Technische Universität München

Fakultät für Mathematik
Lehrstuhl für Angewandte Geometrie und Diskrete Mathematik

Incidence Structures of Power Diagrams

Final Report for an Interdisciplinary Project by Markus Kaiser

Examiner:	Prof. Dr. Peter Gritzmann
Advisor:	M.Sc. Fabian Klemm
Submission Date:	22. October, 2015

I hereby confirm that this is my own work, and that I used only the cited sources and materials.

München, 22. October, 2015

Markus Kaiser

Abstract

Power diagrams are a generalization of Voronoi diagrams which define a cell decomposition of an euclidean space based on a finite set of spheres, assigning every sphere a polyhedral cell for which it minimizes the power function of the points with respect to all spheres. This interdisciplinary project is concerned with the implementation of an algorithm that yields the incidence structure of such a diagram. First, power diagrams are introduced and some fundamental results provided. It continues with the investigation of the close relationship of power diagrams in d dimensions to arrangements of hyperplanes in $d + 1$ dimensions and the convex hull of points obtained via their polarization. This connection gives rise to an efficient algorithm for computing power diagrams, for which both a formal definition and the description of an implementation is presented.

Contents

Chapter 1

Introduction

Power diagrams define a cell decomposition of the euclidean space $\mathbb{R}^d$ into cells defined by a finite set of spheres. A point is part of the cell of some sphere if there is no other sphere for which the power of the point with respect to the sphere is lower. They can be understood as a generalization of Voronoi diagrams with a different distance function with respect to the sphere centers.

Power diagrams can be used to solve geometric problems like finding the volume of a union of spheres [ABI88] or to solve packing problems with spheres [Tot72]. Applications outside of geometry include the solution of weighted balanced clusterings [BG12] and the description of the internal structure of polycrytals [Alp+15].

A way of describing a power diagram is via the adjacency structure of the different cells. Two cells are adjacent if they share points which have the same minimizing power with respect to both spheres. In the euclidean case, they share a polyhedron which is incident to both cells. This interdisciplinary project is concerned with describing power diagrams formally, describing algorithms to construct such incidence structures and implementing them in C++. The theoretical parts of this project are guided by a paper by Aurenhammer [Aur87]. While all statements presented here can be found in [Aur87], only the ones with an explicit citation contain a proof in the original paper. For the other statements, proofs are provided as a service to the reader.

The first chapter of this paper defines power diagrams, introduces some geometric notation and gives both geometric and algebraic interpretations of the distance metric together with some fundamental results about power diagrams. Using a transformation from spheres to hyperplanes, the next chapter shows that there exists an affine equivalency between power diagrams in d dimensions and polyhedra which can be described as the intersection of halfspaces pointing upwards in $d+1$ dimensions. This equivalency is used to also show a dual relationship between power diagrams in $\mathbb{R}^d$ and the convex hull of a set of points in $\mathbb{R}^{d+1}$, which can be exploited to give an efficient algorithm to construct incidence structures of power diagrams. With incidence lattices, a data structure to store them is introduced. In the last chapter of the paper, the implementation of both the algorithms and the incidence lattices accompanying this paper is described.

Chapter 2

Power Diagrams

Power diagrams in d-dimensional Euclidean space $\mathbb{R}^d$ can be understood as a particular generalization of the more well known Voronoi diagrams. A *Voronoi diagram* is a cell decomposition of the space introduced by a finite set of points $M \subset \mathbb{R}^d$ and a metric $\mathrm{d}(\cdot, \cdot)$. It assigns every point $p \in M$ its *region* of points for which there is no point closer than p in M. Using the euclidean metric, these regions are guaranteed to be polyhedra. The normal vectors of the hyperplanes at the boundaries are given by the connecting line of neighbouring points with the hyperplane bisecting this line. Figure 2.1 shows an example of a Voronoi diagram in $d = 2$ dimensions.

While obtaining the hyperplanes which potentially separate the regions is easy, it must be identified which of them actually exist, since not every region is a neighbour of every other region. It can be shown that there exists a dual relationship between Voronoi diagrams in two dimensions and Delaunay triangulations [Aur91]. A *Delaunay triangulation* of a set of points M is a graph in which three points form a clique iff the smallest circle containing all three points does not contain any other point in M. Two regions in the Voronoi diagram then are neighbours iff there exists an edge in the Delaunay triangulation between their corresponding points.

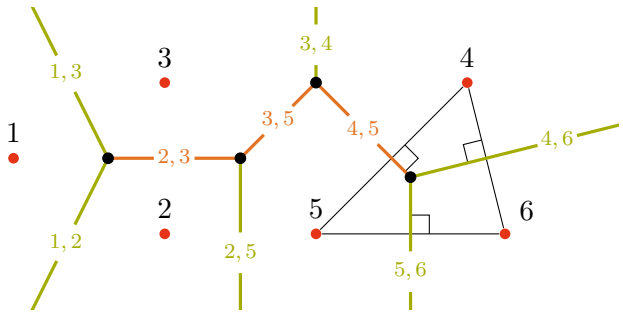


Figure 2.1: 2D Voronoi diagrams partition the plane into regions of points with a common closest input point. The edges separating two regions can be calculated as the perpendicular bisector of the line connecting their centers. Two regions are neighbours if there exists an edge between their points in the Delaunay triangulation of the same set.

This gives rise to efficient algorithms for constructing the diagrams, since the triangulations can be obtained in $\mathcal{O}(n \log n)$ time by incrementally inserting points into an existing triangulation. This algorithm was first described by Green and Sibson [GS78] and is optimal, since one can reduce the sorting problem to finding triangulations [Aur91]. Another algorithm to find Delaunay triangulation described by Aurenhammer involves lifting the two dimensional input into three dimensions by projecting it to a paraboloid. The triangulation can then be found by computing the convex hull of those points, also yielding $\mathcal{O}(n \log n)$ complexity in two dimensions. The efficient algorithm presented in this paper is a generalization of this algorithm for power diagrams.

In this chapter, power diagrams will be defined in terms of the power of a point with respect to a sphere. After introducing some geometric notation, some fundamental results about power diagrams will be shown which in the next chapter can be used to proof a relationship between power diagrams in d dimensions and polyhedra in $d+1$ dimensions.

2.1 Definition of Power Diagrams

There are multiple possible modifications of Voronoi diagrams, a few of which are described in [Aur87]. To obtain *power diagrams*, each point $p \in M$ is assigned a *weight* $w(p)$, where a larger weight results in a larger cell. While the definition given here is applicable to more general metrics, the results presented here assume the usage of the euclidean metric $\mathrm{d}(x, y) = \|x - y\|_2$.

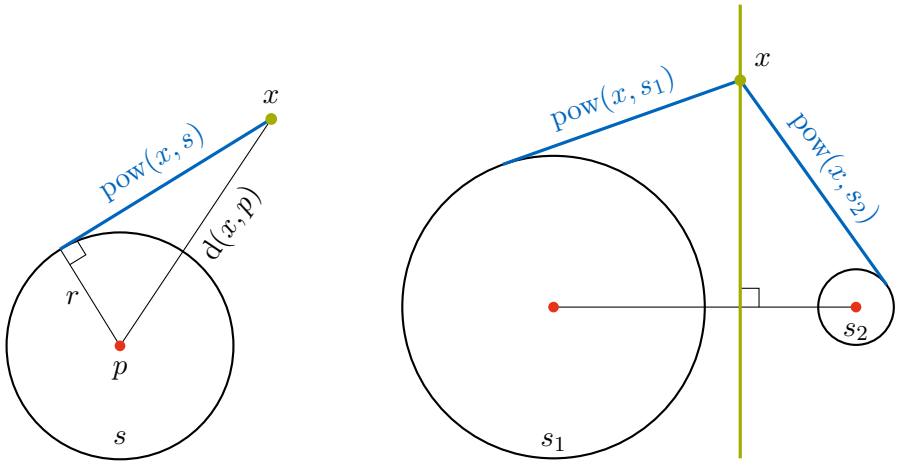
The distance function to be minimized in Voronoi diagrams for points $x \in \mathbb{R}^d$ in a cell is $\mathrm{d}(x, p)$. Power diagrams use the function $\mathrm{d}(x, p)^2 - w(p)$. While other possible combinations of distance and weight have also been investigated [Aur87], this definition has a connection to the power of circles around the points $p \in M$.

Definition 2.1 (Power of a Point)
The *power* of a point $x \in \mathbb{R}^d$ with respect to a sphere $s = (p, r^2)$ with center $p \in \mathbb{R}^d$ and radius $r \in \mathbb{R}_{\geq 0}$ is given by

$$\mathrm{pow}(x, s) = \mathrm{d}(x, p)^2 - r^2. \tag{2.1}$$

The power function was, for example, mentioned at the beginning of the 19th century by Steiner [Ste81] and Laguerre [Bla13] as a measure of the relationship of a point to a circle. A point has a negative power if inside, a positive power if outside and its power vanishes if it lies on the circle. Using the Pythagorean theorem, this function has a geometric interpretation shown in Figure 2.2.

Using this definition, a pair $s = (p, w(p))$ of a point and its weight is called a *site* of the power diagram, which can be interpreted as a sphere with a radius of $\sqrt{w(p)}$. Every such site gets assigned its partition of the space where there is no other site with lower power.



(a) Interpretation of pow **(b)** Equality of powers of two spheres

Figure 2.2: Definition 2.1 has a geometric interpretation for points x lying outside of the sphere $s = (p, r)$. The power $\mathrm{pow}(x, s) = \mathrm{d}(x, p)^2 - r^2$ can be interpreted as the length of a tangent of point p to the sphere s. When comparing the powers of two non-intersecting spheres s_1, s_2, the set of points x for which it holds that $\mathrm{pow}(x, s_1) = \mathrm{pow}(x, s_2)$ is a hyperplane perpendicular to the line connecting the two centers but not necessarily its bisector.

Definition 2.2 (Power Diagram)
For a finite set of sites $S = \{s_1, \ldots, s_n\} \subset \mathbb{R}^d \times \mathbb{R}_{\geq 0}$, the mapping from spheres to their corresponding *cells* is given by

$$\mathrm{cell}_S : \begin{cases} S \to \mathcal{P}(\mathbb{R}^d) \\ s \mapsto \left\{ x \in \mathbb{R}^d \mid \forall t \in S \setminus \{s\} : \mathrm{pow}(x, s) \leq \mathrm{pow}(x, t) \right\} \end{cases}. \tag{2.2}$$

The *power diagram* $\mathrm{PD}(S)$ is the set of all cells corresponding to all sites and their non-empty intersections and given by

$$\mathrm{PD}(S) \coloneqq \left\{ \bigcap_{p \in P} \mathrm{cell}_S(p) \mid P \in \mathcal{P}(S) \setminus \{\varnothing\} \right\} \setminus \{\varnothing\}. \tag{2.3}$$

Figure 2.3 shows an example of a power diagram in two dimensions. After introducing some geometric notation, the remainder of this chapter will show some properties of power diagrams in d dimensions.

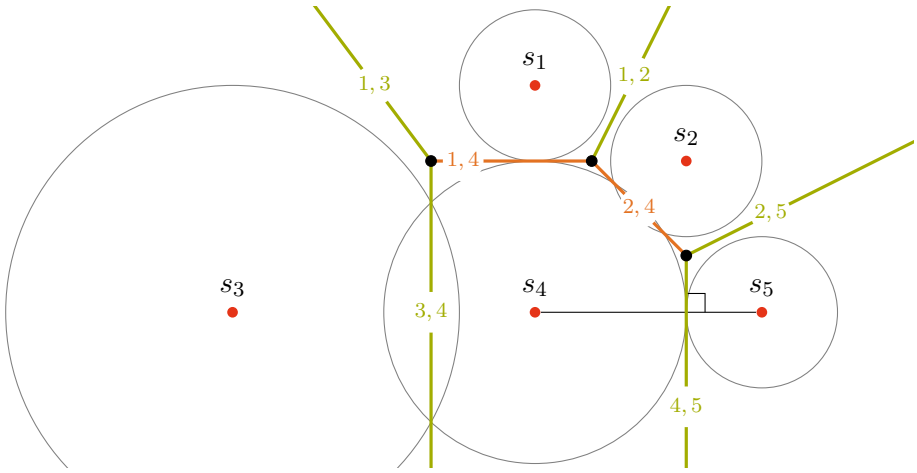


Figure 2.3: The power diagram of the five sites $\mathrm{PD}(\{s_1, \ldots, s_5\})$ is the set of their cells, whose boundaries are shown in the figure. Boundaries are defined by sets of points for which adjacent sites have equal minimizing power.

2.2 Geometric Notation

Like Voronoi diagrams, power diagrams in the euclidean case will be shown to consist of polyhedral regions which share faces at their boundaries. An extensive introduction into polyhedra can be found in [Grü03] by Grünbaum and an introduction into convex analysis is given by Gritzmann in [Gri13]. The following repeats some notations of [Aur87].

Let $n \in \mathbb{R}^d$ with $n \neq 0$ and let $b \in \mathbb{R}$. Then $h = \left\{x \in \mathbb{R}^d \mid n^T x = b\right\}$ is a *hyperplane* and $h^{\leq} = \left\{x \in \mathbb{R}^d \mid n^T x \leq b\right\}$ is a *(closed) halfspace* in $\mathbb{R}^d$. A *j-flat* f is a set expressible as the intersection of $d - j$, but no fewer, hyperplanes. It is an affine subspace of $\mathbb{R}^d$ with dimensionality j and is expressible as the affine hull of $j + 1$ linearly independent points in f.

A set $P \subseteq R^d$ is called a *polyhedron* iff it is expressible as the intersection of a finite number of halfspaces. P is a *j-polyhedron* if there is a j-flat, but no $(j-1)$-flat, which contains P. The boundary of P consists of a finite number of i-polyhedra with $0 \leq i < j \leq d$, which are called the *i-faces* of P. 0-, 1- and $(j-1)$-faces are called *vertex*, *edge* and *facet* respectively. P is called a *polytope* if it is bounded and can then also be expressed as the *convex hull* CH of its 0-faces.

Two d-polyhedra P and Q are called *(combinatorically) dual* if there is a bijection φ from the j-faces of P to the $(d-j-1)$-faces of Q for $0 \leq j \leq d-1$ such that $f \subseteq g$ for any two faces f and g of P iff the converse $\varphi(f) \supseteq \varphi(g)$ holds in Q.

A polytope can be separated into its upper and lower parts with respect to some

vector $v \neq 0$. A facet of the convex hull with outwards pointing normal vector n is part of the *lower convex hull* $\mathrm{CH_b}$ if $\langle n, v\rangle < 0$ and part of the *upper convex hull* $\mathrm{CH_t}$ if $\langle n, v\rangle > 0$. Facets with normal vector orthogonal to v are not attributed to either group. Any other face is in the lower or upper convex hull if it is contained in a facet which is in $\mathrm{CH_b}$ or $\mathrm{CH_t}$ respectively.

A *cell decomposition* C of $\mathbb{R}^d$ is a finite family of polyhedra. In this family, every face of a polyhedron in C is itself a member of C and every non-empty intersection of any two members of C is a face of each of them. Lastly, the union of all polyhedra must be the whole space, i.e. $\bigcup_{f\in C} f = \mathbb{R}^d$. Full-dimensional polyhedra, or d-faces, of C are called *cells*. C and a $(d+1)$-polyhedron P are called *affinely equivalent* if there exists a central or parallel projection φ such that for every face $f \in C$ it holds that $f = \varphi(g)$ for some face g of P.

These notations can now be used to describe power diagrams in an algebraic way and to show that they define a cell decomposition.

2.3 Properties of Power Diagrams

The boundaries of the cells a power diagram are given by sets of points for which the powers with respect to at least two sites are equal. Using the definition of the power function in Definition 2.1, it can be shown that these boundaries are hyperplanes.

Lemma 2.3 ([cf. Aur87, Observation 1])
Let $s = (z_s, r_s^2)$ and $t = (z_t, r_t^2)$ be spheres in $\mathbb{R}^d$ with $z_s \neq z_t$. The points x for which $\mathrm{pow}(x, s) = \mathrm{pow}(x, t)$ *are called the* chordale *of s and t and form the hyperplane*

$$\mathrm{chor}(s,t) = \left\{x \in \mathbb{R}^d \mid 2(z_s - z_t)^T x = r_t^2 - r_s^2 - z_t^T z_t + z_s^T z_s\right\} \tag{2.4}$$

which is perpendicular to the line connecting z_s and z_t.

The chordal of two concentric spheres is not defined, but can be assumed to be at infinity. Note that if $r_s = r_t$, the chordal of the two spheres is the bisector of the line connecting z_s and z_t. A power diagram for which all radii are equal is therefore equivalent to the Voronoi diagram of the centers.

The normal vectors of the chordales of three spheres are given by the pairwise differences of their centers. Since any of these differences can be expressed through the other two, their intersection has higher dimensionality than a general intersection of three hyperplanes can have.

Lemma 2.4 ([cf. Aur87, Observation 2])
Let s, t and u be spheres in $\mathbb{R}^d$ with $d \geq 2$. If the spheres are not collinear, the intersection of their chordales $\mathrm{chor}(s,t) \cap \mathrm{chor}(t,u) \cap \mathrm{chor}(u,s)$ *is a $(d-2)$-flat. If they are collinear, the three chordales are parallel. More generally, the chordales of a sphere with $k \leq d$ non-collinear other spheres in $\mathbb{R}^d$ intersect in a $(d-k)$-flat.*

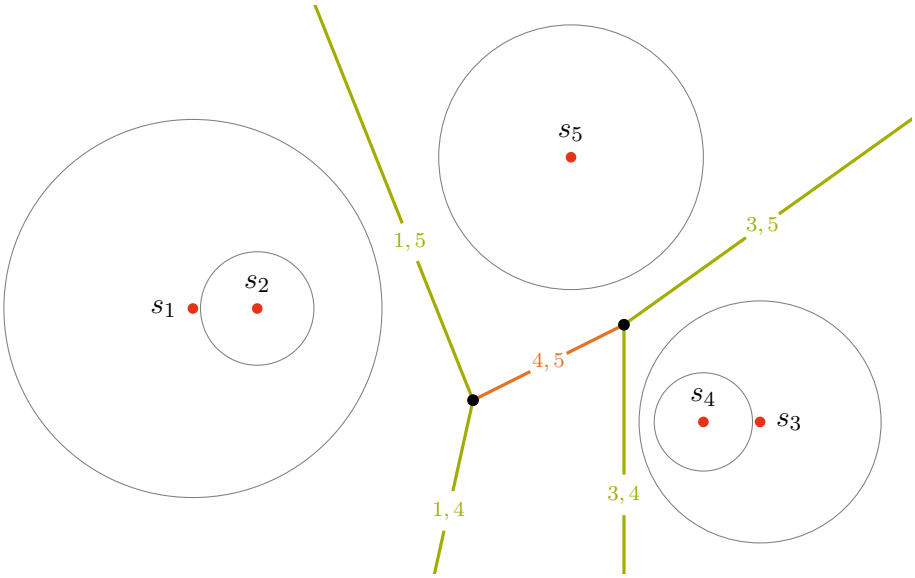


Figure 2.4: The cell of site s_2 in this two-dimensional power diagram is empty. This can happen, if the sphere of a site is completely contained in a sphere of an other site. However, this condition is not sufficient. The sphere s_4 does have a non-empty cell, but its center is not part of its cell.

Every cell in the power diagram is either unbounded or it is bounded by chordales or intersections of other chordales.

Lemma 2.5 ([cf. Aur87, section 2.2, p. 80])
The power diagram $\mathrm{PD}(S)$ *of a finite set of spheres in* $\mathbb{R}^d$ *is a cell decomposition of* $\mathbb{R}^d$.

Proof. Let S be a finite set of n spheres in $\mathbb{R}^d$ whose centers have an affine dimension of d. Since the cell of every sphere $s \in S$ is bounded by halfspaces defined by its $n-1$ chordales (which are not necessarily all part of the power diagram), it can be expressed as the intersection of $n-1$ halfspaces and is therefore a d-polyhedron. Together with the fact that every $x \in \mathbb{R}^d$ is part of some cell of $\mathrm{PD}(S)$ by definition, this shows that $\mathrm{PD}(S)$ is a cell decomposition of $\mathbb{R}^d$, since the combinatorical structure of the shared faces of the polyhedra is guaranteed by definition. □

Figure 2.4 shows that in contrast to Voronoi diagrams, there can be spheres which have empty cells in a power diagram. This can happen if a is sphere completely contained in another sphere. This is not a sufficient condition however. Such a constellation can also lead to a sphere s having a cell for which it holds that $z_s \notin \mathrm{cell}_S(s) \neq \varnothing$.

While it is easy to calculate the chordale of two spheres, the main question to answer when constructing power diagrams is which chordales have non-empty intersection with their cells and which j-faces (for $0 \leq j < d-1$) shared by which cells exist. To

describe all existing bounded faces, one can find all 0-faces and remember which sets of them describe the existing faces of higher dimensionality via their convex hulls. The following lemma shows that if a power diagram contains a 0-face, finding those faces and the cells which are adjacent to them is sufficient to construct a set of all non-empty cells and therefore decide whether some specific cell is empty.

Lemma 2.6
Let $S \subset \mathbb{R}^d \times \mathbb{R}_{\geq 0}$ be a finite set of spheres. If $\mathrm{PD}(S)$ *contains a* 0*-face, then every cell in* $\mathrm{PD}(S)$ *contains a* 0*-face.*

Proof. Let $S = \{s_1, \ldots, s_k\} \subset \mathbb{R}^d \times \mathbb{R}_{\geq 0}$ be a finite set of spheres $s_i = (z_i, r_i^2)$ with $k \geq d + 2$ and $s_1, \ldots, s_d, s_{d+1}, s_{d+2} \in S$ such that (the cell of) s_{d+1} shares a 0-face with $s_1, \ldots, s_d$ in $\mathrm{PD}(S)$. Since the chordales of s_{d+1} with these sites intersect in a 0-face, their d normal vectors $\{z_i - z_{d+1} \mid 1 \leq i \leq d\}$ must be linearly independent, or equivalently, it holds that $\dim(\mathrm{aff}(\{z_1, z_2, \ldots, z_{d+1}\})) = d$.

Suppose that s_{d+2} does not contain a 0-face. Then, a face f of lowest dimension in the polyhedron of s_{d+2} has at least dimension $\dim(f) = 1$, giving the polyhedron a non-empty lineality space.

The vectors in this lineality space must be perpendicular to all chordales of s_{d+2}, since the polyhedron is bounded by them. A subset of the normal vectors of these chordales is the set $C = \{z_i - z_{d+2} \mid 1 \leq i \leq d + 1\}$. Note that

$$\dim(C) = \dim\left(\mathrm{aff}\left(\{z_1, z_2, \ldots, z_{d+2}\}\right)\right) \geq \dim\left(\mathrm{aff}\left(\{z_1, z_2, \ldots, z_{d+1}\}\right)\right) = d \qquad (2.5)$$

and therefore, the lineality space must be empty, which is a contradiction. Note that if $k < d + 1$, the power diagram cannot contain a 0-face and if $k = d + 1$, the power diagram can only contain a zero face if all spheres are adjacent to it. □

Chapter 3

Embedding in $d + 1$ Dimensions

To derive an algorithm that efficiently computes the cells of a power diagram via the incidence structure of the faces of the boundaries, power diagrams of d dimensions will be embedded into $d + 1$ dimensions using a transformation function to map the spheres to hyperplanes. This will lead to a result identifying these power diagrams with polyhedra in $d + 1$ dimensions which can be expressed as the intersection of halfspaces pointing upwards. Using this identity together with a duality mapping for polyhedra will finally connect power diagrams to convex hulls and be the basis of the algorithm to compute power diagrams.

3.1 The Transformation Function

The embedding of power diagrams in a space of one dimension higher relies on the fact that the power function of a sphere in $\mathbb{R}^d$ can be calculated using distances to some hyperplane in $\mathbb{R}^{d+1}$. These hyperplanes can be obtained from the spheres using a transformation function.

Inspect the space $\mathbb{R}^{d+1}$ and let the original space $\mathbb{R}^d$ be identified with the hyperplane $h_0 : x_{d+1} = 0$. Let $\mathcal{H}$ be the set of all hyperplanes not parallel to the x_{d+1} axis

$$\mathcal{H} \coloneqq \left\{ h \subseteq \mathbb{R}^{d+1} \mid \exists a \in \mathbb{R}^d, a_{d+1} \in \mathbb{R} : x \in h \Leftrightarrow x_{d+1} = a^T x_{[d]} + a_{d+1} \right\}. \tag{3.1}$$

Thus, $h_0 \in \mathcal{H}$. Lastly let U denote the paraboloid $x_{d+1} = x_{[d]}^T x_{[d]}$ with $x_{[d]} = (x_1, \ldots, x_d)^T$. Then the transformation function is defined as follows.

Definition 3.1 (Transformation Function)
The *transformation function* Π connects spheres in $\mathbb{R}^d$ to hyperplanes in $\mathbb{R}^{d+1}$ and is given by

$$\Pi : \begin{cases} \mathbb{R}^d \times \mathbb{R}_{\geq 0} \to \mathcal{H} \\ \quad (z, r^2) \mapsto \left\{ x \in \mathbb{R}^{d+1} \mid x_{d+1} = 2x_{[d]}^T z - z^T z + r^2 \right\} \end{cases}. \tag{3.2}$$

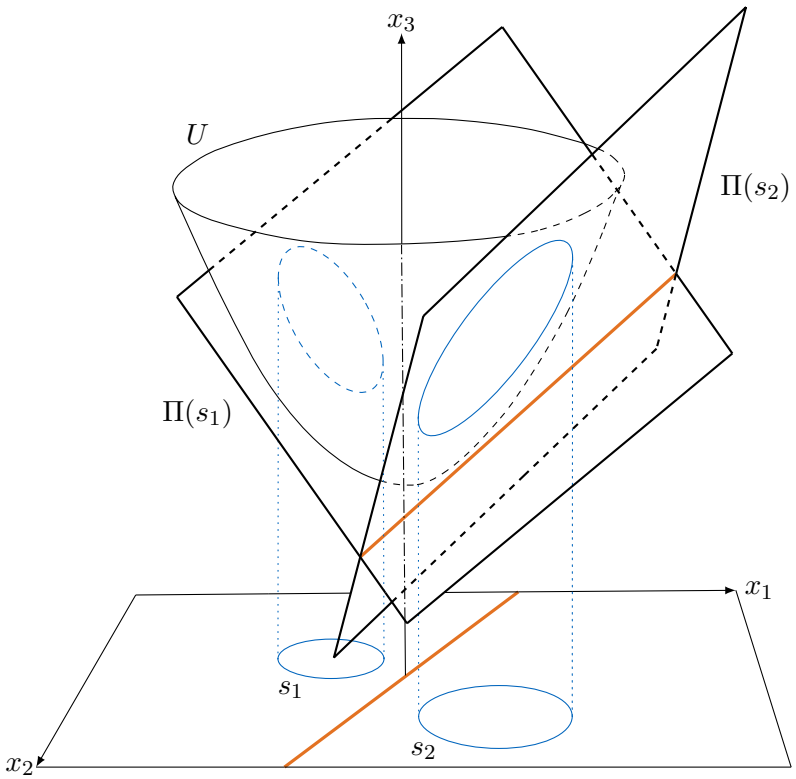


Figure 3.1: The projection function Π maps the spheres s_1 and s_2 to hyperplanes in $d+1$ dimensions. The intersection of the projections $\Pi(s_i)$ and the paraboloid U are the orthogonal projections of s_i onto the paraboloid. The orthogonal projection onto the x_1-x_2-plane of the intersection of the two hyperplanes is the chordale of the two spheres. This figure is a recreation of [Aur87, Figure 4].

This transform has a direct connection to U since for any sphere $s = (z, r^2)$ it holds that

$$\begin{aligned} x \in \Pi(s) \cap U \Leftrightarrow \quad & x_{d+1} = 2x_{[d]}^T z - z^T z + r^2 \wedge x_{d+1} = x_{[d]}^T x_{[d]} && (3.3)\\ \Leftrightarrow \quad & 2x_{[d]}^T z_{[d]} - z_{[d]}^T z_{[d]} + r^2 = x_{[d]}^T x_{[d]} && (3.4)\\ \Leftrightarrow \quad & \mathrm{d}(x_{[d]}, z) = r^2 && (3.5)\\ \Leftrightarrow \quad & x_{[d]} \in z + r \cdot \mathbb{S}^d. && (3.6) \end{aligned}$$

The intersection of $\Pi(s)$ and U is therefore the vertical projection of s onto U. A visualization of this can be seen in Figure 3.1. The following lemma proofs that this mapping is an identification of the two concepts.

Lemma 3.2
The transform Π *is a bijection from spheres in* $\mathbb{R}^d$ *to hyperplanes in* $\mathcal{H}$ *with nonempty intersection with* U*.*

Proof. Clearly, Π is injective since two spheres with different centers map to hyperplanes with different normal vectors n normalized to $n_{d+1} = -1$ and two concentric spheres need to share the same radius r to map to hyperplanes with the same offset. Π is also surjective if every intersection of U with a hyperplane in $\mathcal{H}$ is an ellipsoid which maps to a sphere if projected to h_0.

Let $n \in \mathbb{R}^{d+1}$ be the normal vector with $n_{d+1} = -1$, let $b \in \mathbb{R}$ be an offset and let $h = \left\{ y \in \mathbb{R}^{d+1} \mid n^T y = b \right\}$ be a hyperplane with $h \in \mathcal{H}$. The intersection of h with U is given by

$$h \cap U = \left\{ x \in \mathbb{R}^{d+1} \mid x_{d+1} = b - n_{[d]}^T x_{[d]} = x_{[d]}^T x_{[d]} \right\} \tag{3.7}$$

where the projection to h_0 is

$$b - n_{[d]}^T x_{[d]} = x_{[d]}^T x_{[d]} \tag{3.8}$$

$$\Leftrightarrow \quad x_{[d]}^T x_{[d]} + n_{[d]}^T x_{[d]} - b = 0. \tag{3.9}$$

To show that this projection is a sphere, it must be described by a center $v \in \mathbb{R}^d$ and a radius $\alpha \geq 0$ in the equation

$$\mathrm{d}(x_{[d]}, v)^2 = \alpha. \tag{3.10}$$

Now let $\alpha = b + \frac{1}{4} n_{[d]}^T n_{[d]}$ and $v = -\frac{1}{2} n_{[d]}$. Then

$$\mathrm{d}(x_{[d]}, -\frac{1}{2} n_{[d]})^2 = b + \frac{1}{4} n_{[d]}^T n_{[d]} \tag{3.11}$$

$$\Leftrightarrow \quad x_{[d]}^T x_{[d]} + n_{[d]}^T x_{[d]} + \frac{1}{4} n_{[d]}^T n_{[d]} = b + \frac{1}{4} n_{[d]}^T n_{[d]} \tag{3.12}$$

$$\Leftrightarrow \quad x_{[d]}^T x_{[d]} + n_{[d]}^T x_{[d]} - b = 0, \tag{3.13}$$

so the projection of the intersection is indeed a sphere if $b > -\frac{1}{4} n_{[d]}^T n_{[d]}$, a point if $b = -\frac{1}{4} n_{[d]}^T n_{[d]}$ and h does not intersect U otherwise. □

Because $\Pi(s)$ is the vertical projection of spheres to U, the resulting hyperplanes can also be used to compute the power of some point with respect to s.

Lemma 3.3 ([cf. Aur87, Observation 4])
Let $s = (z, r^2)$ *be a sphere in* h_0 *and* $x \in h_0$*. Let* x' *and* x'' *denote the vertical projections of* x *onto* U *and* $\Pi(s)$*. It then holds that*

$$\mathrm{pow}(x_{[d]}, s) = \mathrm{d}(x, x') - \mathrm{d}(x, x''). \tag{3.14}$$

Proof. Since the projections of x do not change the first d coordinates, it holds that $\mathrm{d}(x, x') = x_{[d]}^T x_{[d]}$ and $\mathrm{d}(x, x'') = 2x_{[d]}^T z - z^T z + r^2$. The difference $\mathrm{d}(x, x') - \mathrm{d}(x, x'')$ is given by $(x_{[d]} - z)^T (x_{[d]} - z) - r^2$, which is equal to $\mathrm{pow}(x_{[d]}, s)$. □

This directly implies the following lemma, which connects chordales to the newly formed hyperplanes by recognizing that the intersection of two hyperplanes in $\mathbb{R}^{d+1}$ contains projected points with the same power value for the respective spheres.

Lemma 3.4
Let s and t be non-concentric spheres in h_0. Then $\mathrm{chor}(s,t)$ is the vertical projection of $\Pi(s) \cap \Pi(t)$ onto h_0.

3.2 Duality of points and hyperplanes

The hyperplanes established with the mapping Π will now be connected to specific *polar points* to establish a bijective mapping between j-flats defined by the intersection of these hyperplanes and $(d-j)$-flats in this dual domain.

Definition 3.5 (Polarity Function)
Let $\mathcal{H}$ as defined above and let $\mathcal{F}$ be the set of all flats which do not contain any vectors parallel to the x_{d+1} axis, i.e.

$$\mathcal{F} \coloneqq \left\{ f \subseteq \mathbb{R}^{d+1} \mid \exists 1 \leq k \leq d+1 : \exists h_1, \ldots, h_k \in \mathcal{H} : \varnothing \neq f = \bigcap_{i=1}^{k} h_i \right\}. \tag{3.15}$$

The *polarity function* Δ then is a mapping in $\mathcal{F}$ given by

$$\Delta : \begin{cases} \mathcal{F} \to \mathcal{F} & \\ f \mapsto \begin{cases} \left\{ \begin{pmatrix} \frac{1}{2}a \\ -a_{d+1} \end{pmatrix} \right\} & \text{if } h \in \mathcal{H} \text{ with } x \in h \Leftrightarrow x_{d+1} = a^T x_{[d]} + a_{d+1} \\ \bigcup_{\substack{h \in \mathcal{H}: \\ f \subseteq h}} \Delta(h) & \text{if } \dim(f) < d \end{cases} \end{cases}. \tag{3.16}$$

Let $h \in \mathcal{H}$ be a hyperplane and $p \in \mathbb{R}^{d+1}$. The images $\Delta(h)$ and $\Delta(p)$ are called the *pole* and *polar hyperplane* of p and h respectively. The following lemma shows that Δ is involutory, i.e. its own inverse, and that it connects flats of different dimensionalities in a dual manner.

Lemma 3.6
Δ is an involutory function and a bijection from j-flats to $(d-j)$-flats in $\mathcal{F}$ for $0 \leq j \leq d$.

Proof. Let $F = f + X \subset \mathbb{R}^{d+1} \in \mathcal{F}$ be some flat with $f \in \mathbb{R}^{d+1}$, X a linear subspace with $\dim(X) = j$ for $0 \leq j \leq d$ and let

$$X^{\perp} = \left\{ n \in \mathbb{R}^{d+1} \mid \forall x \in X : \langle n, x \rangle = 0 \right\} \tag{3.17}$$

be the set of all vectors orthogonal to X for which $\dim(X^{\perp}) = d - j + 1$. Then all hyperplanes containing F have a normal vector which is in $X^{\perp}$. Let $n \in X^{\perp}$ and

$$\begin{aligned} h(n) &= \left\{ x \in \mathbb{R}^{d+1} \mid \langle n, x - f \rangle = 0 \right\} && (3.18) \\ &= \left\{ x \in \mathbb{R}^{d+1} \mid n_{d+1} \cdot x_{d+1} = -n_{[d]}^T x_{[d]} + n^T f \right\} && (3.19) \end{aligned}$$

its corresponding hyperplane containing F. Note that by definition of $\mathcal{F}$, for all $n \in X^{\perp}$ it holds that $n_{d+1} \neq 0$. Then let $N = \left\{ n \in X^{\perp} \mid n_{d+1} = 1 \right\}$ be the set of all possible normalized normal vectors with $\dim(N) = d - j$ and

$$\begin{aligned} \Delta(F) &= \Delta(h(N)) && (3.20) \\ &= \bigcup_{n \in N} \Delta(h(n)) && (3.21) \\ &= \bigcup_{n \in N} \Delta\left(\left\{ x \in \mathbb{R}^{d+1} \mid x_{d+1} = -n_{[d]}^T x_{[d]} + n^T f \right\} \right) && (3.22) \\ &= \bigcup_{n \in N} \left\{ \begin{pmatrix} -\frac{1}{2} n_{[d]} \\ -n^T f \end{pmatrix} \right\} && (3.23) \\ &= \bigcup_{n \in N} \left\{ \begin{pmatrix} -\frac{1}{2} n_{[d]} \\ -n_{[d]}^T f_{[d]} - f_{d+1} \end{pmatrix} \right\}. && (3.24) \end{aligned}$$

$\Delta(F)$ is therefore a $(d-j)$-flat which is the result of an affine transformation of N.

Now assume that $\Delta(F) \notin \mathcal{F}$. Then $\Delta(F)$ is not expressible as the intersection of hyperplanes in $\mathcal{H}$ and therefore, there must exist two different $d^{(1)}, d^{(2)} \in \Delta(F)$ with $d_{[d]}^{(1)} - d_{[d]}^{(2)} = 0$. Let $n^{(1)}, n^{(2)} \in N$ with $\Delta(h(n^{(1)})) = \left\{ d^{(1)} \right\}$ and $\Delta(h(n^{(2)})) = \left\{ d^{(2)} \right\}$. Then it holds that

$$\begin{aligned} 0 &= d_{[d]}^{(1)} - d_{[d]}^{(2)} && (3.25) \\ &= \Delta(h(n^{(1)}))_{[d]} - \Delta(h(n^{(2)}))_{[d]} && (3.26) \\ &= -\frac{1}{2} n_{[d]}^{(1)} + \frac{1}{2} n_{[d]}^{(2)} && (3.27) \end{aligned}$$

and $n_{[d]}^{(1)} = n_{[d]}^{(2)}$. But since $n^{(i)} \in N$ implies that $n_{d+1}^{(1)} = n_{d+1}^{(2)} = 1$ it follows that $n^{(1)} = n^{(2)}$ and therefore $d^{(1)} = d^{(2)}$, which is a contradiction, so $\Delta(F) \in \mathcal{F}$. This proofs the well-definedness of Δ.

The last step is to show that Δ is involutory. It holds that

$$x^* \in \Delta(\Delta(F)) = \bigcup_{\substack{h \in \mathcal{H}: \\ \Delta(F) \subseteq h}} \Delta(h) \tag{3.28}$$

$$\Leftrightarrow \exists h \in \mathcal{H} \quad : \Delta(F) \subseteq h \wedge x^* \in \Delta(h) \tag{3.29}$$

$$\Leftrightarrow \forall x \in \Delta(F) : x_{d+1} = 2\left(x^*_{[d]}\right)^T x_{[d]} - x^*_{d+1} \tag{3.30}$$

$$\Leftrightarrow \forall n \in N \quad : -n^T f = -2\left(x^*_{[d]}\right)^T \cdot \frac{1}{2} n_{[d]} - x^*_{d+1} \tag{3.31}$$

$$\Leftrightarrow \forall n \in N \quad : x^* \in h(n) \tag{3.32}$$

$$\Leftrightarrow x^* \in F \tag{3.33}$$

and therefore $\Delta(\Delta(F)) = F$ for any face $F \in \mathcal{F}$. □

An additional property of Δ is its preservation of relative positions of points and hyperplanes.

Lemma 3.7 ([cf. Aur87, Observation 5])
Let $p \in \mathbb{R}^{d+1}$ and h a hyperplane in $\mathbb{R}^{d+1}$. Then p is above, in or below h with respect to the $(d+1)$th component iff $\Delta(h)$ is above, in or below $\Delta(p)$ respectively.

Proof. Let $x, n \in \mathbb{R}^{d+1}$ and $h = \left\{y \in \mathbb{R}^{d+1} \mid y_{d+1} = n^T_{[d]} y_{[d]} + n_{d+1}\right\}$ a hyperplane in $\mathbb{R}^{d+1}$. Then by definition and by Lemma 3.6 it holds that

$$\Delta(h) \coloneqq \begin{pmatrix} \frac{1}{2} n \\ -n_{d+1} \end{pmatrix} \in \mathbb{R}^{d+1} \tag{3.34}$$

$$\Delta(p) = \left\{y \in \mathbb{R}^{d+1} \mid y_{d+1} = 2x^T_{[d]} y_{[d]} - x_{d+1}\right\}. \tag{3.35}$$

Any point $y \in \mathbb{R}^{d+1}$ is above $\Delta(p)$ if

$$2x^T_{[d]} y_{[d]} - y_{d+1} - x_{d+1} > 0 \tag{3.36}$$

and on or below the hyperplane when it is equal or smaller. Inserting $\Delta(h)$ into the equation yields

$$2x^T_{[d]} \left(\frac{1}{2} n_{[d]}\right) + n_{d+1} - x_{d+1} = n^T_{[d]} x_{[d]} - x_{d+1} + n_{d+1} \tag{3.37}$$

which is equal to the expression determining the relative position of x to h. □

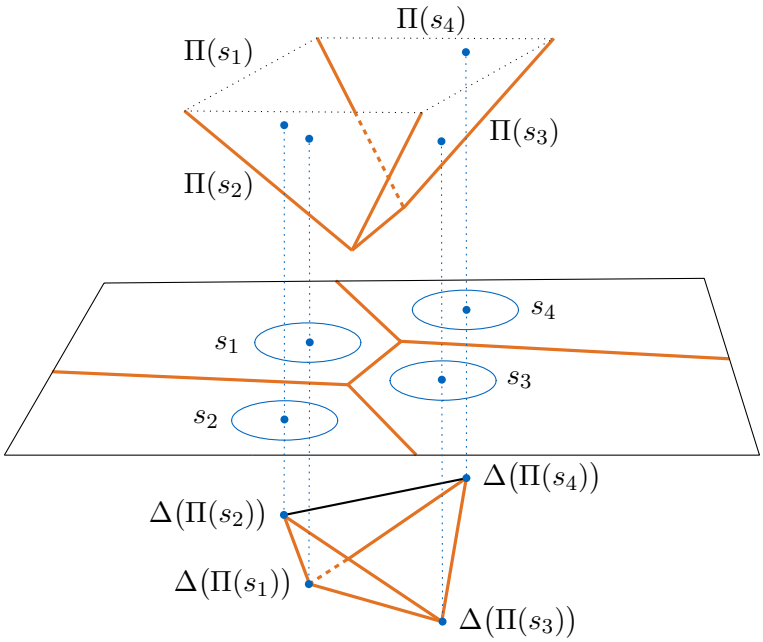


Figure 3.2: The polar point $\Delta(\Pi(s_i))$ of the hyperplane $\Pi(s_i)$ shares the first d coordinates with the sphere center z_i. The polar of a face of lower dimension is the polar point of all hyperplanes containing the face. The polar of the 1-face contained in $\Pi(s_1) \cap \Pi(s_3)$ is the convex hull of the polars of the two hyperplanes. The lower convex hull of the polar points is dual to the power diagram.

3.3 Power Diagrams and Polyhedra

Having established the bijectivity of Π it can be shown that power diagrams in $\mathbb{R}^d$ can be identified with boundaries of $(d+1)$-polyhedra. Since Π provides a mapping to hyperplanes which intersect U, only polyhedra which are bounded by halfspaces defined by such hyperplanes can be considered. Any other polyhedron can however be converted to such a polyhedron by moving it up in x_{d+1}-direction, preserving their combinatorical structure.

Theorem 3.8 ([cf. Aur87, Theorem 4])
For any $(d+1)$-polyhedron P which is expressible as the intersection of upper halfspaces of hyperplanes with normal vectors with their $(d+1)$th coordinate equal to -1, there exists an affinely equivalent power diagram in h_0 and vice versa.

Proof. Let $P = \bigcap_{i=1}^{n} h_i^{\leq}$ be the intersection of upper halfspaces of hyperplanes h_i with normal vectors with $(d+1)$th coordinates equal to 1 such that for facet f_i of P it holds that $f_i \subseteq h_i$. Also assume that $h_i \cap U \neq \varnothing$ for all i which can be achieved by moving the polyhedron in x_{d+1}-direction. This movement does not affect the desired affine equivalence.

Lemma 3.2 implies that there is a set $S = \{s_1, \ldots, s_n\}$ of spheres in h_0 such that $\Pi(s_i) = h_i$. Using Lemma 3.3 it holds that the vertical projection of any $x \in h_0$ onto P is in f_i (and therefore h_i) iff $x \in \text{cell}_S(s_i)$: The point x is in the cell of s_i iff s_i minimizes the power function for this point. Since $\text{pow}(x, s_i) = \text{d}(x, x') - \text{d}(x, x'')$ with x' being its projection to U and x'' being its projection to $\Pi(s_i)$, the distance of x to $\Pi(s_i)$ must therefore be maximal, i.e. f_i must contain x''. This means that $\text{cell}_S(s_i)$ is exactly the vertical projection of f_i onto h_0 for all $1 \leq i \leq n$, giving that $\text{PD}(S)$ is affinely equivalent to P.

Conversely, given any set of spheres S in h_0 a corresponding polyhedron can be constructed by intersecting the upper halfspaces of the hyperplanes in $\Pi(S)$. □

To also describe polyhedra P generated using the intersection of upper halfspaces, one can inspect the reflection of P through h_0 as described in [Aur87].

Having established the existence of an affinely equivalent polyhedron for each set of spheres in h_0, the following theorem will show that using the polarity function Δ, it is possible to relate power diagrams to convex hulls. A power diagram is called dual to a convex hull if there exists a polyhedron such that the power diagram is affinely equivalent to the polyhedron as described in Theorem 3.8 and this polyhedron is dual to the convex hull. An example of this duality can be seen in Figure 3.2.

Remember that a convex hull splits into its top and bottom part with respect to some vector v given by the sign of the scalar product of v with outwards pointing normal vectors, where orthogonal facets are assigned to neither parts. In the following, separations with respect to the x_{d+1}-axis will be considered.

Theorem 3.9 ([cf. Aur87, Theorem 5])
For any finite set $M \subset \mathbb{R}^{d+1}$ there exists a set S of spheres in h_0 such that $\text{PD}(S)$ *is dual to* $\text{CH}_\text{b}(M)$.

Proof. By Theorem 3.8 there exists an affinely equivalent polyhedron P for each $\text{PD}(S)$ in h_0 which is defined by the intersection of the upper halfspaces of the hyperplanes $H = \{h_i \mid h_i = \Pi(s_i), 1 \leq i \leq n\}$ with $n = |S|$ and vice versa. Let $M = \Delta(H)$ be the set of all polars of the hyperplanes corresponding to all spheres. It is to be shown that $\text{CH}_\text{b}(M)$ is dual to P.

Let f_i be the facet associated with h_i (see the proof of Theorem 3.8) and let $p_i = \Delta(h_i)$. Let also f_i and f_j be adjacent in a $(d-1)$-face g $(1 \leq i < j \leq n)$. Then every $x \in g$ is in $h_i \cap h_j$ and above each $h \in H \setminus \{h_i, h_j\}$ via Lemma 3.3.

Lemma 3.7 now implies that exactly in this case it holds that $p_i, p_j \in \Delta(x)$ and every other point $p \in M \setminus \{p_i, p_j\}$ is above $\Delta(x)$. The line defined by p_i and p_j, which is $\Delta(g)$, therefore defines an edge of $\text{CH}_\text{b}(M)$.

An analogous argument holds for all other j-faces $(0 \leq j < d-1)$ by substituting the pair of faces by a set of faces adjacent in a j-face in this proof. P is thus dual to $\text{CH}_\text{b}(M)$ and therefore, $\text{PD}(S)$ is, too. □

Chapter 4

Constructing Power Diagrams

In this chapter, the theoretical results of the previous chapters will be applied to the formulation of algorithms to construct the power diagram $\mathrm{PD}(S)$ of some set of spheres S in d dimensions. Lemma 2.4 implies that the cells of those power diagrams are polyhedra which share faces at their boundaries.

To represent those boundaries, incidence lattices will be introduced. Also based on Lemma 2.4, a naive algorithm constructing all 0-faces can be formulated, which is based on the observation that every such face is defined by at least $(d+1)$ spheres. Lastly, an efficient algorithm based on the results of Chapter 3 and the known problem of constructing convex hulls will be presented and its running time analyzed.

4.1 Incidence Lattices

Theorem 3.8 establishes that for every power diagram in d dimensions there is a $(d+1)$-polyhedron which is affinely equivalent. This means that all chordales which exist in the power diagram and their intersections which form flats of lower dimensionality are represented as faces in this polyhedron as can be seen from Lemma 3.4. To represent all faces which are boundaries of cells in the power diagram, one can therefore choose a data structure which represents the faces of the equivalent polyhedron.

Grünbaum [Grü03] observes that by creating a partial order on faces of a polyhedron induced by set inclusion of the points in the faces, one can obtain the so called incidence lattice of the polyhedron. The two-dimensional case is also described in more detail in [EOS86]. After the following definition, an important result about the size of these lattices will be presented.

Definition 4.1 (Incidence Lattice)
Let P be a polyhedron whose set of non-empty faces is denoted by F. The *incidence lattice* of P is a directed graph

$$\mathrm{IL}(P) \coloneqq (F, E) \tag{4.1}$$

with the set of edges E given by

$$E \coloneqq \left\{ (f_1, f_2) \in F^2 \middle| f_1 \subsetneq f_2 \wedge \nexists f_3 \in F : f_1 \subsetneq f_3 \subsetneq f_2 \right\}. \tag{4.2}$$

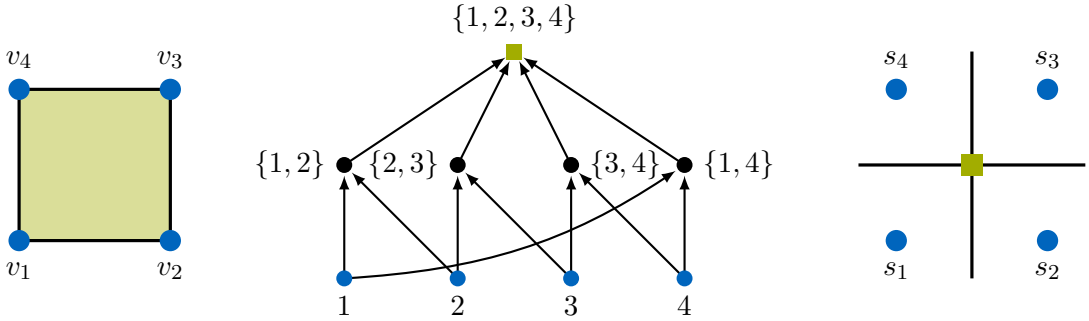


Figure 4.1: The polytope on the left is given as the convex hull of $\{v_1, \ldots, v_4\}$. The minimal nodes of the incidence lattice are the four vertices, while the maximal node is the complete polytope. The 1-faces (edges) form the middle layer of the lattice. The dual power diagram is a Voronoi diagram which can be seen on the right side which has an incidence structure described by the same lattice with reversed edges.

The minimal vertices of an incidence lattice are the faces of lowest dimension in the polyhedron, while the single maximal node is the polyhedron itself.

Lemma 3.6 implies that a dual polyhedron obtained using the polarity function Δ has a dual combinatorical structure compared to the original polyhedron. Since every j-face has been replaced by a $(d-j)$-face, the subset-relationship between two faces are reversed, i.e. if $f_1 \subsetneq f_2$, then $\Delta(f_1) \supsetneq \Delta(f_2)$. This directly yields that the incidence lattice of this dual polyhedron is isomorphic to the graph created by reverting all edges in the original incidence lattice. Together with Theorem 3.8 it follows that the incidence lattice describing the boundaries of a power diagram can be obtained from an incidence lattice of the lower part of a convex hull by reverting all edges in the lattice.

Figure 4.1 shows an example of this relationship. A square defined by four vertices can be interpreted as a degenerated lower convex hull in three dimensions. It is then dual to a Voronoi diagram in two dimensions whose incidence structure is described by the same incidence lattice with reverted edges.

Since every non-empty face at the boundary between two cells is represented by a node in the incidence lattice, the following result by Brondsted [Bro12] bounds the size of incidence lattices describing a power diagram (and therefore the size of incidence lattices of polyhedra which are expressible as intersections of upwards pointing halfspaces).

Lemma 4.2
Let S be a set of n spheres in $\mathbb{R}^d$ with $n > d > 0$ and let f_j denote the maximal number of j-faces of a $(d+1)$ polyhedron with n facets. Then $f_j \in \mathcal{O}(n^{\lceil d/2 \rceil})$ and $\mathrm{PD}(S)$ contains at most n cells, f_j j-faces for $1 \leq j \leq d-1$ and $f_0 - 1$ vertices.

A direct consequence of this lemma is that any algorithm constructing the complete incidence lattice of some power diagram will have a running time in $\Omega(n^{\lceil d/2 \rceil})$ for n spheres in d dimensions. To lessen the impact of higher dimensions somewhat, the implementation presented here will create shallow incidence lattices which only contain the 0-faces (vertices), 1-faces (edges) and d-faces (cells) of a power diagram.

4.2 Naive Algorithm

For any power diagram which contains a cell with a 0-face, every cell contains a 0-face, as stated by Lemma 2.6. The proof of the lemma yields that the existence of a 0-face in a power diagram is guaranteed if there are at least $d+1$ linearly independent spheres, which also is a necessary condition.

Faces are called internal faces of a power diagram if they can be expressed as the convex hull of 0-faces, i.e. if they are polytopal. The naive algorithm presented here uses these observations to find all 0-faces. Starting from the set of all existing 0-faces together with the spheres which define them, it is possible to reconstruct all internal faces inductively.

Since all 0-faces must be defined by at least $d+1$ spheres, the naive algorithm iterates over all groups of $d+1$ spheres and checks whether they define a 0-face as described in Algorithm 1. Operations of the incidence lattice are denoted using addition and subtraction of nodes or edges to simplify the notation. To analyze the running time, the different steps are analyzed separately and lead to a bound derived from the combinatorical structure of $\mathcal{G}$.

Theorem 4.3
Let $S = \{s_1, \ldots, s_n\}$ a set of spheres in d dimensions with $n > d$ and at least $d+1$ spheres linearly independent. Algorithm 1 then finds the 0-faces of $\mathrm{PD}(S)$ in $\mathcal{O}(\binom{n}{d+1} \cdot \max(d^3, n))$ time.

Proof. Line 2 is in $\mathcal{O}(1)$, line 3 in $\mathcal{O}(|\mathcal{G}| \cdot d)$ and the merge phase starting in line 10 can be achieved in $\mathcal{O}(|\mathcal{G}| \cdot d)$, where the factor comes from adding the edges from G_2 to the incidence lattice, assuming that adding an edge to the lattice can be done in $\mathcal{O}(1)$. Finding the intersection p in line 5 amounts to solving a system of linear equations with a matrix of size $(d+1) \times d$, which takes $\mathcal{O}(d^3)$ time. Line 8 needs to check every sphere in S which is not in G and therefore can be bounded by $\mathcal{O}(n)$, while line 9 only operates on the spheres in G and therefore only takes $\mathcal{O}(d)$ time. The running time of this algorithm is therefore clearly dominated by the first loop since lines 5 and 7 could both have to be performed $|\mathcal{G}| = \binom{|S|}{d+1} = \binom{n}{d+1}$ times. This leads to the desired bound. □

If one assumes $d \ll n$, this running time can be approximated as $\mathcal{O}(n^{d+2})$. To construct the complete incidence lattice of the power diagram, this shallow representation

Algorithm 1 Naive approach to finding 0-faces of power diagrams

Let $S = \{s_1, \ldots, s_n\}$ be a set of spheres in d dimensions with $n > d$ and at least $d+1$ spheres linearly independent.

function FindZeroFaces(S)
  IL $\leftarrow \varnothing$ ▷ Create empty incidence lattice
  $\mathcal{G} \leftarrow \binom{S}{d+1}$ ▷ Find the set of all possible groups

  for G **in** $\mathcal{G}$ **do** ▷ Find 0-faces
    $P \leftarrow \bigcap_{s_1,s_2 \in G} \text{chor}(s_1, s_2)$ ▷ Find intersection of all pairs of spheres in G.
    if $P = \{p\}$ **then** ▷ P is a single point
      Choose $g \in G$
      if $\forall o \in S \setminus G : \text{pow}(p, o) \geq \text{pow}(p, g)$ **then** ▷ p is a 0-face
        IL $\leftarrow$ IL $+ G + \{p\} + \{(g, p) \mid g \in G\}$ ▷ Add 0-face to IL

  for 0-faces $p_1, p_2 \in$ IL **do** ▷ Merge overdefined 0-faces
    if $p_1 = p_2$ **then** ▷ Merge p_1 and p_2
      Find G_2 corresponding to p_2
      IL $\leftarrow$ IL $- \{p_2\}$ ▷ Remove p_2 from IL
      IL $\leftarrow$ IL $+ \{(g_2, p_1) \mid g_2 \in G_2\}$ ▷ Add edges from G_2 to p_1
  return IL

has to be filled by adding the internal j-faces for $1 \leq j < d$ and unbounded faces would need to be found. Since this naive algorithm is not applicable in practice due to its running time, these questions were not explored further and it was only used as a test implementation for the efficient algorithm presented in the next section.

4.3 Dual Algorithm

According to Theorem 3.9, an incidence lattice which is equivalent to the power diagram PD(S) of some set of spheres S in d dimensions can be obtained using the lower part of a certain convex hull in $\mathbb{R}^{d+1}$. Seidel [Sei81] describes an optimal algorithm to obtain convex hulls in even dimensions, while Preparata and Hong [PH77] describe one which is optimal for $d \in \{2, 3\}$. The implementation to this paper uses the QuickHull algorithm described in [BDH96]. Their results can be summarized as follows.

Lemma 4.4
Let M be a set of n points in $\mathbb{R}^d$. The convex hull of M can be determined in $\mathcal{O}(n \log n)$ time for $d = 3$ and in $\mathcal{O}(n \log n + n^{\lceil d/2 \rceil})$ time for $d > 3$. These bounds are optimal for $d = 3$ or d even.

Algorithm 2 Power diagrams using embedding in $d+1$ dimensions

Let $S = \{s_1, \ldots, s_n\}$ be a set of spheres in d dimensions.

```
1:  function PowerDiagram(S)
2:      H ← Π(S)                                  ▷ Compute hyperplanes of spheres
3:      P ← Δ(H)                                  ▷ Compute poles of the hyperplanes

4:      IL ← CH(P)                                ▷ Construct convex hull of P
5:      (IL_b, IL_t) ← Split IL into top and bottom parts   ▷ IL_b represents CH_b(P)
6:      for face f in IL_b do                     ▷ Dualize IL_b
7:          if f is a facet then
8:              Find the hyperplane h containing f
9:              Replace f by the polar point Δ(h)
10:         else
11:             Replace j-face f by a (d − j)-face          ▷ 0 ≤ j < d

12:     for vertex v in IL_b do
13:         Project v vertically onto h_0.        ▷ Forget (d + 1)st coordinates
14:     return IL_b
```

To construct a power diagram, it is now enough to follow the path of Chapter 3 to transform it to $d+1$ dimensions, calculate the convex hull $\mathrm{CH_b}$ of the polar points and project the results back to the original space. A more formal description can be found in Algorithm 2. The different steps of the algorithm allow for a straightforward analysis of its running time.

Theorem 4.5 ([cf. Aur87, Theorem 7])
Let S be a set of n spheres in $\mathbb{R}^d$ and denote the amount of time necessary to compute the convex hull of n points in $\mathbb{R}^d$ as $\mathrm{T}_d(n)$. Algorithm 2 then constructs $\mathrm{PD}(S)$ in $\mathcal{O}(\mathrm{T}_{d+1}(n))$ time.

Proof. Lines 1 and 2 can be performed in $\mathcal{O}(n)$ time. Let IL be the incidence lattice constructed by line 4. Then line 5 and the loop at line 6 are bounded by $\mathcal{O}(|\mathrm{IL}|)$. Since line 4 must take at least $\Omega(|\mathrm{IL}|)$ time, the construction of the convex hull of the polar points dominates the running time of the algorithm. □

Since Lemma 4.4 ensures the optimality of the convex hull algorithms and every other operation in Algorithm 2 is dominated by them, this is an optimal algorithm to construct power diagrams in d dimensions for $d = 2$ or d odd.

Chapter 5

Implementation

In addition to their theoretical description, the data structure and algorithms presented in Chapter 4 were also implemented as part of this project. The implementation is written in C++, making heavy use of features in the C++11 standard which introduces a more functional and generic style to the language. References for these features can be found in the newer editions of [Str86] and [Str14] and also in [Mey14]. To improve readability, code presented in this paper will be pseudo code rather than concrete C++.

This chapter first describes the general architecture of the code written for this interdisciplinary project. It continues by introducing a concrete implementation of the incidence lattices presented in Section 4.1 based on directed graphs and then describes how it is used to implement the dual algorithm.

5.1 Code Architecture

The program uses a command line interface which is mainly implemented in powerdiagram_main.cpp. The coordinates of the sphere centers and their radii are read from textfiles, a parser for which is implemented in FromCSV.hpp. Based on parameters supplied on the command line, both the naive algorithm of Section 4.2 implemented in PowerDiagramNaive.hpp or the dual algorithm presented in Section 4.3 and implemented in PowerDiagramDual.hpp can be performed.

Both of the algorithms return an incidence lattice which contains the 0-faces present in the power diagram. The dual algorithm also calculates the 1-faces and the directions of unbounded 1-faces. The incidence lattices are implemented in IncidenceLattice.hpp and depend on an internal graph representation which is implemented in BidirectionalGraph.hpp.

While the naive algorithm only depends on a permutation generator implemented in AllChoices.hpp, the dual algorithm requires a convex hull algorithm. The implementation supplies an interface to one such algorithm implemented in ConvexHullQhull.cpp which is described in Section 5.3 and Appendix A.

In the following, both the implementations of the incidence lattices and the dual algorithm will be described in more detail.

5.2 Incidence Lattices

The incidence lattices described in Section 4.1 and [EOS86] can be used to represent the subset-relationships between different faces of a polyhedron. Since it was shown in Theorem 3.8 that a power diagram in d dimensions can be represented by a polyhedron in $d+1$ dimensions, incidence lattices can also be used to represent the cells of power diagrams via their shared boundaries.

According to Definition 4.1, incidence lattices are graphs where every face is a node and two faces are connected with a directed edge if the source is a true subset of the sink with no faces in between. The implementation of these lattices in IncidenceLattice.hpp uses the graph data structure in BidirectionalGraph.hpp as an internal representation. These graphs represent finite graphs with directed edges. They are called bidirectional since it is possible to traverse the edges in both directions in $\mathcal{O}(1)$ time.

To achieve this, every node is identified by some unique (numeric) identifier. The graph is then modelled as a hash-map from those identifiers to entries. The entries are tuples of two incidence lists and some value the node holds. In the case of incidence lattices this value is used to store normal vectors or points. Element access of the graph map has an amortized cost of $\mathcal{O}(1)$. The incidence lists for every node hold both all successors and predecessors, allowing the direct access in either direction. Maintaining both incidence lists increases the work necessary to insert and delete both nodes and edges in terms of constant factors, but not in terms of complexity. With n being the number of nodes, deletion of a node is in $\mathcal{O}(n)$ and the other operations are in $\mathcal{O}(1)$.

Incidence lattices are Hasse diagrams and therefore directed acyclic graphs. A node in such a graph is called *minimal* if it does not have a predecessor and *maximal* if it does not have a successor. Internal faces in the incidence lattice can be described as the convex hull of their vertices. Since there are no non-empty faces which are true subsets of vertices, they are the minimal nodes of an incidence lattice. Finding the vertices which define internal faces therefore translates to finding the minimal nodes in the graph which are connected to the face. For unbounded faces, the minimal nodes connected to it also represent all adjacent vertices. Their convex hull does not produce the face however, since it contains rays.

While constructing power diagrams, an empty incidence lattice has to be filled using the information generated by some convex hull algorithm. During this process, the lattice is usually *shallow* in the sense that it contains minimal nodes (the vertices) and maximal nodes (the d-dimensional facets), but it does not yet contain all j-faces for $1 \leq j < d$. Adding a face which is not minimal requires finding existing edges in the lattice which get cut when inserting the new face, an example of which can be seen in Figure 5.1.

Every face which is not minimal needs to (not necessarily directly) be connected to the minimal faces it contains, which in non-degenerate cases are vertices. Similarly, it

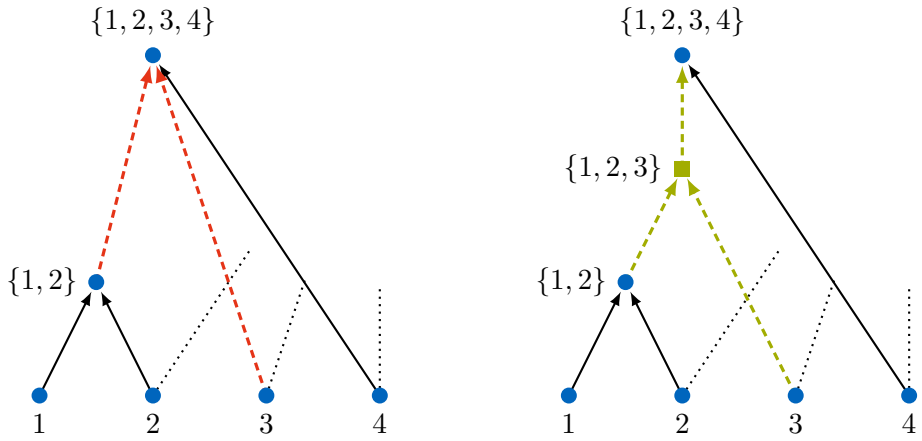


Figure 5.1: The left figure shows part of an incidence lattice in the process of being filled before the insertion of the new face $\{1, 2, 3\}$. To identify the red dashed edges which need to be bisected by the new face, lower and upper boundaries have to be found. The face $\{1, 2\}$ is a largest subset of the new face and therefore an outgoing edge needs to be replaced. The face $\{1, 2, 3, 4\}$ is a smallest superset of the new face and therefore, the new face has to point to it.

also needs to be connected to faces of higher dimensionality it is a subset of. Since the lattice must not contain transitive edges, it is necessary to identify the correct edges to be modified. In the following, the algorithm to find those edges will be described by the example defined in Figure 5.1.

The algorithm for finding those maximal and minimal relevant faces can be seen in Algorithm 3. Given a shallow incidence lattice as described in the left side of Figure 5.1, inserting a new face into an incomplete lattice requires finding all the edges in the existing lattice which have to be bisected by the new face.

Suppose the face $f_{123} = \{1, 2, 3\}$ which represents the intersection of the cells represented by the nodes 1, 2 and 3 should be added to the lattice. Since it already contains the face $f_{1234} = \{1, 2, 3, 4\}$, there must be paths connecting those three minimal nodes to f_{1234}. The structure of the power diagram gives that $f_{1234} \subseteq f_{123}$ and therefore, the lattice must contain the edge (f_{123}, f_{1234}) after the update. The direction of the edge is a convention and could be chosen differently, depending on whether the lattice is based on the primal or dual polyhedron. For the same reason, the lattice must also contain the edge $(3, f_{123})$, which would make the edge $(3, f_{1234})$ transitive, so it has to be removed. However, the edge $(2, f_{1234})$ does not exist because of the face $f_{12} = \{1, 2\}$. Lastly, suppose there existed a face $f_{12345} = \{1, 2, 3, 4, 5\}$ in the lattice. Then although $f_{12345} \subseteq f_{123}$ holds, the edge (f_{123}, f_{12345}) must not be added, since after connecting f_{123} to f_{1234}, this edge would be transitive.

Algorithm 3 Adding faces to an incidence lattice

Let IL be an incidence lattice, V_{minimals} the set of vertices to be combined to a face, FILTER and CONTINUE predicates mapping from nodes to boolean truth values and NEXTNODES mapping from nodes to sets of nodes.

function FINDNODESBFS(v_0, FILTER, CONTINUE, NEXTNODES)
    $V_{\text{visited}} \leftarrow \{v_0\}$
    $V_{\text{tovisit}} \leftarrow$ Queue containing only v_0
    $V_{\text{found}} \leftarrow \{\}$
    while V_{tovisit} **is not** empty **do**
        $v \leftarrow$ Pop the first element of V_{tovisit}
        if CONTINUE(v) **then**
            for n **in** NEXTNODES(v) **do**
                if n **not in** V_{visited} **then**
                    $V_{\text{visited}} \leftarrow V_{\text{visited}} \cup \{n\}$
                    Append n to V_{tovisit}
        if FILTER(v) **then**
            $V_{\text{found}} \leftarrow V_{\text{found}} \cup \{v\}$
    return V_{found}

function FINDGROUPS(IL, V_{minimals}) ▷ Find maximal faces below of V_{minimals}
    function ISGROUP(f)
        return MINIMALSOF(IL, f) $\subseteq V_{\text{minimals}}$
    $\mathcal{G} \leftarrow \{\}$
    for $v \in V_{\text{minimals}}$ **do**
        $G \leftarrow$ FINDNODESBFS(IL, v_0, ISGROUP, ISGROUP, SUCCESSORS(IL, $\cdot$))
        $G \leftarrow \{g \in G \mid \nexists g' \in G : (g, g') \in \text{IL}\}$ ▷ Choose maximal nodes in G
        $\mathcal{G} \leftarrow \mathcal{G} \cup G$
    return $\mathcal{G}$

function FINDLUBS(IL, V_{minimals}) ▷ Find minimal faces above of V_{minimals}
    function ISUB(f)
        return $V_{\text{minimals}} \subseteq$ MINIMALSOF(IL, f)
    function ISNOTUB(f)
        return not ISUB(f)
    Choose $v_0 \in V_{\text{minimals}}$
    $G \leftarrow$ FINDNODESBFS(IL, v_0, ISUB, ISNOTUB, SUCCESSORS(IL, $\cdot$))
    $G \leftarrow \{g \in G \mid \nexists g' \in G : (g', g) \in \text{IL}\}$ ▷ Choose minimal nodes in G
    return G

procedure ADDFACE(IL, V_{minimals})
    $V_{\text{below}} \leftarrow$ FINDGROUPS(IL, V_{minimals})
    $V_{\text{above}} \leftarrow$ FINDLUBS(IL, V_{minimals})
    IL $\leftarrow$ IL $- \{(a, b) \mid a \in V_{\text{above}}, b \in V_{\text{below}}\}$ ▷ Remove old edges
    IL $\leftarrow$ IL $+ \{V_{\text{minimals}}\} + \{(b, V_{\text{minimals}}) \mid b \in V_{\text{below}}\}$ ▷ Add new face and edges

To identify the existing nodes which are incident to edges which should be bisected, faces *above* and *below* in the lattice are treated separately. A face is considered *above* another face in the lattice if the set of minimal nodes it combines is a superset of the minimal nodes of the other face and considered *below* if it is a subset. In the example, f_{1234} is above of f_{123} while f_{12} is below it. Adding the face f_{123} now requires to bisect exactly those edges which connect maximal faces below it to minimal faces above it. A face f is considered maximal below (or a *group*) of f_{123} if there is no face which is also below f_{123} but above f and minimal above (or a *lub*) in a symmetric manner. There cannot exist other edges which have to be bisected since those would be transitive.

To find both groups and lubs, the program uses breadth first search. The groups can be found by starting from each of the minimal nodes associated with the new face f_{123} (i.e. 1, 2 and 3) and only continuing to new nodes which are also below the new face, i.e. whose minimal nodes are a subset of the minimal nodes of the new face. After finding all candidates, every node which is not maximal has to be removed. The example in Figure 5.1 contains two groups, the faces $\{1, 2\}$ and 3. The lubs can be identified similarly by starting from any one of the minimal nodes of the new face and exploring the lattice to find nodes which are above the new face. Note that those faces must all be above this minimal node. Successors of faces which are above the new face need not be considered. Again, after finding the candidates, every node which is not minimal must be removed. The example contains one lub, the face f_{1234}.

The graph data structure contains a generalized search which can be called using function parameters to implement the similar searches described above. All searches have a worst-case running time of $\mathcal{O}(n^2)$ with n being the number of faces and therefore nodes in the graph. To speed up the program, the incidence lattices cache the minimal nodes of every node they contain, making the MinimalsOf-function an $\mathcal{O}(1)$ operation. This is possible since it is not necessary to remove faces besides the restriction operations.

This data structure forms the basis for both the naive algorithm and the efficient algorithm using the convex hull of polar points. Since the implementation of the naive algorithm in PowerDiagramNaive.cpp follows Algorithm 1 rather directly, the following section will only describe the implementation of the efficient algorithm.

5.3 Dual Algorithm

The dual algorithm can be found in PowerDiagramDual.cpp. The main challenge in its implementation is the construction of the convex hull of the polar points, which are obtained by applying the functions Π and Δ to the input spheres. Since the creation of a convex hull is a known geometrical problem, this implementation uses libraries to perform the task.

While there exists a multitude of algorithms to construct convex hulls in both the two and three-dimensional cases, there are surprisingly few alternatives available for the general d-dimensional case. To make it possible to easily switch the algorithm used, the concrete convex hull algorithms are encapsulated and are required to implement a simple interface which takes a set of points and returns an incidence lattice. Since these lattices are of exponential size in general, the algorithms are expected to return a shallow version which only contain vertices, facets and ridges ($(d-2)$-faces).

This implementation uses the generalization of the QuickHull algorithm for arbitrary dimensions, a description of which can be found in [BDH96] and an interface to which is implemented in ConvexHullQhull.cpp. QuickHull assumes that the input is full-dimensional. It starts by creating a simplex from $d+1$ points and then iteratively adds the remaining points to the convex hull. It does so by inspecting the facets of the current convex hull and finding all points in the upper halfspace of a hyperplane containing the facet with a normal vector which points to the outside of the convex hull. If a point is above a facet in this sense, the facet is called *visible* from this point. Out of the visible points for some facet the one with maximal distance to it gets chosen. It is added to the convex hull by creating new simplices using horizontal ridges. A ridge is called *horizontal* for a point if it connects two facets out of which one is visible to the point and one is not. For well-behaved inputs, this operation should also add some of the points closer to the facet to the convex hull. Those do not have to be considered in later iterations. QuickHull is therefore a variant of the Beneath-Beyond Algorithm described in [Grü63] and [CS89]. An alternative implementation of convex hulls in general dimensions can be found in the CGAL library [CGA15].

After obtaining the shallow incidence lattice of the convex hull of the polar points, this lattice needs to be split into the top and bottom parts as described in Algorithm 2. Since only the bottom lattice is needed to construct the power diagram, all faces which are not contained in a bottom-facing facet are removed from the incidence lattice. To dualize the incidence lattice, the only non-trivial operation is to replace facets with polar points of hyperplanes containing the facet. Replacing j-faces by $(d-j)$-faces can be achieved by reversing all edges in the lattice or, equivalently, by interpreting minimal nodes in the lattice as the maximal ones of the dual polyhedron and considering predecessors instead of successors for the subset relationship. See Algorithm 4 for a more formal description of this part of the algorithm.

If the convex hull algorithm returns a shallow incidence lattice which, besides the vertices and facets, only contains ridges, the dualized lattice only contains edges (1-faces) of the power diagram. To facilitate the output of these edges, an additional step of the algorithm is the calculation of their directions. For internal edges, the direction can be calculated as the difference of the two vertices which define the edge via their convex hull. For unbounded edges, the direction must be contained in the chordales of all pairs of spheres defining the edge (i.e. the minimal nodes of the node representing the edge in the lattice). Assuming there is a 0-face in the power diagram,

Algorithm 4 Finding IL_b and dualizing it

Let IL be the incidence lattice of the convex hull of polar points.

```
1:  procedure RestrictToBottomHull(IL)
2:      V_bottoms ← {}
3:      for f in MaximalNodes(IL) do                    ▷ f is a dual facet
4:          V_minimals ← MinimalsOf(f)                  ▷ V_minimals contains dual vertices
5:          n ← OutwardsNormalToAffineSpace(V_minimals)
6:          if n_{d+1} < 0 then                         ▷ n points downwards
7:              V_bottoms ← V_bottoms ∪ {f}
8:      for f in IL do
9:          if MaximalsOf(f) ∩ V_bottoms = ∅ then
10:             RemoveFace(IL, f)

11: procedure DualizeLattice(IL)
12:     for f in MaximalNodes(IL) do
13:         Find the hyperplane h containing f
14:         Replace f by Δ(h)
                                                        ▷ No replacement of other faces necessary
```

Algorithm 5 Finding the direction of an extremal ray

Let IL be the incidence lattice of a power diagram, let f be an unbounded 1-face and v a 0-face with $v \subset f$.

```
1:  function FindDirectionOfUnboundedEdge(IL, f, v)
2:      V_edge ← MinimalsOf(f)                          ▷ V_edge contains primal spheres
3:      V_vertex ← MinimalsOf(v)
4:      Choose s_active ∈ V_edge
5:      Choose s_inactive ∈ V_vertex \ V_edge

6:      d ← NormalToAffineSpace(V_edge)                 ▷ Possible direction
7:      t ← v + d                                       ▷ Test point
8:      if pow(t, s_inactive) < pow(t, s_active) then
9:          d ← (−1) · d
10:     return d
```

the unbounded edge is a ray starting from some vertex (see Lemma 2.6). Since a vector orthogonal to all chordales could point both inwards and outwards, the correct direction can be established by testing the power of one point as described in Algorithm 5.

Adding the directions of edges is the last step in constructing a (shallow) incidence lattice describing the power diagram. This lattice can be used to extract information about the power diagram like finding spheres whose cells are non-empty or identifying adjacency relationships between spheres. Appendix B describes how the implementation can be used to generate output which can be transformed to a tikz-picture like Figure 2.3 or Figure 2.4.

Chapter 6

Conclusion

This paper introduced power diagrams which are a generalization of Voronoi diagrams. It adds the notion of a real weight to every site of the diagram and introduces a new distance function, the power of a point with respect to a site. The power function gives rise to a geometric interpretation which identifies the site centers and weights with spheres. The power diagram of a set of spheres is then a collection of polyhedral cells where each cell is generated by a sphere. A point is part of its cell if a sphere minimizes the power function in comparison to all other spheres.

After introducing some general properties of power diagrams, the main focus of the project lies on the affine equivalency of power diagrams in d dimensions and those polyhedra in $d+1$ dimensions which can be described as the intersection of half spaces which point upwards in x_{d+1} direction. Using a polarity function which maps these polyhedra to combinatorically dual ones, a dual relationship between power diagrams and convex hulls in $d+1$ dimensions can be established.

This relationship can be used to derive an efficient algorithm to construct power diagrams whose main geometric operation is the construction of a convex hull. After describing this algorithm formally and introducing a data structure which can be used to represent polyhedra and therefore power diagrams, this paper lastly presents a concrete implementation of the results.

Appendix A

Building the Code

The implementation presented in Chapter 5 is written in C++ making use of features introduced into the language in the C++11 standard. These features have been part of all major C++ compilers for some time. Usage of the standard should therefore not pose problems for the build process on reasonably new systems. The build process has been tested with GCC and Clang on Linux and MSVC on Windows.

The code depends on libraries for the processing of command line parameters, for linear algebra and for the calculation of convex hulls.

gflags Gflags is a library for the simple processing of command line parameters developed by Google [SS15]. It is licensed under the BSD license which poses minimal restrictions on the redistribution of the covered software. Gflags is a comparatively small and compact library which is not quite as feature rich as alternatives like the program options of Boost, but is much easier to integrate in a small project and provides enough possibilities to create a simple command line interface to choose between different algorithms and outputs.

Eigen Eigen is an open library for fast and reliable linear algebra developed by Guennebaud and Jacob [G+10]. It is licensed under the MPL license which is aims to find a compromise between proprietary and open source developers and also poses little restrictions on the redistribution. Eigen provides both data structures for vectors and matrices and also implements many important primitives of linear algebra like the solution of systems of linear equations.

Qhull Qhull is a library which implements the QuickHull algorithm for convex hulls and is developed by Barber, Dobkin, and Huhdanpaa [BDH96]. It is licensed under a custom license which allows both the redistribution and alteration of the code. Qhull is one of the few choices for libraries implementing convex hulls in arbitrary dimensions and has been chosen for this project for its speed and numerical stability.

To simplify the compilation, the implementation of this interdisciplinary project includes CMake scripts which automate most of the build process. CMake [MH07] is a tool which allows programmers to specify how a program should be compiled in a

```
# Debug build using bundled libraries
$ make
# Clean bundled libraries and program
$ make distclean

# Release build using system libraries
$ make CMAKE_BUILD_TYPE=Release USE_BUNDLED_DEPS=0
# Clean program only
$ make clean
```

Listing 1: A simple invocation of `make` creates a debug build using bundles libraries. The computer will download the libraries, build them and then build the power diagram program. To specify the creation of an optimized release build, the variable `CMAKE_BUILD_TYPE` can be used, while `USE_BUNDLED_DEPS` can be set to 0 to use system libraries. `make clean` and `make distclean` can be used to remove the compiled files.

way which is independent of operating systems and compiler or IDE choices. CMake generates project files in a toolchain of the user's choice. It can, for example, create Makefiles for a command line based build on Linux or it can create solutions which can be used in Visual Studio on Windows. There are many parameters and customizable choices to create build files using CMake. The following two sections will describe a typical build process on both Linux and Windows, which can be customized via options described in the documentation of CMake [MH15].

A.1 Linux

The CMake scripts can either use libraries installed on the target system (for example via a package manager) or compile all libraries from source. The latter is achieved by separating the build process into two steps, first downloading and building all dependencies and then using those dependencies to create the final program.

Since a correct usage of these different steps can be cumbersome, the implementation also provides a top level MAKEFILE which automates the process. This make script will invoke CMake to build the library dependencies or only build the main program. It can be configured using variables shown in Listing 1.

A.2 Windows

Since Windows does not support Makefiles by default, the build process has to be completed manually. After installing CMake, the easiest way to use it on Windows is to use the `cmake-gui` program, which offers a graphical interface. The general structure

of the build remains the same. If the bundled libraries should be built, CMake has to be invoked twice, once for the libraries and once for the program itself.

The most common compiler on Windows is the MSVC compiler by Microsoft, which is usually installed alongside Visual Studio. Both the libraries and the program can be built in both 32 and 64 bit and while untested, using alternative compilers should be possible. It is important to either build both parts with the same generator and to either build both with debug information or both parts without it. While it is possible to choose in Visual Studio in which mode to build the program, the choice has to be made in CMake in case of the libraries.

To create a build on Windows, start with opening `cmake-gui` and creating the build files for the libraries. Figure A.1 shows the correct path to ensure that the later step will automatically find the dependencies. The source CMake files can be found in THIRD-PARTY, while the binaries should be built into the folder .DEPS (including the period).

After choosing the folders, click the "Configure"-button and choose the desired generator as shown in Figure A.2. CMake checks whether the compiler works correctly and searches for correct paths. By default, the libraries will be built with debug information. If this is not desired (e.g. for an optimized build), change the `CMAKE_BUILD_TYPE` variable as shown in Figure A.3.

After pressing the "Generate"-button, .DEPS should contain a Visual Studio solution file. Open the solution with Visual Studio and start a build process, ignoring the build type options. Visual Studio will download the source files and compile the libraries, installing them into the .DEPS folder. This concludes the first step.

Open a new `cmake-gui` and prepare the second step as shown in Figure A.4. The destination folder which will contain the final executable can be chosen freely. After clicking "Configure" again, choose the same generator as before and click "Generate" — changing the `CMAKE_BUILD_TYPE` here has no effect. After this, the destination directory contains a Visual Studio solution which can be used to build the final program as shown in Figure A.5.

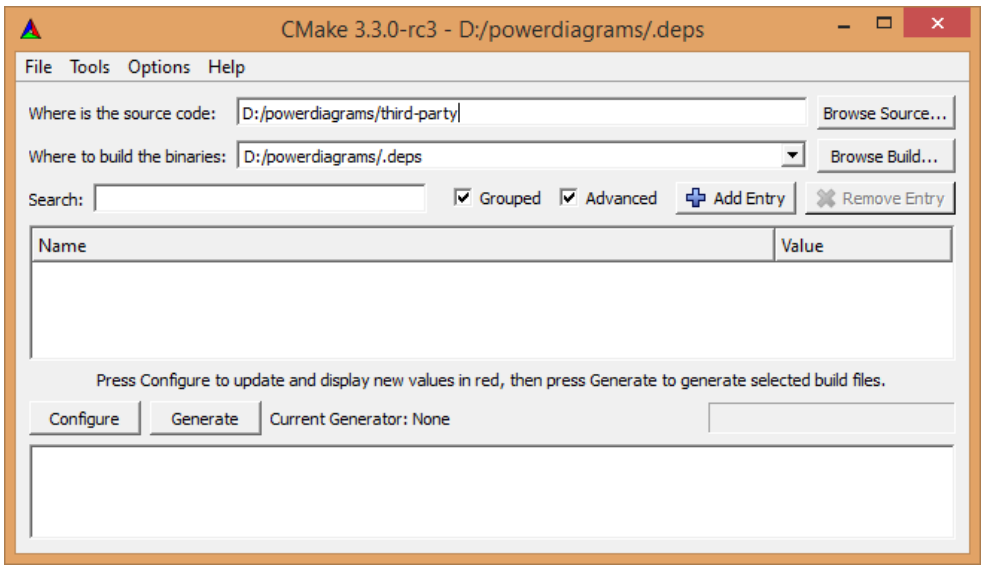


Figure A.1: The THIRD-PARTY directory contains a CMake script which downloads and builds the libraries needed by the `powerdiagram` executable. Choose .DEPS as the destination folder to ensure that the second step will automatically find the dependencies.

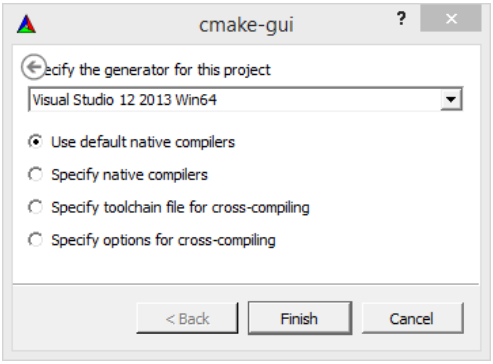


Figure A.2: After clicking the "Configure"-button, CMake prompts the user to choose a generator or toolchain. Visual Studio and MSVC are the most common toolchain on Windows. Both 32 and 64 bit builds are possible, but both libraries and the final program have to be built using the same generator.

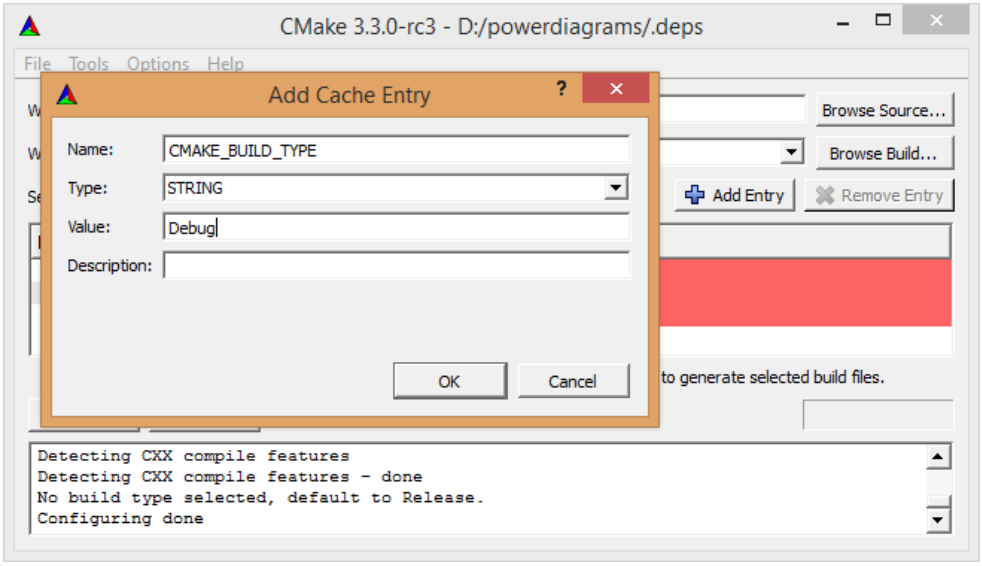


Figure A.3: To change the build type of the libraries from the default optimized release build to a debug build, add the according entry after the "Configure" step.

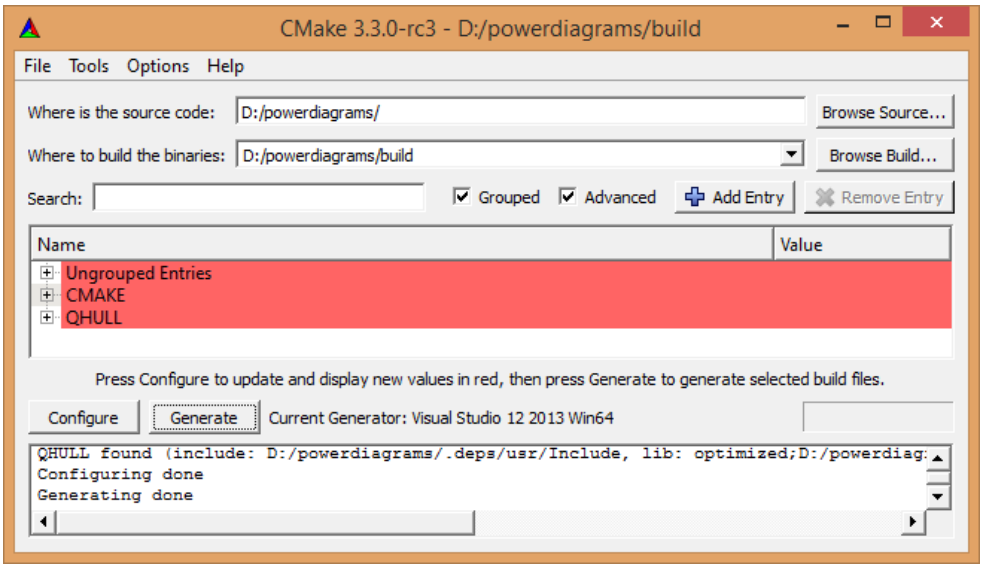


Figure A.4: To build the program executable, CMake has to be run again with the top level directory as the source folder.

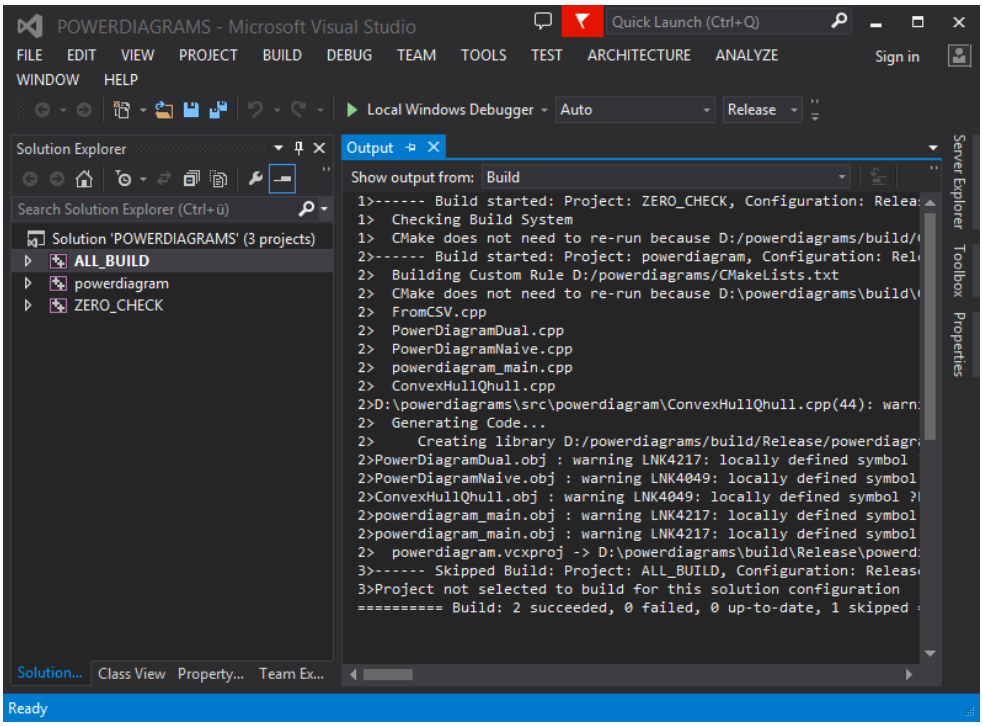


Figure A.5: The Visual Studio solution created by CMake for the program can be used to build the code in both debug and release mode. CMake found the correct paths for all libraries and configured Visual Studio accordingly. The solution explorer shows the `powerdiagram` executable which contains the source files which can be edited and debugged as usual.

Appendix B

Running the Program

The compiled program has a simple command line interface which allows the selection of power diagram sites, algorithms to be executed and output options. The program includes a short documentation which can be viewed by invoking `./powerdiagram -help`. The output of this invocation can be seen in Listing 2.

This chapter shows how to use the program to create a visualization of a two-dimensional power diagram in the LaTeX-package tikz with Figure 2.3 as an example. The first step is to choose the sphere centers and radii. They are defined in two different text files where each line defines a site. While the file defining the radii contains one number per line, the different entries of the vectors of the sphere centers are separated with commas, examples of which can be found in the EXAMPLES folder. A d dimensional input must contain at least $(d+1)$ linearly independent sphere centers. The files corresponding to Figure 2.3 are called GARAGE_SITES.CSV for the centers and GARAGE_GAMMA.CSV for the radii.

```
Sample usage:
    ./powerdiagram [Options] <centers> <radii>
For a complete help, use options --help or --helpfull.

  Flags from (...)/powerdiagram/ConvexHullQhull.cpp:
    -qhullout (Output string for Qhull (e.g. "f i s"))
        type: string default: ""

  Flags from (...)/powerdiagram_main.cpp:
    -draw (Output Information needed to draw the Diagram)
        type: bool default: false
    -dual (Run the Dual Algorithm) type: bool default: true
    -naive (Run the Naive Algorithm) type: bool default: false
    -verbose (Verbose output) type: bool default: false
```

Listing 2: The output of running `./powerdiagram -help`.

```
# Manual invocation
$ ./powerdiagram -draw garage_sites.csv garage_gamma.csv > /tmp/intrmdt
$ util/tikz.py /tmp/intermdt
# Using the Makefile
$ make TIKZ_OUT=1 garage
```

Listing 3: To create a tikz picture from sphere centers and radii defined in text files, the `./powerdiagram -draw` parameter can be used. If the program is invoked with this parameter, it outputs a easily parseable format describing vertices and edges of a power diagram, which gets saved to a temporary file. This output is then parsed by a Python script which in turn creates a tikz picture. The Makefile mentioned in Appendix A.1 can also generate tikz output.

Creating a tikz output is a two-step process. The C++ executable itself cannot produce tikz but rather outputs a format which is easily parseable. The file UTIL/TIKZ.PY contains a Python-script which converts this intermediate format to tikz output. The commands that need to be executed can be found in Listing 3 with the intermediate output of the C++ program described in Listing 4. There is also a template in UTIL/TEMPLATE.TEX which can be used to quickly compile the output of the script files into a standalone image.

```
# Spheres
s1 2 3 1
s2 5 0 1
s3 -2  0  3
s4 2 0 2
s5 4 2 1

# Vertices
p1 0.625     2
p2    4 0.75
p3 2.75    2

# Internal Edges
ei p1 p3 s1 s4
ei p2 p3 s4 s5

# External Edges
ee p1 s1 s3 d-0.6  0.8
ee p1 s3 s4 d 0 -1
ee p2 s2 s4 d-0 -1
ee p2 s2 s5 d0.894427 0.447214
ee p3 s1 s5 d0.447214 0.894427
```

Listing 4: The intermediate output format generated by `./powerdiagram -draw`. The output first enumerates both the spheres and vertices. Every internal edge can be defined as the connection of two of the vertices, while the external edges are defined as a point and a normalized direction of the ray. The additional annotations of the edge describe to which cells of spheres the edge is incident.

List of Figures

List of Algorithms

List of Listings

Bibliography

[ABI88] D. Avis, B. K. Bhattacharya, and H. Imai. "Computing the volume of the union of spheres". In: *The Visual Computer* 3.6 (1988), pp. 323–328.

[Alp+15] A. Alpers, A. Brieden, P. Gritzmann, A. Lyckegaard, and H. F. Poulsen. "Generalized balanced power diagrams for 3D representations of polycrystals". In: *Philosophical Magazine* 95.9 (2015), pp. 1016–1028.

[Aur87] F. Aurenhammer. "Power diagrams: properties, algorithms and applications". In: *SIAM Journal on Computing* 16.1 (1987), pp. 78–96.

[Aur91] F. Aurenhammer. "Voronoi diagrams – a survey of a fundamental geometric data structure". In: *ACM Computing Surveys (CSUR)* 23.3 (1991), pp. 345–405.

[BDH96] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. "The quickhull algorithm for convex hulls". In: *ACM Transactions on Mathematical Software (TOMS)* 22.4 (1996), pp. 469–483.

[BG12] A. Brieden and P. Gritzmann. "On optimal weighted balanced clusterings: gravity bodies and power diagrams". In: *SIAM Journal on Discrete Mathematics* 26.2 (2012), pp. 415–434.

[Bla13] W. Blaschke. *Vorlesungen über Differentialgeometrie I: Elementare Differentialgeometrie*. Vol. 1. Springer-Verlag, 2013.

[Bro12] A. Brondsted. *An introduction to convex polytopes*. Vol. 90. Springer Science & Business Media, 2012.

[CGA15] CGAL Project. *CGAL User and Reference Manual*. 4.6.1. CGAL Editorial Board, 2015.

[CS89] K. L. Clarkson and P. W. Shor. "Applications of random sampling in computational geometry, II". In: *Discrete & Computational Geometry* 4.1 (1989), pp. 387–421.

[EOS86] H. Edelsbrunner, J. O'Rourke, and R. Seidel. "Constructing arrangements of lines and hyperplanes with applications". In: *SIAM Journal on Computing* 15.2 (1986), pp. 341–363.

[G+10] G. Guennebaud, B. Jacob, et al. *Eigen v3*. `http://eigen.tuxfamily.org`. 2010.

[Gri13] P. Gritzmann. *Grundlagen der Mathematischen Optimierung: Diskrete Strukturen, Komplexitätstheorie, Konvexitätstheorie, Lineare Optimierung, Simplex-Algorithmus, Dualität*. Springer Spektrum, 2013.

[Grü03] B. Grünbaum. *Convex polytopes, volume 221 of Graduate Texts in Mathematics*. 2003.

[Grü63] B. Grünbaum. "Measures of symmetry for convex sets". In: *Convexity: Proceedings of the Seventh Symposium in Pure Mathematics of the American Mathematical Society*. Vol. 7. American Mathematical Soc. 1963, p. 233.

[GS78] P. J. Green and R. Sibson. "Computing Dirichlet tessellations in the plane". In: *The Computer Journal* 21.2 (1978), pp. 168–173.

[Mey14] S. Meyers. *Effective Modern C++: 42 Specific Ways to Improve Your Use of C++ 11 and C++ 14*. " O'Reilly Media, Inc.", 2014.

[MH07] K. Martin and B. Hoffman. "An open source approach to developing software in a small organization". In: *Ieee Software* 1 (2007), pp. 46–53.

[MH15] K. Martin and B. Hoffman. *CMake reference documentation*. `http://www.cmake.org/documentation/`. 2015.

[PH77] F. P. Preparata and S. J. Hong. "Convex hulls of finite sets of points in two and three dimensions". In: *Communications of the ACM* 20.2 (1977), pp. 87–93.

[Sei81] R. Seidel. "A convex hull algorithm optimal for point sets in even dimensions". PhD thesis. University of British Columbia, 1981.

[SS15] C. Silverstein and A. Schuh. *gflags*. `https://github.com/gflags/gflags`. 2015.

[Ste81] J. Steiner. *Jacob Steiners gesammelte Werke*. Vol. 1. American Mathematical Soc., 1881.

[Str14] B. Stroustrup. *Programming: principles and practice using C++*. Pearson Education, 2014.

[Str86] B. Stroustrup. *The C++ programming language*. Pearson Education India, 1986.

[Tot72] F. L. Toth. *Lagerungen in der Ebene auf der Kugel und im Raum*. Vol. 65. Springer-Verlag, 1972.