Capacity-Constrained Voronoi Tessellations: Computation and Applications

---

Dissertation

zur Erlangung des akademischen Grades des
Doktors der Naturwissenschaften (Dr. rer. nat.)
an der Universität Konstanz, Fachbereich Informatik und
Informationswissenschaft,

vorgelegt von
Michael Balzer

Tag der mündlichen Prüfung: 9. Juli 2009
Referent: Prof. Dr. Oliver Deussen
Referent: Prof. Dr. Daniel Keim

Advisor
Prof. Dr. Oliver Deussen, University of Konstanz, Germany

Reviewers
Prof. Dr. Oliver Deussen, University of Konstanz, Germany
Prof. Dr. Daniel Keim, University of Konstanz, Germany

Date of Submission
May 5th, 2009

Date of Defense
July 9th, 2009

Abstract

Voronoi tessellations specify a partition of a given space according to a set of sites where all points in that space are assigned to the closest site. Capacity-constrained Voronoi tessellations are special cases of Voronoi tessellations in which the region of each site has a predefined area. For example, the capacity constraint could state that each region in a Voronoi tessellation must have the same area, or it could state that all regions have individually defined areas. Thus, the capacity constraint allows us to control the spatial influence of the sites in the tessellation.

Unfortunately, there exists no straightforward approach for computing capacity-constrained Voronoi tessellations. Rather, they have to be generated with iterative optimization techniques. In this thesis, we present three different approaches for the computation of capacity-constrained Voronoi tessellations. The algorithms are tailored for either continuous or discrete spaces, and allow us to employ different distance functions that result in Voronoi tessellations with different characteristics. The presented *continuous space* algorithms modify the locations and weights of the sites, thereby reducing the error between the current and the desired areas of the regions. Although a proof of convergence is still an open research problem, our experiments showed their reliable convergence towards arbitrarily precise capacity-constrained Voronoi tessellations. In contrast, our *discrete space* algorithm starts with an arbitrary assignment of the points in the discrete space to the sites that fulfills the capacity constraint. This assignment is then optimized until it achieves an equilibrium state that represents a valid Voronoi tessellation. The convergence of the algorithm to such an equilibrium state is guaranteed.

Based on these three algorithms, we present two applications that utilize capacity-constrained Voronoi tessellations. The first application are *Voronoi treemaps* for the visualization of attributed hierarchies in the domain of information visualization. This application focuses on the resulting polygonal regions of the Voronoi tessellations. The second application are *capacity-constrained point distributions* as a general purpose method for sampling-related tasks in the domain of computer graphics. This application focuses not on the tessellations itself but on the resulting distributions of sites. Both applications represent significant advances in their field with respect to the flexibility of the method and the quality of their results.

Zusammenfassung

Eine Voronoi-Tessellierung beschreibt eine Partition eines gegebenen Raumes entsprechend einer Menge sogenannter Zentren, die alle Punkte des Raumes dem nächstgelegenen Zentrum zuweist. Kapazitätsbeschränkte Voronoi-Tessellierungen sind Spezialfälle von Voronoi-Tessellierungen in denen die Region jedes Zentrums einen vorbestimmten Flächeninhalt besitzt. Solch eine Kapazitätsbeschränkung kann zum Beispiel fordern, dass alle Regionen den gleichen Flächeninhalt haben sollen, oder sie kann fordern, dass jede Region einen individuell definierten Flächeninhalt haben soll. Mit Kapazitätsbeschränkungen wird somit der räumliche Einfluss der einzelnen Zentren in der Partitionierung gesteuert.

Kapazitätsbeschränkte Voronoi-Tessellierungen können leider nicht direkt berechnet werden. Stattdessen erfolgt ihre Berechnung mithilfe von iterativen Optimierungsverfahren. In dieser Arbeit präsentieren wir drei Ansätze für die Berechnung von kapazitätsbeschränkten Voronoi-Tessellierungen. Die entsprechenden Algorithmen sind dabei entweder für kontinuierliche oder für diskrete Räume konzipiert, und erlauben eine flexible Verwendung verschiedener Metriken für die Konstruktion von Voronoi-Tessellierungen mit unterschiedlichen Eigenschaften. Unsere Algorithmen für *kontinuierliche Räume* verändern die Position oder das Gewicht eines Zentrums, um dadurch schrittweise die Differenzen zwischen den tatsächlichen und den gewünschten Flächeninhalten der Regionen zu verringern. Auch wenn wir keinen Nachweis für die Konvergenz dieser Algorithmen geben können, so haben unsere Experimente trotzdem gezeigt, dass sich damit kapazitätsbeschränkte Voronoi-Tessellierungen verlässlich und beliebig genau berechnen lassen. Im Gegensatz dazu beginnt unser Algorithmus für *diskrete Räume* mit einer beliebigen Zuordnung von Punkten des diskreten Raumes zu den Zentren, welche zunächst einzig und allein die Kapazitätsbeschränkung erfüllen muss. Diese Zuordnung wird dann schrittweise optimiert bis ein Gleichgewichtszustand erreicht ist, in dem die Zuordnung einer gültigen Voronoi-Tessellierung entspricht. Die Konvergenz des Algorithmus in solch einen Gleichgewichtszustand ist dabei garantiert.

Aufbauend auf diesen Algorithmen präsentieren wir zwei Anwendungen für kapazitätsbeschränkte Voronoi-Tessellierungen. Die erste Anwendung sind *Voronoi Treemaps* für die Visualisierung von attributierten Hierarchien im Bereich der Informationsvisualisierung. Diese Anwendung konzentriert sich dabei auf die polygonalen Regionen der Voronoi-Tessellierungen. Die zweite Anwendung sind *kapazitätsbeschränkte Punktverteilungen* als eine universelle Methode für Diskretisierungsverfahren im Bereich der Computergrafik. Diese Anwendung konzentriert sich dabei auf die entstehenden Verteilungen der Zentren. Beide Anwendungen repräsentieren signifikante Fortschritte in ihrem jeweiligen Bereich in Bezug auf die Flexibilität der Methode als solche und in Bezug auf die Qualität der Ergebnisse.

Acknowledgements

This work was done with the kind help and support of many collaborators and friends.

First of all, I thank my advisor, Prof. Dr. Oliver Deussen, for the opportunity of working in his group, for his help, encouragement, and guidance during the course of this work, for supporting the publications that originated from this work as a co-author, and for reviewing this thesis. I also thank Prof. Daniel Keim for his helpful comments, and for reviewing this thesis.

Many thanks go to my colleagues and friends from the computer graphics group at the University of Konstanz. I especially thank Daniel Heck, and Thomas Schlömer for the intense and successful work on our papers that became an integral part of this thesis, for the numerous proof-readings, and last but not least, for the relaxed and funny atmosphere in our office. I am also very thankful to Thomas Luft and Joachim Böttger for our joint works on different topics in computer graphics and visualization. Besides the ones already mentioned, I thank Martin Röder, Carsten Colditz, Stefan Hiller, Boris Neubert, Johannes Kopf, Andreas Urra, Hendrik Strobelt, and Sören Pirk, for the many interesting and sometimes even fruitful discussions, their sharing of ideas, and for the nice time we spent outside the university. It has been a great pleasure working with you.

I also thank various other people from the Department of Computer and Information Science at the University of Konstanz that helped me with their academical, technical, and administrative skills.

I am deeply grateful to my family that supported me during my years of study, and also tolerated the things in my life that have been incomprehensible to them. Finally, I thank all my friends that directly or indirectly supported me, and which have been a great source of divertissements beside work.

This work was partially supported by the DFG Graduiertenkolleg 1042 “Explorative Analysis and Visualization of Large Information Spaces”.

Contents

Chapter 1

Introduction

Voronoi tessellations are a mathematical concept which is applied to various fields of science, ranging from astronomy and geography, over biology and chemistry, to mathematics and computer science. Their general idea is to specify a partition of a given space according to a set of sites, where all points in that space are assigned to the closest site. Figure 1.1 shows an example of a Voronoi tessellation in two-dimensional Euclidean space.

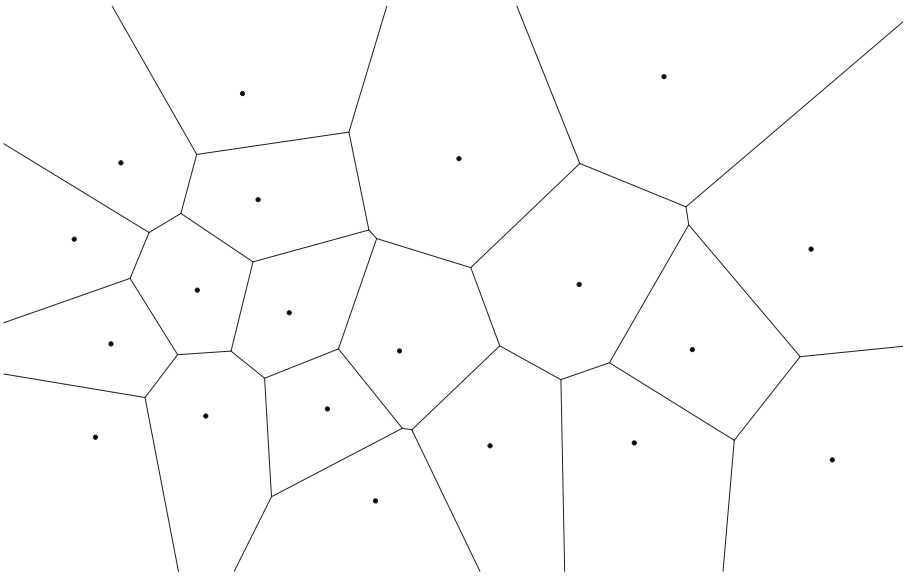
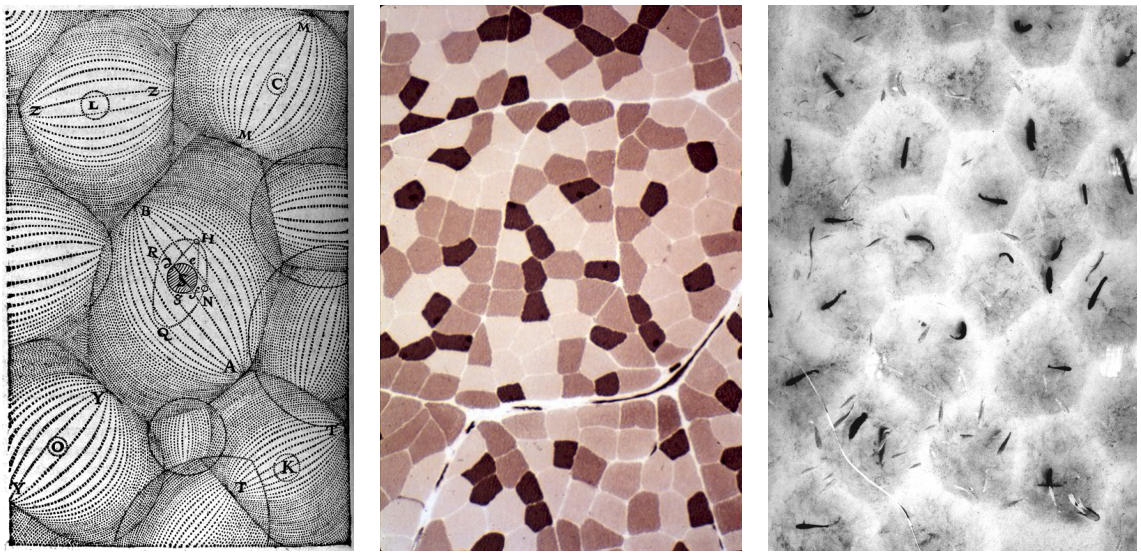


Figure 1.1: Voronoi tessellation of a set of 20 sites in two-dimensional Euclidean space

The origins of Voronoi tessellations date back to the 17th century, when René Descartes used them in illustrations of the solar system [25]. An example of these illustrations is shown in Figure 1.2(a). Since then, this concept has independently emerged in biology and physiology under the name *medial axis transform*, in chemistry and physics as *Wigner-Seitz cells*, in crystallography as *domains of action*, or in meteorology and



(a) Descartes' decomposition of space into vortices [25]

(b) human muscle fibres [43]

(c) territories of male mouthbreeder fishes *Tilapia mossambica* [14]

Figure 1.2: Examples of Voronoi tessellations in science and nature.

geography as *Thiessen polygons*. In the late 19th and beginning 20th century, the mathematicians Dirichlet [27] and Voronoi [77, 78] were the first to formally introduce and study this concept as *Dirichlet tessellations* and/or *Voronoi diagrams*. Voronoi was also the first to consider the dual of this structure, which was later defined by Delaunay [24] and is now commonly known as the *Delaunay graph*.

Beside their application in various scientific fields, Voronoi tessellations are ubiquitous in nature. Two such examples are given in Figure 1.2. Image (b) shows human muscle fibres that form hierarchical Voronoi tessellations, and image (c) shows territories of the mouthbreeder fish (*tilapia mossambica*) that form centroidal Voronoi tessellations. Voronoi tessellations are ubiquitous in nature because of their ability to model growth processes and equilibrium states that are efficient with respect to some energy model.

The formulation of equilibrium states and/or energy efficiency makes Voronoi tessellation also interesting for applications in computer science, especially computer graphics. In these contexts, they are employed for a wide variety of optimization problems that search for a finite set of representatives of a much larger set or a continuous function. The efficiency of such sets can be evaluated by analyzing the corresponding Voronoi tessellation, and it can be improved by generating so-called centroidal Voronoi tessellations that represent somehow optimal configurations.

1.1 Scope of the Thesis

In this thesis, we are interested in so-called *capacity-constrained Voronoi tessellations*. These tessellations are special cases of Voronoi tessellations in which the region of each site has a predefined area. For example, the capacity constraint could state that each region in a Voronoi tessellation must have the same area, or it could state that all regions have different, individually defined, areas. With such capacity constraints, we can precisely specify the importance of the individual sites in the given space with respect to their actual spatial influence.

Unfortunately, there exists no straightforward approach for the computation of capacity-constrained Voronoi tessellations. The areas of the Voronoi regions are implicitly determined by the neighbor topology in the Voronoi tessellation and the distances between these neighboring sites. If a site has no nearby neighbors then it forms a larger Voronoi region, whereas if a site is closely surrounded by their neighbors then it forms a smaller Voronoi region. This implicit relation between topology and site distances is the reason that there exists neither an intuitive approach nor an efficient heuristic to control the region areas in a Voronoi tessellation. Rather, capacity-constrained Voronoi tessellations have to be generated as iteratively optimized approximations.

In this context, this thesis presents three different approaches for the efficient generation of capacity-constrained Voronoi tessellations as iterative optimization techniques. The algorithms are tailored either for continuous or discrete spaces, and allow the utilization of different distance functions that result in Voronoi tessellations with different characteristics. Based on these algorithms, we present two applications that benefit from capacity-constrained Voronoi tessellations. The capacity constraint is the key concept for both applications, which represent major advances in their field with respect to flexibility and quality of the achieved results. The first application are *Voronoi treemaps* for the visualization of attributed hierarchies in the domain of information visualization. This application focuses on the resulting polygonal regions of the Voronoi tessellations. The second application are *capacity-constrained point distributions* as a general purpose method for sampling-related tasks in the domain of computer graphics. This application is a combination of capacity-constrained Voronoi tessellations and centroidal Voronoi tessellations, and focuses on the resulting distributions of sites.

1.2 Contributions

The main contributions of this thesis are:

- An algorithm for computing capacity-constrained Voronoi tessellations in continuous spaces using ordinary (non-weighted) distance functions. This is the first algorithm that allows the generation of such tessellations.

- An algorithm for computing capacity-constrained Voronoi tessellations in continuous spaces using weighted distance functions. This is the first algorithm that is

based on arbitrary weighted distance functions, and allows a precise computation of capacity-constrained Voronoi tessellations for large datasets with many thousands of sites.

- An algorithm for computing capacity-constrained Voronoi tessellations in discrete spaces using weighted distance functions. This algorithm is a very general, flexible, and easily implemented method that works in arbitrary discrete spaces, whereas other algorithms rely on implementations that are tailored for specific spaces.
- A treemap layout method based on capacity-constrained Voronoi tessellations. This method improves on the existent methods that are solely based on rectangular layouts by enabling polygonal treemap layouts, which are advantageous with respect to the aspect ratio of the treemap nodes and the interpretability of their hierarchical structure.
- A general purpose method for generating capacity-constrained point distributions. This method improves on the commonly used Lloyd's method in computer graphics with respect to the blue noise characteristic of the resulting distributions and their adaptation to arbitrary density functions.

1.3 Selected Publications

The results of this thesis and other related topics have been or will be presented in the following publications, listed in chronological order of appearance. The detailed references for theses publications are given in the bibliography at the end of the thesis.

- Michael Balzer, Andreas Noack, Oliver Deussen, and Claus Lewerentz. Software landscapes: Visualizing the structure of large software systems. In *Proceedings of the Joint Eurographics and IEEE TVCG Symposium on Visualization*, 2004.
- Michael Balzer, Oliver Deussen, and Claus Lewerentz. Voronoi treemaps for the visualization of software metrics. In *Proceedings of the ACM Symposium on Software Visualization*, 2005.
- Michael Balzer and Oliver Deussen. Exploring relations within software systems using treemap enhanced hierarchical graphs. In *Proceedings of the 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, 2005.
- Michael Balzer and Oliver Deussen. Voronoi treemaps. In *Proceedings of the IEEE Symposium on Information Visualization*, 2005.
- Joachim Böttger, Michael Balzer, and Oliver Deussen. Complex logarithmic views for small details in large contexts. *IEEE Transactions on Visualization and Computer Graphics: IEEE Visualization Conference and IEEE Symposium on Information Visualization Proceedings*, 12(5), 2006.

- Michael Balzer and Oliver Deussen. Level-of-detail visualization of clustered graph layouts. In *Proceedings of the Asia-Pacific Symposium on Visualisation*, 2007.

- Thomas Luft, Michael Balzer, and Oliver Deussen. Expressive illumination of foliage based on implicit surfaces. In *Proceedings of the Eurographics Workshop on Natural Phenomena*, 2007.

- Joachim Böttger, Martin Preiser, Michael Balzer, and Oliver Deussen. Detail-in-context visualization for satellite imagery. In *Proceedings of Eurographics*, 2008.

- Michael Balzer and Daniel Heck. Capacity-constrained Voronoi diagrams in finite spaces. In *Proceedings of the 5th Annual International Symposium on Voronoi Diagrams in Science and Engineering*, 2008.

- Michael Balzer. Capacity-constrained Voronoi diagrams in continuous spaces. *Proceedings of the 6th Annual International Symposium on Voronoi Diagrams in Science and Engineering*, 2009.

- Michael Balzer, Thomas Schlömer, and Oliver Deussen. Capacity-constrained point distributions: A variant of Lloyd's method. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2009)*, 28(3), 2009.

1.4 Outline

The remainder of the thesis is organized as follows:

A short introduction to Voronoi tessellations and the concept of capacity-constrained Voronoi tessellations is given in Chapter 2. The content of this chapter considers the necessary theoretical background for the following chapters, without being a comprehensive treatment of Voronoi tessellations.

In Chapter 3, we present two algorithms for computing capacity-constrained Voronoi tessellations in continuous spaces. The first approach is based on ordinary (non-weighted) distance functions, whereas the second approach generates tessellations based on weighted distance functions. The results of this chapter are currently submitted for publication [5].

In Chapter 4, we present an algorithm for computing capacity-constrained Voronoi tessellations in discrete spaces based on weighted distance functions. The flexibility of this approach allows its generic application to arbitrary dimensions and various distance functions. The results of this chapter were published in [11].

In Chapter 5, we present the application of capacity-constrained Voronoi tessellations to treemap layouts in information visualization. Therefore, we discuss existing treemap layout algorithms, present our Voronoi treemap layout algorithm, and give some examples of resulting Voronoi treemap visualizations. The results of this chapter were published in [8, 10].

In Chapter 6, we present the application of capacity-constrained Voronoi tessellations to point distributions as a variant of Lloyd's method in computer graphics. We evaluate our results by an extensive comparison to those of Lloyd's method, and illustrate why the capacity constraint is the reason for the improved quality of the distributions. The results of this chapter have been accepted for publication in [13].

The thesis is concluded in Chapter 7.

Chapter 2

Definitions and Properties

In this chapter, we provide the necessary theoretical background to Voronoi tessellations. For the ease of exposition, we restrict ourselves to aspects relevant in the scope of this thesis, and, without loss of generality, to Voronoi tessellations of point sites within the Euclidean plane. The definitions and properties can easily be transferred to other continues or discrete spaces, and to a variety of distance functions. For a comprehensive treatment of Voronoi tessellations, see [58, 2]. All definitions and denotations are according to [58, 28].

In the following Section 2.1, we give the basic definition of an ordinary Voronoi tessellation. This definition is extended to weighted Voronoi tessellations in Section 2.2, and to centroidal Voronoi tessellations in Section 2.3. Finally, we present the concept of capacity and the corresponding definition of capacity-constrained Voronoi tessellations in Section 2.4.

2.1 Ordinary Voronoi Tessellations

We consider a set $S = \{s_1, \ldots, s_n\}$ of n points in the Euclidean plane $\mathbb{R}^2$, and assume that $2 \leq n < \infty$. The points have the Cartesian coordinates $(s_{11}, s_{12}), \ldots, (s_{n1}, s_{n2})$ and are distinct in the sense that $(s_{i1}, s_{i2}) \neq (s_{j1}, s_{j2})$ for $i \neq j$ and $i, j \in I_n = \{1, \ldots, n\}$. These points in the set S are the *sites*. Let x be an arbitrary point in $\mathbb{R}^2$ with coordinates (x_1, x_2). The Euclidean distance between the point x and a site $s_i \in S$ is given by

$$d_E(s_i, x) = \|s_i - x\| = \sqrt{(s_{i1} - x_1)^2 + (s_{i2} - x_2)^2}. \tag{2.1}$$

If $s_i \in S$ is the nearest site from x or one of the nearest sites from x, we have the relation $d_E(s_i, x) \leq d_E(s_j, x)$ for $i \neq j$, $j \in I_n$. We call the points with the nearest site s_i, given by

$$V_E(s_i) = \{x \mid d_E(s_i, x) \leq d_E(s_j, x), i \neq j, j \in I_n\}, \tag{2.2}$$

the *ordinary Voronoi region* associated with s_i, and the set given by

$$\mathcal{V}_E(S) = \{V_E(s_1), \ldots, V_E(s_n)\} \tag{2.3}$$

the *ordinary Voronoi tessellation* generated by S. An example with a set of 20 random sites is given in Figure 2.1(a)

In an ordinary Voronoi tessellation, a *bisector* of two regions $V_E(s_i)$ and $V_E(s_j)$, with $i \neq j$, is the line perpendicular to the line segment $\overline{s_i s_j}$, formed by points equidistant to both s_i and s_j. The bisector divides the plane in two half-planes and defines

$$\mathrm{Dom}_E(s_i, s_j) = \{x \mid \|s_i - x\| \leq \|s_j - x\|\},\ i \neq j. \tag{2.4}$$

The region $\mathrm{Dom}_E(s_i, s_j)$ is called the *dominance region* of s_i over s_j. Therefore, the Voronoi tessellation may alternatively be understood as the intersection of dominance regions with

$$\mathscr{V}_E(S) = \bigcap_{i \in I_n \setminus \{i\}} \mathrm{Dom}_E(s_i, s_j),\ i \neq j, j \in I_n. \tag{2.5}$$

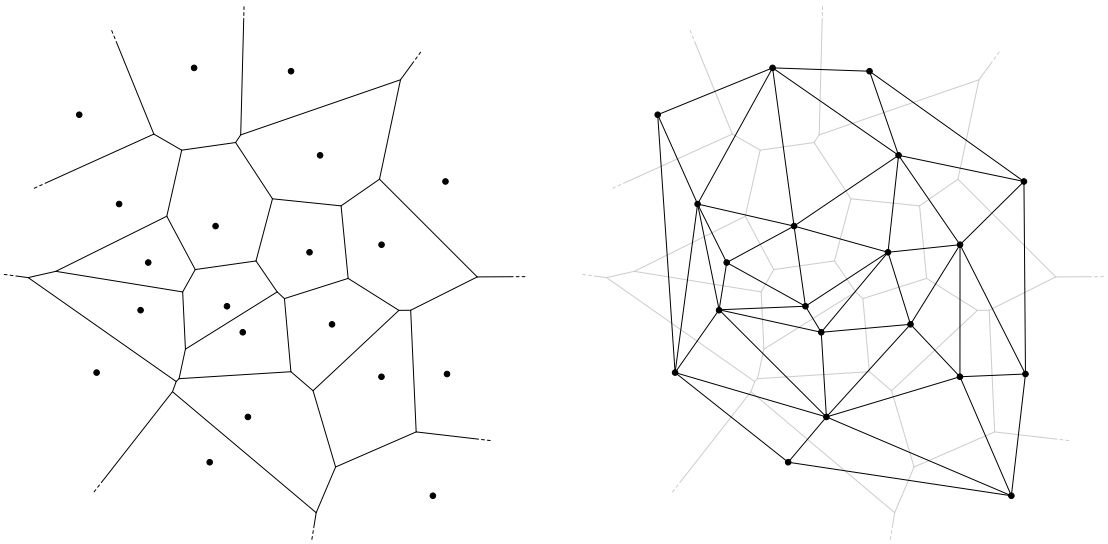
A *Voronoi edge* of an ordinary Voronoi tessellation is a non-empty line segment, half line or infinite line that is formed by all points of the *bisector* of two regions $V_E(s_i)$, $V_E(s_j)$, $i \neq j$, that are closer to s_i and/or s_j than to a third site s_k, $k \neq i \neq j$, $k \in I_n$. The sites s_i and s_j that share a Voronoi edge are called *neighbors*. A *Voronoi vertex* of an ordinary Voronoi tessellation is formed by the intersection of three or more Voronoi edges, and has an equal distance to the sites that form these edges.

The dual of an ordinary Voronoi tessellation is a *Delaunay graph*, which connects sites that are neighbors in the tessellation. A Delaunay graph can be derived from the corresponding ordinary Voronoi tessellation with linear time and memory complexity, and vice versa. An example for a corresponding Delaunay graph of the Voronoi tessellation in Figure 2.1(a) is shown in Figure 2.1(b).

A lower bound on the worst-case time complexity for constructing an ordinary Voronoi tessellation $\mathscr{V}_E(S)$ of n sites in two-dimensional Euclidean space is $O(n \log n)$. A lower bound on the memory complexity for computing $\mathscr{V}_E(S)$ is $O(n)$ in the worst case. All these lower bounds for $\mathscr{V}_E(S)$ are tight in the sense that there exist optimal algorithms having these lower bounds. An extensive overview of a variety of algorithms for computing different kinds of Voronoi tessellations can be found in [58]. Efficient and exact implementations for computing Voronoi tessellations, and their weighted derivatives presented in the next section, are provided by the CGAL library [20].

2.2 Weighted Voronoi Tessellations

In an ordinary Voronoi tessellation of a set S of n sites it is implicitly assumed that the sites are identical except for their locations, or that each site has the same weight. As an extension, a set of parameters $W = \{w_i \mid i \in \{1, \ldots, n\}\}$ may be given. These parameters are the *weights*. By using weighted sites, we can define *weighted distances* that are used for generating *weighted Voronoi tessellations* $\mathscr{V}(S, W)$. The Voronoi regions $V(s_i, w_i) \in \mathscr{V}(S, W)$ not only depend on the locations of the sites, but also on their weights, where usually sites with larger weights form larger Voronoi regions.



(a) ordinary Voronoi tessellation (b) corresponding Delaunay graph

Figure 2.1: Ordinary Voronoi tessellations of a set of 20 random sites and the corresponding Delaunay graph that connects neighboring sites in the tessellation.

Since the weighted distance allows many functional forms, a wide variety of weighted Voronoi tessellations exist, all having their own characteristics. In the following sections, we introduce two different weighted tessellations that are either relevant for the computation of capacity-constrained Voronoi tessellations in Chapter 3 and Chapter 4 or their applications in Chapter 5 and Chapter 6.

2.2.1 AW Voronoi Tessellations

An *additively weighted Voronoi tessellation* $\mathcal{V}_{AW}(S,W)$ [2], briefly AW Voronoi tessellation, is characterized by the following *AW distance function* between a site $s_i \in S$ with its assigned weight $w_i \in W$ and a point x:

$$d_{AW}(s_i,w_i,x) = \|s_i - x\| - w_i. \tag{2.6}$$

The dominance region of s_i over s_j in an AW Voronoi tessellation is given by

$$\mathrm{Dom}_{AW}(s_i,s_j,w_i,w_j) = \{x \mid \|s_i - x\| - \|s_j - x\| \leq w_i - w_j\}, i \neq j. \tag{2.7}$$

An example for an AW Voronoi tessellation is shown in Figure 2.2(a).

The shape of the dominance region $\mathrm{Dom}_{AW}(s_i,s_j,w_i,w_j)$ varies according to the parameter values $\alpha = \|s_i - s_j\|$ and $\beta = w_i - w_j$, where $\beta \geq 0$ is assumed without loss of generality. If $0 < \alpha < \beta$, the site s_i dominates the whole plane, and the dominance region of s_j disappears. If $\alpha = \beta$, the dominance region is the whole plane except for

the half-line radiating from s_j in the direction from s_i to s_j. Finally, if $\alpha > \beta$, the bisector of $V_{AW}(s_i, w_i)$ and $V_{AW}(s_j, w_j)$ forms a hyperbolic curve with foci s_i and s_j. Note that if $\beta = 0$, the bisector again becomes a straight line perpendicularly bisecting the line segment $\overline{s_i s_j}$ through its midpoint. If at least one weight w_i is different from the other weights and the relation $\alpha > \beta$ holds, there exists at least one non-convex AW Voronoi region, only guaranteed that it is star-shaped [42].

For the third case of $\alpha > \beta$, the AW Voronoi tessellation may additionally be illustrated as a Voronoi tessellation that uses circles as sites with the Euclidean distance function. This is comprehensible if circles are considered as points with a size or weight parameter. Here, the weight w_i in the AW distance function directly represents the radius of the circle. Necessarily, these circles do not overlap due to the constraint of $\alpha > \beta$. However, the abstraction of circles with negative radii has to be made in order to achieve the complete spectrum of possible AW Voronoi tessellations.

The dual of an AW Voronoi tessellation is an *Apollonius graph*, which connects neighboring sites in the AW Voronoi tessellation. Again, both representations, the AW Voronoi tessellation and the Apollonius graph, can be directly derived from each other with linear time and memory complexity.

2.2.2 PW Voronoi Tessellations

An *additively weighted power Voronoi tessellation* $\mathcal{V}_{PW}(S, W)$ [1], briefly PW Voronoi tessellation, is characterized by the following *PW distance function* between a site $s_i \in S$ with its assigned weight $w_i \in W$ and a point x:

$$d_{PW}(s_i, w_i, x) = \|s_i - x\|^2 - w_i. \tag{2.8}$$

The dominance region of s_i over s_j in a PW Voronoi tessellation is given by

$$\mathrm{Dom}_{PW}(s_i, s_j, w_i, w_j) = \{p \mid \|s_i - p\|^2 - \|s_j - p\|^2 \leq w_i - w_j\}, i \neq j. \tag{2.9}$$

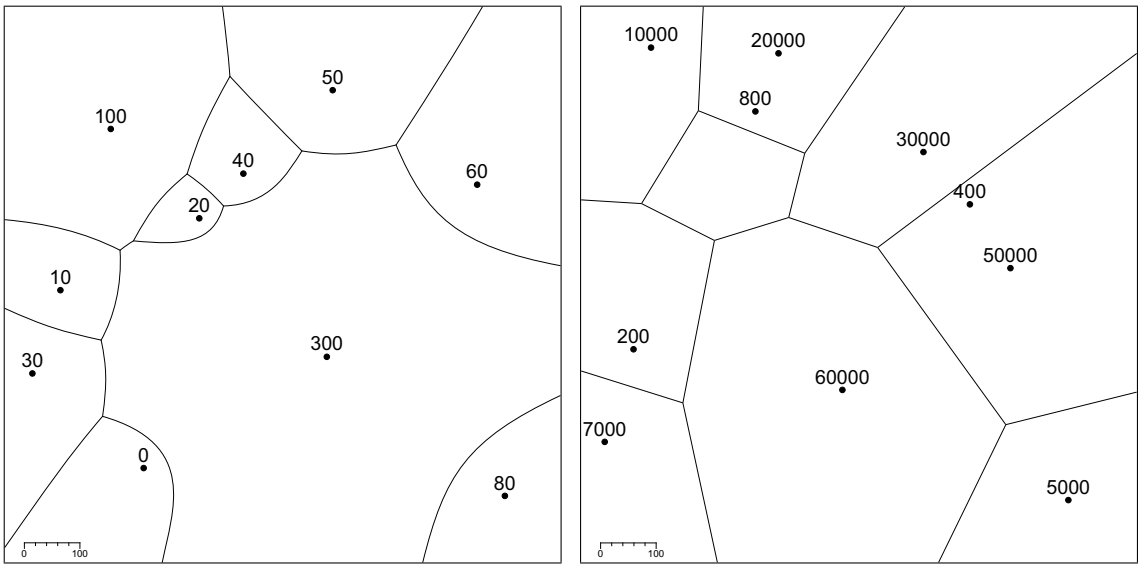
An example for a PW Voronoi tessellation is shown in Figure 2.2(b).

The bisector of two Voronoi regions $V_{PW}(s_i, w_i)$ and $V_{PW}(s_j, w_j)$ is a straight line. It corresponds to the line perpendicularly bisecting the line segment $\overline{s_i s_j}$ through the point

$$p^*_{ij} = \frac{\|s_j\|^2 - \|s_i\|^2 + w_i - w_j}{2\|s_j - s_i\|^2}(s_j - s_i). \tag{2.10}$$

In contrast to the AW Voronoi tessellation, the dominance region $\mathrm{Dom}_{PW}(s_i, s_j, w_i, w_j)$ cannot disappear or degenerate to a line due to the squaring of the Euclidean distance, which necessarily exceeds any difference between the assigned weights w_i and w_j for large enough distances to both s_i and s_j. However, if $\|s_i - s_j\|^2 < w_j - w_i$ then the site s_i may not reside in the dominance region $Dom_{PW}(s_i, s_j, w_i, w_j)$. In contrast to the AW Voronoi tessellation, every non-empty region in a PW Voronoi tessellation is convex.

Similar to the AW Voronoi tessellation, the PW Voronoi tessellation can also be regarded as a Voronoi tessellation of circles. Here, the slightly different distance function



(a) AW distance function (b) PW distance function

Figure 2.2: Weighted Voronoi tessellations of two sets of 10 random sites using the AW and the PW distance function. The numbers represent the weights of the sites.

is based on the so-called Laguerre geometry [37] whereby the weights correspond to the square of the radii of the circles. However, this analogy holds only if considering positive weights, but PW Voronoi tessellation allow negative weights as well.

The dual of a PW Voronoi tessellation is a *regular Delaunay graph*, which connects neighboring sites in the PW Voronoi tessellation. Again, both representations, the PW Voronoi tessellation and the regular Delaunay graph, can be directly derived from each other with linear time and memory complexity.

2.3 Centroidal Voronoi Tessellations

A *centroidal Voronoi tessellation* [28] is a Voronoi tessellation of a set S of sites in a bounded space $\Omega \subset \mathbb{R}^2$ with the property that each site $s_i \in S$ coincides with the centroid of the corresponding Voronoi region $V(s_i) \in \mathscr{V}(S)$. Given a Voronoi region $V(s_i)$ and a density function $\rho(x) \geq 0$, defined for $x \in \Omega$, the centroid, or center of mass, p_i of the Voronoi region $V(s_i)$ is calculated by

$$p_i = \frac{\int_{x \in V(s_i)} x\rho(x)dx}{\int_{x \in V(s_i)} \rho(x)dx}. \tag{2.11}$$

In other words, a centroidal Voronoi tessellation $\mathscr{V}(S)$ with n sites $s_i \in S$ fulfills the condition

$$\sum_{i=1}^{n} \|s_i - p_i\|^2 = 0. \tag{2.12}$$

The importance of the centroidal Voronoi tessellation is founded on its relationship to the energy function

$$\mathscr{F}(S) = \sum_{i=1}^{n} \int_{x \in V(s_i)} \rho(x) \|x - s_i\|^2 dx, \tag{2.13}$$

with $\mathscr{V}(S)$ as the Voronoi tessellation consisting of the Voronoi regions $V(s_i)$. This energy function describes the centrality of the sites within their regions, and the uniformity of the distribution of all sites. A variety of applications demand low values for $\mathscr{F}(S)$ thereby representing somehow optimal sets of sites. A necessary condition for $\mathscr{F}(S)$ to be minimized is that each site $s_i \in S$ coincides with the centroid of the corresponding Voronoi region $V(s_i)$, which means that $\mathscr{V}(S)$ is a centroidal Voronoi tessellation [28].

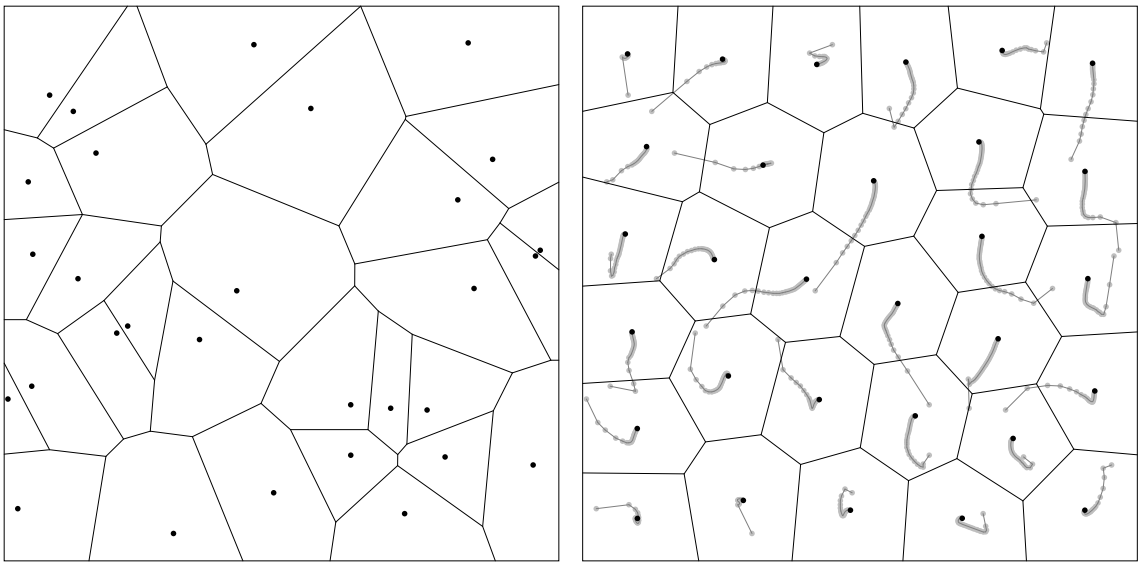
Trivial examples of centroidal Voronoi tessellations are regular tessellations of $\Omega \subset \mathbb{R}^2$ into triangles, squares or hexagons, thereby producing regular lattices of sites. However, more relevant in practical applications is the generation of non-regular centroidal Voronoi tessellations for an arbitrary number of sites. In the one-dimensional case, a configuration of sites that results in a centroidal Voronoi tessellation can be computed analytically [66]. If a centroidal Voronoi tessellation has to be found for two or more dimensions, an analytical solution is intractable. Rather, it is necessary to iteratively compute approximations until a sufficient solution is achieved, meaning that the distance between the actual centroid of each region and the associated site is below a chosen threshold ε.

A straightforward method for such an iterative computation is based on the energy function in Equation 2.13, leading to the so-called *spatial-temporal adjustment model* by Hasegawa and Tanemura [36]. To formulate this model mathematically, let $S^{(t)}$ be a set of sites in the bounded space $\Omega \subset \mathbb{R}^2$ at a time t, with $t = 0, 1, \ldots$. It is assumed that the initial sites $S^{(0)}$ are uniformly randomly distributed over Ω. For each site $s_i^{(t)} \in S^{(t)}$, the centroid $p_i^{(t)}$ of the associated Voronoi region $V\left(s_i^{(t)}\right)$ of the Voronoi tessellation $\mathscr{V}\left(S^{(t)}\right)$ in Ω is given by Equation 2.11. In these terms, the spatial-temporal adjustment model is described as

$$s_i^{(t+1)} = s_i^{(t)} + \alpha \left(p_i^{(t)} - s_i^{(t)}\right), \tag{2.14}$$

with $0 < \alpha$ as an adjustment coefficient. When $\|p_i^{(t)} - s_i^{(t)}\| < \varepsilon$ for all $s_i^{(t)} \in S^{(t)}$ the configuration of the sites $S^{(t)}$ is *stable*. Figure 2.3 illustrates this method. For $\alpha = 1$, this model is also known as *Lloyd's method* [50].

Algorithms based on this spatial-temporal adjustment model will generate different centroidal Voronoi tessellations for the same number of sites, depending on the initial locations of these sites. Therefore, the minimization of the energy function in Equation 2.13 will end up in a local minimum. The convergence of the spatial-temporal adjustment model to a centroidal Voronoi tessellation as a local minimum of $\mathscr{F}$ was proven in some



(a) initial configuration (b) resulting centroidal Voronoi tessellation

Figure 2.3: Voronoi tessellation of 30 random sites and their resulting centroidal Voronoi tessellation generated by using Lloyd's method. The traces illustrate the movements of the sites during the computation.

particular cases by Du et al. [28, 29], such as in the one-dimensional case with any positive and smooth density function. In this context, they showed that besides being a fixed-point iteration that solves Equation 2.11, the adjustment model can be understood as a gradient descent method that always decreases the energy $\mathscr{F}$ without step-size control. The same type of analysis can be applied to similar gradient descent methods for $\mathscr{F}$ that result in equivalent results, such as Liu et al. [49], which also showed the C^2 smoothness of $\mathscr{F}$ for convex boundings. A general proof of convergence in a two- or higher-dimensional space is still not existent. An effective method for finding global minima of $\mathscr{F}$ is also unknown.

Due to the fact that the notions of Voronoi regions and centroids may be generalized to more abstract spaces and to distance functions other than the Euclidean norm, it is possible to generalize the concept of centroidal Voronoi tessellations. This includes its extension to the application of weighted Voronoi tessellations [39]. The generalization of centroidal Voronoi tessellations also allows the substitution of points as sites by arbitrary objects, such as lines, polygons, or similar entities as long as a suitable distance function exists. However, it is not guaranteed that the iterative computation with the presented spatial-temporal adjustment model will show a feasible convergence for such generalizations.

2.4 Capacity-Constrained Voronoi Tessellations

Consider a set S of n sites that determines a Voronoi tessellation $\mathscr{V}(S)$ in the space $\Omega \subseteq \mathbb{R}^2$ with the density function $\rho(x) \geq 0, x \in \Omega$. The area of the Voronoi region $V(s_i) \in \mathscr{V}(S)$ of a site $s_i \in S$ is given by

$$|V(s_i)| = \int_{x \in V(s_i)} \rho(x) dx. \tag{2.15}$$

This Voronoi region area $|V(s_i)|$ is denoted as the *capacity* of s_i.

A set of parameters $C = \{c_i \mid i \in \{1, \ldots, n\}\}$ with $0 < c_i \leq \infty$ and $\sum C = \int_{x \in \Omega} \rho(x) dx$ may be given, which is called the *capacity constraint*. Based on this set, Aurenhammer et al. [3] defined the *capacity-constrained Voronoi tessellation* $\mathscr{V}(S,C)$ as a Voronoi tessellation of S in Ω in which each site $s_i \in S$ has the corresponding capacity $c_i \in C$. In other words, a capacity-constrained Voronoi tessellation $\mathscr{V}(S,C)$ with n sites $s_i \in S$ and a capacity constraint C with n parameters fulfills the condition

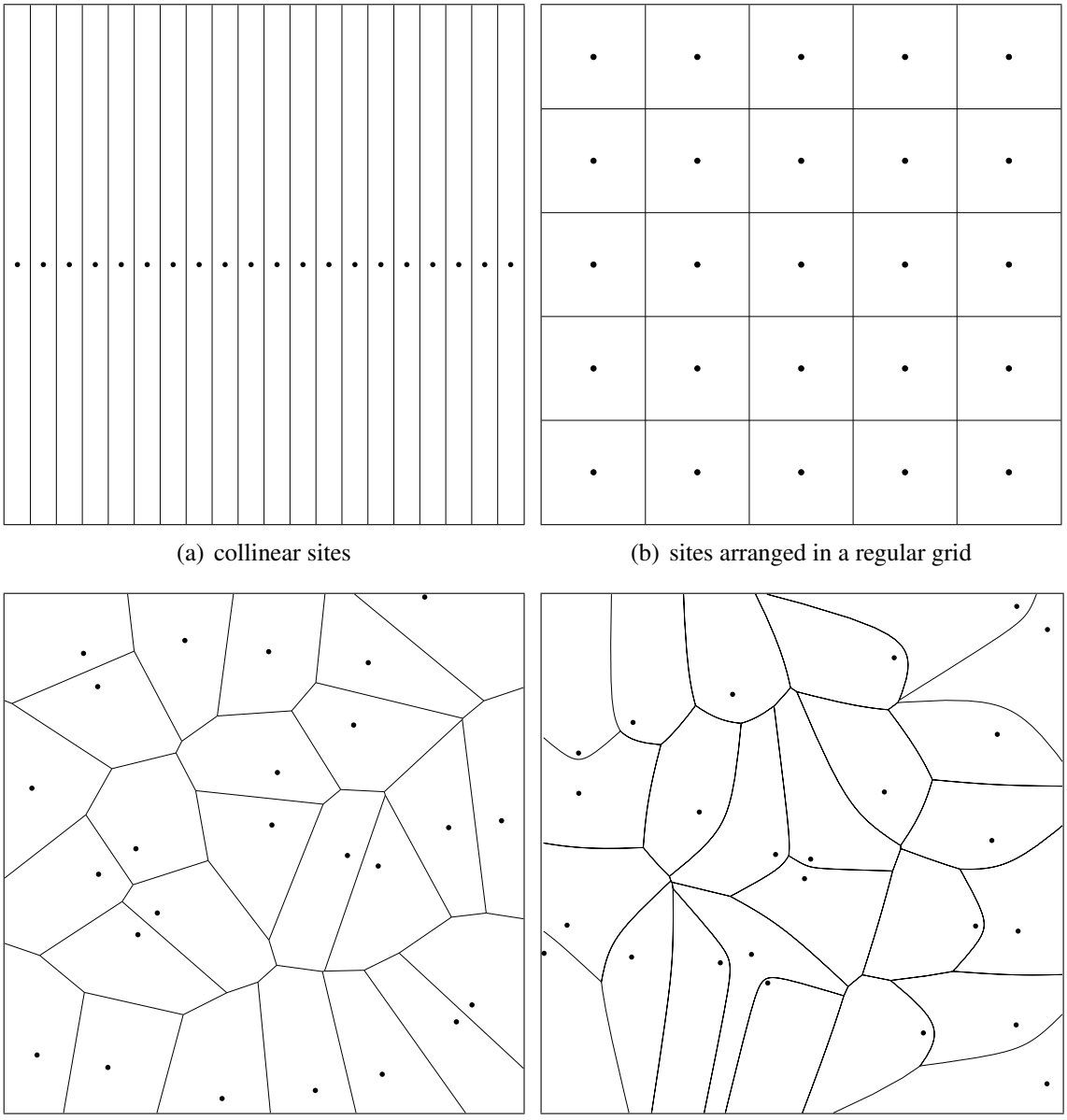
$$\sum_{i=1}^{n} (|V(s_i)| - c_i)^2 = 0. \tag{2.16}$$

Note that capacity-constrained Voronoi tessellations are a general concept that emphasizes the area of the Voronoi regions. It is not restricted to particular spaces or distance functions, and can also be combined with other concepts such as centroidal Voronoi tessellations.

The space Ω can be bounded or unbounded as long as the area of that space combined with the density function ρ is equal to the sum of capacities in C. In this thesis, we utilize only bounded spaces without loss of generality, for which each site has a finite capacity, to allow a clearer presentation of the concept and the algorithms.

Trivial examples for capacity-constrained Voronoi tessellations are sets of collinear sites as in Figure 2.4(a) or regular lattices of sites as in Figure 2.4(b), both with equal capacities for all sites. More relevant in practical applications are capacity-constrained Voronoi tessellations with non-regular site distributions as shown in Figure 2.4(c) and Figure 2.4(d), or with capacity constraints where each Voronoi region has a different, individually defined, area.

Currently there exist only three algorithms that generate capacity-constrained Voronoi tessellations. All three algorithms are based on weighted distance functions, and adjust the weights of the sites according to the difference between the actual areas of the Voronoi regions and the capacity constraint. Ohyama [57] uses the PW distance function, and Reitsma et al. [64] use the *multiplicatively weighted distance function* $d_{MW}(s_i, w_i, x) = \frac{1}{w_i}\|s_i - x\|, w_i > 0$. Both approaches adjust the weights of the sites individually with a variant of Newton's method, and perform their optimization in an underlying discrete space. The drawback of both algorithms are numerical problems due to the discrete approximation that impact the convergence of the algorithms and the precision of the results. Therefore, Reitsma et al. also presents an adaptive discrete refinement model that diminishes the approximation error. The third existent algorithm is presented by Aurenhammer



(a) collinear sites

(b) sites arranged in a regular grid

(c) non-regularly distributed sites using the ordinary Euclidean distance function

(d) non-regularly distributed sites using the weighted AW distance function

Figure 2.4: Examples of capacity-constrained Voronoi tessellations with 25 sites. Each Voronoi region in these tessellations has the same capacity.

et al. [3]. This algorithm uses the PW distance function as well, but is designed exclusively for two-dimensional continuous Euclidean space, whereas the other two algorithm in principle also work in higher dimensions. Even though, the algorithm by Aurenhammer et al. is not prone to approximation errors, and reliably generates solutions even for large datasets.

Chapter 3

Computation in Continuous Spaces

The shapes and areas of the regions in a Voronoi tessellation are implicitly given by the neighbor topology and the pairwise distances between these neighbors. Sites with no nearby neighbors form larger regions than sites that are closely surrounded by their neighbors. Weighted distance functions allow to increase the area of a region by increasing the weight of the respective site. Unfortunately, due to the interdependency of the sites, there exists no direct relation between the locations and/or weights of the sites and the resulting Voronoi region areas. To generate a capacity-constrained Voronoi tessellation it is therefore necessary to perform an iterative optimization of the site parameters based on the difference between the actual areas of the Voronoi regions and the given capacity constraint.

In this section, we present two approaches for computing capacity-constrained Voronoi tessellations in continuous spaces. Both methods are iterative algorithms that improve the region area of one single site at a time, and converge towards a stable state in which the capacity constraint is fulfilled. The first approach is based on ordinary (non-weighted) distance functions and modifies the locations of the sites. It is the first algorithm that allows the generation of capacity-constrained Voronoi tessellations based on ordinary distance functions. The second approach uses weighted distance functions and modifies the weights of the sites. It is the first algorithm that is based on arbitrary weighted distance functions, and allows a precise computation of capacity-constrained Voronoi tessellations for large datasets with many thousands of sites.

The prerequisite for the convergence of both algorithms is that they are used in a continuous space, or a sufficiently precise discrete representation of a continuous space based on real numbers. This is because the algorithm require that small changes in the locations and/or weights of the sites result in similarly small changes in the area of the Voronoi regions. In other words, both algorithm are based on the minimization of a function that has to be continuous, otherwise, they may not converge. In contrast, Chapter 4 presents an algorithm that is especially tailored to discrete spaces.

3.1 Ordinary Distance Functions

The Voronoi region $V(s_i)$ of a site $s_i \in S$ in an ordinary Voronoi tessellation $\mathscr{V}(S)$ is solely determined by the relative location of all sites $s_j \in S$ that are neighbors to s_i. The generation of a capacity-constrained ordinary Voronoi tessellation $\mathscr{V}(S,C)$ for n sites with their capacities in C in the d-dimensional space $\Omega \subseteq \mathbb{R}^d$ can be formulated as an optimization problem in nd dimensions. The variables in this optimization are the d-dimensional locations of all n sites. The continuous function $\mathscr{D}$ that has to be minimized is the total difference between the capacity constraint in C and the actual areas of the Voronoi regions:

$$\mathscr{D}(\mathscr{V}(S),C) = \sum_{i=1}^{n} (|V(s_i)| - c_i)^2 \text{ with } s_i \in S, c_i \in C. \tag{3.1}$$

This function is equivalent to Equation 2.16 in the definition of capacity-constrained Voronoi tessellations in Section 2.4. The global minimum $\mathscr{D} = 0$ is achieved if the Voronoi tessellation $\mathscr{V}(S)$ fulfills the given capacity constraint C. Such global minimum always exists, which is obvious if one considers a collinear set of n sites for which a solution can easily be found—we presented an algorithm concerning this problem in [7]. Actually, there exists not only one set of sites in S that fulfills the capacity constraint in C, but rather a large number of variants for which $\mathscr{D}$ attains the global minimum.

3.1.1 Algorithm

The minimization of the continuous function $\mathscr{D}$ can be performed with general optimization techniques that use the site locations as the parameter vector of dimensionality nd. The drawback of such direct minimization approach is that for large sets of sites the high dimensionality of the parameter vector entails a prohibitive runtime and numerical problems during the optimization. This problem of the high dimensionality of the optimization can be remedied by not optimizing all site locations at once, but rather iteratively optimizing just one site location at a time. This reduces the dimensionality of an optimization step from nd to d, where d—the dimension of the space Ω—is usually small, and does not depend on the number of sites. Ultimately, this iterative improvement of single site locations is equivalent to the optimization of all sites at once. The only difference is that the iterative approach subdivides the full optimization into smaller fragments, which are easier to solve.

The result is the following approach for the computation of a capacity-constrained ordinary Voronoi tessellation in continuous spaces, which is outlined in Algorithm 3.1. The input for the algorithm is a set of n sites S, a set of n associated capacities C, and a d-dimensional space Ω. The result of the algorithm is a capacity-constrained ordinary Voronoi tessellation $\mathscr{V}(S,C)$ in Ω with $\mathscr{D}(\mathscr{V}(S),C) = 0$. Initially, the Voronoi tessellation $\mathscr{V}(S)$ is computed with the given sites in S. This tessellation is then iteratively optimized towards $\mathscr{D} = 0$ by performing the following steps in each iteration:

Algorithm 3.1: Ordinary Distance Functions in Continuous Spaces

Input: Sites $S = \{s_1, \ldots, s_n\}$, Capacity constraint $C = \{c_1, \ldots, c_n\}$, d-dimensional space Ω with $\sum C = \int_{x \in \Omega} dx$

Output: Capacity-constrained ordinary Voronoi tessellation $\mathscr{V}(S, C)$

```
1  Compute the Voronoi tessellation V(S);
2  repeat
3      stable := true;
4      Initialize a set I := {1, ..., n};
5      while I ≠ ∅ do
6          Choose the site s_i with i ∈ I, s_i ∈ S and |V(s_i)| − c_i < |V(s_j)| − c_j for each
           j ∈ I, j ≠ i, s_j ∈ S;
7          Minimize D(V(S), C) by adjusting the location of s_i and recomputing the
           Voronoi tessellation V(S);
8          if the location of s_i changed then
9              stable := false;
10         end
11         Remove i from I;
12     end
13 until stable = true ;
```

1. A set $I := \{1, \ldots, n\}$ is initialized, which represents the indices of the sites in set S that have not been optimized in this iteration.

2. While the set I is not empty, which means that there are sites that have not been optimized in this iteration:

   (a) Choose the site s_i with index $i \in I$ for which $|V(s_i)| - c_i$ is minimal for all the sites that have not been optimized in this iteration. The site s_i is thereby the smallest non-optimized site in this iteration with respect to its intended capacity c_i.

   (b) Minimize $\mathscr{D}(\mathscr{V}(S), C)$ by adjusting the location of s_i and simultaneously recomputing the Voronoi tessellation $\mathscr{V}(S)$.

   (c) Remove index i from the set I, which indicates that s_i has been optimized in this iteration.

3. If none of the site locations has changed during this iteration, then the optimization converged to a stable state, and the algorithm terminates.

The priority for the sites implemented in step 2(a) and line 6 in Algorithm 3.1 significantly improves the convergence of the algorithm. The reason for this is that a site

with an undersized Voronoi region just has to identify the most oversized and/or least undersized neighboring site, and then directly moves into this direction thereby increasing its own area and decreasing the other's area. In contrast, a site with an oversized region must avoid other sites to decrease its area. Rather, oversized regions have to wait until undersized regions indirectly decrease them. Hence, prioritizing the sites with undersized regions effectively moves them towards oversized regions which profit thereof later in the iteration.

The minimization in step 2(b) and line 7 in Algorithm 3.1 can be performed by any optimization algorithm that works on function samples. A good choice is the Downhill simplex method [56] because it reliably minimizes the function locally around a given starting location, which is preferably the current site location. Furthermore, it is not necessary that the minimization for each individual site is performed until it finds a local minimum. Rather, a few minimization steps are sufficient, which results in a much shorter runtime than performing a complete minimization.

Due to the fact that the given space Ω may be bounded, it is possible that some resulting site locations are not in Ω if the minimization is solely based on Equation 3.1. This can be avoided by adding a penalty to site locations that are not in Ω. Good results were generated by using a penalty function $\mathscr{P}_1$ that is applied to the currently optimized site s_i and then added to the value of $\mathscr{D}$ with

$$\mathscr{P}_1(s_i) = \begin{cases} 0 & \text{if } s_i \in \Omega, \\ (\|s_i - \Omega\| c_i)^2 & \text{if } s_i \notin \Omega \end{cases}. \tag{3.2}$$

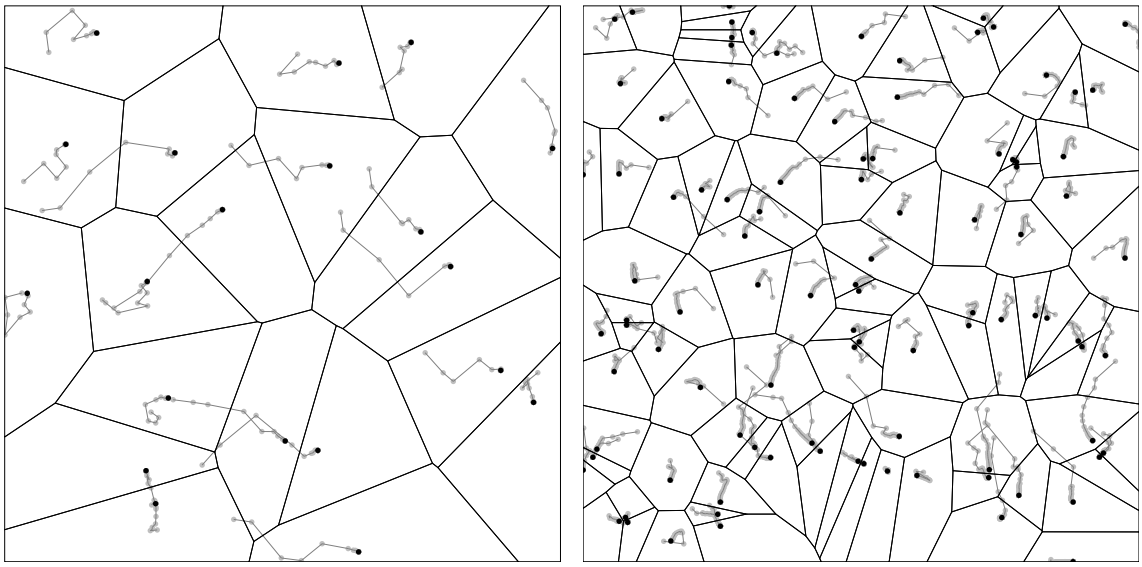
This function adds a capacity-dependent penalty based on the distance of the site location to Ω.

Another effect of the presented algorithm is that sometimes two or more neighboring sites are moved very close together, almost having the same site location. This occurs if neighboring sites possess regions that are much larger then their desired capacity. Then the minimization moves these sites closer together, and as result, their combined region area is reduced. Even though such Voronoi tessellations are still valid and fulfill the capacity constraint, the resulting close sites are often unfavorable with regard to the intended application. This problem is remedied by applying another penalty $\mathscr{P}_2$ to the currently optimized site s_i that is then added to the value of $\mathscr{D}$ with

$$\mathscr{P}_2(s_i) = \sum_{j=1}^{n} \begin{cases} 0 & \text{if } i = j \text{ or } \delta >= 1, \\ (1-\delta)\, c_i c_j & \text{if } i \neq j \text{ and } \delta < 1, \end{cases} \text{ with } \delta = \frac{\|s_i - s_j\|}{\sqrt{c_i + c_j}} \tag{3.3}$$

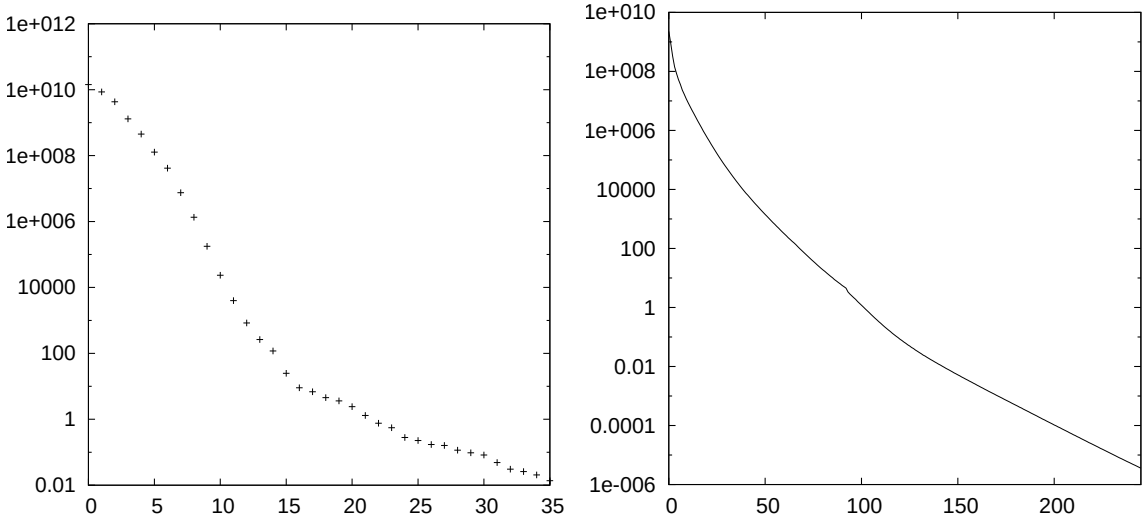
This function adds a capacity-dependent penalty if the site is very close to another site, and no penalty if the site does not have nearby neighboring sites.

Two examples for capacity-constrained ordinary Voronoi tessellations generated with Algorithm 3.1 are given in Figure 3.1. The tessellation in example (a) consists of 20 sites with equal capacities, whereas the tessellation in example (b) consists of 100 sites with



(a) 20 sites with equal capacities (b) 100 sites with different capacities

Figure 3.1: Capacity-constrained ordinary Voronoi tessellations computed with Algorithm 3.1. The traces illustrate the site movements during the computation.



(a) 20 sites with equal capacities (b) 100 sites with different capacities

Figure 3.2: Development of the function $\mathscr{D}$ for each iteration of the computation examples in Figure 3.1.

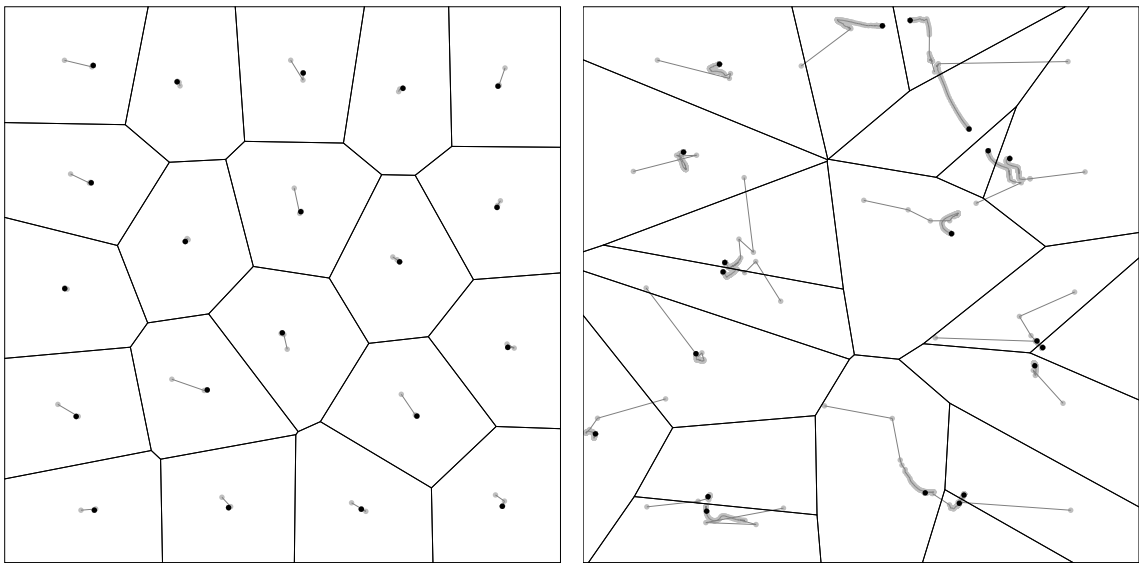
different capacities. The computation of example (a) is further illustrated in Figure 3.10 at the end of this chapter by a sequence of iterations. The development of the capacity error $\mathscr{D}$ of these two examples is given in Figure 3.2, showing a clear logarithmic convergence. The computation of the example with 20 sites required 35 iterations in less than 5 seconds on Intel Core 2 hardware, computing more than 21000 function samples of $\mathscr{D}$ for the minimization in step 2(b), including the update of the Voronoi tessellation for each sample and the extraction of the individual polygons. The computation time for the necessary 245 iterations for the 100 sites example was approximately 16 minutes, computing more than 3.2 million function samples.

The convergence and the result of an individual computation depends on the initial distribution of sites. We can use this to our advantage if all sites have similar capacities. In this case, we can at first generate an approximation of a centroidal Voronoi tessellation with a few iterations of Lloyd's method, and then use the resulting sites as the initial set for the computation of a capacity-constrained ordinary Voronoi tessellation. The result of this combination is usually a homogeneous distribution of sites that all reside near the centroids of their corresponding Voronoi regions. Additionally, the convergence of the algorithm is greatly improved by this variant due to the fact that a centroidal Voronoi tessellation is a good first approximation of a capacity-constrained ordinary Voronoi tessellation with equal capacities. An example for the development of a Voronoi tessellation in this variant of our algorithm is given in Figure 3.3(a). Unfortunately, the same effect cannot be observed for sets of sites with differing capacities, such as the Voronoi tessellation in Figure 3.3(b), because of their dissimilarity to centroidal Voronoi tessellations. The development of the function $\mathscr{D}$ for these two examples that used centroidal Voronoi tessellations as initial sets of sites is shown in Figure 3.4.

3.1.2 Convergence and Computational Complexity

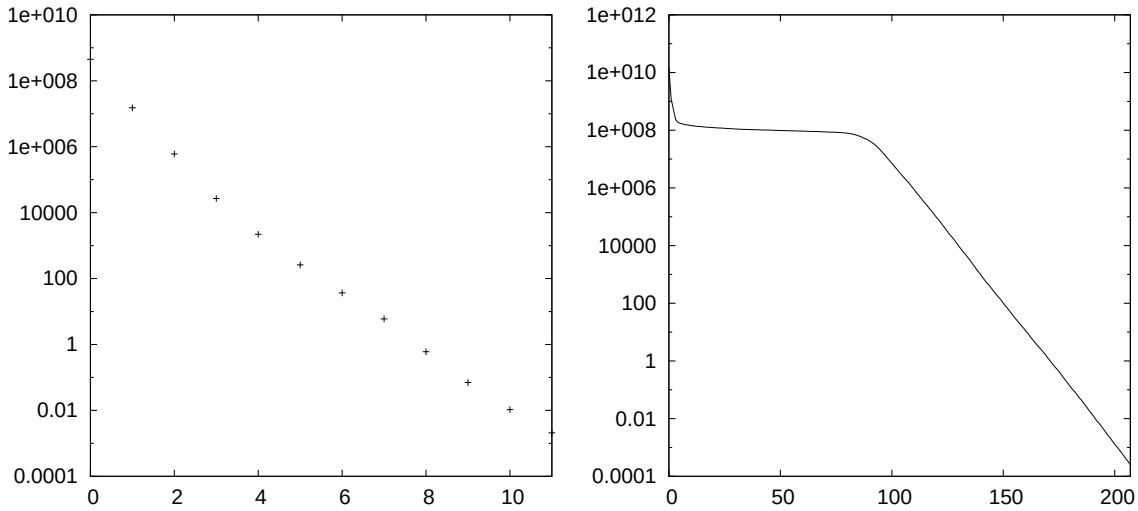
The convergence of the minimization can be guaranteed if the minimization algorithm in step 2(b) and line 7 does not increase the value of $\mathscr{D}$. It is not guaranteed that the algorithm terminates in a global minimum. However, during extensive tests, the algorithm always terminated in the global minimum $\mathscr{D} = 0$, and never in a local minimum $\mathscr{D} > 0$. If it happens that a local minimum with $\mathscr{D} > 0$ is achieved, it should be sufficient to move all sites that do not fulfill their capacity constraint to new random locations within Ω, and then to proceed with the minimization. If these potential cases of local minima are neglected, the algorithm shows clear logarithmic convergence towards $\mathscr{D} = 0$. This convergence is preserved if one or both of the penalty functions $\mathscr{P}_1$ and $\mathscr{P}_2$ are utilized.

The values for the computation time of the presented examples show that the computation of a capacity-constrained ordinary Voronoi tessellations is very expensive. Especially large sets of sites, with many thousands or even millions of sites are beyond computational feasibility. The main reason for this is that each function sample during the minimization consists of the update of one site in the Voronoi tessellation with $O(\log n)$ and the eval-



(a) 20 sites with equal capacities (b) 20 sites with different capacities

Figure 3.3: Capacity-constrained ordinary Voronoi tessellations with 20 sites that used centroidal Voronoi tessellations as initial sets of sites for the computation.



(a) 20 sites with equal capacities (b) 20 sites with different capacities

Figure 3.4: Development of the function $\mathscr{D}$ for each iteration of the computation examples that used centroidal Voronoi tessellations as initial sets of sites in Figure 3.3.

uation of the area of each Voronoi region with $O(n)$. Hence, the resulting computational time complexity for one single function sample is $O(\log n + n)$. The time complexity for a complete iteration is $O(n^2 + n \log n)$.

3.2 Weighted Distance Functions

The Voronoi region $V(s_i, w_i)$ of a site $s_i \in S$ in a weighted Voronoi tessellation $\mathscr{V}(S,W)$ is determined by the relative location of all sites $s_j \in S$ that are neighbors to s_i. The distance function that determines the neighbor topology and forms the bisectors additionally incorporates a weight $w_i \in W$ for each site $s_i \in S$. Thus, the set W of weights offers an additional degree of freedom to control the resulting Voronoi regions.

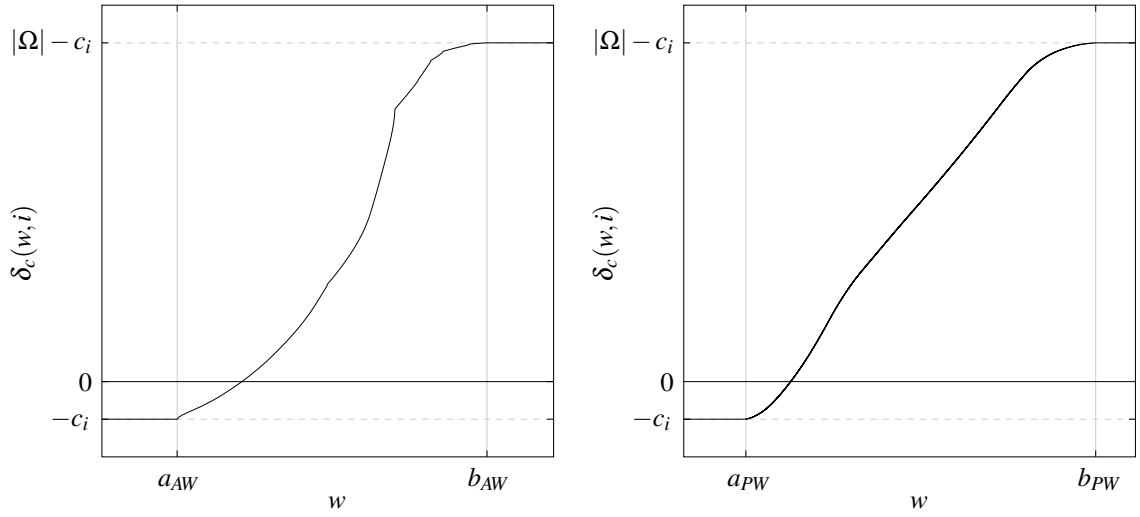
The incorporation of a weight parameter in the distance computation for a weighted Voronoi tessellation $\mathscr{V}(S,W)$ can have many functional forms. All weighted distance functions allow to increase or decrease the area of a Voronoi region by solely increasing and/or decreasing the weight of the respective site. The modification of the site locations is typically not necessary to achieve a given capacity constraint [3]. The common distance functions either use an *additive* or a *multiplicative* weight term. Instances for an additive term are the AW and PW distance functions presented in Section 2.2. An example for the multiplicative term is the *multiplicatively weighted Voronoi tessellation* that uses the distance function $d_{MW}(s_i, w_i, x) = \frac{1}{w_i}\|s_i - x\|, w_i > 0$. All other weighted Voronoi tessellations can either be reduced to these two types, or at least possess similar characteristics [58]. The advantage of distance functions with additive weight terms is that they result in connected Voronoi regions, whereas multiplicative weight terms often generate disconnected regions.

The generation of a capacity-constrained weighted Voronoi tessellation $\mathscr{V}(S,W,C)$ for n sites with their respective weights in W and their capacities in C in the d-dimensional space $\Omega \subset \mathbb{R}^d$ can be formulated as an optimization problem in n dimensions. The variables in this optimization are the weights of all n sites. The continuous function $\mathscr{D}(\mathscr{V}(S,W),C)$ that has to be minimized is equivalent to the function for the ordinary case in Equation 3.1. A weighted Voronoi tessellation $\mathscr{V}(S,W)$ that fulfills the given capacity constraint C is achieved if $\mathscr{D} = 0$, representing the global minimum of $\mathscr{D}$. Similar to ordinary distance functions, such global minimum can be determined by general optimization methods that work on function samples. Here, the same problem occurs as for ordinary distance functions: for large sets of sites the high dimensionality of the parameter vector entails a prohibitive runtime and numerical problems during the optimization.

The solution to this problem of high dimensionality for the optimization can again be remedied by not optimizing all site weights at once, but rather iteratively optimizing just one site weight at a time. In contrast to the algorithm for ordinary distance functions, we optimize the following function $\mathscr{D}'$ during each minimization of one single site:

$$\mathscr{D}'\left(\mathscr{V}\left(S,W\right),C,i\right) = \left(|V(s_i,w_i)| - c_i\right)^2. \tag{3.4}$$

This reduces the dimensionality of an optimization step from n to 1. Similar approaches



(a) AW distance function (b) PW distance function

Figure 3.5: Correlation between the weight w of a site s_i and the resulting capacity error $\delta_c(w,i)$. The weights of all other sites $s_j \in S, i \neq j$ remained fixed.

of individually adjusting the site weights for generating capacity-constrained Voronoi tessellations were also chosen by Ohyama [57] and Reitsma [64].

3.2.1 Correlation Between Site Weight and Region Area

The minimization of $\mathscr{D}'$ for a single site $s_i \in S$ is performed by adjusting the weight $w_i \in W$. For analyzing the correlation between a weight w that is associated with a site s_i, and the area of the resulting Voronoi region $V(s_i,w)$ in a bounded continuous space Ω, we consider the capacity error

$$\delta_c(w,i) = |V(s_i,w)| - c_i. \tag{3.5}$$

It becomes apparent that this capacity error δ_c is a continuous function with three characteristic intervals, illustrated in Figure 3.5. In the first interval $w \in (-\infty, a]$, the site s_i is dominated by the other sites $s_j \in S, i \neq j$, thus the Voronoi region $V(s_i,w)$ disappears. The capacity error in this interval is constant with $\delta_c(w,i) = -c_i$. In the second interval $w \in (a,b)$, the Voronoi region of the site s_i occupies some part but not the whole area of Ω. In this interval, $\delta_c(w,i)$ is monotonically increasing, and passing its root $\delta_c(w,i) = 0$. In the third interval $w \in [b,\infty)$, the site s_i dominates the whole bounded space Ω, having a constant capacity error of $\delta_c(w,i) = |\Omega| - c_i$.

This behavior of the capacity error is in general independent of the utilized distance function. The lower bound is always $\delta_c = -c_i$ and the upper bound is always $\delta_c = |\Omega| - c_i$. Of course, the actual weights a and b that determine the intervals highly depend on the utilized distance function, and on the individual locations and weights of the other sites.

For example, the plots in Figure 3.5 were generated in a square with edge length 1000 using a set of 10 random sites. The weights of all sites s_j other than s_i were $w_j = 0$. The actual interval bounds for the AW distance function were at $a_{AW} \approx -200$ and $b_{AW} \approx 626$, and for the PW distance function they were at $a_{PW} \approx -89860$ and $b_{PW} \approx 538067$.

3.2.2 Algorithm

The global minimum of $\mathscr{D}'$ is achieved if the capacity error for each site is zero, $\delta_c(w_i, i) = 0$. Hence, we have to find the root of the function δ_c for each site, which is simple due to its predictable behavior. A good method for finding the root of δ_c is the false position method, which reliably converges towards $\delta_c = 0$ at a faster rate than the also reliable bisection method. In contrast, methods that utilize the function's gradient, such as the Newton's method or the secant method, are not sufficient due to the constant values of $\delta_c(w,i)$ for $w \leq a$ and $w \geq b$ with respect to Figure 3.5. The result is the following approach for the computation of capacity-constrained weighted Voronoi tessellations in continuous spaces, which is outlined in Algorithm 3.2.

Algorithm 3.2: Weighted Distance Functions in Continuous Spaces

Input: Sites $S = \{s_1, \ldots, s_n\}$, Capacity constraint $C = \{c_1, \ldots, c_n\}$, d-dimensional space Ω with $\sum C = \int_{x \in \Omega} dx$

Output: Capacity-constrained weighted Voronoi tessellation $\mathscr{V}(S, W, C)$

```
1  Initialize a set of n weights W := {0, ..., 0};
2  repeat
3      stable := true;
4      foreach site s_i ∈ S do
5          Find the weight w such that δ_c(w, i) = 0 via the false position method;
6          if w_i ≠ w then
7              stable := false;
8          end
9          w_i := w;
10     end
11 until stable = true ;
```

The input for the algorithm is a set S of n sites, a set C of n associated capacities, and a d-dimensional space Ω. The result of the algorithm is a set W of n weights that determines a capacity-constrained weighted Voronoi tessellation $\mathscr{V}(S, W, C)$ in Ω with $\mathscr{D}(\mathscr{V}(S, W), C) = 0$. The algorithm starts with the initialization of the set W of n weights with $w_i = 0$, $w_i \in W$. Then, the initial Voronoi tessellation $\mathscr{V}(S, W)$ is iteratively optimized towards the capacity constraint C by adjusting the weights of the sites one at a time. The weight adjustment for each site s_i is performed by finding the root w of the function δ_c

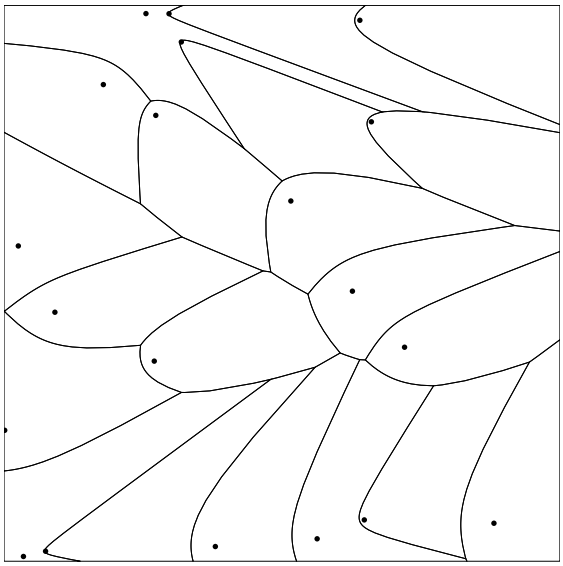
via the false position method. This root w is then assigned to the weight w_i. The overall algorithm stops if the weight of not even one site can be further improved.

Four examples of capacity constrained weighted Voronoi tessellations generated with Algorithm 3.2 are presented in Figure 3.6. Two of them are based on 20 sites with equal capacities, and the other two are based on 100 sites with different capacities. The computation of the examples in Figure 3.6(a) and Figure 3.6(c) is further illustrated in Figure 3.11 and Figure 3.12 at the end of this chapter by a sequence of iterations. In the AW distance function results, the elongated Voronoi regions emerge due to the fact that the sites always reside within their corresponding Voronoi regions. In contrast, the results using the PW distance function exhibit much more compact Voronoi regions, but the sites do not necessarily reside within their corresponding regions. The development of the function $\mathscr{D}$ that lead to these result is shown in Figure 3.7. These plots clearly illustrate the superlinear convergence of $\mathscr{D}$ throughout the computation using Algorithm 3.2. An overview of the computation time, the number of iterations, the number of necessary Voronoi tessellation updates for evaluating δ_c during the root finding, and the minimum and maximum weights of the results are presented in Table 3.1. Note that the much larger computation time for the AW distance function examples is reasoned by the hyperbolic bisectors of the resulting regions. Most of their computation time was used to extract these hyperbolic bisectors and intersect them with the bounding polygon.

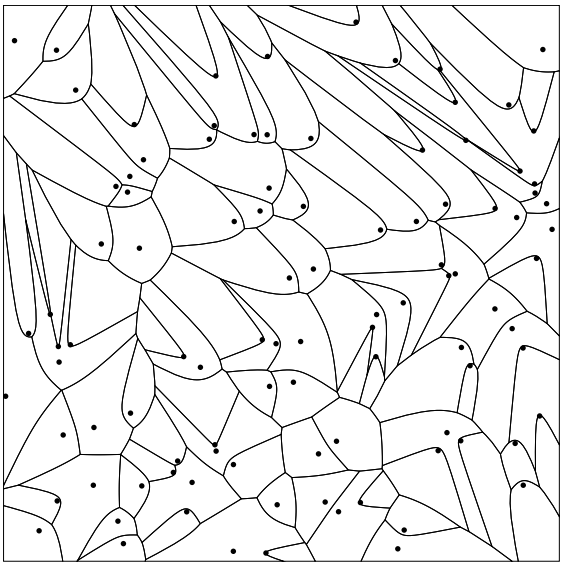
3.2.3 Centroidal Variant

The computation of capacity-constrained Voronoi tessellations in Algorithm 3.2 is solely based on the modification of the weights of the sites. Hence, elongated regions appear in the case of the AW distance function, and sites that do not reside within their corresponding Voronoi region in the case of the PW distance function. Even though such Voronoi tessellations are valid and fulfill the capacity constraint, they may be unfavorable with regard to the intended application. The solution to both problems is to additionally modify the locations of the sites to achieve more homogeneous distributions. Here, the best results are achieved if the sites reside in the centroids of their corresponding regions. In such case, the resulting tessellation becomes a *centroidal capacity-constrained Voronoi tessellation*.

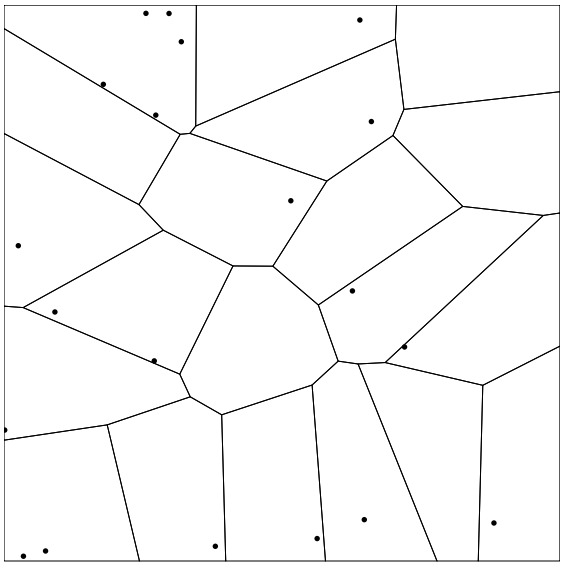
A straightforward approach to achieve the additional constraint of centroidal sites is to directly integrate the site relocation into Algorithm 3.2. We therefore insert the additional step of relocating the site s_i to the centroid p_i of the corresponding region $V(s_i, w_i)$ between line 4 and 5 of the algorithm. The advantage of this additional relocation step is that the sites will now be distributed homogeneously within the underlying space. The disadvantage is that this additional operation deteriorates the convergence of the algorithm. A small site location change may entail a couple of other relocations that result in a dramatic increase of the value of $\mathscr{D}$. Nevertheless, experiments showed that the algorithm still exhibits a reliable convergence towards the global minimum $\mathscr{D} = 0$.



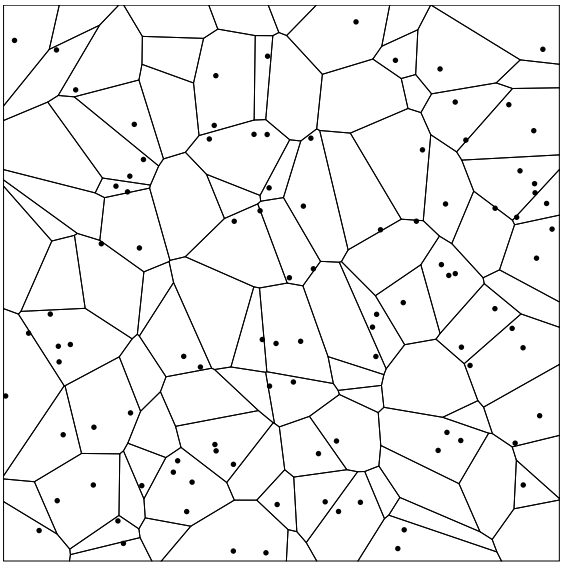
(a) 20 sites with equal capacities using the AW distance function



(b) 100 sites with different capacities using the AW distance function

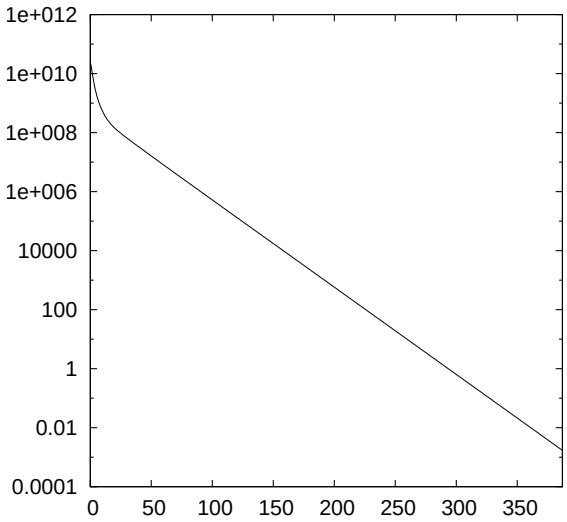


(c) 20 sites with equal capacities using the PW distance function

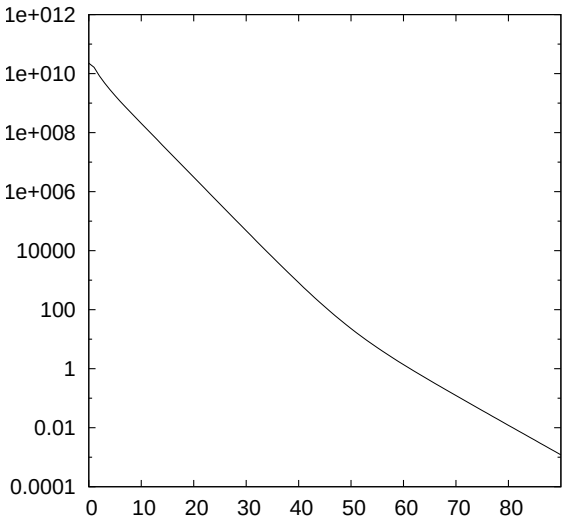


(d) 100 sites with different capacities using the PW distance function

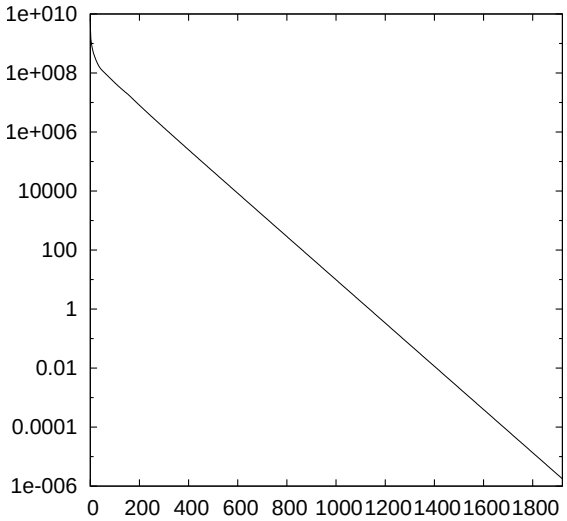
Figure 3.6: Capacity-constrained weighted Voronoi tessellations computed with Algorithm 3.2.



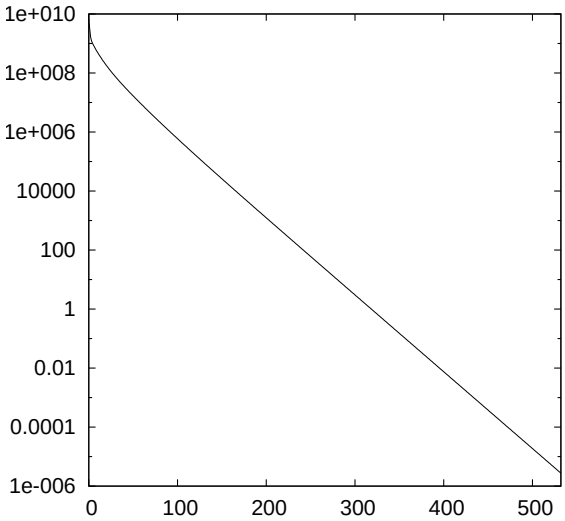
(a) 20 sites with equal capacities using the AW distance function



(b) 100 sites with different capacities using the AW distance function



(c) 20 sites with equal capacities using the PW distance function



(d) 100 sites with different capacities using the PW distance function

Figure 3.7: Development of the function $\mathscr{D}$ for each iteration in the examples in Figure 3.6.

dataset	time	iterations	updates	min. weight	max. weight
Figure 3.6(a)	4 sec	387	28379	−322	300
Figure 3.6(b)	152 sec	1921	527475	−414	291
Figure 3.6(c)	0.1 sec	90	8364	−138646	136133
Figure 3.6(d)	5 sec	532	141572	−101732	−61562
Figure 3.8(a)	3 sec	219	17474	−53	22
Figure 3.8(b)	96 sec	988	352274	−50	2
Figure 3.8(c)	0.2 sec	196	13284	−22045	−14685
Figure 3.8(d)	7 sec	882	238472	−6951	19

Table 3.1: Overview of the computation time, the number of iterations, the number of necessary Voronoi tessellation updates during the root finding, and the minimum and maximum weights of the results for all weighted examples.

Four examples for the generation of centroidal capacity-constrained Voronoi tessellations are presented in Figure 3.8. These examples are based on the same data as the previous four examples, whereas, in contrast, their computation now included the additional site relocation step. All four examples exhibit homogeneous site distributions with very compact Voronoi regions, and sites that always reside in the region centroids. An overview of the computation time, the number of iterations, et cetera, is given in Table 3.1. By analyzing the development of the function $\mathscr{D}$ for these four examples in Figure 3.9, it becomes apparent that during the first part of the computation, the centroidal variant converges faster than the original non-centroidal algorithm. Furthermore, during these iterations, the permanent change of site locations results in an alternation of the value of $\mathscr{D}$. In the second part, the distribution becomes stable, sometimes with a last large change of various site locations that results in a major increase of $\mathscr{D}$. Afterwards, the distribution is stable, only minor site location changes occur, and the weights are adjusted until the capacity constraint is finally fulfilled, showing a slightly slower convergence than the original non-centroidal algorithm.

3.2.4 Convergence and Computational Complexity

The convergence of the algorithm is not guaranteed due to the fact that the optimization is based on the per site function $\mathscr{D}'$, and not on the global function $\mathscr{D}$. Thus, the improvement of the capacity error for one site may result in a higher deterioration of the capacity error of another site, which increases the value of $\mathscr{D}$. However, similar to the algorithm for ordinary distance functions, we could not observe cases where the algorithm did not converge to the global minimum $\mathscr{D} = 0$. Rather, the expansion and contraction of the regions always combined in a way that consistently reduced the overall capacity error towards the global minimum.

The computation times of the examples in this section are significantly smaller than

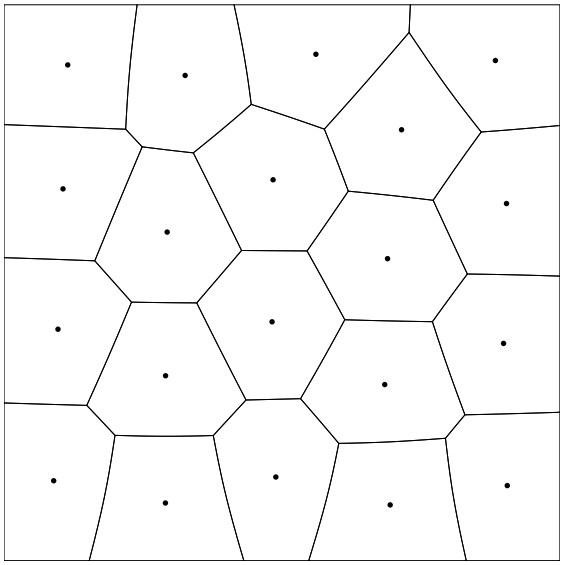
those of the ordinary distance function examples in the previous section. Actually, the weighted approach even allows to generate capacity-constrained Voronoi tessellations for large datasets with thousands or even millions of sites. For example, the computation of a dataset with one million sites of equal capacity using the PW distance function was performed in less than one day. The reason for this significantly reduced computational effort is the simplification of the optimization from a function for all sites to a much simpler function for just one single site. The resulting optimization for one single site is reduced to a simple root finding that only incorporates the update and extraction of one single Voronoi region. The computational time complexity for this operation is $O(\log n)$, reasoned by the update of the Voronoi tessellation. The time complexity for a complete iteration is $O(n \log n)$. In contrast, the ordinary case included the extraction of all Voronoi regions in the tessellation with a computational time complexity of $O(\log n + n)$ for a single function evaluation.

3.3 Conclusion

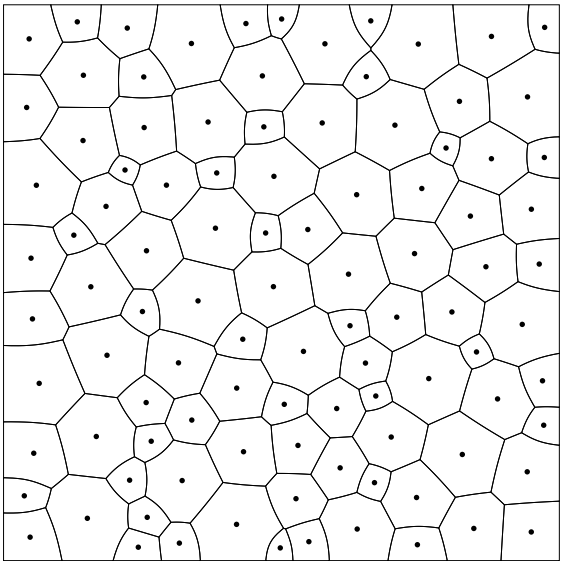
We presented two approaches for generating capacity-constrained Voronoi tessellations in continuous spaces based on ordinary and weighted distance functions. Although we cannot present a proof for the convergence of these algorithms, our experiments showed their reliable convergence towards arbitrarily precise capacity-constrained Voronoi tessellations. The computational time complexity of our approaches is $O(n^2 + n \log n)$ per iteration for ordinary distance functions, and $O(n \log n)$ per iteration for weighted distance functions. Unfortunately, the iterative structure of our algorithms and the costly function evaluation actually result in a high computational effort for large sets of sites. However, especially our weighted approach enables the generation of capacity-constrained Voronoi tessellations with thousands or even millions of sites.

3.4 Computation Sequences

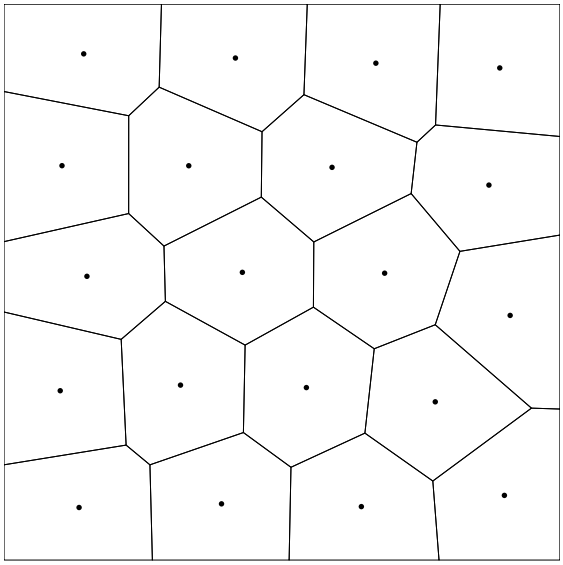
The computation of capacity-constrained Voronoi tessellations in continuous spaces is further illustrated in Figure 3.10, Figure 3.11, and Figure 3.12. All three examples are based on a set of 20 sites with equal capacities. These figures illustrate the development of the region areas as well as the site locations and/or weights during the computation. The color of each Voronoi region denotes its relative capacity error, where blue regions are too large, and red regions are too small. In the weighted examples, the circles additionally illustrate the weights of the sites, where green circles denote positive weights, and red circles denote negative weights.



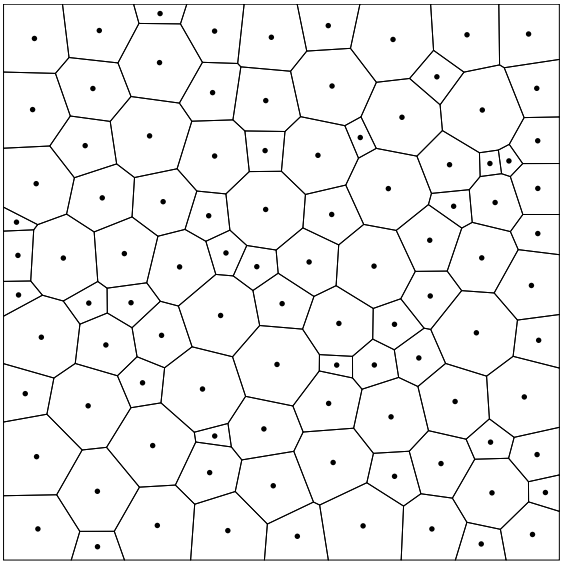
(a) 20 sites with equal capacities using the AW distance function



(b) 100 sites with different capacities using the AW distance function

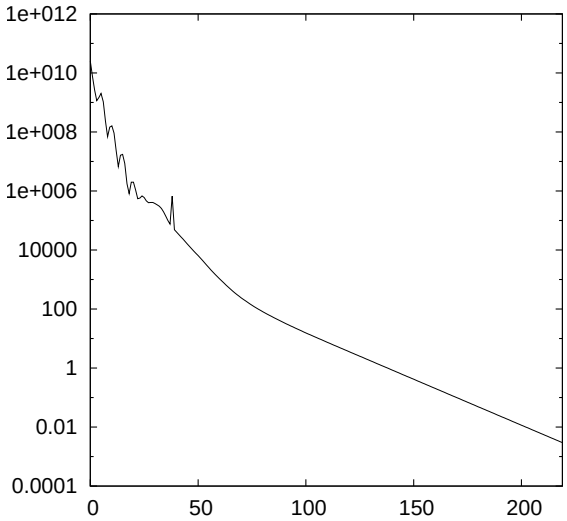


(c) 20 sites with equal capacities using the PW distance function

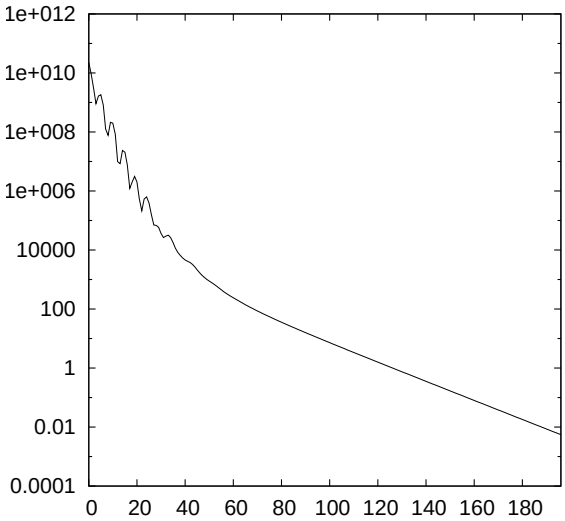


(d) 100 sites with different capacities using the PW distance function

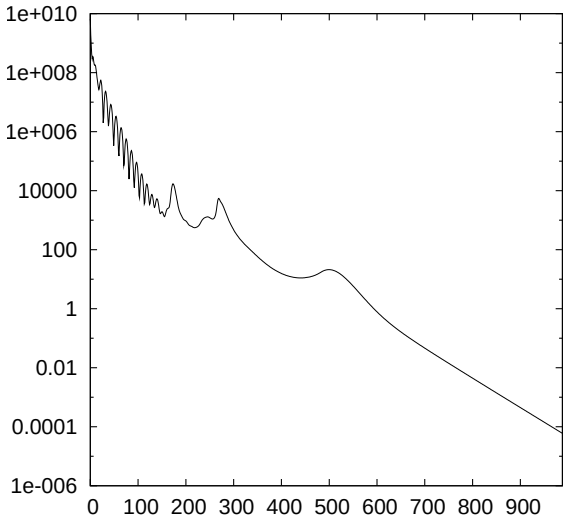
Figure 3.8: Centroidal capacity-constrained weighted Voronoi tessellations computed with the centroidal variant of Algorithm 3.2.



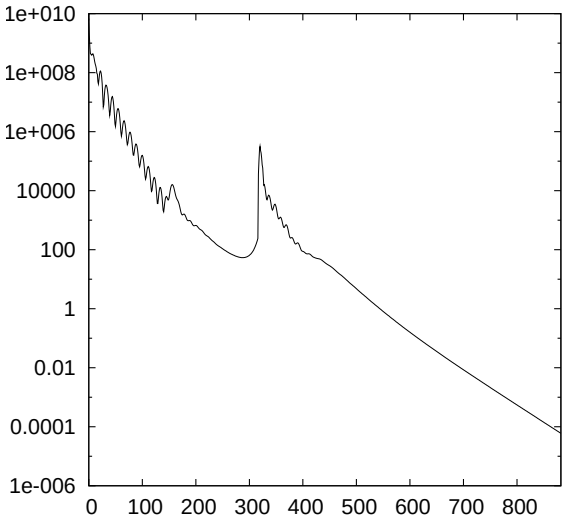
(a) 20 sites with equal capacities using the AW distance function



(b) 100 sites with different capacities using the AW distance function



(c) 20 sites with equal capacities using the PW distance function



(d) 100 sites with different capacities using the PW distance function

Figure 3.9: Development of the function $\mathscr{D}$ for each iteration in the examples in Figure 3.8.

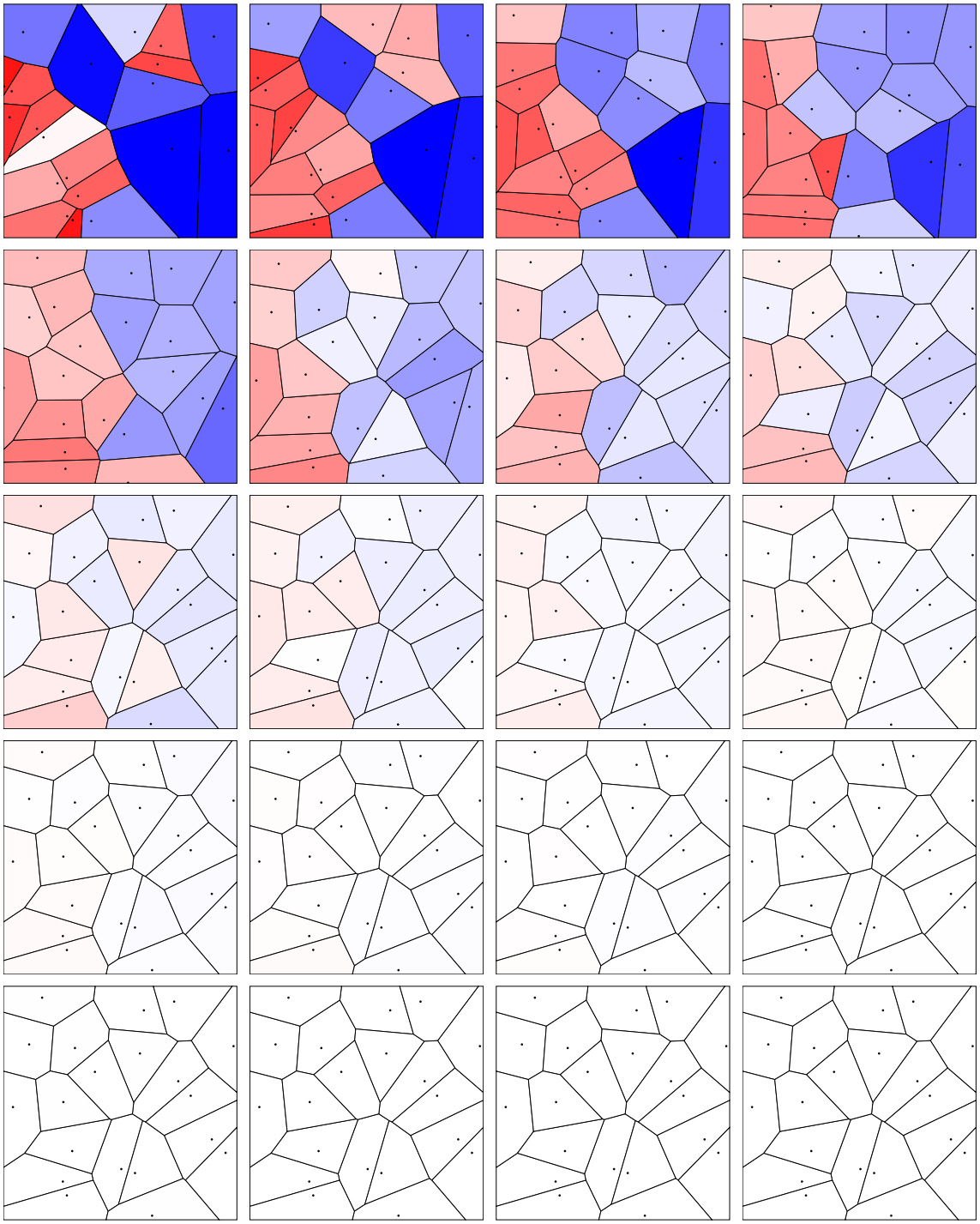


Figure 3.10: Development of the site locations during the computation of the capacity-constrained ordinary Voronoi tessellation of 20 sites with equal capacities in Figure 3.1(a): initial state, iterations 1–14, 17, 20, 25, 30, final result after iteration 35. The colors denote the relative capacity error of each Voronoi region, where blue regions are too large and red regions are too small.

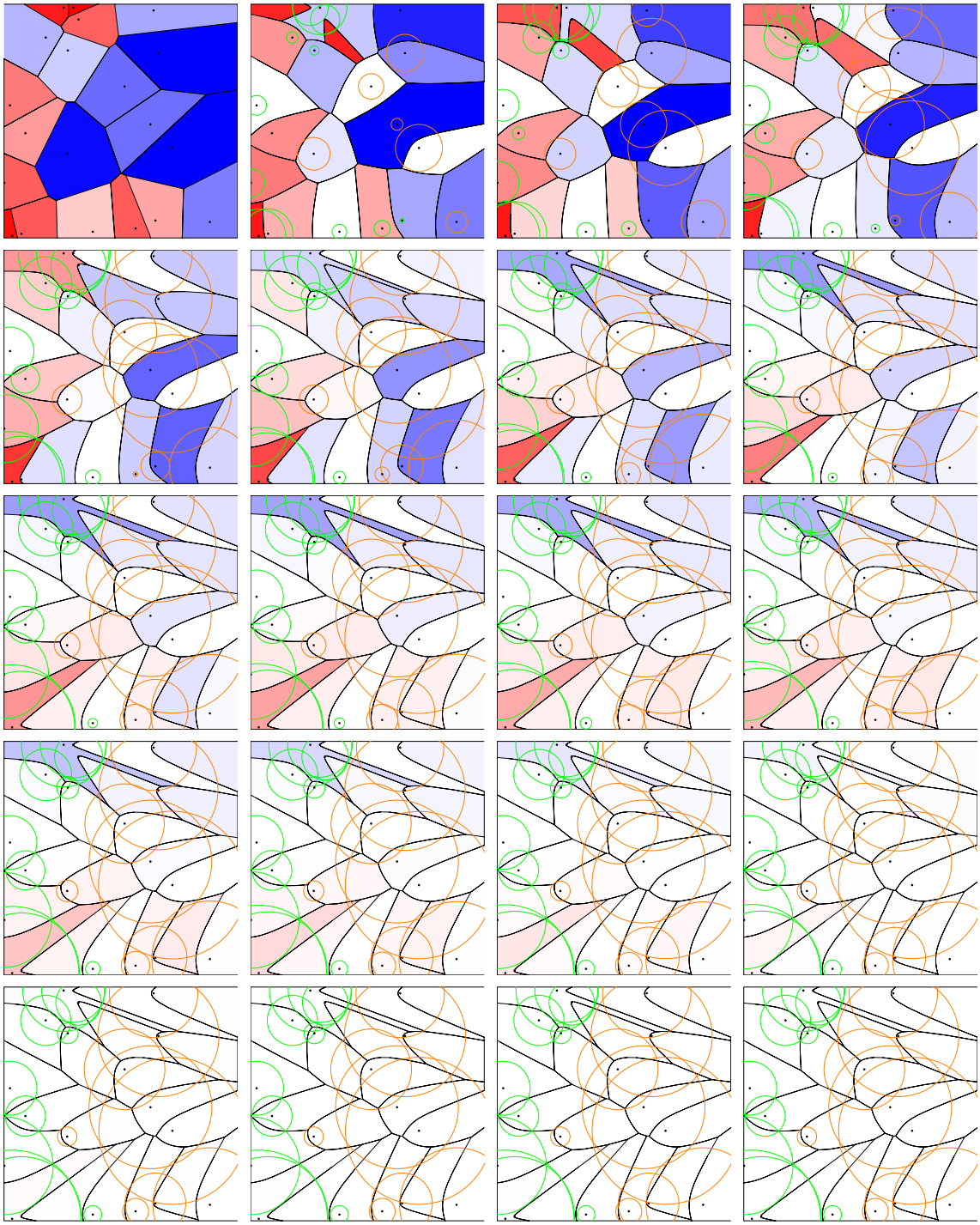


Figure 3.11: Development of the site weights during the computation of the capacity-constrained AW Voronoi tessellation of 20 sites with equal capacities in Figure 3.6(a): initial state, iterations 1–3, 5, 7, 10, 15, 20, 25, 30, 40, 50, 75, 100, 150, 200, 250, 300, final result after iteration 378. The circles additionally illustrate the weights, where green circles denote positive weights, and red circles denote negative weights.

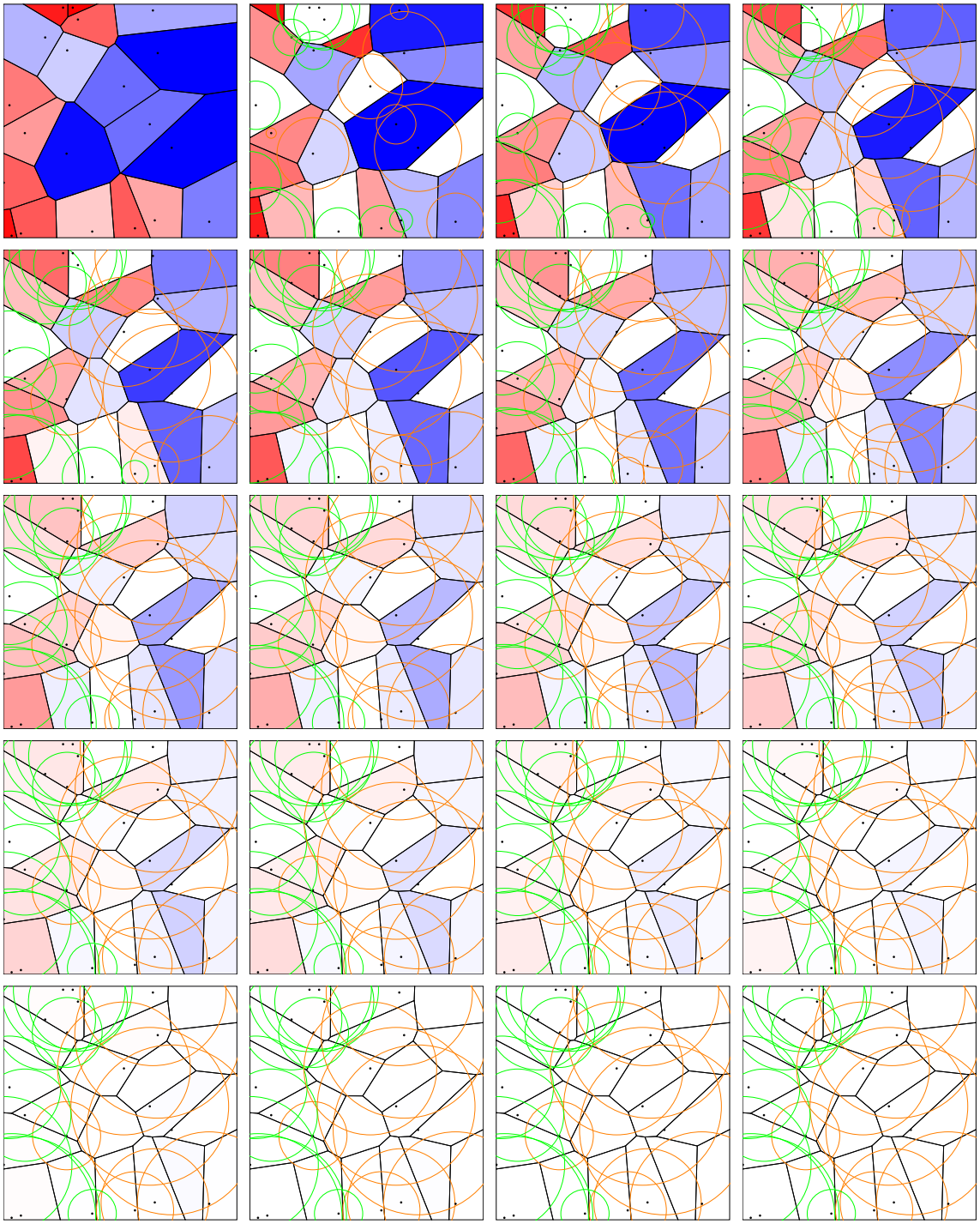


Figure 3.12: Development of the site weights during the computation of the capacity-constrained PW Voronoi tessellation of 20 sites with equal capacities in Figure 3.6(c): initial state, iterations 1–6, 8, 10, 12, 14, 16, 18, 20, 25, 30, 50, 70, final result after iteration 90. The circles additionally illustrate the square root of the weights, where green circles denote positive weights, and red circles denote negative weights.

Chapter 4

Computation in Discrete Spaces

The idea of the previous two approaches for the computation of capacity-constrained Voronoi tessellations is to start with an arbitrary set of sites, and then iteratively improve these sites until their Voronoi tessellation fulfills the given capacity constraint. Both approaches are particularly suited for continuous spaces for which minimal modifications of the locations and/or weights of the sites result in minimal modifications of the Voronoi region areas. However, these approaches are less suited for discrete spaces. The reason for this is that in discrete spaces, small modifications of the locations and/or weights of the sites may result in no area change at all if this change is smaller than the distance between neighboring discrete elements. In contrast, such small modifications may result in major area changes due to regular arrangements of the discrete elements. These discontinuities prohibit a smooth convergence of the previous algorithms, and may result in an alternation of the capacity error between two or more suboptimal states.

Nevertheless, capacity-constrained Voronoi tessellations can be generated in discrete spaces as well. In this chapter, we presents such discrete space algorithm. This algorithm is the first that is not restricted to a specific distance function or two dimensions, but rather allows a flexible and simple implementation of arbitrary spaces and density functions. Before we present our algorithm, we first provide some theoretical background that is necessary to motivate and understand our algorithmic approach.

4.1 Background

A set S of n sites in discrete Euclidean space $\mathbb{R}^d$ induces a partition of $\mathbb{R}^d$ into n regions. The region $V(s_i)$ of a site $s_i \in S$ consists of all points x that are closer to s_i than to any other site $s_j \in S$, $i \neq j$. This partition is known as the Voronoi tessellation of S in $\mathbb{R}^d$. For any finite set $X \subset \mathbb{R}^d$ of m points, which determines a *finite discrete space* in $\mathbb{R}^d$, the Voronoi tessellation of S induces a partition of X into subsets. It defines an *assignment* $A : X \to S$, given by $A(x) = s_i \Longleftrightarrow x \in V(s_i), x \in X, s_i \in S$. Note that points that are equally close to more than one site are not covered by this definition. Rather, the function A assigns such point to an arbitrary but fixed closest site. The number of points assigned to a particular site s_i, $|V(s_i)|$, is called the *capacity* c_i, and the sum of the capacities of all sites is $m = |X|$.

The capacity of a site depends on the arrangement of all sites, their pairwise Euclidean

distances, and the underlying distance function that governs the assignment of points to sites. Ordinary Voronoi tessellations use the Euclidean distance as their distance function. Weighted Voronoi tessellations attach an additional set W of *weights* to the sites, and vary the distance function in such a way that the weights influence the distances between points and sites. Thus, the capacities of the sites can be controlled by adjusting their weights. Again, we obtain an assignment function $A_W : X \to S$ which now additionally depends on the set of weights.

If we want to determine the assignment A_W that partitions a finite set X of points according to a finite set S of n sites in $\mathbb{R}^d$ in such a way that the set $C = \{c_i | i \in \{1,\ldots,n\}\}$ of capacities consists of given integer values $c \geq 0$, with $\sum C = |X|$, we have to find the set W of weights. For the PW Voronoi tessellation, Aurenhammer et al. [3] show that there always exists a set of weights that fulfills a given capacity constraint C. Furthermore, they demonstrate the equivalence of such a capacity-constrained PW Voronoi tessellation to a similarly constrained Euclidean least-squares assignment. In this process, they show that the PW Voronoi tessellation minimizes the expression

$$\sum_{x \in X} d_{PW}(s_i, w_i, x) = \sum_{x \in X} \|s_i - x\|^2 - \sum_{x \in X} w_i \tag{4.1}$$

over all possible assignments $A_W : X \to S$, regardless of the capacity constraint C. Here, the last sum is a constant for all assignments, and can therefore be omitted. Thus, a capacity-constrained PW Voronoi tessellation can be generated by performing a least-squares minimization.

The basic idea behind our approach to generate capacity-constrained Voronoi tessellations is to ignore the weights but maintain the capacity constraint from the very beginning. As shown in Equation 4.1, and in more detail in the work by Aurenhammer et al. [3], the PW Voronoi tessellation is equivalent to a least-squares minimization for which the set of weights is ignored. Hence, we obtain a PW Voronoi tessellation of the finite point set X for a finite site set S that fulfills a given capacity constraint C by performing a least-squares minimization on an arbitrary assignment $A : X \to S$ with the same capacity constraint C. In other words, the least-squares minimization of any capacity-constrained assignment results in a valid PW Voronoi tessellation with the same capacity constraint, if this capacity constraint is preserved in all minimization steps. Of course, we do not obtain the weights for the sites of the PW Voronoi tessellation, but this does not affect the correctness of the result. In the following, we describe and discuss our approach in detail.

4.2 Algorithm

The goal of our algorithm is to obtain a partition of a finite set X of m points into n subsets according to a set S of n sites with the capacity constraint C, where $\sum C = m$. This partition represents a PW Voronoi tessellation of S for the point set X. We are not interested in the actual weights of the PW Voronoi tessellation. The basic idea of our algorithm is

to start with an arbitrary assignment $A : X \rightarrow S$ that fulfills the capacity constraint C, and then perform an iterative least-squares minimization. Throughout this minimization, we preserve the capacity constraint C by exclusively swapping the assignment of two points belonging to different sites. Finally, the minimization converges to a stable state that represents a valid PW Voronoi tessellation. The time complexity of the algorithm is $O(n^2 + nm \log \frac{m}{n})$ for each iteration step, and its memory complexity is $O(n+m)$. Our method is described in detail in the next paragraphs and is outlined in Algorithm 4.1.

Algorithm 4.1: Capacity-Constrained PW Voronoi Tessellation in Discrete Spaces

Input: Set S of n sites, Set X of m points, Set C of n capacities with $\sum C = m$

Output: Assignment $A : X \rightarrow S$ that represents a PW Voronoi tessellation of S for X with $c_i = |V(s_i)|, s_i \in S, c_i \in C$

```
1  Initialize an arbitrary assignment A that fulfills the capacity constraint in C;
2  repeat
3      stable := true;
4      foreach pair of sites (s_i, s_j) with s_i, s_j ∈ S, i < j do
5          Initialize two max-heap data structures H_i, H_j;
6          foreach point x_i with A(x_i) = s_i do
7              insert x_i into H_i with Δe(x_i, s_i, s_j) as its key;
8          end
9          foreach point x_j with A(x_j) = s_j do
10             insert x_j into H_j with Δe(x_j, s_j, s_i) as its key;
11         end
12         while |H_i| > 0 and |H_j| > 0 and max(H_i) + max(H_j) > 0 do
13             modify the assignment A : X → S to A(max(H_i)) := s_j, and
               A(max(H_j)) := s_i;
14             remove max(H_i) from H_i, and max(H_j) from H_j;
15             stable := false;
16         end
17     end
18 until stable = true ;
```

We start with an arbitrary initialization of the assignment $A : X \rightarrow S$ in which the number of points $|V(s_i)|$ assigned to each site $s_i \in S$ equals the desired capacity c_i. Such initial assignment can be generated by randomly assigning points in X to sites in S as long as their capacity is not exceeded. Note that the sum of all capacities in C must be equal to the number of points in X. The initial assignment is most likely different from the corresponding PW Voronoi tessellation of the sites in S. Rather, we have to perform a least-squares minimization, which is directly related to the distance function of the PW Voronoi tessellation as shown in Equation 4.1.

The basic operation during the energy minimization is to swap the assignment of two points x_i and x_j belonging to different sites s_i and s_j. The restriction to swap operations guarantees that the capacity constraint for each site is preserved throughout the energy minimization. Of course, finding the minimal series of swaps that yields the global minimum of the energy function is computationally prohibitive. However, due to the concave characteristic of the energy function [3], it is possible to achieve a local energy minimum of the assignment with a gradient descent method. This means that any series of swaps, in which each swap individually reduces the energy of the assignment, will converge to an energy minimum.

Nevertheless, a preferably small number of swaps should be performed to reduce the runtime of the energy minimization. Therefore, the energy minimization is performed iteratively. In each iteration, all pairs of sites (s_i, s_j) with $s_i, s_j \in S$, $i < j$, are examined for swaps that improve the overall energy of the assignment. While examining a pair of sites, we perform the minimal number of swaps that yield an energy minimum for these two sites. This combination of pairwise examination and optimal swapping for each pair results in an improved runtime of the algorithm while preserving a linear memory complexity. If no swap is found for all pairs of sites, the iterative minimization process stops, and the resulting assignment is equivalent to the PW Voronoi tessellation of S. The necessary number of iterations until the minimization stops is not predictable, but is guaranteed to be finite because of the strictly monotonic decreasing energy function.

The underlying energy function $e(x,s)$ of the minimization is derived from the distance function of the PW Voronoi tessellation in Section 2.2, for which the additive weight component is omitted:

$$e(x,s) = \|x-s\|^2. \tag{4.2}$$

This naturally models the compactness and the shape of the bisectors of the regions in the PW Voronoi tessellation. To determine whether swapping two points of different sites reduces the energy of the assignment, it is necessary to evaluate their energy before and after the potential swapping. The energy difference Δe for a single point x_i, whose assignment is changed from site s_i to another site s_j, is given by

$$\Delta e(x_i, s_i, s_j) = e(x_i, s_i) - e(x_i, s_j), \tag{4.3}$$

with $\Delta e > 0$ if the energy of the assignment is reduced. To find the optimal swapping candidates between the points of two different sites, we perform a greedy-like approach with the following steps for each pair of sites (s_i, s_j):

1. We initialize two max-heap data structures H_i, H_j for storing the points assigned to s_i, and s_j, respectively.

2. For each point x_i assigned to s_i, we calculate the energy difference $\Delta e(x_i, s_i, s_j)$, and insert x_i into the max-heap H_i with Δe as its heap key.

3. Likewise, for each point x_j assigned to s_j, we calculate the energy difference $\Delta e(x_j, s_j, s_i)$, and insert x_j into the max-heap H_j with Δe as its heap key.

4. Now we can access the points of s_i and s_j in descending order with respect to Δe, starting with those points that give the largest energy improvement, and evaluate the influence of a swap on the overall energy:

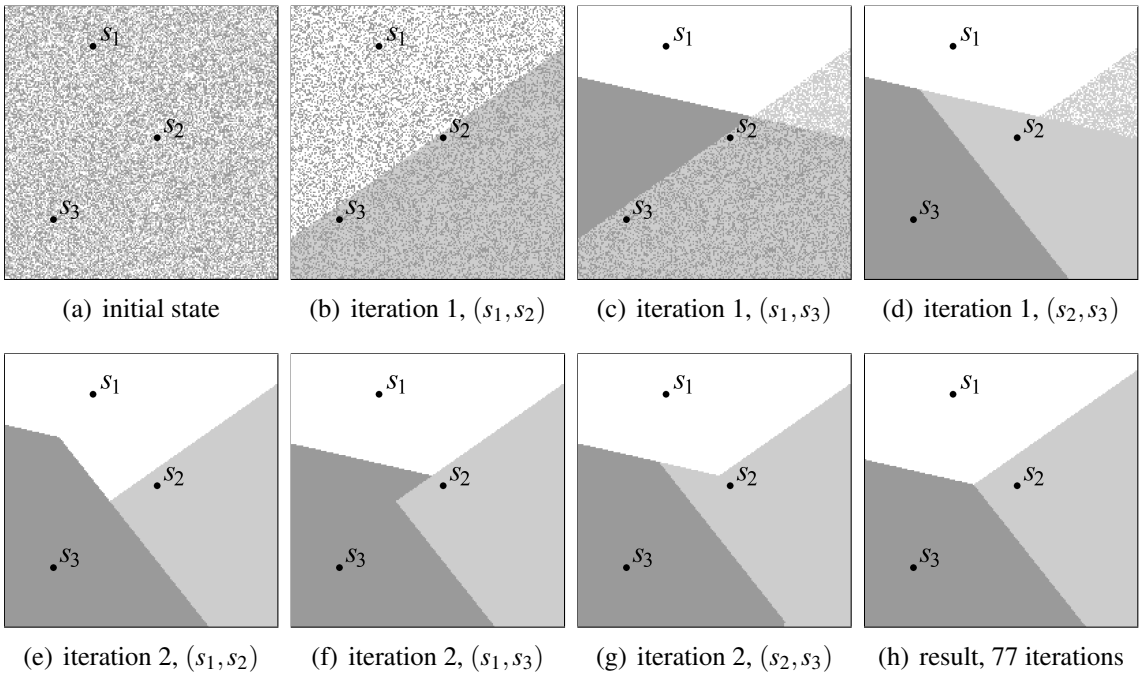
   (a) First, we test if both heaps contain any points. If this is the case, we combine the energy differences of the two maximum elements $max(H_i)$ and $max(H_j)$. If the result is greater than zero, a swap of the assignments of these two elements reduces the total energy, and we proceed with step (b). If the combined energy difference is equal to or less than zero, no swap will reduce the total energy. Hence, we can stop the evaluation of this pair of sites. The same applies if one of the heaps is empty.

   (b) To actually perform the swap, we change the assignment $A : X \rightarrow S$ for the two maximum elements to $A(max(H_i)) := s_j$ and $A(max(H_j)) := s_i$. We then remove these elements from the heap. Finally, we indicate that the assignment is not stable, by setting the corresponding flag to *false*. Then we proceed with step (a).

A further illustration of our algorithm is shown in Figure 4.1. This example consists of three sites with equal capacities, computed on a regular grid of 200×200 points in less than one second on Intel Core 2 hardware. Image (a) presents the random initial state of the assignment. The state after the optimal swapping of the points for the first pair of sites in the first iteration is shown in (b), with the correct orthogonal bisector of the pair of sites (s_1, s_2) evident. Similarly, the orthogonal bisectors of the pairs (s_1, s_3) and (s_2, s_3) are formed in (c) and (d), and the bisector between (s_1, s_2) is necessarily disrupted due to the swapping between the white and dark grey points of s_1 and s_3 in step (c). In the second iteration (e–g), this pairwise swapping is repeated. Again, the correct orthogonal bisector is formed for each pair, while disrupting other bisectors. However, the interval which is alternately occupied by the three sites is getting smaller from iteration to iteration. After 77 iterations, a stable state (h), which represents a valid PW Voronoi tessellation, is finally achieved.

The simple structure of our algorithm allows for its easy adaptation to arbitrary discrete spaces. The only prerequisite is a function to determine the distance between two points, which is used in lines 7 and 10 of Algorithm 4.1. Everything else is generic and independent of the underlying space. This also implies that we can easily implement other geometric objects than just points as sites.

The utilized heap data structure in our algorithm reduces its average runtime due to the fact that only a small fraction of points needs to be swapped, and for this reason we do not have to perform a complete sorting. A further speedup is achieved by introducing a bounding test for reducing the number of pairs of sites and the number of points that are evaluated. Therefore, for each site we construct a hypersphere that encloses all points assigned to that site. From the geometry of the problem it follows that there only exist beneficial swaps in the area where the hyperspheres of two sites overlap. This means that



(a) initial state (b) iteration 1, (s_1, s_2) (c) iteration 1, (s_1, s_3) (d) iteration 1, (s_2, s_3)

(e) iteration 2, (s_1, s_2) (f) iteration 2, (s_1, s_3) (g) iteration 2, (s_2, s_3) (h) result, 77 iterations

Figure 4.1: First steps and final result of the computation of a capacity-constrained PW Voronoi tessellation using Algorithm 4.1 for three sites with equal capacity on a regular grid.

pairs of sites with non-overlapping hyperspheres do not have to be evaluated. Furthermore, points that are not located in the overlapping area do not have to be evaluated as well. This considerably reduces the number of site pairs and points that have to be tested. The bounding test optimization has been omitted in the foregoing section to simplify the presentation.

The first few iterations of the algorithm take significantly longer than later ones, due to the fact that the majority of sites is swapped at the beginning, while later iterations just make minor modifications—see Figure 4.1 for an example. These first iterations can be significantly accelerated if we do not start with a random assignment, but rather with an assignment that already approximates the resulting Voronoi tessellation. Such initial assignments can be generated by assigning each point to the closest site for which the capacity is larger than the number of currently assigned points to this site. To speed up this initialization we can either use an ordinary Voronoi tessellation, or a *kd*-tree which provides a simpler implementation for arbitrary dimensions. An example for such closest site initialization based on a *kd*-tree is given in the complete computation sequence in Section 4.7 at the end of this chapter.

Figure 4.2 presents further examples of capacity-constrained Voronoi tessellations generated with Algorithm 4.1. Here, we computed similar datasets as for the case of weighted distance functions in continuous space in Section 3.2, even though the initial locations of the sites were different. Additionally, Section 4.7 at the end of this chapter

presents an overview of the computation process for the example in Figure 4.2(a). The computation time for these examples was between 12 seconds for example (a) and less than 4 minutes for example (d).

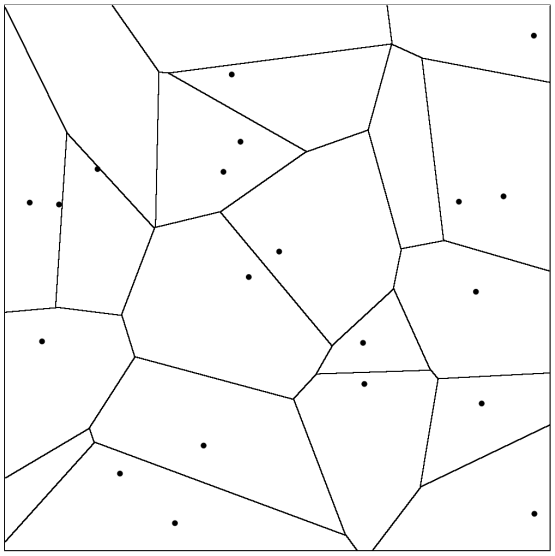
4.3 Convergence and Computational Complexity

Our algorithm performs swaps of the assignment of points if and only if such a swap decreases the total energy of the assignment. Thus, it is guaranteed that it converges to a stable state where no further swap improves the energy. In this stable state, the assignment is equivalent to a PW Voronoi tessellation, as proven by Aurenhammer et al. [3]—an exception to this rule is the extremely unlikely case of a cyclic dependency as described at the end of this section.

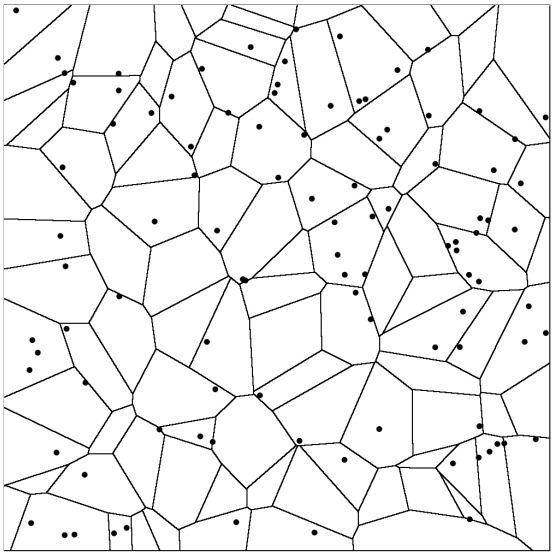
The complexity of our algorithm is influenced by the number of sites n and the number of points m. In the worst case, we have to evaluate n^2 pairs of sites in each iteration. For each pair, we have to initialize a heap data structure and organize $\frac{m}{n}$ points in the heap data structure according to the energy difference of the potential swap, which is equivalent to sorting—note that the worst case for the point heap is $\frac{m}{n}$ due to the fact that the while loop in line 12 stops if just one heap is empty. The resulting complexity is $O(n^2(1+\frac{m}{n}\log\frac{m}{n}))$. By reducing this term, we obtain a worst case time complexity for each iteration of $O(n^2+nm\log\frac{m}{n})$. The number of iterations is not directly related to n and m. Roughly speaking, the number of iterations increases/decreases as $\frac{m}{n}$ increases/decreases. Also note that in practical application, n is orders of magnitude smaller than m, so the quadratic term is usually dominated by the second linearithmic term. The memory complexity of our algorithm is $O(n+m)$.

The memory and time complexity of our algorithm is only linearly dependent on the dimensionality of the discrete space provided that both the number of sites n and the number of points m remain constant. For each additional dimension, we have to store one additional coordinate value, and we have to evaluate one additional dimension for the distance computation in our algorithm. Figure 4.3 illustrates this linear dependency between the dimensionality of the discrete space and the runtime of our algorithm. In image (a), we used 256 random sites and 256^2 random points, and measured the runtime for only the first iteration, in which by far the most assignment swaps occur. In image (b), we used 1024 random sites and 1024^2 random points, and measured the runtime for a complete optimization until convergence. Both images clearly show the linear dependency between number of dimensions and runtime. The outliers in the datasets result from the random placement of points and sites. However, their occurrence is uniformly distributed over the number of dimensions, and of similar magnitude.

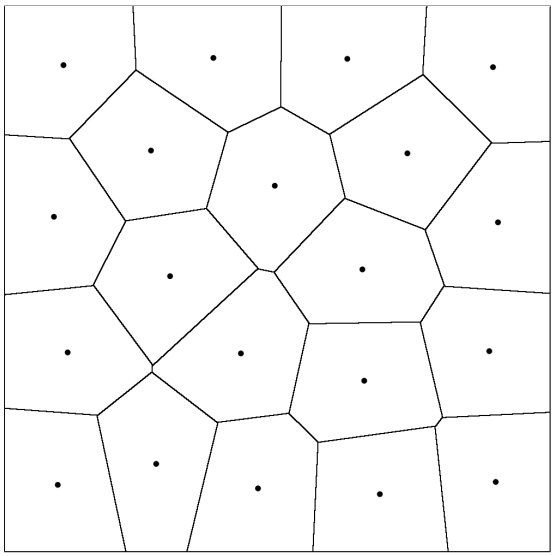
Depending on the application, it may be necessary that the number of points in the discrete space increases with its dimensionality. For example, our capacity-constrained point distributions presented in Chapter 6 require a minimum of around 10 points per



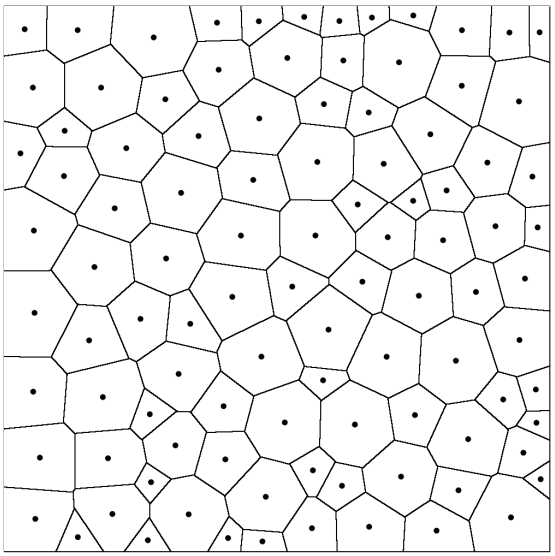
(a) 20 sites with equal capacities



(b) 100 sites with different capacities

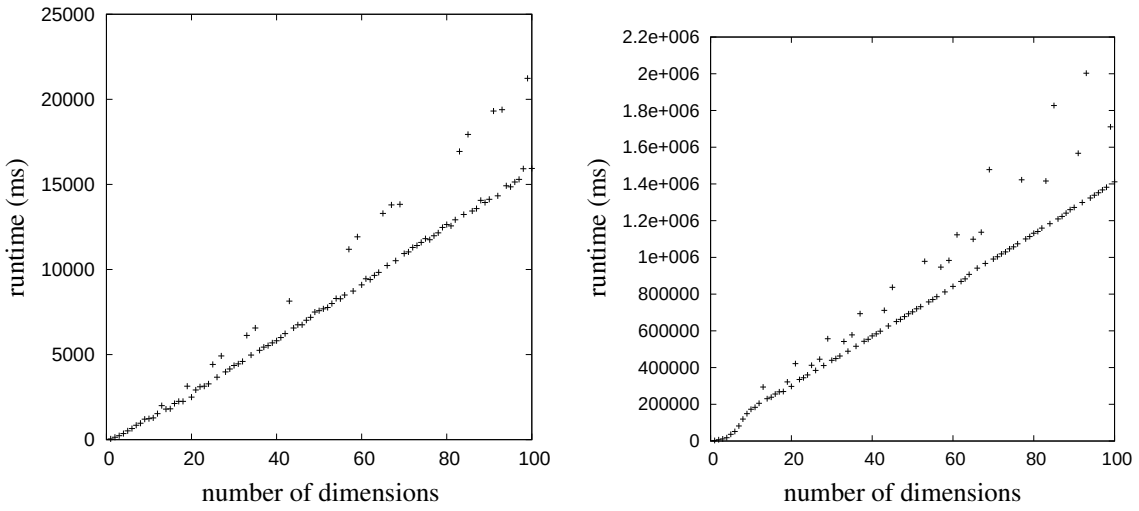


(c) 20 sites with equal capacities, centroidal



(d) 100 sites with different capacities, centroidal

Figure 4.2: Examples of capacity-constrained Voronoi tessellations generated with Algorithm 4.1 computed on a regular grid of 1000×1000 points. The computations of the tessellations in (c) and (d) additionally used the extension of centroidal sites as described in Section 4.4.2.



(a) 256 sites, 256^2 points, only first iteration (b) 1024 sites, 1024^2 points, complete computation

Figure 4.3: Dependency between the dimensionality of the discrete space and the runtime of Algorithm 4.1. The number of sites and the number of points in the discrete space remained constant during the test series.

dimension for each site to achieve blue noise characteristics of good quality. Thus, for this application there exists an exponential correlation between the number of sites and the number of points that are necessary to achieve sufficient results. Once again, note that this correlation completely depends on the application, but it is no restriction for the convergence of the algorithm and its general applicability.

4.3.1 Degeneracy Case

The algorithm does not generate a valid Voronoi tessellation in cases of cyclic dependency. An example for such cyclic dependency is given in Figure 4.4. Here an assignment of 16 points to the sites 1 to 4 is formed based on a regular grid. The assignment of the points in the center would have been rotated by 90 degrees to generate a valid Voronoi tessellation. Unfortunately, the discrete algorithm is not able to perform the according swaps and resolve this state. However, such cases are extremely unlikely in practice and can be resolved by a simple extension of the algorithm. Therefore, if the algorithm stops in a stable state, an additional iteration has to be performed in which a swap between the assignment of two points is performed if the energy difference is *greater or equal* to zero. This means that the last part of the comparison in line 12 of Algorithm 4.1 has to be changed to $max(H_i) + max(H_j) \geq 0$. If a swap occurred in this additional iteration then the normal optimization has to be continued and its result has to be checked again. If no swap occurred then the algorithm converged in a valid Voronoi tessellation.

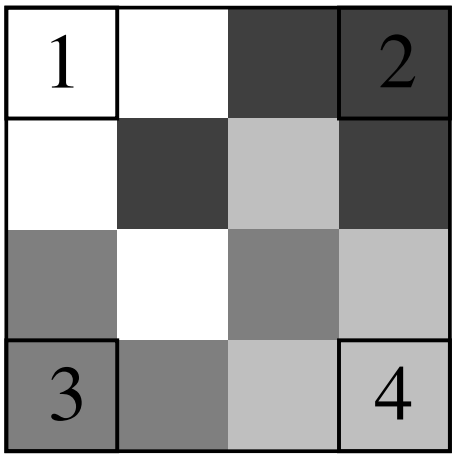


Figure 4.4: Degeneracy case of a cyclic dependency that is not a valid Voronoi tessellation and cannot be resolved with Algorithm 4.1. However, such cyclic dependencies are extremely unlikely in practice, and resolved by a simple extension of the algorithm.

4.4 Extensions

In the previous sections, we described our approach for computing capacity-constrained PW Voronoi tessellations. Its foundation is the least-squares minimization, which is directly related to the distance function of the PW Voronoi tessellations. This section presents extensions of our method which enable the computation of other capacity-constrained Voronoi tessellations beyond the PW distance function. Figure 4.5 presents examples for the extensions in this section, which can also be combined with each other. Each of these examples is based on the same initial set of ten sites with equal capacities, computed on a regular grid of 1000×1000 points in less than one minute on Intel Core 2 hardware.

4.4.1 Other Energy Functions

Our algorithm for discrete spaces generates a PW Voronoi tessellation by minimizing the assignment A based on an energy function that is directly related to the PW distance function. By replacing this energy function, we can generate other capacity-constrained weighted Voronoi tessellations. The range of Voronoi tessellations that can be generated with our approach is limited to those having an additive weight component in their distance function. Voronoi tessellations with a multiplicative weight component cannot be generated. For other energy functions with additive weights it is not guaranteed that there actually exists a Voronoi tessellation for any given sets of sites and capacity constraints. However, our algorithm will generate such a tessellations, if there exists one.

The necessary energy function can be derived directly from the distance function of the intended Voronoi tessellation by omitting the weight component. In fact, we can use

all kinds of energy functions as long as they fulfill the definition of a norm. For example, to achieve an AW Voronoi tessellations we use the following energy function, which is directly derived from the AW distance function:

$$e_{AW}(x,s) = \|x-s\|. \tag{4.4}$$

An example for an AW Voronoi tessellation with equal capacities computed with Algorithm 4.1 is shown in Figure 4.5(a). A similar example is given in Figure 4.5(c). This tessellation is based on the Manhattan distance, and uses the following energy function:

$$e_{Manhattan}(x,s) = \sum_{k=1}^{d} \|x_k - s_k\| \tag{4.5}$$

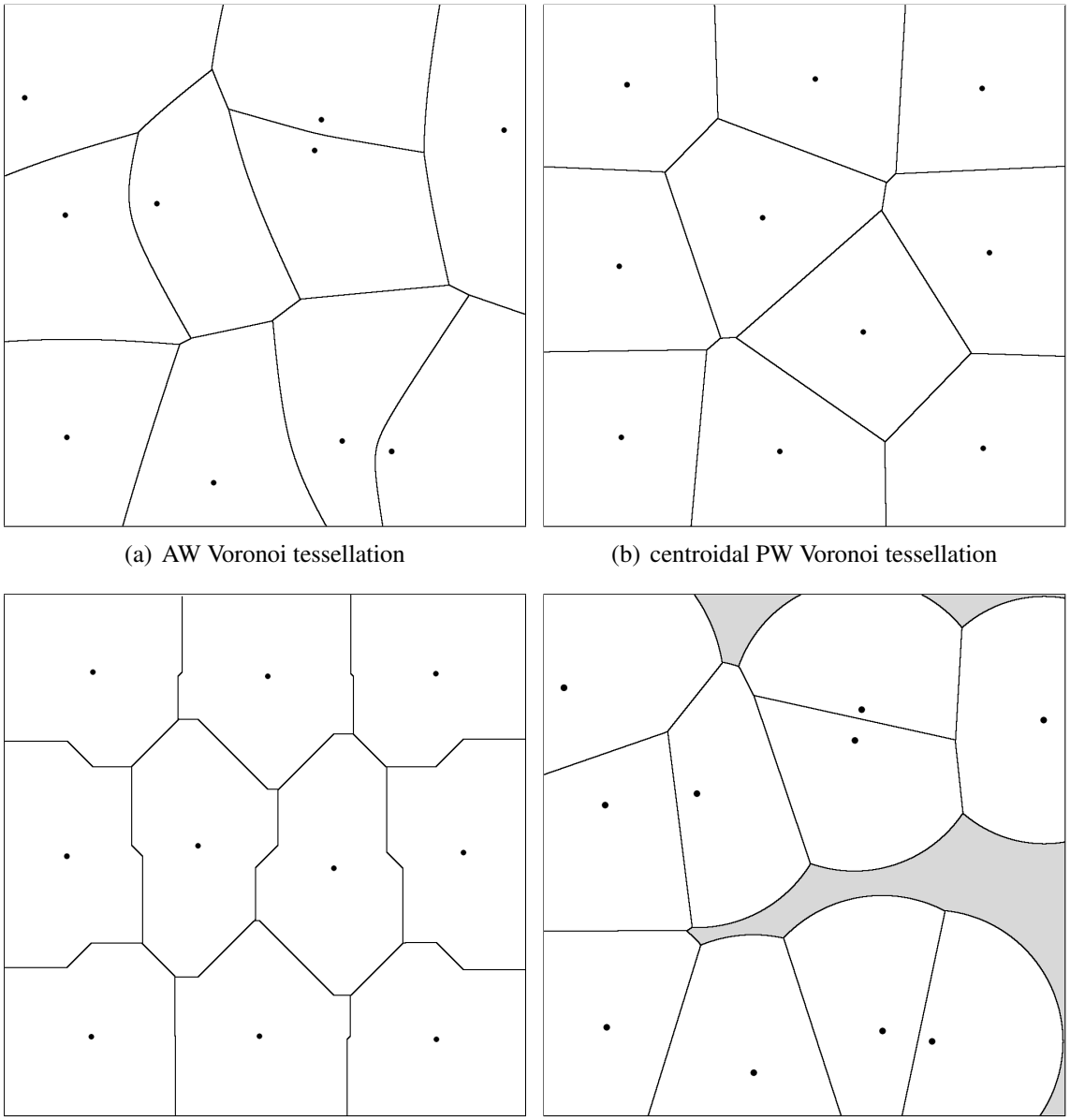
with k as a single dimension in a d-dimensional space. This example based on the Manhattan distance is simultaneously a centroidal Voronoi tessellation. Furthermore, all distance functions can be directly applied to toroidal spaces without any modification of the algorithm—examples for the application of toroidal distance functions are given in the context of our capacity-constrained point distributions in Chapter 6.

4.4.2 Centroidal Voronoi Tessellations

A centroidal capacity-constrained Voronoi tessellation can be generated by just applying our algorithm to the generation of the Voronoi tessellation in the spatial-temporal adjustment model in Section 2.3. However, we achieve a much better convergence if we directly integrate the site relocation into our algorithm. We insert the site relocation after line 16 of the algorithm, as the last statement inside the foreach loop. This means that immediately after the point assignment of the pair of sites has been changed, we relocate both sites. The convergence of this extension is guaranteed because relocating a site into the centroid of its Voronoi region always decreases the energy of the assignment. In fact, the energy function in Equation 4.1 which is minimized by our algorithm is equivalent to the energy function in Equation 2.13 which is minimized by a centroidal Voronoi tessellation. When combining centroidal Voronoi tessellations with energy functions that are not based on the PW distance function, this convergence guarantee may not apply. However, in our experiments with non-monotonic decreasing energy functions we did not identify a case where our algorithm did not converge. An example for a centroidal PW Voronoi tessellation with equal capacities is shown in Figure 4.5(b). Actually, this combination of our discrete space algorithm with centroidal sites is similar to the k-means algorithm with a capacity constraint. Here, the additional capacity constraint eliminates the problem of outliers that form their own clusters.

4.4.3 Void Region

The Voronoi tessellation of a set X of points for a set S of sites is defined as a partition into disjoint subsets, which means that every point in X is assigned to exactly one site



(a) AW Voronoi tessellation

(b) centroidal PW Voronoi tessellation

(c) centroidal Voronoi tessellation based on the Manhattan distance

(d) PW Voronoi tessellation with a void region (grey)

Figure 4.5: Examples for extensions of our algorithm. All tessellations are based on the same initial set of ten sites with equal capacities, and have been computed on a regular grid of 1000×1000 points.

in S. This implies that the sum of the capacities in C is equal to the number of points in X. If we want to create a Voronoi tessellation where the sum of the capacities is smaller than the number of points, we have to add a special site that gathers all points that are not assigned to a site in S. We call the points that are assigned to this special site the *void region*.

The void region is seamlessly integrated into our algorithm by inserting it as an additional site into our set of sites S. The site location of the void region is undetermined, and the energy of a point in X that is assigned to the void region is always zero,

$$e_{void}(x, s_{void}) = 0. \tag{4.6}$$

As result, the void region gathers all leftover points with high energy values. An example for a PW Voronoi tessellation with a void region is shown in Figure 4.5(d). Here, the total capacity equals 90% of the number of points, and again all sites have the same capacity. The void region is colored grey, and covers the remaining 10% of the discrete space.

4.5 Implementation of Density Functions

A major advantage of our algorithm for discrete spaces is its adaptability to d-dimensional spaces with arbitrary density functions. The core of our algorithm just swaps the assignment of points to sites, disregarding the dimension and the character of the density distribution within the discrete space. The only prerequisite is a formulation of the energy between a point and a site that represents a norm, such as in Equation 4.2.

The actual discrete space that is used for the computation is explicitly defined by the point set X. This set models the underlying space and thereby also represents a given density function. For a clearer illustration, we give three basic examples: To define a constant density in a given area, we could sample this area with a regular grid, and add for each sample on this grid a point to the set X. To define a non-constant density based on a continuous function in a given area, we could discretize this function via rejection sampling, and add for each accepted sample a point to the set X. To define a non-constant density based on a grayscale image, we could insert a number of random points within the area of each pixel of the image that corresponds to its grayscale value to the set X. More specific, for a pixel at coordinates $(3,5)$ with grayscale value 27, we add 27 random points within the interval $([2.5,3.5],[4.5,5.5])$ to X.

4.6 Conclusion

We presented an approach for the computation of capacity-constrained Voronoi tessellations in discrete spaces. In contrast to our previous approaches in continuous spaces, which iteratively adjust the site locations and/or weights in a Voronoi tessellation until the capacity constraint is fulfilled, we start with an arbitrary partition that already maintains the capacity constraint, and iteratively optimize this partition until it represents a weighted

Voronoi tessellation. The foundation for this optimization is the swap of assignments of points to sites based on an energy function that is directly related to the distance function of the Voronoi tessellation. By the restriction to assignment swaps, it is guaranteed that the capacity constraint is preserved in all steps of the optimization, including the result.

The results achieved by our approach for discrete spaces are equivalent to those of our approach for weighted distance functions in continuous spaces presented in Section 3.2. The difference is that now the result consists of the assignment of discrete points to sites, and not of the site weights as in the continuous case. Thus, we also do not have a geometric bisector representation of the resulting Voronoi tessellation. This may prohibit our approach for some applications that rely on this information. However, the straightforward structure of our algorithm makes it easy to implement, flexible, and has a runtime similar to the weighted case in continuous spaces. Furthermore, our discrete algorithm is independent of the underlying discrete space, which means that it works in arbitrary dimensions and for arbitrary distance function.

To support the implementation of our algorithm for discrete spaces, we provide C++ template code through an open source project at `http://ccvt.googlecode.com`.

4.7 Computation Sequences

The computation of capacity-constrained Voronoi tessellations in discrete spaces is further outlined in Figure 4.6 and Figure 4.7. These figures illustrate the development of the tessellations during the computation. The first example in Figure 4.6 was computed with our unmodified Algorithm 4.1 based on a set of 20 fixed sites with different capacities, and results in a capacity-constrained PW Voronoi tessellation. The second example in Figure 4.7 uses the same initial set of sites, but generates a centroidal capacity-constrained AW Voronoi tessellation.

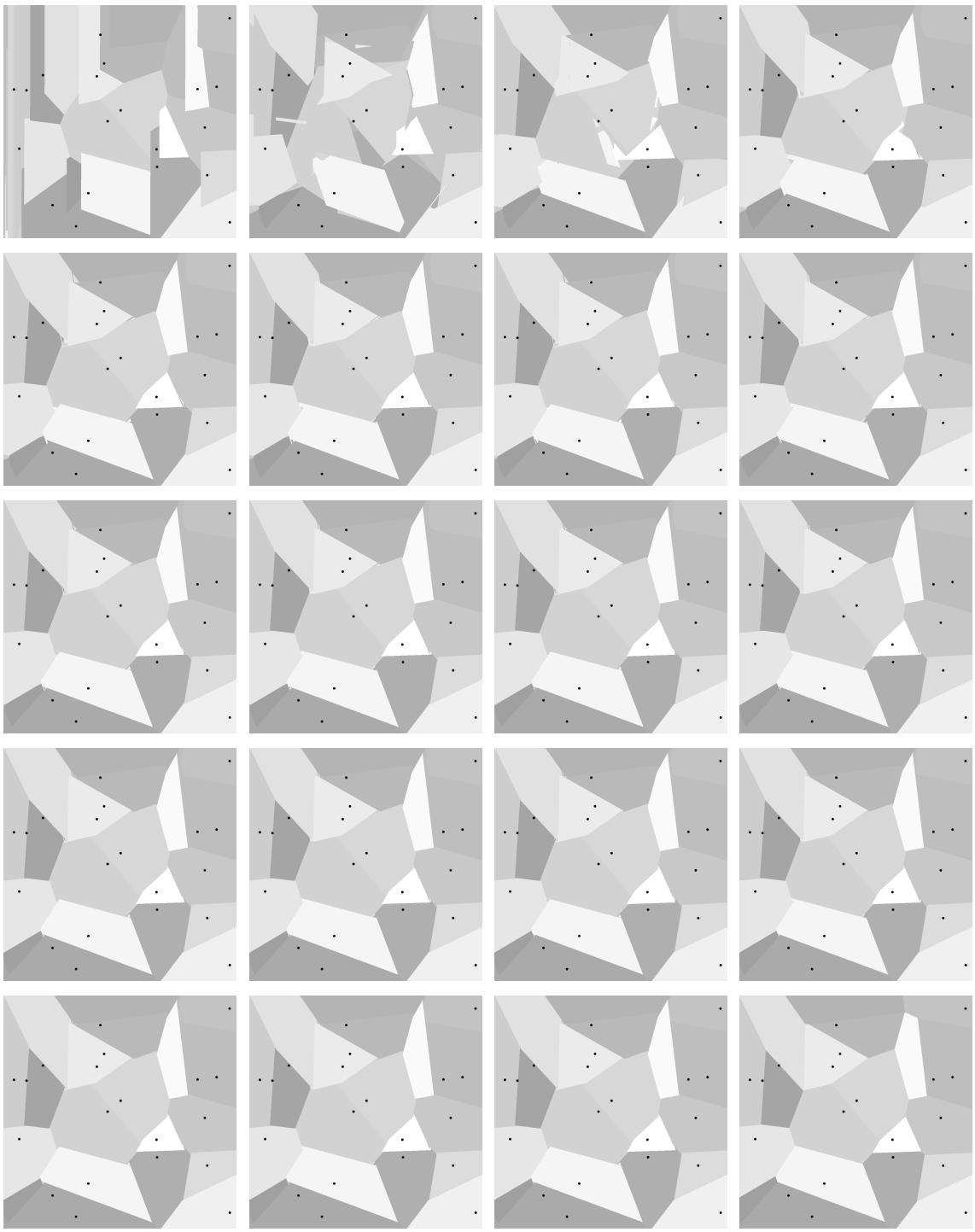


Figure 4.6: Development of the assignment during the computation of the capacity-constrained PW Voronoi tessellation of 20 sites with equal capacities in Figure 4.2(a): initial state, iterations 1–18, final result after iteration 638. The grayscale values denote the assignment of points to sites, where the same grayscale of two points represents an assignment of these points to the same site.

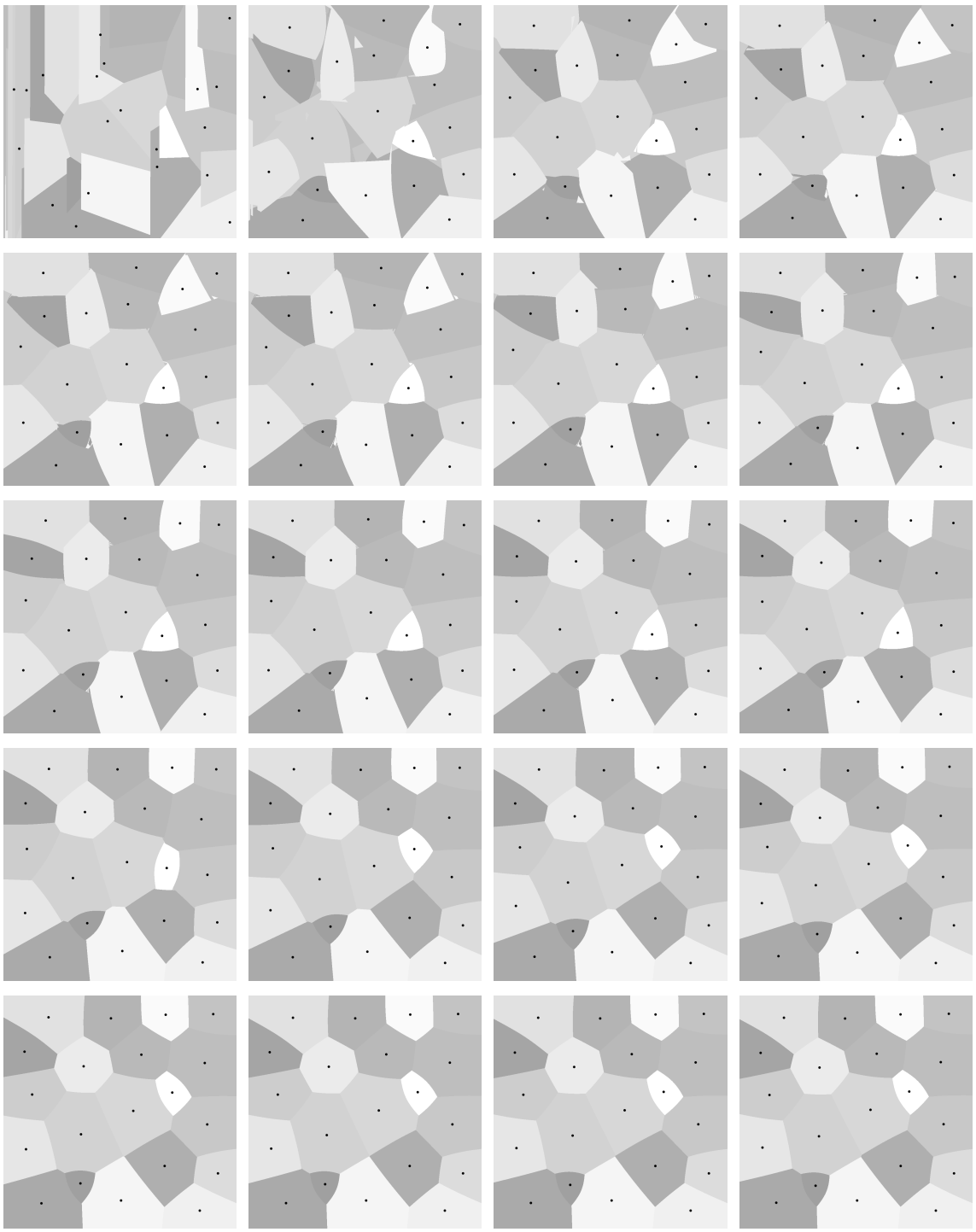


Figure 4.7: Development of the assignment during the computation of a centroidal capacity-constrained AW Voronoi tessellation of 20 sites with different capacities: initial state, iterations 1–5, 10, 15, 20, 30, 50, 100, 250, 500, 1000, 1500, 2000, 3000, final result after iteration 4154. The grayscale values denote the assignment of points to sites, where the same grayscale of two points represents an assignment of these points to the same site.

Chapter 5

Voronoi Treemaps

In this chapter, we present Voronoi treemaps for the visualization of attributed hierarchies. Voronoi treemaps are based on the general concept of treemaps as two-dimensional subdivisions of a given area without producing holes or overlappings. In contrast to the existent layout algorithms that are based on rectangular subdivisions, our Voronoi treemap layout algorithm is the first algorithm that generates polygonal subdivisions. The resulting layouts are advantageous with respect to the aspect ratios of the contained nodes and the clarity of the representation of the hierarchical structure. The basis for the polygonal layouts are centroidal capacity-constrained Voronoi tessellations. The Voronoi regions of these tessellations are used as the nodes in the treemap layout, whereas the resulting site distributions are of no interest. The enhancement of our Voronoi treemap layout with additional borders, coloring or shading, results in information visualizations with a unique visual appearance.

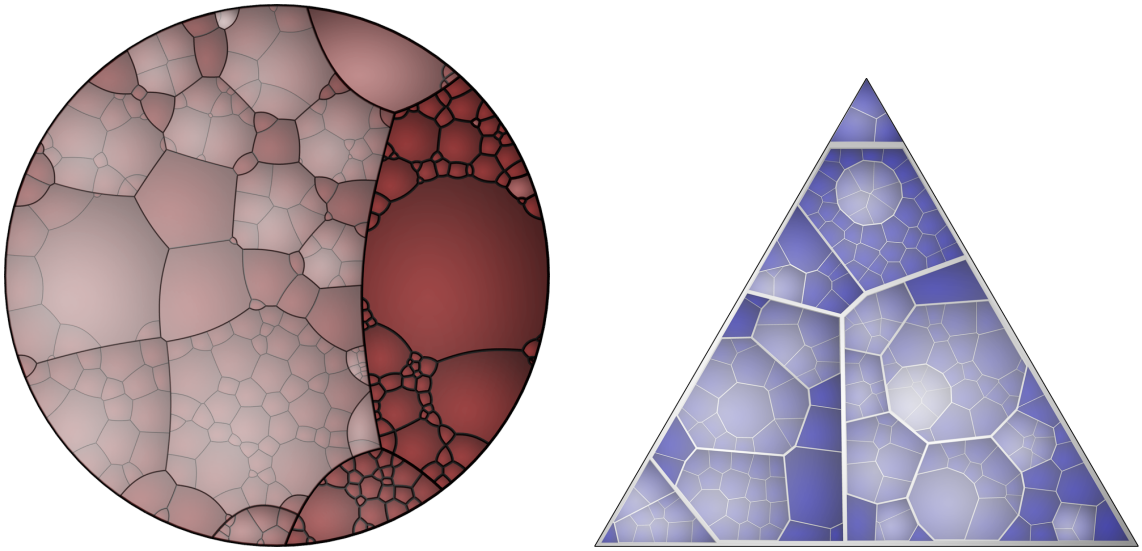


Figure 5.1: Voronoi treemap visualizations

5.1 Introduction

Treemaps are two-dimensional subdivisions of a given area according to an attributed hierarchy without producing holes or overlappings. Here, the term 'attributed' signifies that each node in the hierarchy has a size which represents a value that is related to a given measurement. In a treemap, this size is encoded as the area that is covered by the node. The hierarchical structure of the nodes is encoded implicitly as a result of the recursive construction of the treemap. Therefore, treemaps are especially useful for the visualization of hierarchical datasets in which the node size is important. As the size of the area of a node is proportional to this attribute, one can easily identify the largest nodes or the largest branches in the tree. Admittedly, the interpretation of the hierarchical structure of the treemap may lead to unclear or even ambiguous results for the majority of current treemap layout algorithms. Nevertheless, treemaps have been used successfully in real-world applications to visualize the stock market [79, 70], hard disk usage [32], as well as news and image collections [81, 15]. Due to the fact that treemaps partition the complete given area, they scale very well with the size of the datasets, and have been applied to trees with millions of nodes [33].

Treemaps were invented by Johnson and Shneiderman [40] in 1991 and applied to different kinds of hierarchial data. There exist various alternative layout algorithms, and a number of visual enhancements have been proposed. In contrast to the existent layout algorithms that are based on rectangular subdivisions, our Voronoi treemap layouts utilize polygonal subdivisions of the plane. To motivate our approach, we first explain the principle generation of treemap layouts, and then analyze existing layout algorithms. We discuss constraints and optimization criteria for treemaps, and reason the advantages of centroidal capacity-constrained Voronoi tessellations for treemap layouts.

5.2 Treemap Construction

The construction of treemaps is straightforward, and illustrated in Figure 5.2. Given is a hierarchy as a rooted tree. Each node in the hierarchy tree has a unique identifier and an associated size, where the size of an internal node is the sum of the sizes of its contained leaf nodes. The treemap is constructed within the given area via recursive subdivision. The initial area is associated with the root node A of the hierarchy tree. This area is then subdivided according to the sizes of the child nodes B, C, and D of root node A, where the area of the subareas is proportional to the size of the corresponding nodes. Then each of these subareas is again subdivided according to the sizes of the child nodes of the corresponding node. This recursive subdivision is performed until every leaf node of the hierarchy tree is associated with a subarea in the treemap layout. When employing the original layout algorithm [40], the direction of each one-dimensional subdivision step alternates per level: first horizontally, next vertically, again horizontally, and so on. As a result of its construction, the treemap reflects the structure of the hierarchy tree and the sizes of its nodes, which are directly related to the sizes of the subareas. The hierar-

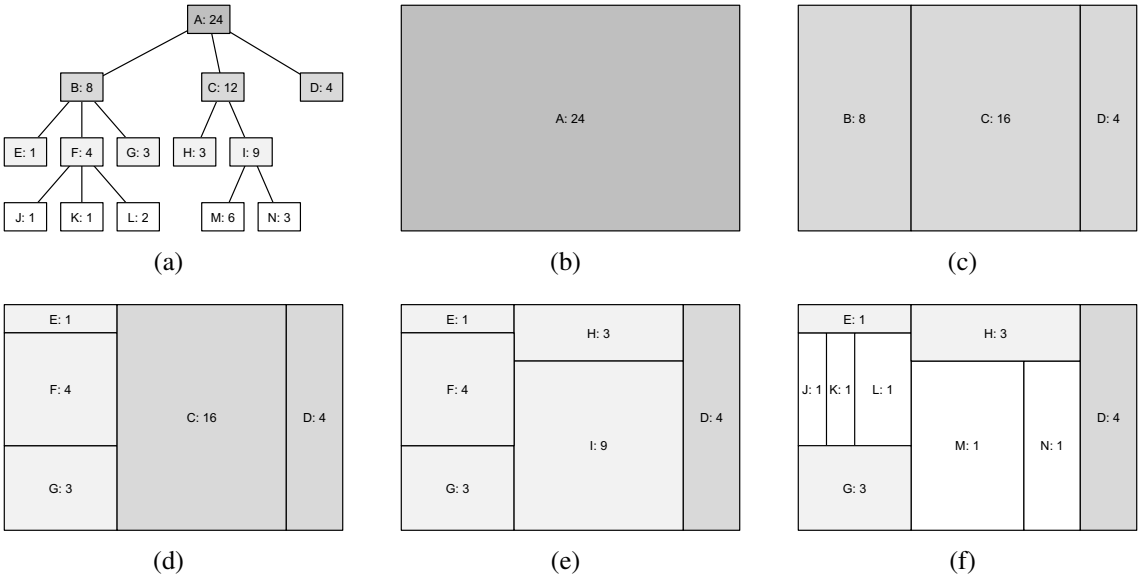


Figure 5.2: Basic treemap algorithm: By having an attributed hierarchy (a), the root node covers the given initial area (b), which is then subdivided horizontally in subareas for the children of the root node proportional to their size (c). These children are then again subdivided vertically in subareas for their children (d,e), and so on, until the final subdivision (f) is achieved.

chy of the nodes is extracted by evaluating the alternation of the horizontal and vertical subdivisions. This original treemap layout is called Slice-and-Dice.

5.3 Existing Treemap Layout Algorithms

A substantial problem of the initial layout algorithm is the restriction of subdividing the plane in each step solely in one dimension, either horizontally or vertically. As result, thin elongated rectangles with a high aspect ratio between width and height emerge. Such rectangles are difficult to see, select, compare in size, and label [34, 73, 19]. Figure 5.3 presents an example of a medium-sized real-world dataset containing 698 nodes at 5 hierarchy levels. The left image (a) visualizes this hierarchy with leaf nodes of different sizes, and the same hierarchy with leaf nodes of equal sizes is presented in the right image (b). Both images clearly illustrate the aforementioned problems of thin elongated rectangles in Slice-and-Dice layouts that causes the hard recognition of small nodes, and the bad perception of the individual node sizes. The reason for both problems is the high aspect ratio.

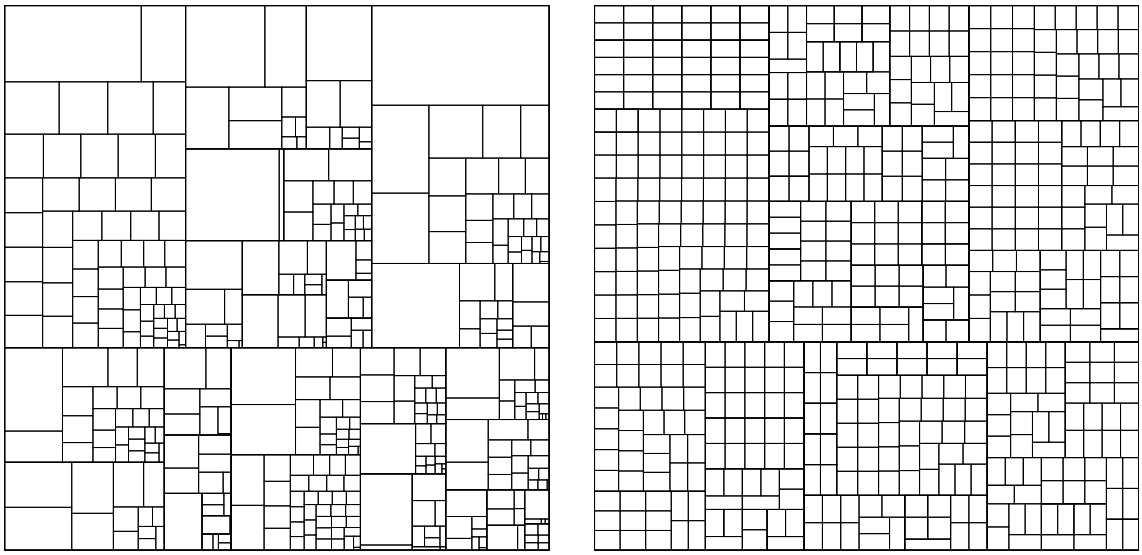
Consequentially, in the further developments of treemap layout algorithms mainly

(a) leaf nodes of different sizes (b) leaf nodes of equal sizes

Figure 5.3: Slice-and-Dice treemap layouts of 698 nodes at 5 hierarchy levels. The high aspect ratio between width and height causes the hard recognizability of small nodes, and the bad perception of node sizes.

the issue of high aspect ratios was addressed, for instance in Clustered treemaps [79], Squarified treemaps [19], Ordered treemaps [69], Quantum treemaps [16], Modifiable treemaps [76], Bubblemaps [15], and ET-Maps [65]. These algorithms employ a two-dimensional subdivision in each recursion step, meaning horizontally and vertically at the same time. Their main optimization criterion is the approximation of the subareas to the shape of a square, so that the average aspect ratio of the rectangles converges to one. Aside from the aspect ratio criterion, other criteria have also been considered, mostly related to the respective application domain. For example, the order of nodes or the distance between two nodes. A representative and popular member of this group of advanced treemap layout algorithms is presented in Figure 5.4, visualizing the same dataset as in Figure 5.3 with a Squarified treemap layout. Similarly, nodes of different sizes are shown in the left image (a), and nodes of equal sizes in the right image (b). This layout algorithm maintains a much better aspect ratio, resulting in a better separation of the nodes and perception of their sizes.

Currently, their exists one treemap layout algorithm that is not based on axis-aligned rectangles, but rather generates polygonal subdivisions. This algorithm by Onak and Sidiropoulos [59] uses sequential binary subdivisions based on so-called ‘circular partitions’. An example layout of the original paper is given in Figure 5.5. This algorithm is related to our previous approach in [7], where we presented subdivisions of convex polygons for treemap layout algorithm within non-rectangular boundaries. However, the



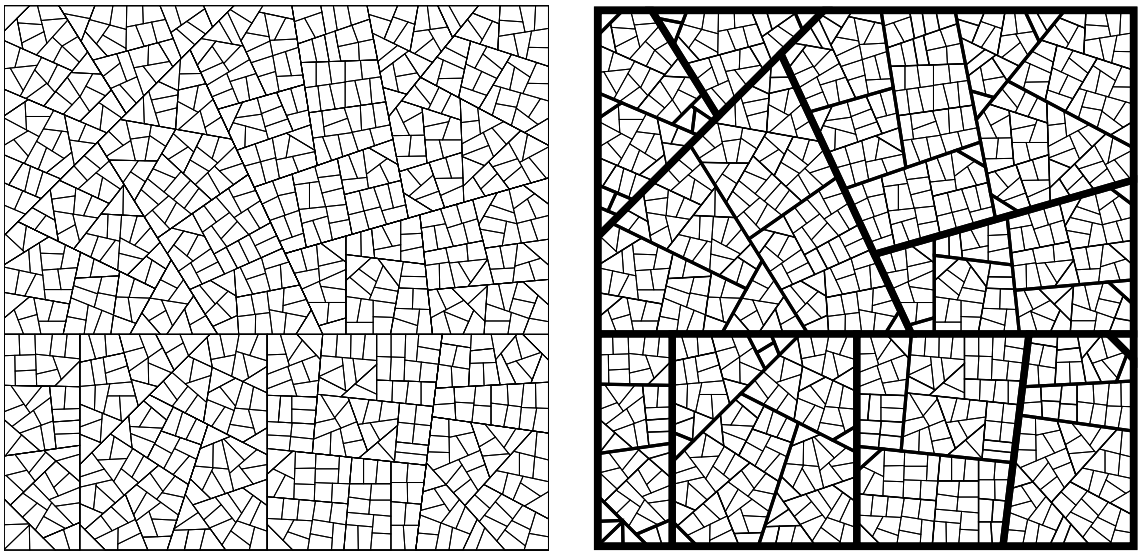
(a) leaf nodes of different sizes (b) leaf nodes of equal sizes

Figure 5.4: Squarified treemap layouts of the same dataset as in Figure 5.3 containing 698 nodes at 5 hierarchy levels. The interpretation of the hierarchical structure is ambiguous, and often edges seemingly run into each other.

approach of Onak and Sidiropoulos additionally identifies the best axis for a binary subdivision, and offers guarantees for the aspect ratios of the treemap nodes. Unfortunately, the utilized binary subdivisions dramatically distort the hierarchical structure of the dataset, which results in a very ambiguous hierarchy representation.

A major disadvantage of the discussed sophisticated treemap layout algorithms is the ambiguous representation of the hierarchical structure. For example, the Slice-and-Dice treemap layout in Figure 5.3 unambiguously represents a hierarchy with a root node containing six child nodes, which is conform to the dataset. The indication therefor is the alternating horizontal and vertical subdivision. In contrast, this observation cannot clearly be made in Figure 5.4. Here the number of child nodes of the root node is ambiguous, as illustrated in Figure 5.6. Aside from the correct interpretation of the dataset as shown in image (a), other interpretation variants are valid as well, while still maintaining all characteristics of the generating algorithm. This also implies that datasets with different hierarchical structures may result in identical treemap layouts, again inducing errors when interpreting and comparing hierarchical datasets. The same applies to the layout algorithm by Onak and Sidiropoulos [59].

The reason for not producing a one-to-one mapping between the hierarchical structure and the corresponding treemap layout is the grouping of nodes during the subdivision step of the algorithms. This grouping introduces pseudo-branchings in the hierarchical structure. For example, in the Squarified treemap layout shown in Figure 5.4, the two largest



(a) equally thick boundaries for all hierarchy levels

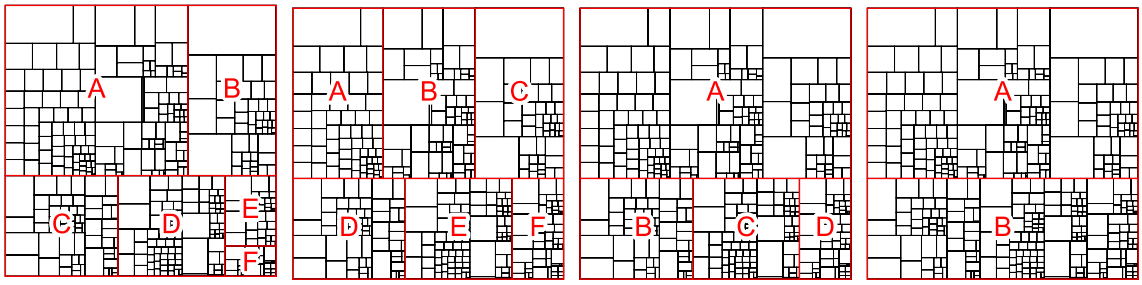
(b) thicker boundaries correspond to higher hierarchy levels

Figure 5.5: Polygonal treemap layout based on circular partitions [59]. Both images show one identical layout. The interpretation of the hierarchical structure in (a) is almost impossible without the additional visual enhancements in (b).

nodes are vertically divided from the other four smaller nodes due to the functioning of the algorithm. This 'internal' subdivision step may be interpreted as branching in the hierarchical structure, and thereby provokes misinterpretations.

Another problem of the existent algorithms are edges that seemingly run into each other, making a determination of the location of nodes within the different hierarchy levels even more difficult. The reason is again the limited degree of freedom in each subdivision step. By aligning each edge only horizontally or vertically, the probability that the endpoint of an edge is near the endpoint of another equally aligned edge is rather high, which results in the impression of one single edge instead of two separate edges. Depending on the hierarchical position of the nodes that are separated by these two edges, the interpretation of the hierarchical structure may be further deluded by the treemap layout.

The representation of the hierarchical structure is a crucial requirement for treemaps. Therefore, already Johnson and Shneiderman enhanced their original treemap layouts by assigning borders to the subareas in every subdivision step, which they called Nested treemaps [40]. Another approach as an enhancement to treemap layouts are Cushion treemaps [75], which employ shading to improve the perception of structure by simulating specular reflection and thereby imitating a curved surface. This method is also adapted for the enhancement of Nested treemaps to Framed treemaps [19], resulting in quasi-three-dimensional borders. Each of these layout-independent methods can be applied to existing



(a) correct interpretation (b) variant with 6 nodes (c) variant with 4 nodes (d) variant with 2 nodes

Figure 5.6: Correct and incorrect interpretations of the hierarchical structure of the Squarified treemap layout in Figure 5.4(b).

treemap layouts and may reduce, but do not prevent, misinterpretations concerning the hierarchical structure. Thus, they are no compensation for layout-induced ambiguities, but rather additions to support the perception of general treemap layouts.

5.4 Constraints and Optimization Criteria

By evaluating the definition and the purpose of treemaps, as well as the existent layout algorithms and their results, we identify a set of global application-independent constraints and optimization criteria for treemap layouts:

- **Constraint C1:** The subdivision must fully utilize the given overall area, avoiding holes and overlappings.
- **Constraint C2:** The nodes in the treemap must have predefined areas.
- **Constraint C3:** Siblings in the hierarchy must be treated separately during the subdivision to prevent the introduction of pseudo-branchings in the hierarchy by grouping, and thereby avoiding ambiguous representations of the hierarchical structure.
- **Optimization Criterion O1:** The subareas should have an overall aspect ratio between width and height that converges to one on average, to allow a good recognition of the individual nodes and perception of their sizes.
- **Optimization Criterion O2:** The probability for edges that seemingly run into each other should be minimized to avoid ambiguous representation of the hierarchical structure.

In contrast to the constraints C1 and C2, the constraint C3 is not directly given by the definition and purpose of treemaps. Actually, constraint C3 is violated by the majority of the existent treemap layout algorithms. But this nonobservance directly introduces errors

to the representations of the datasets. This may be acceptable for flat datasets, which contain no hierarchical structure, but for all other datasets, their exploration and analysis is falsified and deluded by such representations. Thus, the separate treatment of siblings in the hierarchy during the subdivision step is formulated as a constraint, and not just as an optimization criterion.

5.5 Centroidal Capacity-Constrained Voronoi Treemap Layouts

So far, all existent treemap layout algorithms have one thing in common: they are restricted to axis-aligned rectangles—an exception to this rule are the recent circular partitions by Onak and Sidiropoulos [59], which however suffer from similar effects as rectangular layouts. This limited degree of freedom drastically restricts the space of layout variability. The issues of high aspect ratios and misinterpretations concerning the hierarchical structure are consequential symptoms. Additionally, the restriction to rectangles implies that the layout of treemaps can only take place within rectangular display areas. More complex shapes like circles, triangles, and arbitrary polygons are not possible. Even though these shapes may not be necessary if treemap visualizations are used independently, by embedding treemap layouts within more complex visualizations, a better adaptability is useful or even necessary. An example for such a visualization is presented in [7], where we used treemaps as representations for collapsed graph nodes. To overcome the problems that are caused by the restriction to axis-aligned rectangles, we present an approach that generates treemaps with polygonal subareas. The generic shape of polygons dramatically enhances the degree of freedom for the subdivisions, and remedies the aforementioned problems of rectangular treemap layout algorithms.

Our approach for generating treemap layouts is based on centroidal capacity-constrained Voronoi tessellations. The suitability of such tessellations for the generation of treemap layouts is clarified by examining the constraints and optimizations criteria given in the previous section:

- Voronoi tessellations enable the subdivision of a given area without producing holes and overlappings, which satisfies constraint C1.

- Capacity-constrained Voronoi tessellations enable the partition into regions with predefined areas, which satisfies constraint C2.

- In a Voronoi tessellation, each generator is treated separately from the other generators, thereby avoiding any grouping during the computation, which refers to constraint C3.

- Centroidal Voronoi tessellations minimize the overall energy of the Voronoi tessellation. This energy minimization is equivalent to the minimization of the overall

aspect ratio of the subareas of the treemap layout, which refers to optimization criterion O1.

- The energy minimization in centroidal Voronoi tessellations simultaneously minimizes the number of degenerate vertices, and maximizes the distances between vertices and the minimum angles between the edges of a vertex [28, 31, 58]. In combination with the general non-regular topology of centroidal Voronoi tessellations, this refers to optimization criterion O2.

Apparently, centroidal capacity-constrained Voronoi tessellations are suitable for generating treemap layouts in reference to the aforementioned constraints and optimization criteria. Such treemaps based on centroidal capacity-constrained Voronoi tessellations are called *Voronoi treemaps*.

5.6 Voronoi Treemap Algorithm

The principal structure of our algorithm for generating Voronoi treemaps is similar to the original treemap layout algorithm as outlined in Section 5.2. Our modification is applied to the subdivision in each recursion step. This subdivision is based on centroidal capacity-constrained Voronoi tessellations. The property of centroidal sites is necessary to minimize the aspect ratio of the subareas and the number of edges that seemingly run into each other. To compute such subdivisions we use our continuous space algorithm based on weighted distance functions presented in Section 3.2. The continuous space algorithm based on ordinary distance functions is insufficient because it does not allow to implement centroidal sites. Furthermore, we use the weighted algorithm in continuous space because its result is a geometric bisector representation of the resulting subdivision. This polygonal representation is necessary to generate treemaps from large datasets that consist of nodes at many hierarchy levels. In contrast, a discrete representation would not allow a precise partition over a large number of hierarchy levels. Furthermore, the polygonal representations are advantageous for the visualization of the treemap layouts because of their flexibility for post processing steps such as scaling, coloring, or texturing, and their efficient render capabilities on modern graphics hardware.

The input for our Voronoi treemap layout algorithm is an attributed hierarchy of nodes given as a rooted tree. Each node s in this hierarchy has an associated size $c \geq 0$, and a set of n child nodes $S = \{s_1, \ldots, s_n\}$ of the same type as s. During the computation of the layout, a region $V \subset \mathbb{R}^2$ is assigned to each node s. This region represents the node in the treemap layout, and defines the bounded space for the layout of all nodes below this node according to the hierarchy. The region V may be defined as a polygon, a set of connected curve segments, or any other form that defines a subset of $\mathbb{R}^2$. This yields to the following definition of a node for Voronoi treemap layout computations:

$$\text{Node } s = (\text{Value } c \geq 0, \text{Child nodes } S = \{s_1, \ldots, s_n\}, \text{Region } V \subset \mathbb{R}^2). \tag{5.1}$$

The generation of a Voronoi treemap layout for an attributed hierarchy of nodes is implemented as a recursion. By having a root node s_0 of the type defined in Equation 5.1 representing an attributed hierarchy, and an initial bounded space defined by the region V_0, the Voronoi treemap layout of s_0 within V_0 is generated by first assigning V_0 to s_0. Then, Algorithm 5.1 is executed for the root node s_0. This algorithm generates the Voronoi treemap layout of a node s by generating a centroidal capacity-constrained Voronoi tessellation within its region V according to its child nodes in S, then assigning the resulting regions to these child nodes, and finally executing this algorithm for each of the child nodes recursively.

Algorithm 5.1: Voronoi Treemap Layout

Input: Node $s = (\text{Value } c \geq 0, \text{Child nodes } S = \{s_1, \ldots, s_n\}, \text{Region } V \subset \mathbb{R}^2)$

Output: Voronoi treemap layout of Node s within region V

```
1  if C ≠ ∅ then
2      Initialize an empty set of values C := ∅;
3      foreach s_i ∈ S do
4          if c_i > 0 then
5              Insert c_i into C;
6          end
7      end
8      if S ≠ ∅ then
9          Compute a centroidal capacity-constrained weighted Voronoi tessellations
           𝒱(S,W,C);
10         foreach s_i ∈ S do
11             if c_i > 0 then
12                 V_i := V(s_i,w_i) with V(s_i,w_i) ∈ 𝒱(S,W,C);
13                 Compute the Voronoi treemap layout of s_i;
14             end
15         end
16     end
17 end
```

The first part of Algorithm 5.1 extracts all capacities $c_i > 0$ of the child nodes S of the considered node s. Capacity values $c_i \leq 0$ are omitted due to the fact that they would result in empty regions, which do not appear in the final treemap layout, but would slow down the computation and entail degenerate cases. The capacity constraint C is then used in line 9 of the algorithm to generate a centroidal capacity-constrained weighted Voronoi tessellation for the subdivision of region V of the considered node s according to the capacities of its child nodes in S. After completion of the subdivision, the resulting regions are assigned to the corresponding child nodes, and finally, the algorithm is executed re-

cursively for each of the child nodes. This procedure is similar to the other treemap layout algorithms.

The resulting treemap layouts depend on the distance function that is used to compute the capacity-constrained Voronoi tessellation in line 9 of Algorithm 5.1. If the AW distance function is used, then the final layout consists of hyperbola segments. Such result is called *AW Voronoi treemap*. If the PW distance function is used, then the final layout consists of straight line segments, and called *PW Voronoi treemap*. Figure 5.7 presents an example of a AW Voronoi treemap, and Figure 5.8 presents an example of a PW Voronoi treemap. All four images use the dataset with 698 nodes at 5 hierarchy levels, as in the previous examples of Slice-and-Dice and Squarified treemap layouts in Section 5.3. Again, the left images visualize the hierarchy with nodes of different sizes, and the right images with nodes of equal sizes. The computation time for the AW example was less than 3 minutes, and for the PW example it was less than 1 minute, on Intel Core 2 hardware.

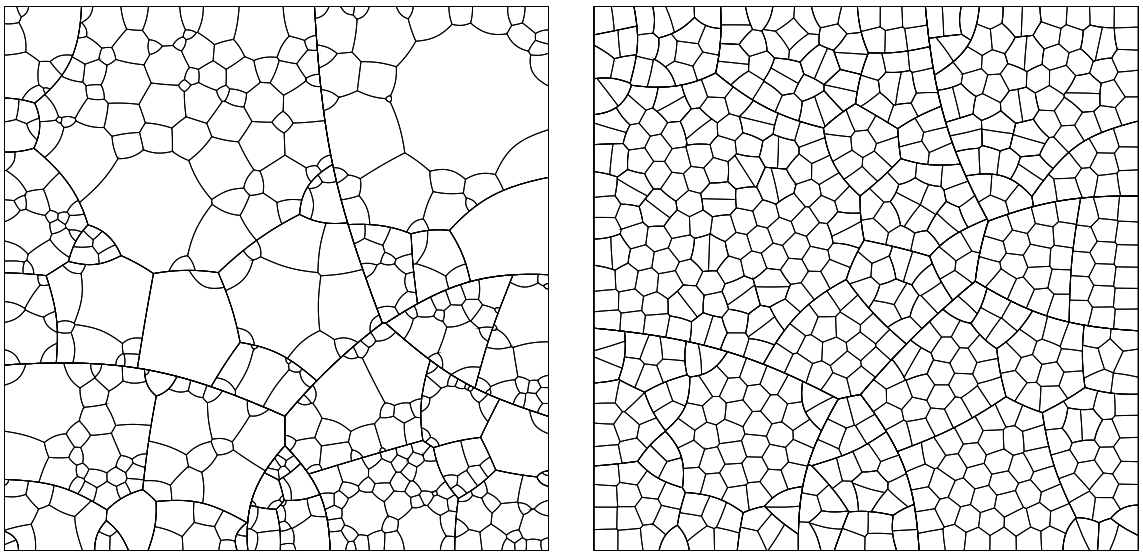
By comparing these Voronoi treemap layouts with the Slice-and-Dice and the Squarified layouts in Section 5.3, we observe that the aspect ratios of the individual treemap nodes is very low, and that the number of edges that seemingly run into each other is highly reduced. Especially the curved hyperbola segments in the AW Voronoi treemap layout are helpful for identifying the different hierarchy levels. Furthermore, no treemap nodes are grouped throughout the layout process, which allows an unambiguous interpretation of the hierarchical structure.

5.7 Resulting Visualizations

Similarly to other treemap layout algorithms, enhancements like borders, adaptive edge sizes, cushions, coloring, et cetera, may also be applied to Voronoi treemap layouts. These enhancements additionally support the user in the perception and interpretation of the treemap visualization. Examples for such enhanced Voronoi treemap layouts are presented in the following figures.

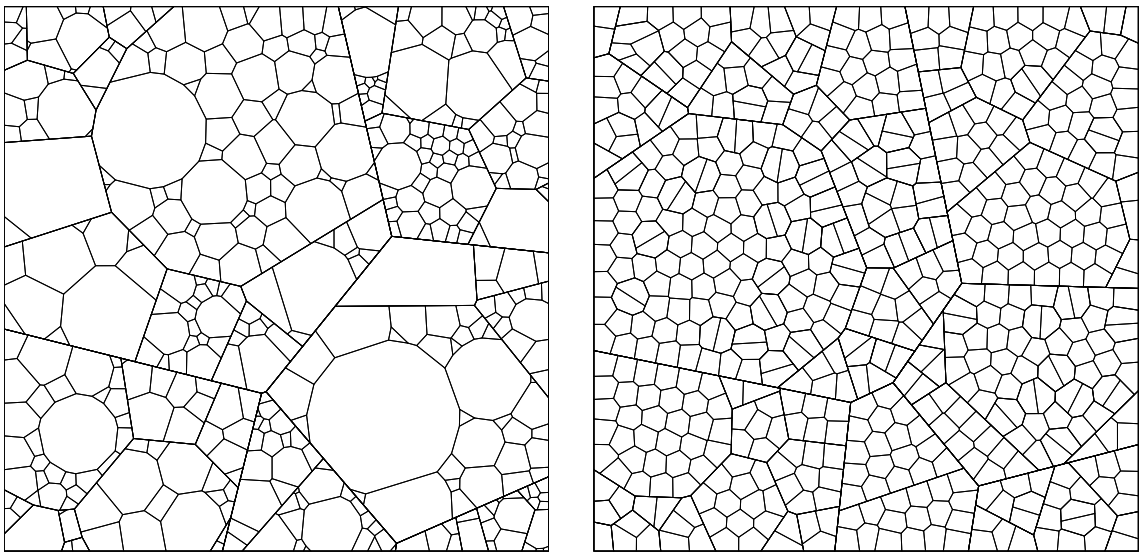
The visualizations in Figure 5.9 are based on datasets generated by an analysis of the Java Development Kit, JDK 1.4.2. Image (a) shows the files and/or directories in the JDK with their filesize denoted in the treemap. This dataset contains 4075 nodes at 10 hierarchy levels. The generation of the treemap layouts was performed with a parallel implementation that generates layouts of different parts of the hierarchy on different processors. The computation time was less than 8 minutes using 8 Intel Xeon CPUs with 2.4 Gigahertz each. Image (b) shows the hierarchy of classes in the JDK with each treemap node having the same size. This dataset contains 16288 treemap nodes at 7 hierarchy levels. The generation of the treemap layout was performed with a parallel implementation in less than 6 minutes using 8 Intel Xeon CPUs with 2.4 Gigahertz each.

Figure 5.10 presents a visualization that combines four different layouts in one tree-



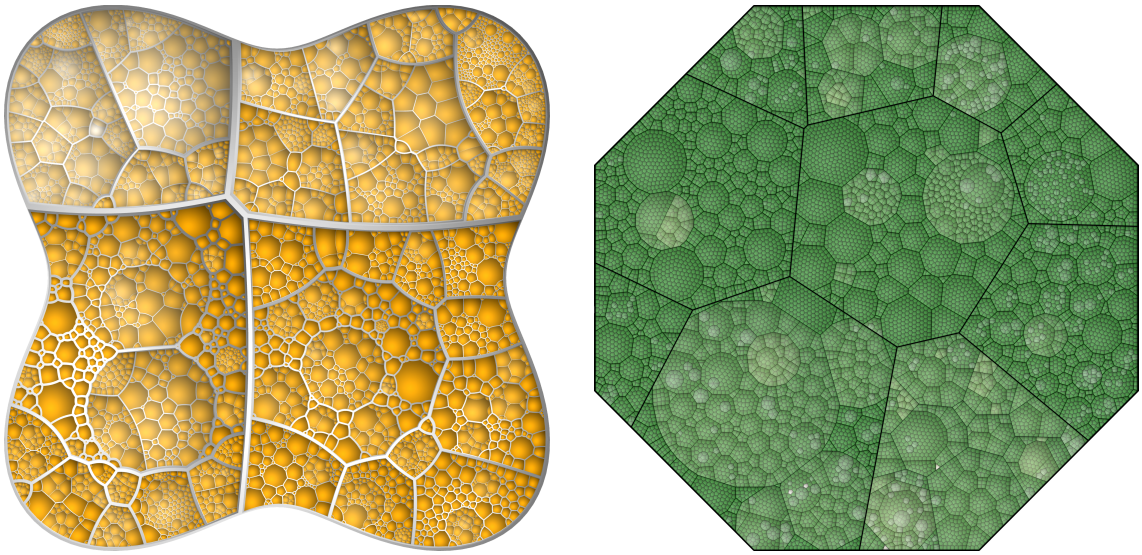
(a) leaf nodes with different sizes (b) leaf nodes with equal sizes

Figure 5.7: AW Voronoi treemap layouts of the same dataset as in Figure 5.3 containing 698 nodes at 5 hierarchy levels. The bisectors between the Voronoi regions are formed by hyperbolic curve segments.



(a) leaf nodes with different sizes (b) leaf nodes with equal sizes

Figure 5.8: PW Voronoi treemap layouts of the same dataset as in Figure 5.3 containing 698 nodes at 5 hierarchy levels. The bisectors between the Voronoi regions are formed by straight line segments.



(a) AW Voronoi treemap layout of 4075 nodes at 10 hierarchy levels

(b) PW Voronoi treemap layout of 16288 nodes at 7 hierarchy levels

Figure 5.9: Enhanced Voronoi treemap visualizations based on datasets generated by the analysis of the Java Development Kit, JDK 1.4.2.

map. The layout of the root node was generated with a Squarified treemap layout. Each subtrees of the four children of the root node used a different treemap layout algorithm: AW Voronoi treemap layout, PW Voronoi treemap layout, Squarified treemap layout, and Slice-and-Dice treemap layout.

Figure 5.11 presents a visualization of the inflation in the United States between March 2007 and March 2008. The nodes in the treemap represent spending categories, and their size is weighted according the spending of the average consumer. The color of the regions illustrate the inflation in this category. This visualization was published in The New York Times [23]. We generated the layout upon request based on data provided by the U.S. Bureau of Labor Statistics. The coloring, labeling, and text was done by Amanda Cox.

Beside these examples, Voronoi treemaps were also utilized for interactive visualizations, such as a visualization of museum exhibits [62], or a multi-user environment for exploring phylogenetic trees [67]. In these interactive applications, the users are able to create views of, or navigate within, large hierarchical datasets that are visualized by a Voronoi treemap.

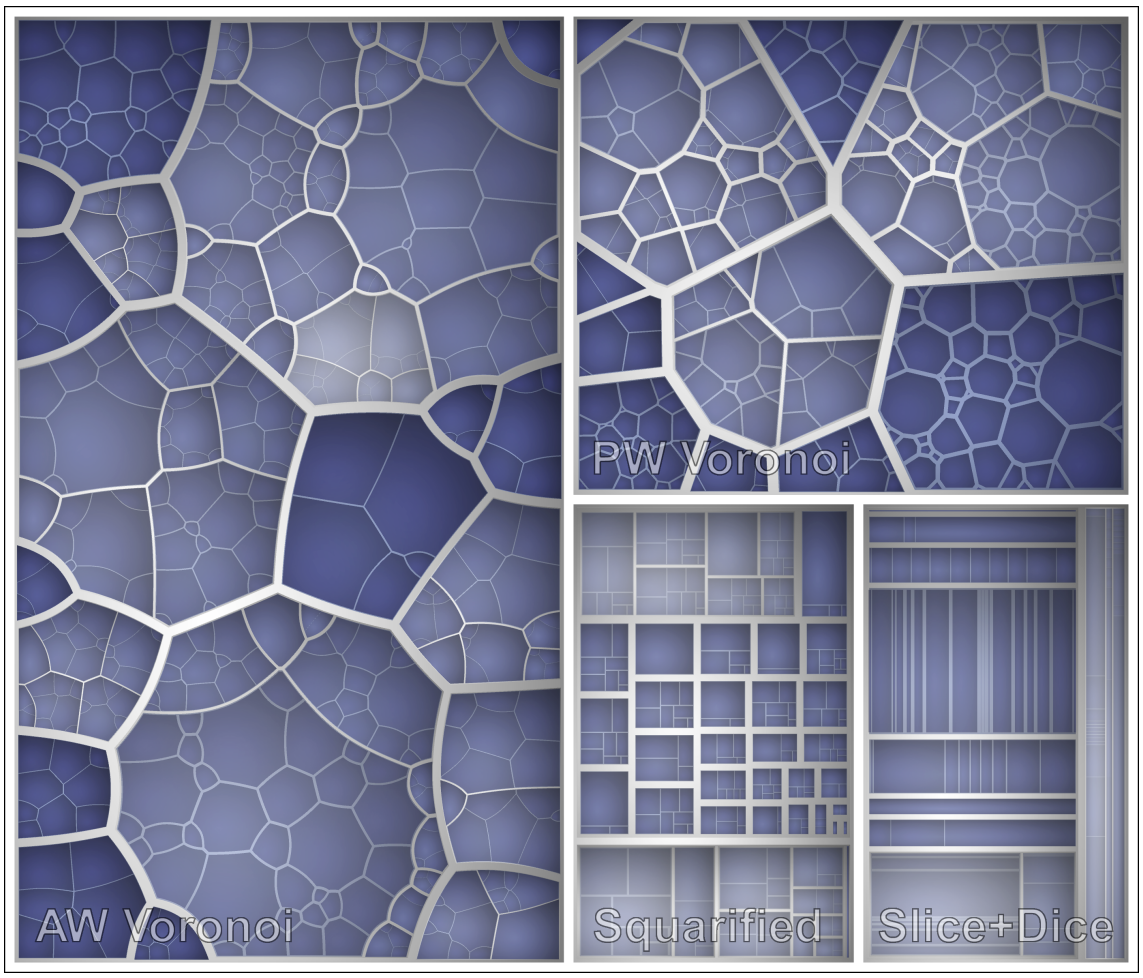


Figure 5.10: Combination of four different layout algorithms in one treemap.

5.8 Conclusion

We presented a novel method for generating treemap layouts based on centroidal capacity-constrained Voronoi tessellations. Our Voronoi treemaps improve on the existent rectangular treemap layouts by enabling polygonal subdivisions. The resulting layouts consist of treemap nodes with low aspect ratios, and they clearly represent the hierarchical structure of the underlying dataset. The algorithm also allows to create treemap visualizations within areas of arbitrary shape, such as circles, triangles, or even non-convex polygons. The suitability of the introduced layout method has been shown by the constraints and general application-independent optimization criteria for treemaps.

Of course, the computational effort for generating Voronoi treemap layouts is much higher than for generating rectangular layouts, which disqualifies them for some real-time applications. However, this problem is diminished by using distributed computing envi-

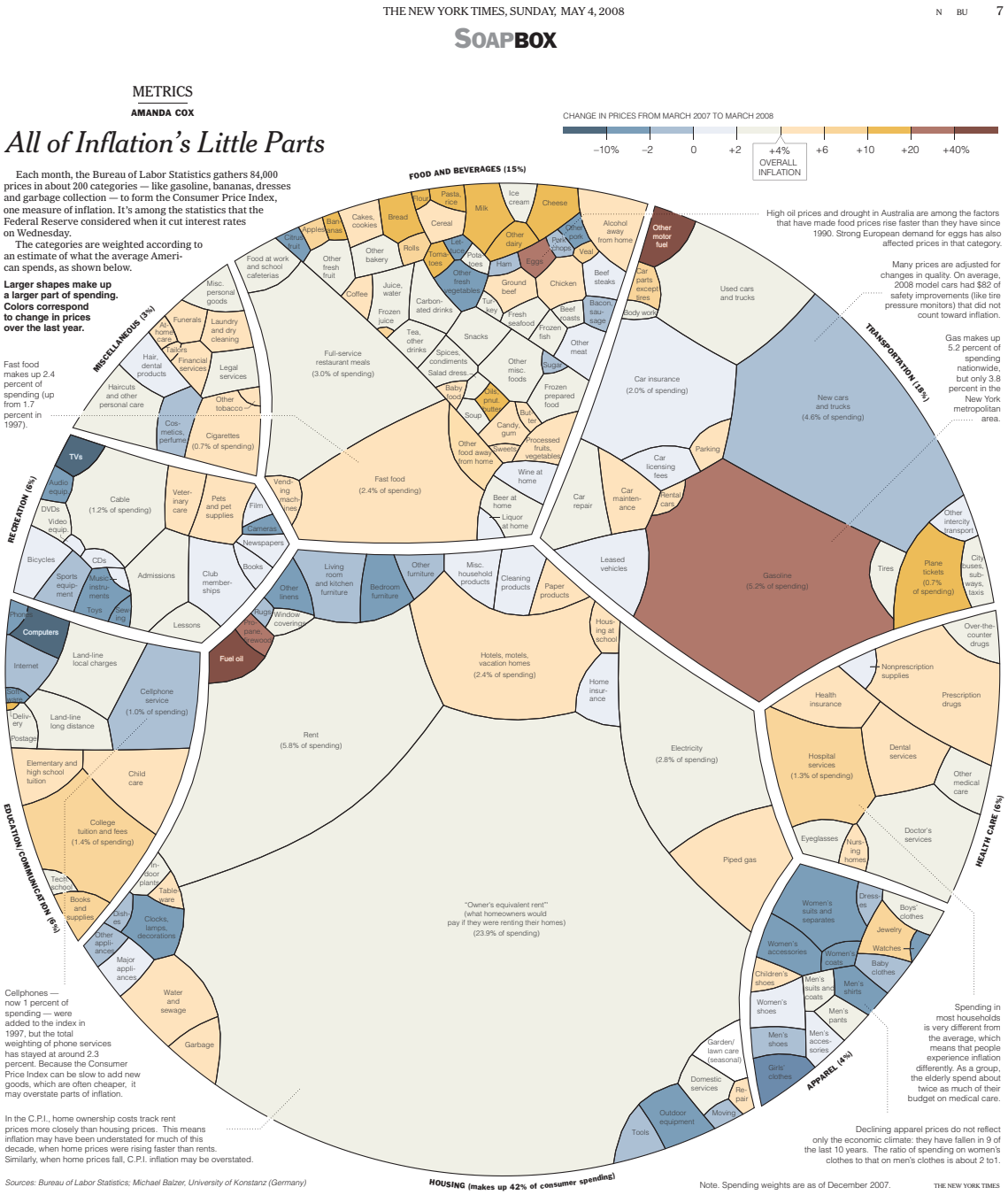


Figure 5.11: Voronoi treemap visualization published in The New York Times [23]. This visualization earned a gold medal at the 17th Malofiej Infographics Awards, 2009. It was also part of the portfolio of The New York Times entry that won the 'Communication Design Award' at the 10th U.S. National Design Awards in 2009. © 2008, The New York Times, reprinted with permission.

ronments where where the different subtrees of the hierarchy are computed on different processors. The recursive structure of the general treemap algorithm enables an almost perfect linear scalability with the number of processors used.

Chapter 6

Capacity-Constrained Point Distributions

In this chapters, we present a general-purpose method for optimizing existing point sets in the field of computer graphics. The resulting distributions possess high-quality blue noise characteristics and precise adaptation to given density functions. Our method is similar to the commonly used Lloyd's method whereas avoiding its drawbacks. We achieve our results by utilizing the concept of capacity, which is determined for each point by the area of its Voronoi region weighted with an underlying density function. We demand that each point has the same capacity. In combination with our discrete optimization algorithm in Chapter 4, this capacity constraint enforces that each point obtains equal importance in the distribution. Our method can be used as an easy drop-in replacement for Lloyd's method, and combines enhancement of blue noise characteristics and density function adaptation in one operation. In this application, we are not interested in the actual Voronoi regions, but rather in the locations of the sites and their distribution.

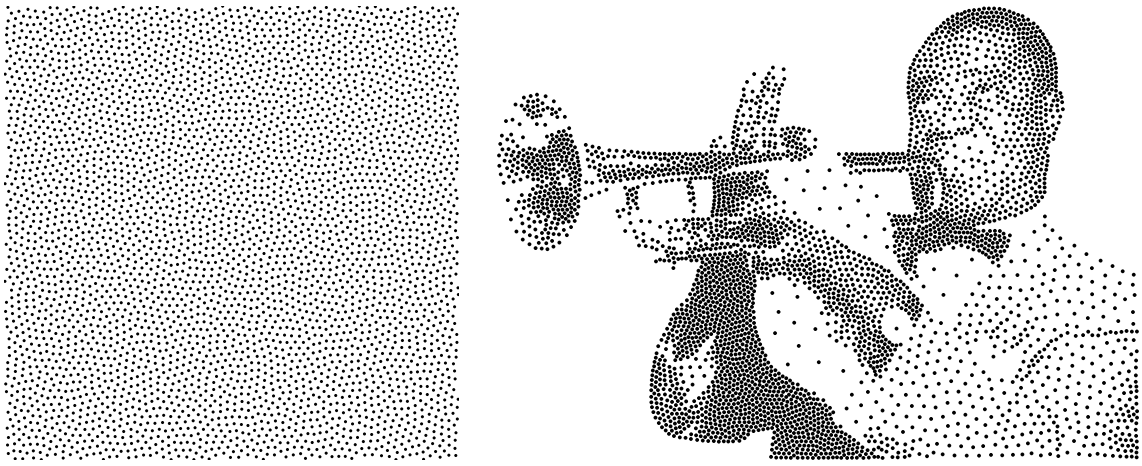


Figure 6.1: Capacity-constrained point distributions with constant density in toroidal space, and with a density function based on a grayscale image

6.1 Point Distributions in Computer Graphics

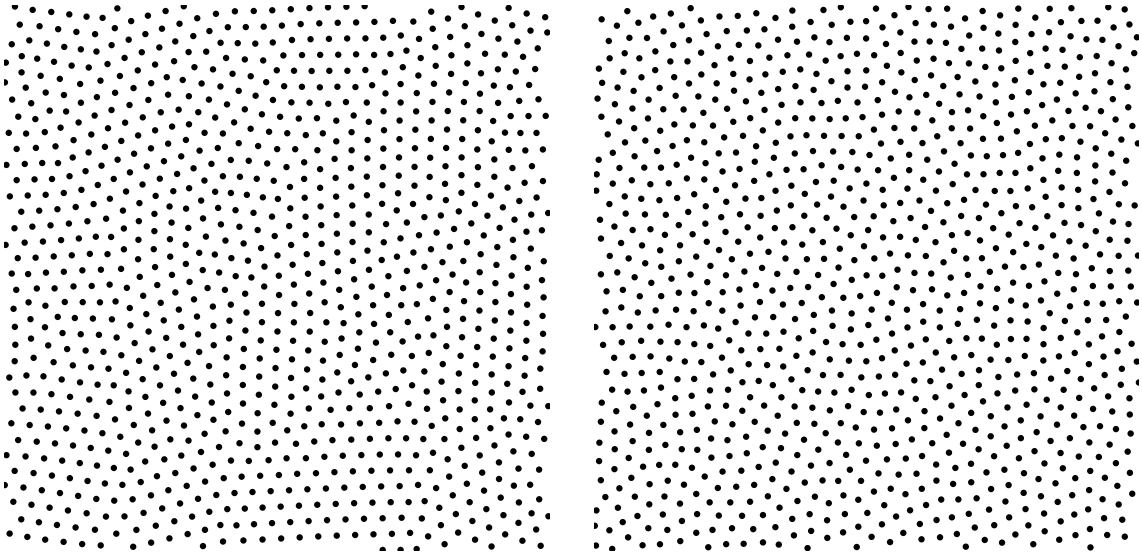
Point distributions are ubiquitous in computer graphics and are used in such diverse domains as sampling, object and primitive distribution, halftoning, point-based modeling and rendering, and geometry processing [63]. One desirable property of point distributions in these contexts is that they possess blue noise characteristics, with large mutual distances between points and no apparent regularity artifacts. Another desirable property is that a point distribution adapts to a given density function in the sense that the number of points in an area is proportional to the density.

The iterative method by Lloyd [50] is a powerful and flexible technique that is commonly used to enhance the spectral properties of existing distributions of points or similar entities. However, the results from Lloyd's method are satisfactory only to a limited extent. First, if the method is not stopped at a suitable iteration step, the resulting point distributions will develop regularity artifacts, as shown in Figure 6.2(a). A reliable universal termination criterion to prevent this behavior is unknown. Second, the adaptation to given heterogenous density functions is suboptimal, requiring additional application-dependent optimizations to improve the results.

In this chapter, we present a variant of Lloyd's method which reliably converges towards distributions that exhibit no regularity artifacts and precisely adapt to given density functions. Like Lloyd's method it can be utilized to optimize arbitrary input point sets to improve their spectral properties whereas avoiding its drawbacks. We achieve the quality of our results by utilizing our capacity constraint. This constraint enforces each point in a distribution to have the same capacity. By demanding that each point's capacity is the same, we ensure that each point obtains equal importance in the resulting distribution. This is a direct approach to generating uniform distributions whereas Lloyd's method achieves such distributions only indirectly by relocating the sites into the corresponding centroids.

Based on this constraint of equal capacities, we utilize our discrete space algorithm presented in Chapter 4 to optimize our point distributions. An evaluation of the results confirms that the constraint of equal capacity is responsible for improved blue noise characteristics and precise density function adaptation. Due to its similarity to Lloyd's method, our method can be used as a substitute in applications that currently benefit from Lloyd's method although at a higher computational cost. In addition, it combines enhancement of blue noise characteristics and density function adaptation in one single operation.

The remainder of this chapter is structured as follows: In the next Section 6.2, we review work related to Lloyd's method and its application in computer graphics. In Section 6.3, we give a short introduction to Lloyd's method. In Section 6.4, we present our variant of capacity-constrained point distributions. In Section 6.5, we provide a critical evaluation of the results of Lloyd's method and those of our method. Finally, in Section 6.6, we draw some conclusions.



(a) Lloyd's method (b) our method

Figure 6.2: Lloyd's method generates point distributions with regular structures, whereas our method does not exhibit such regularities.

6.2 Related Work

To solve the aliasing problem in computer graphics, Dippé and Wold [26], Cook [22], and Mitchell [55] introduced non-uniform sampling. In this process, Poisson disk distributions were identified as a spectrally near optimal sampling pattern, having blue noise characteristics similar to those of receptors in primate's retina [83]. The associated early dart throwing algorithm to generate such point distributions was provided by Cook [22] and accelerated by McCool and Fiume [53] with their relaxation dart throwing algorithm. McCool and Fiume [53] also introduced Lloyd's method to computer graphics, as they recognized that it further improved the spectral properties of their results. They already found that Lloyd's method should be stopped after a few iterations to prevent regularity artifacts.

Since then, several techniques have emerged that try to generate point distributions with blue noise characteristics while offering reduced runtime. These techniques typically involve some tiling approach, allow progressive refinement, and decrease the generation time to real-time speeds. The approaches are either deterministic tilings, such as the Penrose or polyomino tilings of Ostromoukhov et al. [61, 60], or they work on precomputed sets of single tiles, such as by Hiller et al. [38], Lagae and Dutré [47], or Kopf et al. [45]. Each of these techniques require preprocessing to gain blue noise characteristics, and incorporate Lloyd's method at this point.

Next to these extensive construction methods, recent techniques showed significant progress in the generation of Poisson disk distributions, equivalent to those generated by

dart throwing, such as by Jones [41], Dunbar and Humphreys [30], White et al. [82], or Wei [80]. Since none of these techniques is able to combine blue noise characteristics, progressive refinement, and density function adaptation in an equally effective way, Lloyd's method remains pivotal in the context of sampling in computer graphics.

In the aforementioned real-time techniques, Lloyd's method was not used for the adaptation to given density functions. However, Lloyd's method was used for density adaptation by offline techniques, such as by Secord [68] for non-photorealistic rendering, Kollig and Keller [44] for image based lighting using high dynamic range images, or Surazhsky et al. [71] for geometry processing. In these cases, Lloyd's method had to be combined with other heuristics or optimization approaches to improve the density function adaptation. On the other side of the spectrum, Chen [21] suggests a mesh optimization method based on centroidal Delaunay graphs. This method aims at optimizing shape regularities, and tends to produce hexahedral artifacts as well.

6.3 Lloyd's Method

Centroidal Voronoi tessellations, which have already been introduced in Section 2.3, are frequently used in computer graphics because of their relation to the energy function

$$\mathscr{F}(S) = \sum_{i=1}^{n} \int_{x \in V(s_i)} \rho(x) \|x - s_i\|^2 dx, \tag{6.1}$$

where $\mathscr{V}(S)$ is a Voronoi tessellation consisting of the Voronoi regions $V(s_i), s_i \in S$, formed by the set S of sites with cardinality n. This energy function describes the centrality of the sites within their regions, and the uniformity of the distribution of all sites.

A common way to generate a centroidal Voronoi tessellation is to employ the method of Lloyd [50], which is an implementation of the spatial-temporal adjustment model with $\alpha = 1$. This iterative algorithm performs the following steps:

1. generate the Voronoi tessellation $\mathscr{V}(S)$ in Ω;
2. move each site $s_i \in S$ to the centroid p_i of the corresponding Voronoi region $V(s_i) \in \mathscr{V}(S)$;
3. if the new sites in S meet some convergence criterion, then terminate; otherwise return to step 1.

Due to the fact that each relocation of a site to its centroid reduces the energy $\mathscr{F}$, the algorithm converges to a local minimum of $\mathscr{F}$, in which each site coincides with the centroid of its corresponding Voronoi region.

Lloyd's method can either be performed in continuous or in discrete spaces. An implementation in continuous spaces is usually faster, but also more complex and less flexible. Especially the incorporation of a density function is intricate in the continuous case. Therefore it is common to use discrete implementations, where the bounded space Ω with

density function ρ is represented by a set X of m sample points. Based on this point set X, the Voronoi tessellation is formed by an assignment $A : X \rightarrow S$, where each point in X is assigned to the closest site in S. Consequentially, the energy function $\mathscr{F}$ in Equation 6.1, which is minimized by Lloyd's method, is substituted by

$$\mathscr{F}'(X,A) = \sum_{i=1}^{m} \|x_i - A(x_i)\|^2. \tag{6.2}$$

The computational time complexity of Lloyd's method for n sites in two dimensions with constant density is $O(n+n\log n)$ for each iteration [29]. Its memory complexity is at least $O(n)$, or $O(n+m)$ if the algorithm is based on a discrete space with m sample points.

6.4 Capacity-Constrained Method

Following the definition of capacity-constrained Voronoi tessellations in Section 2.4, we say that a distribution of sites in S adapts optimally to the density function ρ, if the capacity c_i of each site $s_i \in S$ fulfills the constraint

$$c_i = c^*, \tag{6.3}$$

where c^* describes a portion of the underlying space Ω related to the number of sites in S with

$$c^* = \frac{\int_\Omega \rho(x)dx}{n}. \tag{6.4}$$

The term $c_i = c^*$ for each site $s_i \in S$ is our capacity constraint.

Intuitively, the capacity of a site is equivalent to the area of its corresponding Voronoi region weighted with the density function. It can be understood as a measure of importance. Our capacity constraint $c_i = c^*$ enforces that each site in a distribution is equally important. This is analogous to the approach of importance sampling, in which samples are distributed so that the chance of a location being sampled is proportional to the density at that location [35]. Thus, our capacity constraint $c_i = c^*$ naturally models an underlying density function, providing us with a precise way of specifying a sample's weight in a distribution.

An arbitrary distribution of n sites in S does usually not fulfill our capacity constraint $c_i = c^*$. Rather, such distribution has to be determined by utilizing our discrete space algorithm presented in Chapter 4. We use this algorithm because of the fact that we are just interested in the locations of the sites, and do therefore not need the geometric bisector representation of our continuous approaches. Furthermore, our discrete algorithm allows to easily represent arbitrary density functions by the discrete point set X.

The capacity-constrained assignments $A : X \rightarrow S$ generated with the our discrete algorithm represent weighted Voronoi tessellations. For the application of point distribution for sampling in computer graphics, we are actually interested in ordinary Voronoi tessellations, which are in general different from the weighted Voronoi tessellations that result

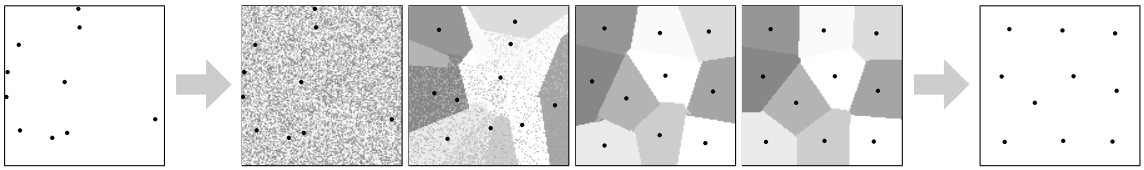


Figure 6.3: Our method takes an existing site distribution and transfers it to a random discrete assignment in which each site has the same capacity. This assignment is then optimized with our discrete space algorithm in Chapter 4, whereby the sites are simultaneously relocated to the centroids of their regions. The optimization stops at an equilibrium state with the final capacity-constrained distribution.

from our discrete space algorithm. However, this difference is almost eliminated if two additional constraints are fulfilled: first, if the capacity for each site is equal, and second, if each site resides in the centroid of its Voronoi region. Our experiments showed that in this case less than 3 percent of all points in X are assigned to a site $s_i \in S$ in the capacity-constrained Voronoi tessellation $\mathscr{V}(S,C)$ while they are assigned to another site $s_j \in S$ in the ordinary Voronoi tessellation $\mathscr{V}(S)$ of the identical set of sites S. Fortunately, the first constraint is already assured by our capacity constraint $c_i = c^*$ for each site, and the second constraint can be easily achieved by the centroidal variant of our discrete space algorithm.

To generate our capacity-constrained distributions we perform the following steps, which are additionally illustrated by Figure 6.3:

0. create a set X of m points that is a discrete representation of the space Ω weighted with the density function ρ;

1. generate the capacity-constrained Voronoi tessellation $\mathscr{V}(S,C)$ for the set S of n sites as an assignment $A : X \to S$ where each site $s_i \in S$ has the capacity $c_i = \frac{m}{n}$;

2. move each site $s_i \in S$ to the center of mass of all points $x_i \in X$ that are assigned to this site, $A(x_i) = s_i$;

3. if the new sites in S meet some convergence criterion, then terminate; otherwise return to step 1.

These steps are a variant of Lloyd's method, in which the generation of the Voronoi tessellation is substituted with the capacity-constrained optimization in discrete space.

Note that the number of points in the set X influences the quality of the resulting distributions with respect to their blue noise characteristics. For two-dimensional dimensional space, we achieved results of good quality with around 100 points per site. More points per site slightly improved the blue noise characteristics of the distributions, until around 1000 points per site where a further increase showed no noticeable improvement.

The computational time complexity class for each iteration of our method for a set of n sites is $O(n^2 + nm\log\frac{m}{n})$, which is significantly higher than Lloyd's method with $O(n + n\log n)$ for each iteration. We do not consider this a significant drawback as our method is not primarily a stand-alone technique and typically incorporated as a preprocessing step. Thus, our method practically allows the same application scenarios as Lloyd's method, especially since it shows a much more rapid convergence. After only four or five iterations, our results exhibit very good properties, with most of the remaining computation time necessary for the subtle improvements towards the final equilibrium state, while Lloyd's method converges much slower and is highly dependent on the initial distribution.

6.5 Evaluation of Results

In the previous sections, we state that Lloyd's method generates point distributions that possess suboptimal blue noise characteristics and do not adapt well to given density functions. We also state that the results of our method are better with respect to these two properties. In this section, we substantiate these claims by an extensive analysis of the results of Lloyd's method and those of our method. Furthermore, we illustrate why our capacity constraint is reasonable. Finally, we present three application examples.

For the evaluation of blue noise characteristics, we restrict ourselves to point sets in a toroidal square with constant density. This is necessary to evaluate spectral properties, but the findings still posses validity in other spaces and for arbitrary density functions.

6.5.1 Blue Noise Characteristics

As outlined in Section 6.2, point distributions with blue noise characteristics are desirable in computer graphics, especially in the domain of sampling. Good distributions have a large mutual distance between their points and do not exhibit regularity artifacts. Lloyd's method yields in distributions with locally hexagonal tessellations and suboptimal blue noise characteristics, as shown in Figure 6.4. Here, a representative distribution of 1024 random points was generated with Lloyd's method until no further relocation of sites occurred. The distribution clearly exhibits patches of hexagonal lattices. These regular structures are reflected in the corresponding mean periodogram [74] in Figure 6.5, which is not smooth, having turbulent radial power and significant anisotropy. This mean periodogram is calculated from the results of Lloyd's method for 10 different initial random sets of 1024 points. In contrast, our capacity-constrained distributions, which were computed with the same initial point sets, show substantially better results: regularities are much less apparent in the representative example in Figure 6.4, and the mean periodogram in Figure 6.5 is smoother, with less turbulent radial power and lower anisotropy.

This observation of better blue noise characteristics for the distributions generated with our method is independent of the initial point distribution. Further evidence for

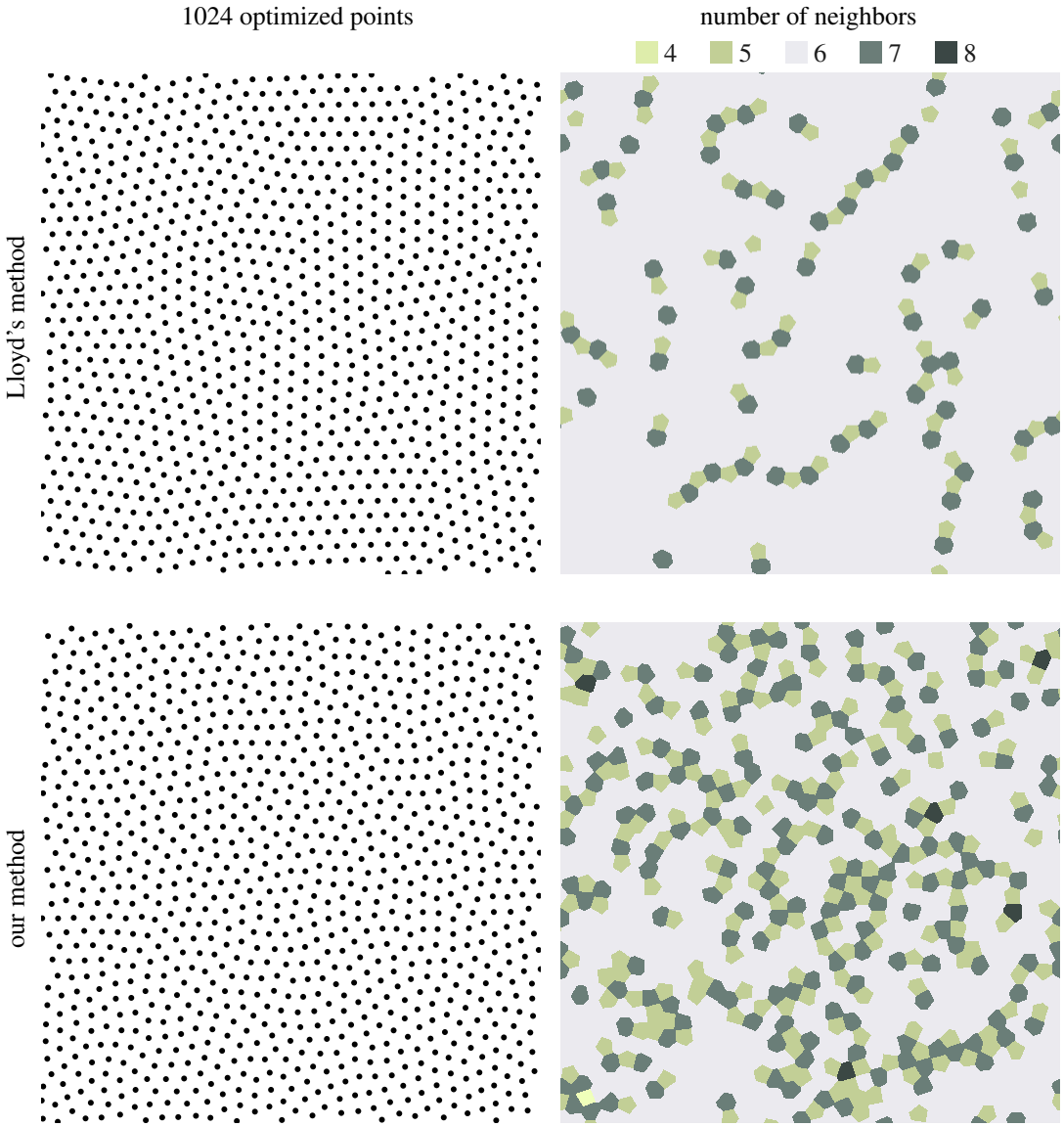
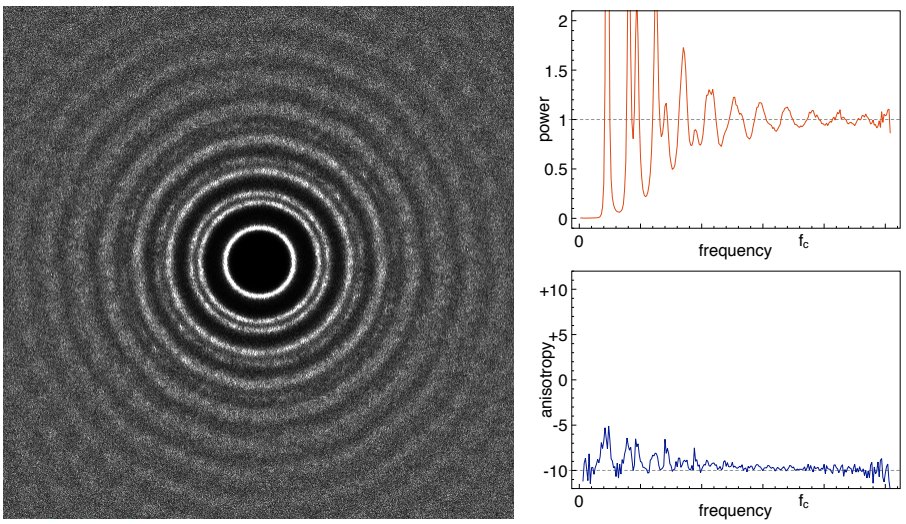
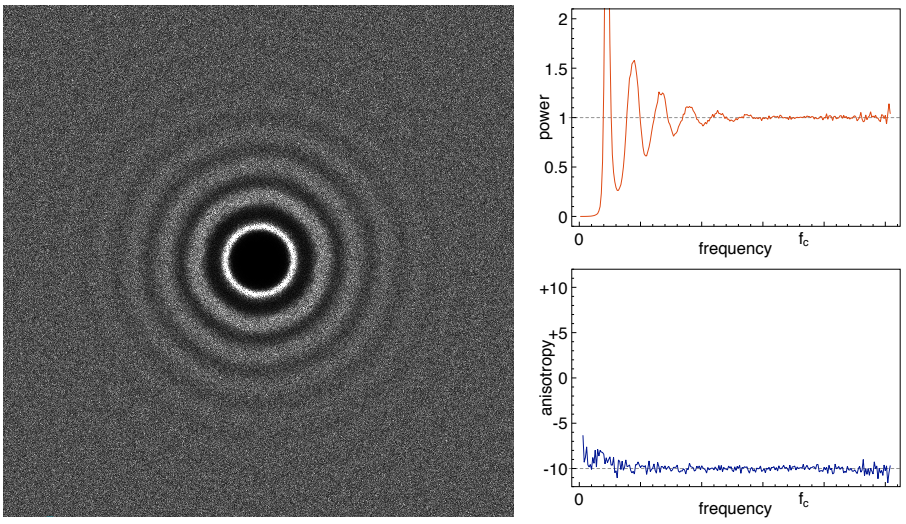


Figure 6.4: Lloyd's method generates point distributions with regular structures if it is not stopped manually. This becomes evident by color coding the number of neighbors for the Voronoi region of each point. The example set of 1024 points was computed with Lloyd's method to full convergence and contains large patches of hexagons. These patches are separated by just a few heptagons and pentagons in between. Due to the higher variance in the number of neighbors, the result of our method does not exhibit such regularities.



(a) Lloyd's method



(b) our method

Figure 6.5: Comparison of the spectral analysis of distributions with 1024 points generated with Lloyd's method and with our method. The distributions generated with our method result in a much smoother mean periodogram with less turbulent radial power and lower anisotropy.

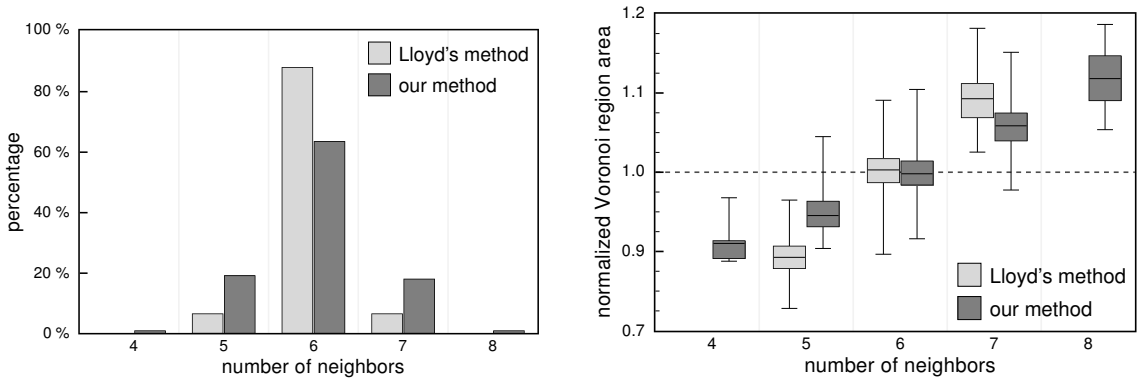
this fact is given in Figure 6.7 and Figure 6.8. For these two figures, and also all similar figures in this chapter, we computed mean periodogram images, radial power, and anisotropy graphs conform to the conventions by Lagae and Dutré [48]. The spectral analysis is performed on 10 sample sets, the radial power is normalized, and the high-magnitude DC-peaks are removed from all plots. Each sample set consists of 4096 points to produce mean periodogram images of 512×512 pixels. The images have been tone-mapped using the mapping $x \mapsto \ln(1+x)$. For the sampling scenario, we utilized point sets of $512^2 = 262144$ points to produce images with a resolution of 512×512 pixels as well. We sampled the zone plate test function

$$(x,y) \mapsto 0.5 + \frac{1}{2}\sin\left(1600(x^2+y^2)\right) \tag{6.5}$$

and reconstructed the images using a cubic b-spline filter. Compared to the reference image in Figure 6.7(a) and Figure 6.8(a), notice in particular the increased radius of the gray, noise-free area in our method's zone images without introducing new artifacts like Lloyd's method. The results are very consistent for each optimization method with our method continuously delivering smoother results, regardless of the input point sets.

The reason for the development of regularity artifacts when using Lloyd's method is the minimization of the energy function $\mathscr{F}$ in Equation 6.1. The minimal value for the summation term in this function would result if the Voronoi region converged to the shape of a circle. Thus, a distribution in two-dimensional space is in a global minimum of $\mathscr{F}$ if it forms a regular hexagonal lattice, since a regular hexagonal lattice offers the best approximation of its Voronoi regions to the shape of circles. This is in line with observations in the domain of circle packing [72]. Of course, perfect hexagonal lattices are not achieved for any number of sites. Rather, arbitrary tessellations that approach local optimality with respect to $\mathscr{F}$ also contain non-hexagons in between. It is apparent, for example, that pentagons are less optimal than hexagons because of their worse circle approximation. The opposite is true for heptagons, which are more optimal than hexagons because of their better circle approximation. In other words, a n-gon is less optimal with respect to $\mathscr{F}$ than a $(n+1)$-gon with the same area. Therefore, an arbitrary distribution of sites optimized by Lloyd's method approaches a tessellation that contains many as large as possible hexagonal patches, which are then connected by a few smaller n-gons with $n < 6$, and a few larger n-gons with $n > 6$. Our constraint of equal capacity significantly improves on this behavior due to the fact that all regions are equally sized throughout the minimization, so that the energy can no longer benefit from area differences between n-gons and $(n+1)$-gons. This effect is illustrated by the neighbor diagrams next to the point sets in Figure 6.4. These diagrams show hexagonal patches for Lloyd's method, whereas our result is much more heterogenous with fewer noticeable regularities.

These theoretical considerations are confirmed by our experiments, as illustrated in Figure 6.6. Here, we present an analysis of 10 different distributions of 1024 sites that have been optimized with Lloyd's method until all sites were stable. This analysis identifies that 87.8% of the Voronoi regions are hexagons, 6.1% are pentagons, and again 6.1% are heptagons. Furthermore, the pentagons are significantly smaller than the hexagons,



(a) number of neighbors percentage

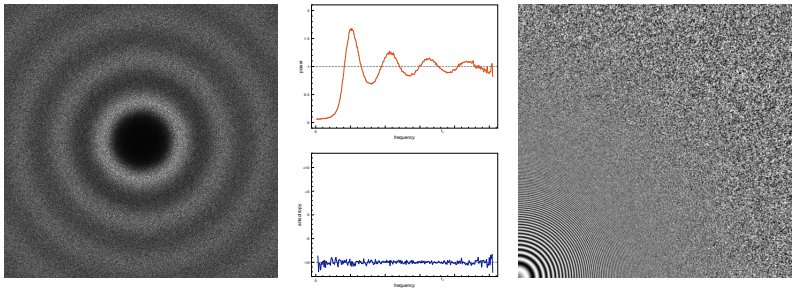
(b) normalized Voronoi region area

Figure 6.6: Comparison of 10 result sets with 1024 sites for Lloyd's and our method. Image (a) illustrates that our method generates less hexagons than Lloyd's method. Image (b) illustrates that our method generates regions with more uniform areas than Lloyd's method.

which in turn are significantly smaller than the heptagons. A similar analysis of the results of our method based on the same initial site sets shows that our method yields a larger variance in shape, with only 69.6% hexagons, 15.5% pentagons, 14.7% heptagons, and even a small fraction of quadrilaterals and octagons. Furthermore, the area of these groups of n-gons are more uniform.

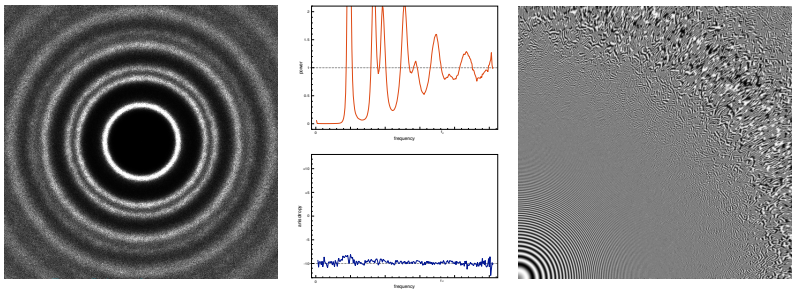
6.5.2 Termination Criterion for Lloyd's Method

The development of regularity artifacts in distributions generated with Lloyd's method is well known in computer graphics. A common solution is to stop Lloyd's method after a few iterations. Lagae and Dutré [48] suggest the normalized Poisson disk radius $\alpha \in [0,1]$ as a quality measure for point distributions. This radius α is equal to zero if any two points in the distribution coincide, equal to one for a hexagonal lattice, and considered optimal for $\alpha \approx 0.75$. The convergence of α can be utilized as a termination criterion, where we stop Lloyd's method if α is stable. However, our experiments show that this is a rather unreliable criterion. Figure 6.9 demonstrates this by a comparison of a set of 1024 points, and a set of $512^2 = 262144$ points used for the sampling of the zone plate test function. In this example, an initial set of 1024 points is optimized by Lloyd's method. After 40 iterations the points are well distributed with a normalized radius of $\alpha \approx 0.75$ and good blue noise characteristics. Further optimization deteriorate the spectral properties and introduces hexagonal structures. In contrast, $\alpha \approx 0.75$ proves to be ill-suited for the sampling of the zone plate test function with 512^2 points as strong artifacts become

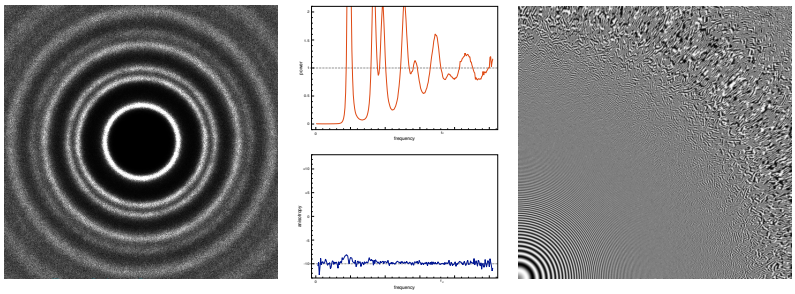


(a) dart throwing reference plots

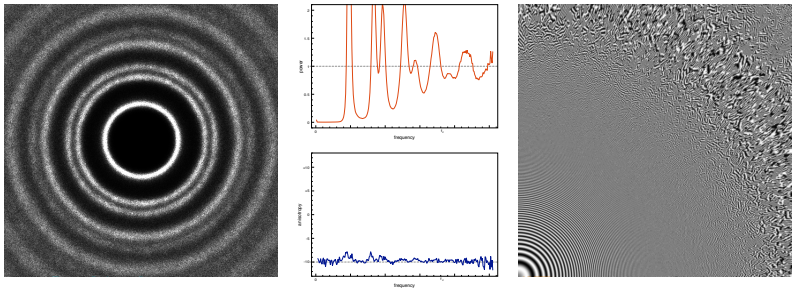
Lloyd's method



(b) input point sets: random

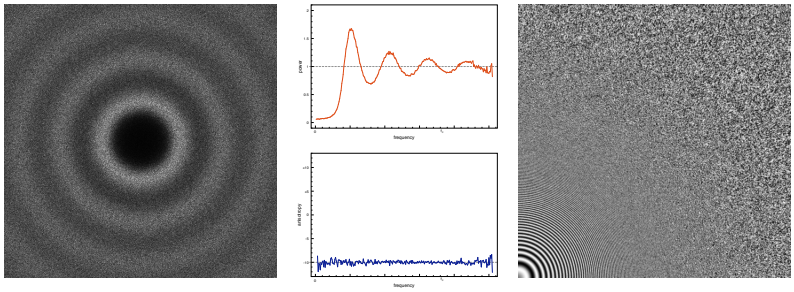


(c) input point sets: traditional dart throwing



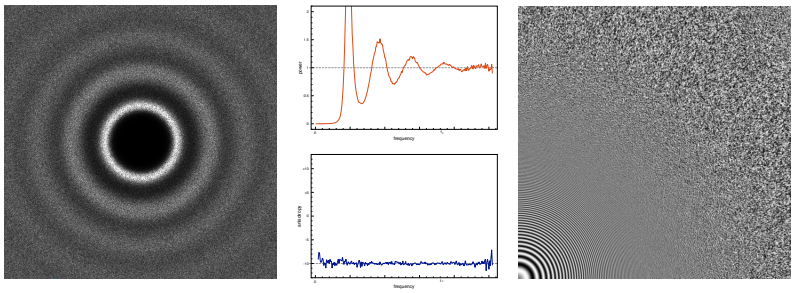
(d) input point sets: relaxation dart throwing

Figure 6.7: Spectral analysis and results from sampling the zone plate test function with various input point sets optimized by Lloyd's method.

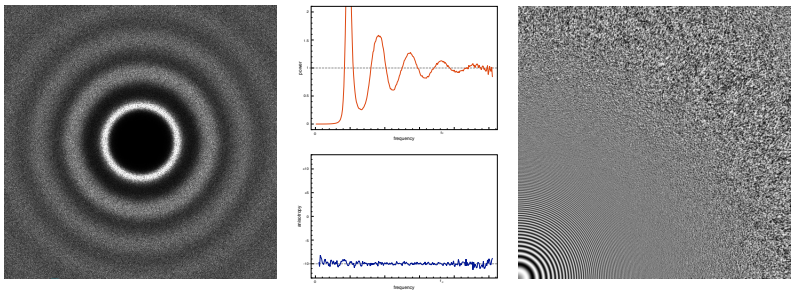


(a) dart throwing reference plots

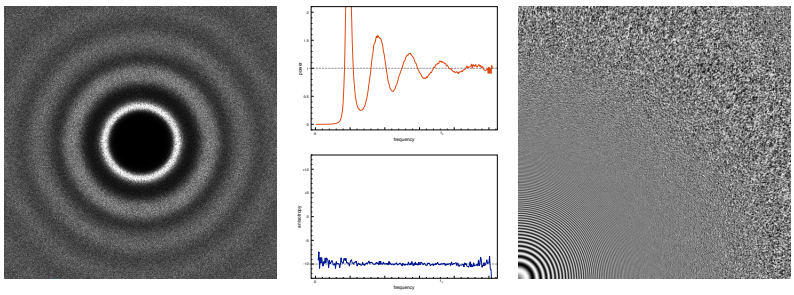
capacity-constrained method



(b) input point sets: random



(c) input point sets: traditional dart throwing



(d) input point sets: relaxation dart throwing

Figure 6.8: Spectral analysis and results from sampling the zone plate test function with various input point sets optimized by our method.

apparent. Relying on the convergence of α is also not an option as marginally less artifacts can be observed. In this sampling scenario, stopping Lloyd's method after about 10 iterations with $\alpha \approx 0.53$ would provide the best sampling results. In contrast, our method converges reliably to an equilibrium with better properties in both scenarios. The example shows that α strongly depends on the number of points and the initial distribution. This is reasoned by the character of the Poisson disk radius, which is determined by the smallest distance between any two points, while being a representative for the overall distribution. During Lloyd's method, it is common that a small fraction of closely packed points remains stable, while the overall distribution is still changing.

Identifying a universal termination criterion for Lloyd's method with respect to the blue noise characteristics is an unsolved problem. This necessitates either a manual intervention, or an intricate search for an application-related criterion. In contrast, our method does not approach any critical states, and terminates reliably in an equilibrium with good properties. Our results demonstrate better blue noise characteristics for the set of 1024 points with $\alpha \approx 0.77$, as well as a low-artifact sampling of the zone plate test function with $\alpha \approx 0.63$. In general, our method generates results that are close to $\alpha \approx 0.75$, the value employed by Lagae and Dutré [48] for a reference point set obtained via dart throwing. This is further illustrated by Figure 6.10. Here, we used a large number of different initial point sets generated by different methods such as random sampling or dart throwing as input for our method. The plot shows that all results are within the recommended interval, and near the reference value $\alpha \approx 0.75$.

6.5.3 Density Function Adaptation

The concept of equal capacity is directly related to importance sampling in computer graphics. It offers the possibility to measure the quality of a density function adaptation by a distribution of sites via the normalized capacity error

$$\Delta_c = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{|V(s_i)|}{c^*} - 1 \right)^2 \tag{6.6}$$

for all Voronoi regions $V(s_i) \in \mathscr{V}(S)$. Site distributions that are well adapted to their underlying density function have Δ_c close to zero.

We observed in our experiments that if the density is constant, Lloyd's method generates a uniform distribution with a small capacity error. Furthermore, if the density is not constant, Lloyd's method generates site distributions with a large capacity error and a suboptimal density function adaptation. In particular, areas with high density contain too few sites, and areas with low density contain too many sites. This indicates that Lloyd's method implicitly blurs the density function. This erroneous behavior can be attributed to the local operation of centroid relocation, which disregards any global characteristics of the underlying density function.

The implicit blurring of the density function in Lloyd's method is illustrated by Figure 6.11. Here, a quadratic ramp is used as density function in (a). The percentages

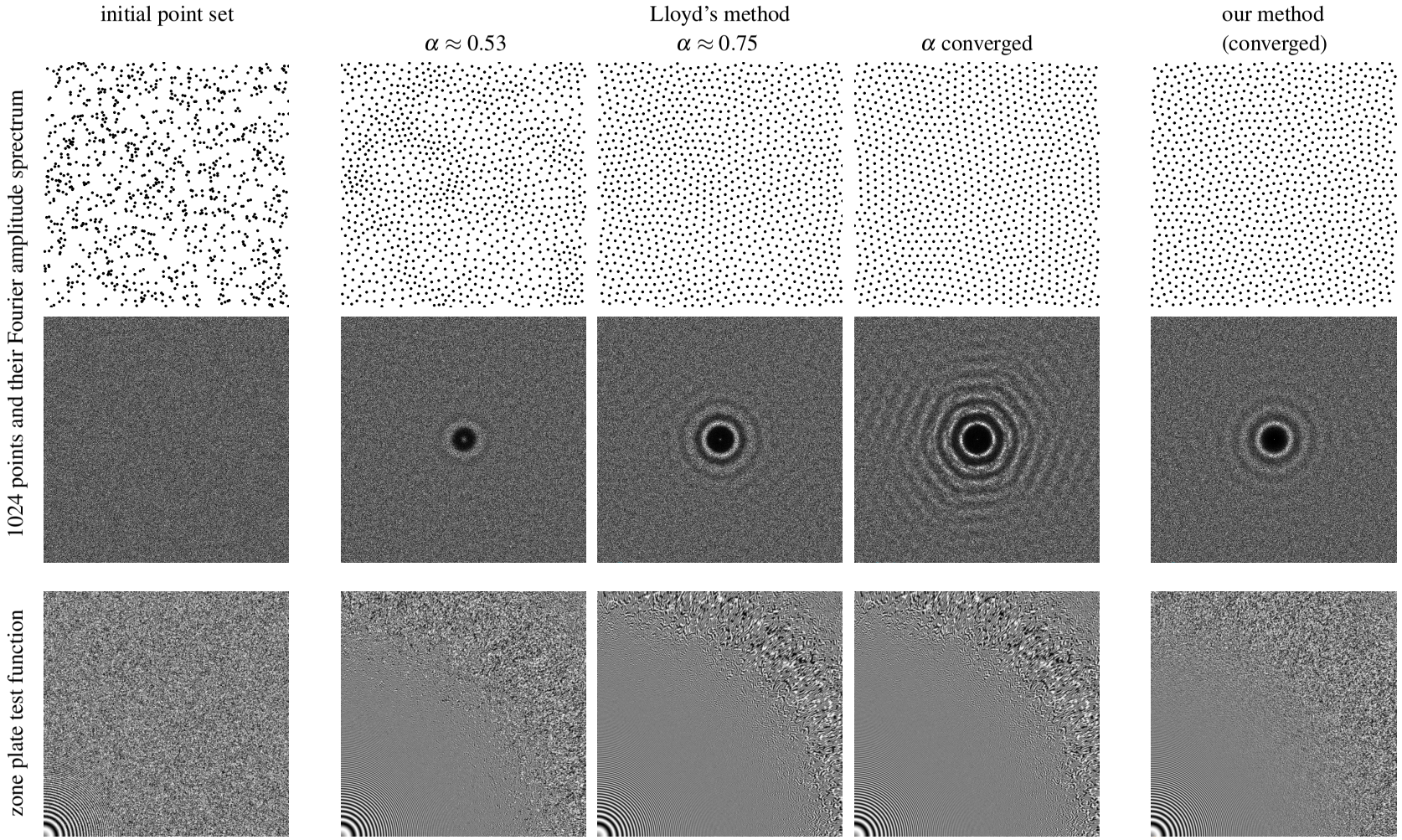


Figure 6.9: An initial point set is optimized by Lloyd's method. After 40 iterations the points are well distributed with $\alpha \approx 0.75$. Further optimization deteriorate the spectral properties. In contrast, $\alpha \approx 0.75$ proves to be ill-suited for the sampling of the zone plate test function. Our method converges reliably to an equilibrium with better properties in both scenarios.

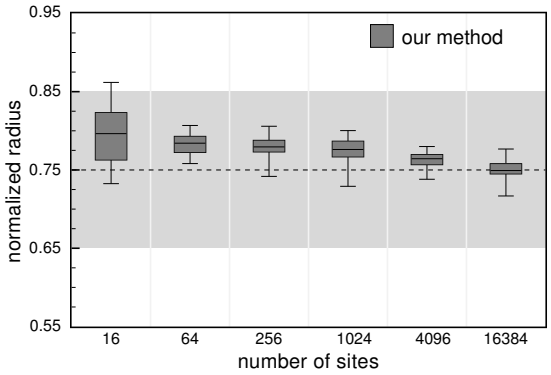
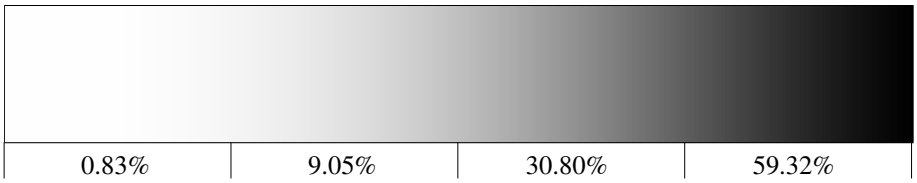


Figure 6.10: Comparison of result sets with different numbers of sites with respect to the normalized radius α. Our results have a normalized radius α within the preferable interval $[0.65, 0.85]$, and near the reference value $\alpha \approx 0.75$ by Lagae and Dutré [48].

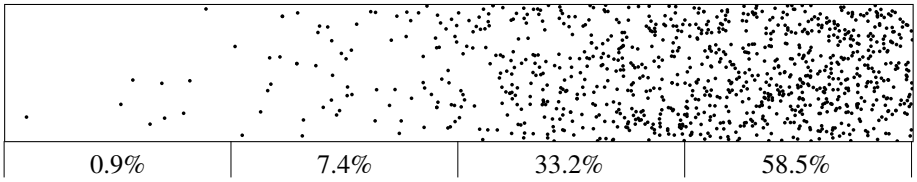
indicate the density ratios that are contained in each quarter of the ramp. An initial set of 1000 sites in (b) is chosen via random sampling, and has a capacity error of $\Delta_c = 0.25622$. The percentages of the quarters denote the ratios of contained sites, showing a reasonable first approximation of the density function. By applying Lloyd's method to this initial distribution, at first the capacity error decreases to a minimum of $\Delta_c = 0.01206$ in (c). Afterwards, the capacity error again steadily increases. The final result of Lloyd's method in (d) has a capacity error of $\Delta_c = 0.08233$, where the leftmost quarter contains only 0.83% of the overall density but 4.0% of the sites, and the rightmost quarter contains 59.32% of the overall density but only 48.9% of the sites. This behavior illustrates the suboptimal density adaptation of Lloyd's method. Of course, the capacity error can be used as a termination criterion for Lloyd's method, but even the minimum capacity error is in general far from optimal. Usually, such minimum already contains regularity artifacts, which can be observed in image (c) as well. In contrast, our method generates the distribution in (e) from the same initial sites. It precisely adapts to the given density, possessing a capacity error of $\Delta_c = 0.00131$. The ratios of sites per quarter are also highly correlated with the density ratios.

An additional example that illustrates this smoothing effect is given in Figure 6.12. Here, we used the density function $\rho(x,y) = e^{(-20x^2-20y^2)} + 0.2\sin^2(\pi x)\sin^2(\pi y)$, and a set of 2048 sites generated with random sampling as initial distribution. Again, Lloyd's method significantly blurs the density function, whereas our method generates a precise reproduction of the underlying density.

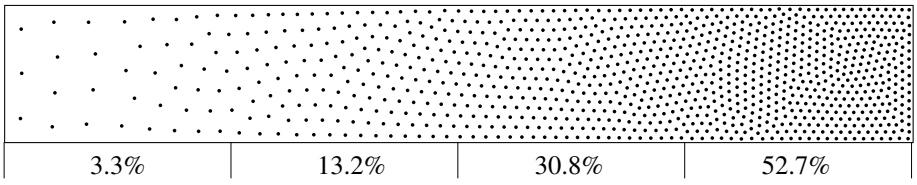
Another measure for the uniformity of the distribution is the *discrepancy* [46]. By analyzing the development of the discrepancy during the computation we observed that our method generates a distribution with low discrepancy already after the first iteration. In contrast, Lloyd's method decreases the discrepancy slowly during the computation.



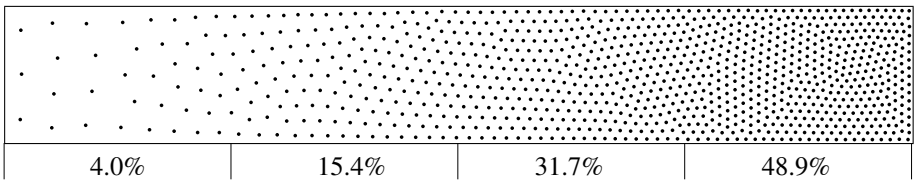
(a) density function



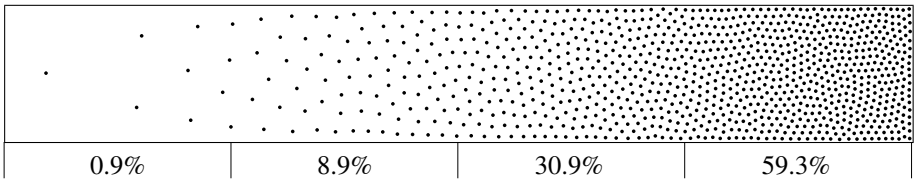
(b) initial set of 1000 sites via random sampling, $\Delta_c = 0.32711$



(c) best intermediate result of Lloyd's method, $\Delta_c = 0.01206$

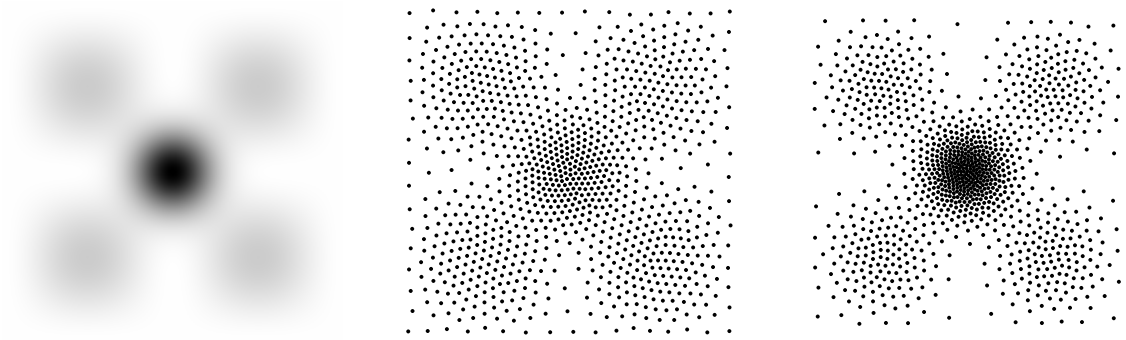


(d) final result of Lloyd's method, $\Delta_c = 0.08233$



(e) final result of our method, $\Delta_c = 0.00131$

Figure 6.11: The quadratic ramp in (a) is used as density function. Starting with the initial sites in (b), Lloyd's method continually blurs the density function. In contrast, our result shows precise adaptation. The percentages show the amount of density or the number of points contained in each quarter.



(a) density function (b) Lloyd's method (c) our method

Figure 6.12: Distributions of 2048 sites based on the density function $\rho = e^{(-20x^2-20y^2)} + 0.2\sin^2(\pi x)\sin^2(\pi y)$. This function is depicted in image (a) as a grayscale rendering. Lloyd's method significantly smoothes the density function, whereas our method generates a precise reproduction of the underlying density.

After the computation stopped for both methods, the discrepancy of our result was also reliably smaller than those of Lloyd's method. However, even Lloyd's method generates results with what is considered as low discrepancy, $O(\frac{\log n}{n})$.

6.5.4 Application Examples

To further demonstrate the quality of our distributions beside their spectral properties, we present three application examples. The first application is non-photorealistic stippling, which places small black dots such that their distribution gives the impression of tone. One prominent example for this technique is the approach by Secord [68], which uses weighted centroidal Voronoi tessellations to generate stipple drawings from a given grayscale image. A result of this approach is shown in Figure 6.14. By applying our capacity-constrained method without any modification to the same grayscale image, we obtain the stipple drawing in (c). The computation time for this distribution was 17 minutes on Intel Core 2 hardware. Both results use 20000 stipples with the same equal radius. Our result better reproduces the contrast of the grayscale original and contains fewer regularity artifacts.

The second application samples high dynamic range images with respect to their radiance and/or luminance. The resulting samples are then used for image based lighting. The density distribution in such images is characterized by large regions with low density, and small light emitting regions with density values that are orders of magnitude higher. Although Lloyd's method usually fails in the density adaptation of such images, it is nevertheless employed in combination with improvements that reduce its erroneous behavior [44, 61]. By applying our capacity-constrained method without any modification, we are

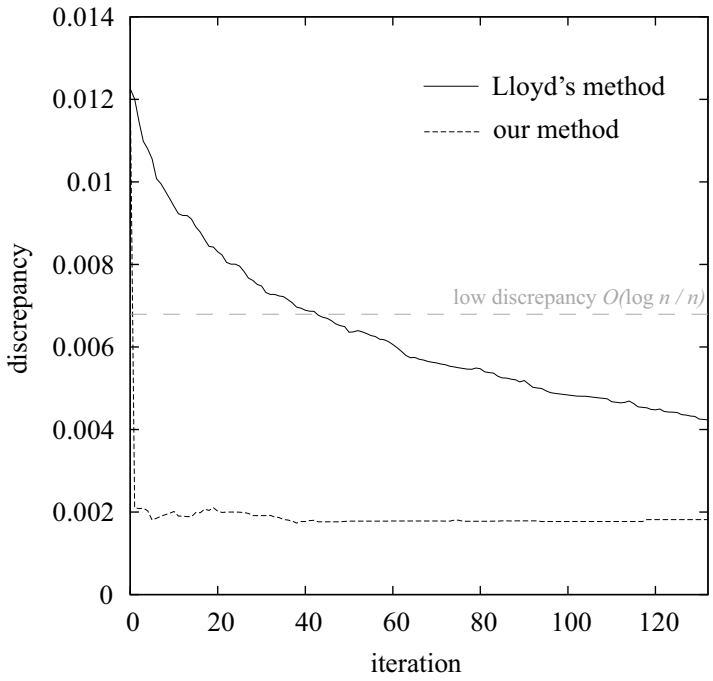
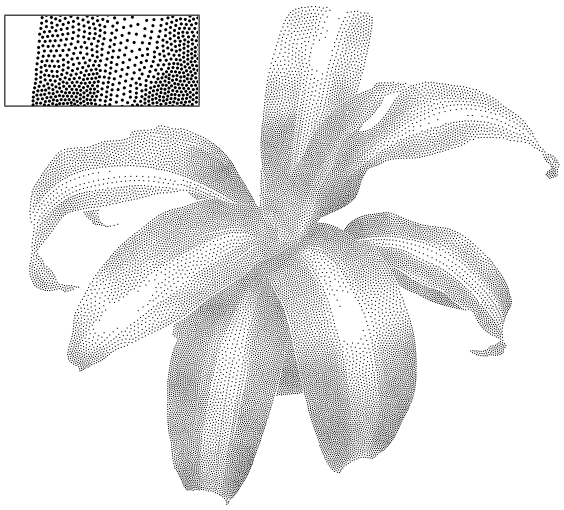


Figure 6.13: Comparison of the *discrepancy* [46] during the computation. Our method achieves a distribution with low discrepancy much faster than Lloyd's method.

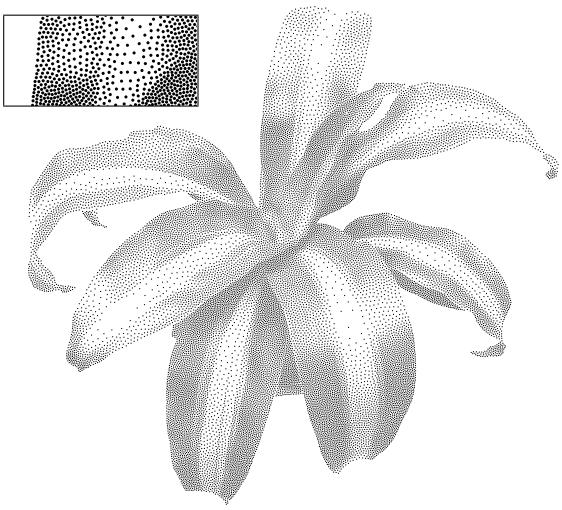
able to create a distribution that precisely adapts to the underlying lighting information. An example of a resulting importance sampling of the luminance of a high dynamic range image is given in Figure 6.15. Here, the distribution of 3000 samples exhibits no regularity artifacts and extracts even subtle features in areas with maximum density such as the cross of the main window. The computation time was 8 minutes on Intel Core 2 hardware.

The third application is color reduction, which converts images with high color resolution to images with low color resolution. For this application, we use a three-dimensional discrete space that represents the LAB color space formed by the individual content of the image. Therefore, we insert one point for each pixel with its LAB values as coordinates. The LAB color space is used because of the fact that the Euclidean distance in LAB space corresponds to human color perception. Now we randomly insert a number of sites depending on the number of colors for the resulting image, and compute their centroidal capacity-constrained Voronoi tessellation. The result is a set of LAB coordinates that is subsequently used for reproducing the original image with a smaller color palette. This approach is directly related to similar methods for color reduction [54] based on k-means clustering. An example for this application is given in Figure 6.16, where we created a palette of 16 color from a 24 Bit color image. Although this result is not state-of-the-art in its field, it illustrates the flexibility of our method. To improve on our current results, we had to modify the distance function to incorporate findings in the domain of human

(a) original grayscale image



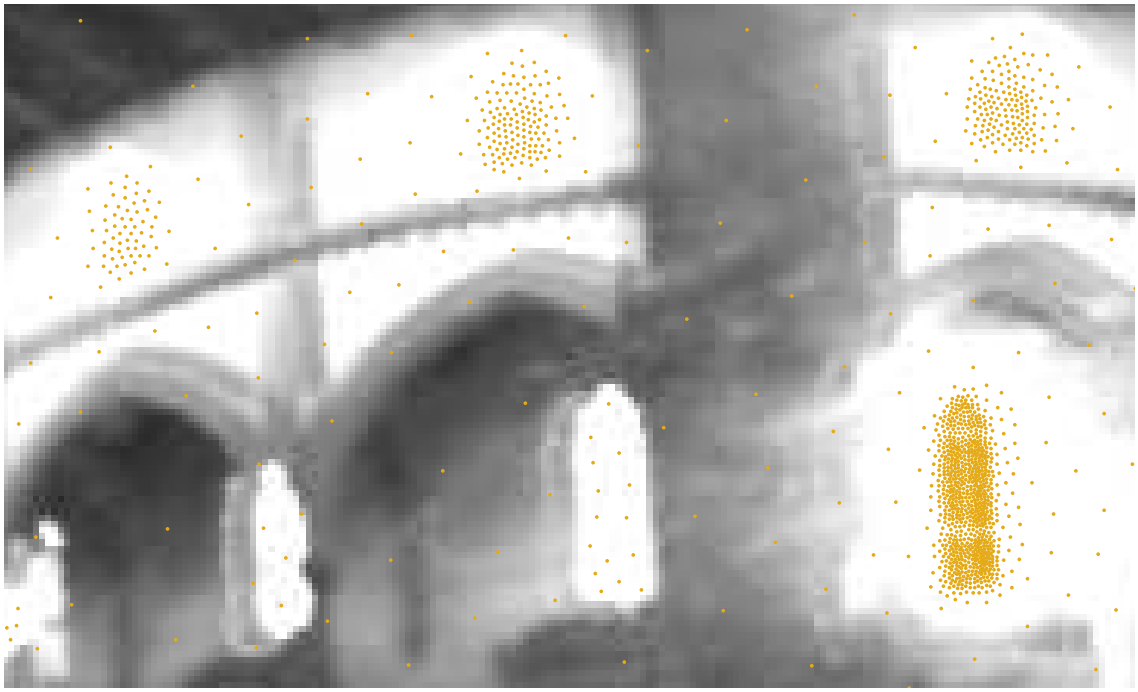
(b) result by Secord [68]



(c) our result

Figure 6.14: The grayscale image in (a) is used as density function to generate stipple drawings. The result of our unmodified method in (c) exhibits no regularities and higher local contrast than the reference in (b). Each stipple drawing uses 20000 points with the same radius.

(a) complete luminance sampling



(b) detail view

Figure 6.15: Image (a) shows a luminance sampling of the high dynamic range image "The Galileo's Tomb" (courtesy of Paul Debevec) using 3000 points. A detail of the same point distribution is shown in the image (b). Our method generates no regularity artifacts and extracts even subtle features such as the the cross of the main window.

(a) original 24 Bit color image (b) 16 colors using dithering

Figure 6.16: Image (a) was used as input for the discrete space in which a distribution of 16 sites was computed with our method. The resulting 16 colors were used for rendering image (b) using dithering.

color perception.

6.6 Conclusion

We presented a new general purpose method for optimizing point distributions. Our method improves on the established Lloyd's method by utilizing equal capacities as an optimization constraint. The resulting capacity-constrained point distributions exhibit improved blue noise characteristics, having a large mutual distance and no apparent regularity artifacts. Arbitrary density functions can be utilized to control our point distributions, and again, the capacity constraint guarantees their precise adaptation.

An important advantage of our method is the reliable quality of the results. After only a few iterations with our method, the point distributions possess significantly improved blue noise characteristics and are precisely distributed according to the underlying density function. In the remaining process, these properties are further improved, until an equilibrium state is finally achieved. The quality of the final result is therefore neither dependent on the initial distribution nor is it less optimal than earlier results during the optimization.

Our method is sufficiently simple, having a similar implementation effort as Lloyd's method. Simultaneously, it avoids the drawbacks of Lloyd's method and does not require a manual intervention or an application-dependent termination criterion. In addi-

tion, the enhancement of blue noise characteristics and the density function adaptation can be combined in one operation. In most applications, our method can be used as a drop-in replacement for Lloyd's method.

To support the implementation of our capacity-constrained point distributions, we provide C++ source code through an open source project at `http://ccvt.googlecode.com`.

Chapter 7

Concluding Remarks

Voronoi tessellations are commonly used for describing, analyzing, and optimizing complex structures or processes. Capacity-constrained Voronoi tessellations are special cases of this general concept that focus on the areas of the Voronoi regions. By considering the capacity, we are able to evaluate the actual spatial influence of the sites in the tessellations, and generate tessellations in which each region has a predefined area. Capacity-constrained Voronoi tessellations cannot be computed directly, rather it is necessary to iteratively compute approximations until some convergence criterion is achieved. In Chapter 3, we presented two such approximation algorithms for continuous spaces that iteratively optimize an initial Voronoi tessellation until it fulfills a given capacity constraint. Both approaches differ in the type of distance function utilized for the tessellations. Although a proof of convergence for these two algorithms is still an open research problem, our experiments showed their reliable convergence towards arbitrarily precise capacity-constrained Voronoi tessellations. In Chapter 4, we presented an algorithm for the computation of capacity-constrained Voronoi tessellations in discrete spaces. The algorithm starts with an arbitrary assignment of the points in the discrete space to the sites that fulfills the capacity constraint. This assignment is then optimized until it achieves an equilibrium state that represents a valid Voronoi tessellation. The convergence of the algorithm to such an equilibrium state is guaranteed.

Based on our algorithms for continuous and discrete spaces, we presented two distinct applications for capacity-constrained Voronoi tessellations. The first application, presented in Chapter 5, are Voronoi treemaps for visualizing attributed hierarchies in the domain of information visualization. The primary interest in this application are the regions of the tessellations. In contrast to existent treemap layout algorithms that are based on rectangles, Voronoi treemap layouts generate polygonal subdivisions. These layouts offer low aspect ratios of the nodes and a clear representation of the hierarchical structure. Although their computation is very expensive in comparison with other algorithms, Voronoi treemaps have been utilized for various static and interactive visualizations since their initial presentation. The second application, presented in Chapter 6, are capacity-constraint point distributions that are used for sampling-related tasks in the domain of computer graphics. The primary interest in this application are not the Voronoi regions, but the site distributions that result from the combination of a constraint of equal capac-

ity and centroidal sites. The results are uniform non-regular point distributions that have high-quality blue noise characteristics, and precisely adapt to given density functions. Our method improves on the established Lloyd's method, and can be utilized in most applications that currently benefit from Lloyd's method. Point distributions are often generated as a pre-processing step, and in these situations the additional computation effort compared to those of Lloyd's method is not a significant drawback. We want to emphasize that the capacity constraint is the key concept for the flexibility and unique quality of the presented applications. By focusing on the Voronoi region areas of the tessellation, we are able to directly control the spatial influence of the individual sites in the tessellation. This control enables optimization schemes that make additional heuristics or necessary limitations of previous methods obsolete. Whether the associated increased computational effort is reasonable depends on the intended application.

Open questions or possible further improvements in the scope of this thesis, and of capacity-constrained Voronoi tessellations in general, include the following topics:

- The convergence of our continuous space algorithms is not guaranteed, even though our experiments showed their reliable convergence. A formal proof of this convergence is still an open problem.

- Improvements of the computational complexity and the overall runtime of our algorithms. Especially our algorithm for ordinary distance functions in continuous space could be improved by a locally restricted evaluation of the optimized function. Additional multi-resolution schemes could improve the runtime of all algorithms.

- Implementation of additional distance functions and of other entities than points as sites, which includes the analysis of the resulting tessellations and the evaluation of their impact on our algorithms.

- Utilization of capacity-constrained Voronoi tessellations for other applications, especially in the domain of information visualization, in which the precise representation of region areas promises to enable interesting visualization approaches.

Bibliography

[1] Franz Aurenhammer. Power diagrams: Properties, algorithms and applications. *SIAM Journal on Computing*, 16(1):87–96, February 1987.

[2] Franz Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, September 1991.

[3] Franz Aurenhammer, Friedrich Hoffmann, and Boris Aronov. Minkowski-type theorems and least-squares clustering. *Algorithmica*, 20(1):61–76, January 1998.

[4] Michael Balzer. Hierarchie-basierte 3D Visualisierung großer Softwarestrukturen. In *Virtuelle und Erweiterte Realität, 1. Workshop der GI-Fachgruppe VR/AR*, pages 291–299, Chemnitz, Germany, September 2004. Shaker Verlag.

[5] Michael Balzer. Capacity-constrained Voronoi diagrams in continuous spaces. Proceedings of the 6th Annual International Symposium on Voronoi Diagrams in Science and Engineering, June 2009.

[6] Michael Balzer and Oliver Deussen. Hierarchy based 3D visualization of large software structures. In *Poster Compendium of the IEEE Visualization and the IEEE Symposium on Information Visualization*, pages 81–82, Austin, TX, USA, October 2004. IEEE Computer Society.

[7] Michael Balzer and Oliver Deussen. Exploring relations within software systems using treemap enhanced hierarchical graphs. In *Proceedings of the 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 89–94, Budapest, Hungary, September 2005. IEEE Computer Society.

[8] Michael Balzer and Oliver Deussen. Voronoi treemaps. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 49–56, Minneapolis, MN, USA, October 2005. IEEE Computer Society.

[9] Michael Balzer and Oliver Deussen. Level-of-detail visualization of clustered graph layouts. In *Proceedings of the 6th Asia-Pacific Symposium on Visualisation*, pages 133–140, Sydney, Australia, February 2007. IEEE Computer Society.

[10] Michael Balzer, Oliver Deussen, and Claus Lewerentz. Voronoi treemaps for the visualization of software metrics. In *Proceedings of the ACM Symposium on Software Visualization*, pages 165–172, St. Louis, MO, USA, May 2005. ACM Press.

[11] Michael Balzer and Daniel Heck. Capacity-constrained Voronoi diagrams in finite spaces. In *Proceedings of the 5th Annual International Symposium on Voronoi Diagrams in Science and Engineering*, number 4(2) in Voronoï's Impact on Modern Science, pages 44–56, Kiev, Ukraine, September 2008.

[12] Michael Balzer, Andreas Noack, Oliver Deussen, and Claus Lewerentz. Software landscapes: Visualizing the structure of large software systems. In *Proceedings of the Joint Eurographics and IEEE TVCG Symposium on Visualization*, pages 261–266, Konstanz, Germany, May 2004. Eurographics Association.

[13] Michael Balzer, Thomas Schlömer, and Oliver Deussen. Capacity-constrained point distributions: A variant of Lloyd's method. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2009)*, 28(3):86:1–8, 2009.

[14] George W. Barlow. Hexagonal territories. *Animal Behavior*, 22:876–878, 1974.

[15] Benjamin B. Bederson. Photomesa: A zoomable image browser using quantum treemaps and bubblemaps. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 71–80, Orlando, FL, USA, November 2001. ACM Press.

[16] Benjamin B. Bederson, Ben Shneiderman, and Martin Wattenberg. Ordered and quantum treemaps: Making effective use of 2d space to display hierarchies. *ACM Transactions on Graphics*, 21(4):833–854, October 2002.

[17] Joachim Böttger, Michael Balzer, and Oliver Deussen. Complex logarithmic views for small details in large contexts. *IEEE Transactions on Visualization and Computer Graphics: IEEE Visualization Conference and IEEE Symposium on Information Visualization Proceedings*, 12(5):845–852, September/October 2006.

[18] Joachim Böttger, Martin Preiser, Michael Balzer, and Oliver Deussen. Detail-in-context visualization for satellite imagery. In *Proceedings of Eurographics*, pages 587–596, Crete, Greece, April 2008. Eurographics Association.

[19] Mark Bruls, Kees Huizing, and Jarke J. van Wijk. Squarified treemaps. In *Proceedings of the Joint Eurographics and IEEE TVCG Symposium on Visualization*, pages 33–42, Amsterdam, The Netherlands, May 2000. IEEE Computer Society.

[20] CGAL. Computational Geometry Algorithms Library. `http://www.cgal.org`.

[21] Long Chen. Mesh smoothing schemes based on optimal Delaunay triangulations. In *Proceedings of the 13th International Meshing Roundtable*, pages 109–120, Williamsburg, VA, USA, September 2004.

[22] Robert L. Cook. Stochastic sampling in computer graphics. In *Computer Graphics (Proceedings of SIGGRAPH 86)*, volume 5, pages 51–72, Dallas, August 1986. ACM Press.

[23] Amanda Cox. All of inflation's little parts. *The New York Times*, page 7, May 8th, 2008. Interactive version at `http://www.nytimes.com/interactive/2008/05/03/business/20080403_SPENDING_GRAPHIC.html`.

[24] Boris N. Delaunay. Sur la sphére vide. A la memoire de Georges Voronoï. *Izvestia Akademia Nauk SSSR, Otdelenie Matematicheskih i Estestvennyh Nauk*, 7:793–800, 1934.

[25] René Descartes. *Principia Philosophiae*. Ludovicus Elzevirius, 1644.

[26] Mark A. Z. Dippé and Erling Henry Wold. Antialiasing through stochastic sampling. In *Computer Graphics (Proceedings of SIGGRAPH 85)*, volume 19, pages 69–78, San Francisco, CA, USA, July 1985. ACM Press.

[27] Johann P. G. L. Dirichlet. Über die Reduktion der positiven quadratischen Formen mit drei unbestimmten Zahlen. *Journal für die reine und angewandte Mathematik*, 40:209–227, 1850.

[28] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal Voronoi tessellations: Applications and algorithms. *SIAM Review*, 41(4):637–676, December 1999.

[29] Quiang Du and Maria Emelianenko. Acceleration schemes for computing centroidal Voronoi tessellations. *Numerical Linear Algebra with Applications*, 13(2–3):173–192, March–April 2006.

[30] Daniel Dunbar and Greg Humphreys. A spatial data structure for fast Poisson-disk sample generation. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006)*, volume 25, pages 503–508, Boston, MA, USA, August 2006. ACM Press.

[31] David Eppstein. The farthest point Delaunay triangulation minimizes angles. *Computational Geometry Theory & Applications*, 1(3):143–148, March 1992.

[32] Expertise Group Visualization, Technische Universiteit Eindhoven. SequoiaView. `http://w3.win.tue.nl/nl/onderzoek/onderzoek_informatica/visualization/sequoiaview/`, 2000.

[33] Jean-Daniel Fekete and Catherine Plaisant. Interactive information visualization of a million items. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 117–124, Boston, MA, USA, October 2002. IEEE Computer Society.

[34] Paul M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6):381–391, June 1954.

[35] John H. Halton. A retrospective and perspective survey of the Monte Carlo method. *SIAM Review*, 12(1):1–63, January 1970.

[36] Masami Hasegawa and Masaharu Tanemura. On the pattern of space division by territories. *Annals of the Institute of Statistical Mathematics*, 28(1):509–519, December 1976.

[37] Armin Herzer. Chain geometries. In Francis Buekenhout, editor, *Handbook of Incidence Geometry: Buildings and Foundations*, pages 781–842. Elsevier, March 1995.

[38] Stefan Hiller, Oliver Deussen, and Alexander Keller. Tiled blue noise samples. In *Proceedings of the Vision Modeling and Visualization Conference*, pages 265–272, Stuttgart, Germany, November 2001. IOS Press.

[39] Hiroshi Imai and Mary Inaba. Geometric clustering with applications. *Zeitschrift für Angewandte Mathematik und Mechanik*, 76:183–186, 1996.

[40] Brian Johnson and Ben Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *Proceedings of the 2nd International IEEE Visualization Conference*, pages 284–291, San Diego, CA, USA, October 1991. IEEE Computer Society.

[41] Thouis R. Jones. Efficient generation of Poisson-disk sampling patterns. *Journal of Graphics Tools*, 11(2):27–36, March 2006.

[42] Rolf Klein. *Concrete and Abstract Voronoi Diagrams*, volume 400 of *Lecture Notes in Computer Science*. Springer, 1989.

[43] Ales Klemencic, Franjo Pernus, and Stanislav Kovacic. Segmentation of muscle fibre images using Voronoi diagrams and active contour models. In *Proceedings of the International Conference on Pattern Recognition*, volume 3, pages 538–542, Washington, DC, USA, August 1996. IEEE Computer Society.

[44] Thomas Kollig and Alexander Keller. Efficient illumination by high dynamic range images. In *Proceedings of the 14th Eurographics Workshop on Rendering*, volume 44, pages 45–50, Leuven, Belgium, June 2003. Eurographics Association.

[45] Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. Recursive Wang tiles for real-time blue noise. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006)*, volume 25, pages 509–518, Boston, MA, USA, August 2006. ACM Press.

[46] Lauwerens Kuipers and Harald Niederreiter. *Uniform Distribution of Sequences*. John Wiley and Sons, 1974.

[47] Ares Lagae and Philip Dutré. An alternative for Wang tiles: Colored edges versus colored corners. *ACM Transactions on Graphics*, 25(4):1442–1459, October 2006.

[48] Ares Lagae and Philip Dutré. A comparison of methods for generating Poisson disk distributions. *Computer Graphics Forum*, 27(1):114–129, March 2008.

[49] Yang Liu, Wenping Wang, Bruno Lévy, Feng Sun, Dong-Ming Yan, Lin Lu, and Chenglei Yang. On centroidal Voronoi tessellation — energy smoothness and fast computation. Technical report, Hong-Kong University and INRIA-ALICE Project Team, 2008.

[50] Stuart P. Lloyd. Least square quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.

[51] Thomas Luft and Michael Balzer. Expressive illumination of foliage based on implicit surfaces. Konstanzer Schriften in Mathematik und Informatik No. 220, University of Konstanz, Germany, December 2006.

[52] Thomas Luft, Michael Balzer, and Oliver Deussen. Expressive illumination of foliage based on implicit surfaces. In *Proceedings of the Eurographics Workshop on Natural Phenomena*, pages 71–78, Prague, Czech Republic, September 2007. Eurographics Association.

[53] Michael McCool and Eugene Fiume. Hierarchical Poisson disk sampling distributions. *Graphics Interface '92*, pages 94–105, May 1992.

[54] Tomáš Mikolov. Color reduction using k-means clustering. In *Proceedings of the Spring Conference on Computer Graphics*, pages 109–120, Wien, Austria, April 2007.

[55] Don P. Mitchell. Generating antialiased images at low sampling densities. In *Computer Graphics (Proceedings of SIGGRAPH 87)*, volume 21, pages 65–72, Anaheim, CA, USA, July 1987. ACM Press.

[56] John A. Nelder and Roger Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, January 1965.

[57] Takashi Ohyama. Division of a region into equal areas using additively weighted power diagrams. In *Proceedings of the 4th International Symposium on Voronoi Diagrams in Science and Engineering*, pages 152–158, Glamorgan, Wales, UK, July 2007. IEEE Computer Society.

[58] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley and Sons Ltd., 2nd edition, May 2000.

[59] Krzysztof Onak and Anastasios Sidiropoulos. Circular partitions with applications to visualization and embeddings. In *Proceedings of the 24th Annual Symposium on Computational Geometry*, pages 28–37, College Park, MD, USA, June 2008. ACM Press.

[60] Victor Ostromoukhov. Sampling with polyominoes. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, volume 26, pages 78(1)–78(6), San Diego, CA, USA, August 2007. ACM Press.

[61] Victor Ostromoukhov, Charles Donohue, and Pierre-Marc Jodoin. Fast hierarchical importance sampling with blue noise properties. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2004)*, volume 23, pages 488–495, Los Angeles, CA, USA, August 2004. ACM Press.

[62] Panopticon Software Inc. Interactive information graphics for the exhibition 'Vårsalongen 2006' at Liljevalchs konsthall, Stockholm, Sweden. `http://demo.panopticon.com/liljevalchs/`, 2006.

[63] Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2004.

[64] René Reitsma, Stanislav Trubin, and Eric Mortensen. Weight-proportional space partitioning using adaptive Voronoi diagrams. *Geoinformatica*, 11(3):383–405, September 2007.

[65] Dmitri G. Roussinov and Hsinchun Chen. A scalable self-organizing map algorithm for textual classification: A neural network approach to thesaurus generation. *Communication and Cognition – Artificial Intelligence*, 15(1–2):81–111, Spring 1998.

[66] Nobuhiko Saitô. Asymptotic regular pattern of epidermal cells in mammalian skin. *Journal of Theoretical Biology*, 95(3):591–599, April 1982.

[67] Scientists' Discovery Room, Harvard University. Involv: Exploring life science with multi-touch. `http://iic.harvard.edu/research/scientists-discovery-room-lab-sdr-lab`, 2008.

[68] Adrian Secord. Weighted Voronoi stippling. In *Proceedings of the Second International Symposium on Non-photorealistic Animation and Rendering*, pages 37–43, Annecy, France, June 2002. ACM Press.

[69] Ben Shneiderman and Martin Wattenberg. Ordered treemap layouts. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 73–78, San Diego, CA, USA, October 2001. IEEE Computer Society.

[70] SmartMoney. Map of the Market. `http://www.smartmoney.com/map-of-the-market/`, 1998.

[71] Vitaly Surazhsky, Pierre Alliez, and Craig Gotsman. Isotropic remeshing of surfaces: A local parameterization approach. In *Proceedings of the 12th International Meshing Roundtable*, pages 215–224, Santa Fe, NM, USA, September 2003.

[72] Peter G. Szabó, Mihaly Cs. Markót, Tibor. Csendes, Eckard Specht, Leocadio G. Casado, and Inmaculada García. *New Approaches to Circle Packing in a Square*, volume 6 of *Springer Optimization and Its Applications*. Springer, March 2007.

[73] David Turo and Brian Johnson. Improving the visualization of hierarchies with treemaps: Design issues and experimentation. In *Proceedings of the 3rd Conference on Visualization*, pages 124–131, Boston, MA, USA, October 1992. IEEE Computer Society.

[74] Robert Ulichney. *Digital Halftoning*. MIT Press, 1987.

[75] Jarke J. van Wijk and Huub van de Wetering. Cushion treemaps: Visualization of hierarchical information. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 73–78, San Francisco, CA, USA, October 1999. IEEE Computer Society.

[76] Frédéric Vernier and Laurence Nigay. Modifiable treemaps containing variable-shaped units. In *Extended Abstracts of the IEEE Symposium on Information Visualization*, pages 28–35, Salt Lake City, UT, USA, October 2000. IEEE Computer Society.

[77] Georges Voronoï. Nouvelles applications des paramètres continus à théorie des formes quadratiques. Premier mémoire. Sur quelques propriétés des formes quadratiques positives parfaites. *Journal für die reine und angewandte Mathematik*, 133:97–178, 1908.

[78] Georges Voronoï. Nouvelles applications des paramètres continus à théorie des formes quadratiques. Deuxième Mémoire. Recherches sur les parallélоèdres primitifs. *Journal für die reine und angewandte Mathematik*, 136:67–181, 1909.

[79] Martin Wattenberg. Visualizing the stock market. In *CHI '99 Extended Abstracts on Human Factors in Computing Systems*, pages 188–189, Pittsburgh, PA, USA, May 1999. ACM Press.

[80] Li-Yi Wei. Parallel Poisson disk sampling. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)*, volume 27, pages 1–9, Los Angeles, CA, USA, August 2008. ACM Press.

[81] Marcos Weskamp. Newsmap. `http://www.marumushi.com/apps/newsmap/index.cfm`, 2004.

[82] Kenric White, David Cline, and Parris Egbert. Poisson disk point sets by hierarchical dart throwing. In *IEEE Symposium on Interactive Ray Tracing*, pages 129–132, Ulm, Germany, September 2007. IEEE Computer Society.

[83] John I. Yellott, Jr. Spectral consequences of photoreceptor sampling in the rhesus retina. *Science*, 12(1):382–385, September 1983.