that belongs to the algebraic model and computes a Steiner weak spanner for a given set $S \subset \mathbb{R}^d$ of $n$ pairwise distinct point with stretch factor $f \geq 1$, need $\Omega(n \log n)$ worst-case time.*

**Proof:** We prove the theorem for one-dimensional point set. This implies the lower bound for the higher dimensional case. In order to apply Theorem 5.1 we need to define an appropriate algorithm $C$ such that

(*i*) $C$ solves the decision problem for $W \subseteq \mathbb{R}^n$,

(*ii*) $C$ has a running time that is within a constant factor of the running time of $A$,

(*iii*) $W$ consists of many (at least $n!$ in our case) distinct connected components.

In the reduction the input $S$ for algorithm $A$ (which is a set of $n$ real numbers) corresponds to a point in $\mathbb{R}^n$ whose $i$th coordinate is the $i$th real number of $S$. We must take care in the definition of the point set $W$, since we only allow distinct real numbers as inputs for $A$. Consider the following example: For $1 \leq i < j \leq n$ let $h_{i,j}$ be the $(d-1)$-dimensional hyperplane in $\mathbb{R}^n$ whose points have equal $i$th and $j$th coordinate. Let $H = \{h_{i,j} : 1 \leq i < j \leq n\}$ and $U = \mathbb{R}^n \setminus H$ (Figure 5.1). The membership problem for $U$ can be solved in constant time if only pairwise distinct real numbers are allowed as input (the algorithm must simply output YES), although the number of connected components $\#U$ is at least $n!$. To see this, consider two distinct permutations $\pi$ and $\rho$ of $1, 2, ..., n$. Let $p$ and $r$ be the points of $\mathbb{R}^n$ with coordinates $p = (\pi(1), ..., \pi(n))$ and $r = (\rho(1), ..., \rho(n))$. It is easy to see that $p$ and $q$ belongs to distinct connected components of $U$: Let $i$ and $j$ be two indices such that $\pi(i) < \pi(j)$ and $\rho(i) > \rho(j)$. Then any continuous curve $g$ in $\mathbb{R}^n$ between $p$ and $r$ contains a point $q$ whose $i$th and $j$th coordinates are equal and so $q \notin U$. Therefore, $p$ and $r$ are contained in different connected components. It implies that the number of connected components is at least $n!$.

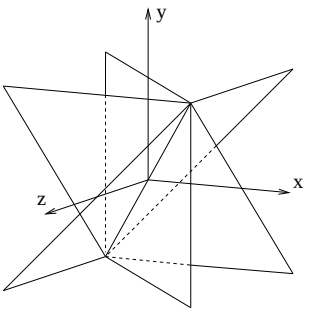


Figure 5.1: The hyperplanes $x = y, x = z$ and $y = z$ in $\mathbb{R}^3$.

The above example shows that we must take care in the definition of the set $W \subseteq \mathbb{R}^n$ and the algorithm $C$ whose valid inputs consist of distinct real coordinates. After the definition of $C$ we define a related algorithm $D$ that takes any point of $\mathbb{R}^n$ as input.

Before introducing algorithm $C$ we define an algorithm $B$ which uses $A$ and will be used by $C$. $B$ does the following on an input consisting of $n$ pairwise distinct real numbers $x_1, ..., x_n$ and a real number $f \geq 1$. It first runs algorithm $A$ on the input $x_1, ..., x_n$ and $f$ to construct a Steiner weak spanner $G$ for this point set with stretch factor $f$. Then $B$ selects the shortest edge of non-zero length of $G$ and outputs the length $ls$ of this edge. Note that because of the Steiner points, $G$ may contain edges with length zero, although the input points are pairwise distinct. Let $T_A(n, f)$ and $T_B(n, f)$ denote the worst-case running time of algorithm $A$ and $B$, respectively. Then $T_B(n, f) \leq T_A(n, f) + o(n \log n)$, because $G$ has $o(n \log n)$ edges.

We now fix an integer $n$ and a real number $f \geq 1$. For any permutation $\pi$ of $1, 2, ..., n$, let $ls_\pi$ be the output of algorithm $B$ for input $\pi(1), \pi(2), ..., \pi(n)$ and $f$. Let $ls^*$ be the minimum of the $n!$ outputs, i.e., $ls^* = \min_\pi\{ls_\pi\}$.

Now we define algorithm $C$. It only accepts inputs of fixed length $n$, consisting of $n$ pairwise distinct real numbers. On input $x_1, ..., x_n$ algorithm $C$ first runs algorithm $B$ with $x_1, ..., x_n$ and $f$. Let $ls$ be the output of $B$. $C$ outputs YES if $ls \geq ls^*$, and NO otherwise.

Since algorithm $C$ only accepts inputs of our fixed length $n$, and since we also fixed $f$, we may assume that $C$ "knows" the value of $ls^*$. Algorithm $C$ exists, although we have not explicitly computed $ls^*$. Therefore, the worst-case running time $T_C(n)$ of $C$ is at most $T_B(n, f) + O(1)$.

Now we define algorithm $D$ which is defined for each input of $(x_1, ..., x_n) \in \mathbb{R}^n$. Algorithm $C$ was defined only for inputs consisting of $n$ pairwise distinct real numbers. As result it can perform some divisions without having to worry whether the denominator is zero (for example divisions in the form $z := y/(x_i - x_j)$). Algorithm $D$ performs the same computation as $C$ on the input $x_1, ..., x_n$ of $n$ not necessary distinct real number, except that each division $z := y/x$ is replaced by

**if** $x = 0$ **then** output NO and terminate **else** $z := y/x$ **fi**.

Since $C$ is a well defined algorithm, if the input consists of $n$ pairwise distinct real numbers then it will always be the case that $x \neq 0$. Therefore, $C$ and $D$ give the same output for each $n$ pairwise distinct real numbers. If the input elements are not pairwise distinct then $C$ is not defined, whereas $D$ is, although its output may not have a meaning at all. Clearly, the worst-case running time of $D$ is within a constant factor

of that of $C$. We prove that the worst-case running time of $D$ is $\Omega(n \log n)$. This also implies the same lower bound for the worst-case running time for $A$.

First we state an important dependence between the output $ls$ of algorithm $B$ and the distance between the closest pair of $n$ distinct real numbers. For $n$ (not necessary distinct) real numbers $x_1, ..., x_n$, let $mingap(x_1, ..., x_n) = \min\{|x_i - x_j| : 1 \leq i < j \leq n\}$.

**Lemma 5.3** *Let $ls$ be the output of algorithm $B$ with $n$ pairwise distinct real numbers $x_1, ..., x_n$ and $f \geq 1$. Then $0 < ls \leq f \cdot mingap(x_1, ..., x_n)$.*

**Proof:** Let $i$ and $j$ be two indices such that $|x_i - x_j| = mingap(x_1, ..., x_n)$. Since the input elements are pairwise distinct, we have $|x_i - x_j| > 0$. Let $G$ be the Steiner weak spanner for the input with stretch factor $f$ constructed by algorithm $A$ and let $P$ be a path in $G$ from $x_i$ to $x_j$ with the property that for each vertex $v \in P$, $|x_i - v| \leq f \cdot |x_i - x_j|$. Clearly, $P$ contains at least one edge of non-zero length. Consider the first such edge $\langle u, v \rangle \in P$. Since each edge on $P$ before $\langle u, v \rangle$ (if any) has length zero, $u$ is coincident with $x_i$. Hence, $ls \leq |u - v| = |x_i - v| \leq f \cdot |x_i - x_j|$. □

In order to apply Theorem 5.1 we now prove that the point set accepted by algorithm $D$ has many connected components. Let $W$ be the set of all points $(x_1, ..., x_n) \in \mathbb{R}^n$ accepted by $D$.

**Lemma 5.4** *The set $W$ has at least $n!$ connected components.*

**Proof:** Let $\pi$ and $\rho$ be two different permutations of $1, 2, ..., n$. Because of the definition of algorithm $C$, the points $p = (\pi(1), ..., \pi(n))$ and $r = (\rho(1), ..., \rho(n))$ are contained in $W$. We show that $p$ and $r$ belong to different connected components of $W$. This will implies the claim of the lemma.

Let $i$ and $j$, $0 \leq i < j \leq n$, be two indices such that $\pi(i) < \pi(j)$ and $\rho(i) > \rho(j)$. Then any continuous curve $g : [0, 1] \to \mathbb{R}^n$, with $g(0) = p$ and $g(1) = r$ intersects the hyperplane $x_i = x_j$. Since $g$ is continuous, it contains points for which the absolute difference between the $i$th and $j$th coordinates is positive but arbitrarily small. Let

$$t_0 = min\{t : 0 \leq t \leq 1, mingap(g(t)) \leq ls^*/(2f)\}.$$

Note that $t_0$ exists, because $g$ passes through the hyperplane $x_i = x_j$, where the function $mingap$ has the value zero, and $mingap$ is continuous along $g$. Let $q = g(t_0)$. Note that $q$ has pairwise distinct coordinates. Let $ls$ be the output of algorithm $B$ started with $q$ and $f$. By Lemma 5.3 we have $ls \leq f \cdot mingap(q) \leq ls^*/2$. Therefore, algorithm $D$ does not accept $q$, and so $q$ does not belongs to $W$.

We have shown that any continuous curve connecting $p$ and $r$ passes through a point outside $W$. Hence, $p$ and $r$ are contained in different connected component of $W$. □

We now return to the proof of Theorem 5.2. Lemma 5.4 and Theorem 5.1 imply that any algorithm that accepts $W$ has a running time $\Omega(\log(\#W) - n) = \Omega(n \log n)$. Since $D$ is such an algorithm, it follows that for our fixed $n$ and $f$, the worst-case running time of $D$ is at least $c\ n \log n$, where $c$ is a positive constant independent of $n$ and $f$. This implies that there is an input on which algorithm $A$ takes at least $c' n \log n$ time for some positive constant $c'$. Since $c'$ does not depend on $n$ and $f$, the lower bound holds for all values of $n$ and $f$. This completes the proof of Theorem 5.2. □

**Remark:** The above reduction does not works if we only allow integer numbers as input: In Theorem 5.2 we showed that each permutation of $1, 2, ..., n$ (more precisely, each point of $\mathbb{R}^n$ with such coordinates) is contained in $W$. But if $W$ only consists of these points and we know that the input consists of integer numbers, we can easily test the membership in $W$ in $O(n)$ time: Consider an array indexed from 1 to $n$ and initialize all its elements to unmarked. We read the input sequentially. When we read the $i$th element $x_i$ we check whether $x_i$ is a number between 1 and $n$. If it is then we set the $x_i$th element of the array to marked. Finally, we have only to check whether all elements of the array are marked.

If we also allow the non algebraic `floor` function and conversion between integer and real numbers in addition to the algebraic functions, we can use the above array to solve the membership problem in $W$ for a point with real coordinates in $O(n)$ time, as well.

**Remark:** In the proof of Theorem 5.2 we defined the value $ls^*$ as the minimum of the outputs of algorithm $B$ over all permutations of $1, 2, ..., n$ as input. If we want to prove Theorem 5.2 only for spanners without Steiner points we can replace the value $ls^*$ by 1, since in each spanner without Steiner points the length of shortest edge is at least the distance between the closest pair of the input point set, which is equal to 1 for each permutation of $1, 2, ..., n$.

# Bibliography


[1] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. Technical Report CS-1997-11, Duke University, Department of Computer Science, 1997. To appear: in Discrete and Computational Geometry: Ten Years Later.

[2] P. K. Agarwal and J. Matoušek. On range searching with semialgebraic sets. *Discrete & Computational Geometry*, 11:393–418, 1994.

[3] A. Aggarwal, M. Hansen, and T. Leighton. Solving query-retrival problems by compact Voronoi diagrams. In *22nd ACM Symposium on Theory of Computing (STOC'90)*, pages 331–340, 1990.

[4] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[5] S. Arora, M. Grigni, D. Karger, and P. Klein A. Woloszyn. A polynomial-time approximation scheme for weighted palnar graph TSP. In *9th ACM-SIAM Symposium on Discrete Algorithms (SODA'98)*, pages 33–41, 1998.

[6] S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid. Euclidean spanners: Short, thin, and lanky. In *27th ACM Symposium on Theory of Computing (STOC'95)*, pages 489–498, 1995.

[7] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. In *5th ACM-SIAM Symposium on Discrete Algorithms (SODA'94)*, pages 573–582, 1994.

[8] S. Arya, D. M. Mount, and M. Smid. Randomized and deterministic algorithms for geometric spanners of small diameter. In *35th IEEE Symposium on Foundations of Computer Science (FOCS'94)*, pages 703–712, 1994.

[9] F. Aurenhammer. A criterion of affine equality of cell complexes in $\mathbb{R}^d$ and convex polyhedra in $\mathbb{R}^{d+1}$. *Discrete & Computational Geometry*, 2:49–64, 1987.

[10] F. Aurenhammer. Voronoi diagrams – a survey of a foundamental geometric data structure. *ACM Computing Surveys*, 23:346–405, 1991.

[11] F. Aurenhammer and R. Klein. Voronoi diagrams. Technical Report TR-198, Fernuniversität Hagen, Praktische Informatik VI, 1996.

[12] M. Barner and F. Flohr. *Analysis II.* Walter de Gruyter, Berlin, 1983.

[13] M. Ben-Or. Lower bounds for algebraic computation trees. In *15th ACM Symposium on Theory of Computing (STOC'83)*, pages 80–86, 1983.

[14] J. L. Bentley and H. A. Maurer. A note on the euclidean near neighbor searching in the plane. *Information Processing Letters*, 8:133–136, 1979.

[15] J. L. Bentley and J. B. Saxe. Decomposable searching problems I: Static-to-dynamic transformation. *J. Algorithms*, 1:301–358, 1980.

[16] M. Bern, L.P. Chew, D. Eppstein, and J. Ruppert. Dihedral bounds for mesh generation in high dimensions. In *6th ACM-SIAM Symposium on Discrete Algorithms (SODA'95)*, pages 189–196, 1995.

[17] M. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. *Journal of Computer and System Sciences*, 48:384–409, 1994.

[18] M. Bern, D. Eppstein, and S. H. Teng. Parallel construction of quadtrees and quality triangulations. In *3rd Workshop on Algorithms and Data Structures (WADS'93)*, pages 188–199, 1993.

[19] F. Bigdeli. *Regular Triangulations of Convex Polytopes and n-Cubes.* PhD thesis, University of Kentucky, Lexington, 1991.

[20] B. Bollobás. *Extremal Graph Theory.* Academic Press Inc. (London) Ltd., 1978.

[21] P. B. Callahan. *Dealing with Higher Dimensions: The Well-Separated Pair Decomposition and Its Applications.* PhD thesis, Johns Hopkins University, Baltimore, Maryland, 1995.

[22] P. B. Callahan and S. R. Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *4th ACM-SIAM Symposium on Discrete Algorithms (SODA'93)*, pages 291–300, 1993.

[23] P. B. Callahan and S. R. Kosaraju. A decompostion of multidimensional point sets with applications to k-nearest neighbors and n-body potential fields. *Journal of the ACM*, 42:67–90, 1995.

[24] B. Chandra, G. Das, G. Narasimhan, and J. Soares. New sparseness results on graph spanners. *International Journal of Computational Geometry & Applications*, 5:125–144, 1995.

[25] B. Chazelle, R. Cole, F. P. Preparata, and C. Yap. New upper bounds for neighbor searching. *Information and Control*, 68:105–124, 1986.

[26] D. Z. Chen, G. Das, and M. Smid. Lower bounds for computing geometric spanners and approximate shortest paths. In *8th Canadian Conference on Computational Geometry (CCCG'96)*, pages 155–160, 1996.

[27] L. P. Chew. There is a planar graph almost as good as the complete graph. In *2nd Annual ACM Symposium on Computational Geometry (SCG'86)*, pages 169–177, 1986.

[28] K. L. Clarkson. Approximation algorithms for shortest path motion planning. In *19th ACM Symposium on Theory of Computing (STOC'87)*, pages 56–65, 1987.

[29] K. L. Clarkson and P. W. Shor. Application of random sampling in computational geometry – II. *Discrete & Computational Geometry*, 4:387–421, 1989.

[30] F. d'Amore, P. G. Franciosa, and G Liotta. A robust region approach to the computation of geometric graphs. In *6th Annual European Symposium on Algorithms (ESA'98)*, pages 175–186, 1998.

[31] G. Das and P. J. Heffernan. Constructing degree-3 spanners with other sparseness properties. *International Journal of Foundations of Computer Science*, 7:11–20, 1996.

[32] M. Dietzfelbinger and F. Meyer auf der Heide. Dynamic hashing in real time. In *Informatik: Festschrift zum 60. Geburtstag von Günter Hotz*, pages 95–119. Teubner, Stuttgart, 1992.

[33] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R. E. Tarjan. Dynamic perfect hashing: Upper and lower bounds. *SIAM Journal on Computing*, 23:748–761, 1994.

[34] D. Dobkin and R. Lipton. On the complexity of computations under varying sets of primitives. *Journal of Computer and System Sciences*, 18:86–91, 1979.

[35] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer Verlag, EATCS Monographs on Theoretical Computer Science, Vol 10, 1987.

[36] H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. *Discrete & Computational Geometry*, 1:25–46, 1986.

[37] D. Eppstein, Z. Galil, and G. F. Italiano. Dynamic graph algorithms. In *CRC Handbook of Algorithms and Theory of Computation, Chapter 22*. CRC Press, 1997.

[38] J. G. Erickson. *Lower Bounds for Fundamental Geometric Problems.* PhD thesis, University of California at Berkeley, 1996.

[39] M. Fischer, F. Meyer auf der Heide, and W.-B. Strothmann. Dynamic data structures for realtime management of large geometric scenes. In *5th Annual European Symposium on Algorithms (ESA'97)*, pages 157–170, 1997.

[40] M. Fischer, T. Lukovszki, and M. Ziegler. Geometric searching in walkthrough animations with weak spanners in real time. In *6th Annual European Symposium on Algorithms (ESA'98)*, pages 163–174, 1998.

[41] M. Fischer, T. Lukovszki, and M. Ziegler. A network based approach for real-time walkthrough of massive models. In *2nd Workshop on Algorithm Engineering (WAE'98)*, 1998.

[42] M. Fischer, T. Lukovszki, and M. Ziegler. Partitioned neighborhood spanners of minimal outdegree. In *11th Canadian Conference on Computational Geometry (CCCG'99)*, pages 47–50, 1999.

[43] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network algorithms. *Journal of the ACM*, 34:596–615, 1987.

[44] M. L. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47:424–436, 1993.

[45] H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30:209–221, 1985.

[46] P. Gupta, R. Janardan, and M. Smid. On intersection searching problems involving curved objects. In *4th Scandinavian Workshop on Algorithm Theory (SWAT'94)*, pages 183–194, 1994.

[47] M. Haiman. A simple and relatively efficient triangulation of the $n$-cube. *Discrete & Computational Geometry*, 6:287–289, 1991.

[48] J. G. Hocking and G. S. Young. *Topology.* Addison-Wesley, 1961.

[49] J. M. Keil. Approximating the complete Euclidean graph. In *1st Scandinavian Workshop on Algorithm Theory (SWAT'88)*, pages 208–213, 1988.

[50] J. M. Keil and C. A. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete & Computational Geometry*, 7:13–28, 1992.

[51] V. King. A simpler minimum spanning tree verification algorithm. *Algorithmica*, 18:263–270, 1997.

[52] V. Klee. On the complexity of $d$-dimensional Voronoi diagrams. *Archiv der Mathematik*, 34:75–80, 1980.

[53] D. T. Lee. On $k$-nearest neighbor Voronoi diagrams in the plane. *IEEE Transactions on Computers*, C-31:478–487, 1982.

[54] C. Levcopoulos, G. Narasimhan, and M. Smid. Efficient algorithms for constructing fault-tolerant geometric spanners. In *30th ACM Symposium on Theory of Computing (STOC'98)*, pages 186–195, 1998.

[55] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36:177–189, 1979.

[56] G. S. Lueker. A data structure for orthogonal range queries. In *19th IEEE Symposium on Foundations of Computer Science (FOCS'78)*, pages 28–34, 1978.

[57] T. Lukovszki. New results on fault tolerant geometric spanners. In *6th Workshop on Algorithms and Data Structures (WADS'99)*, pages 193–204, 1999.

[58] J. Matoušek. Geometric range searching. *ACM Computing Surveys*, 26:421–461, 1991.

[59] J. Matoušek. Reporting points in halfspaces. *Computational Geometry: Theory and Applications*, 2:169–186, 1992.

[60] E. M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14:257–276, 1985.

[61] K. Mehlhorn. *Multi-Dimensional Searching and Computational Geometry, Data Structures and Algorithms 3.* Springer Verlag, 1984.

[62] K. Mehlhorn and S. Näher. Dynamic fractional cascading. *Algorithmica*, 5:215–241, 1990.

[63] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[64] K. Mulmuley. *Computational Geometry, an Introduction through Randomized Algorithms.* Prentice Hall, Englewood Cliffs, 1994.

[65] M. H. Overmars. *The Design of Dynamic Data Structures.* Springer Verlag, 1983.

[66] F. P. Preparata and M. I. Shamos. *Computational Geometry An Introduction.* Springer Verlag, New York, 1985.

[67] M. O. Rabin. Proving simultaneous positivity of linear forms. *Journal of Computer and System Sciences*, 6:639–650, 1972.

[68] S. B. Rao and W. D. Smith. Improved approximation schemes for geometrical graphs via 'spanners' and 'banyans'. In *30th ACM Symposium on Theory of Computing (STOC'98)*, pages 540–550, 1998.

[69] E. M. Reingold. On the optimality of some set algorithms. *Journal of the ACM*, 19:649–659, 1972.

[70] C. A. Rogers. Covering a sphere with spheres. *Mathematika*, 10:157–164, 1963.

[71] J. Ruppert and R. Seidel. Approximating the $d$-dimensional complete Euclidean graph. In *3rd Canadian Conference on Computational Geometry (CCCG'91)*, pages 207–210, 1991.

[72] J. F. Sallee. The middle-cut triangulations of the $n$-cube. *SIAM Journal on Algebraic and Discrete Methods*, 5:407–419, 1984.

[73] J. S. Salowe. Constructing multidimensional spanner graphs. *International Journal of Computational Geometry & Applications*, 1:99–107, 1991.

[74] H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS.* Addison-Wesley, 1990.

[75] H. Samet. *The design and Analysis of Spatial Data Structures.* Addison-Wesley, 1990.

[76] M. Smid. *Closest-Point Problems in Computational Geometry.* Handbook on Computational Geometry, edited by J.-R. Sack, North Holand, Amsterdam, 1998.

[77] J. M. Steele and A. C. Yao. Lower bounds for algebraic decision trees. *Journal of Algorithms*, 3:1–8, 1982.

[78] M. J. Todd. *The Computation of Fixed Points and Applications.* Springer Verlag, Lecture Notes in Economical and Mathematical Systems, Vol 124, 1976.

[79] P. M. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete & Computational Geometry*, 4:101–115, 1989.

[80] P. M. Vaidya. A sparse graph almost as good as the complete graph on points in k dimensions. *Discrete & Computational Geometry*, 6:369–381, 1991.

[81] P. van Emde Boas, R. Kaas, and E. Ziljstra. Design and implementation of an efficient priority queue. *Math. Systems Theory*, 10:99–127, 1977.

[82] A. C. Yao. On constructing minimum spanning trees in $k$-dimensional spaces and related problems. *SIAM Journal on Computing*, 11:721–736, 1982.

[83] A. C. Yao and F. F. Yao. A general approach to $d$-dimensional geometric queries. In *17th ACM Symposium on Theory of Computing (STOC'85)*, pages 163–174, 1985.