

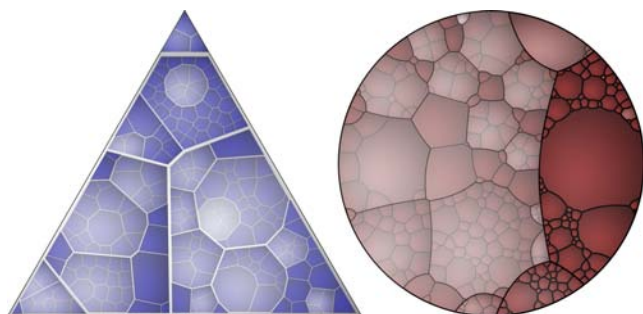
Voronoi Treemaps

Michael Balzer

Oliver Deussen

Department of Computer and Information Science
University of Konstanz, Germany

ABSTRACT



Treemaps are a well-known method for the visualization of attributed hierarchical data. Previously proposed Treemap layout algorithms are limited to rectangular shapes, which causes problems with the aspect ratio of the rectangles as well as with identifying the visualized hierarchical structure. The approach of Voronoi Treemaps presented in this paper eliminates these problems through enabling subdivisions of and in polygons. Additionally, this allows for creating Treemap visualizations within areas of arbitrary shape, such as triangles and circles, thereby enabling a more flexible adaptation of Treemaps for a wider range of applications.

CR Categories: H.5.2 [Information Interfaces and Presentation]: User Interfaces; I.3.6 [Methodology and Techniques]: Interaction Techniques; I.3.8 [Computer Graphics]: Applications

Keywords: Voronoi Treemaps, Information Visualization, Hierarchies, Trees, Treemaps, Voronoi Tessellations

1 INTRODUCTION

Hierarchical structures are an often used abstraction for the classification, sorting, and organization of a broad variety of data. Small structures can be efficiently represented by graphs, whereas for the visualization of large information spaces the approach of Treemaps by Shneiderman and Johnson [11] is an approved method. The idea is the space-filling recursive subdivision of a given area without producing holes or overlappings, whereby area sizes correspond to given attributes in the data set. Their wide variety of applications ranges from family trees and organization structures over file systems and software structures [2] to financial analysis [13, 26, 27], sports reporting [10], and many other areas.

Since the first presentation of Treemaps in 1991, many different layout algorithms have been introduced. The main characteristic of all existing Treemap layout algorithms is that they are based on the subdivision in rectangles. In this paper, Treemaps based on the subdivision in arbitrary polygons are presented. These have advantages concerning the aspect ratio of the subareas and the interpretability of the hierarchical structure. Due to the recursive pattern

of Treemaps, this method is not limited to layouts within rectangles, but rather enables Treemap layouts within circles, triangles or other polygonal shapes. This allows for a more flexible adaptation of Treemaps within a wider range of applications, e.g. the combination of Treemap layouts with other visualization techniques.

The basic idea for the generation of polygonal Treemap layouts is the utilization of centroidal Voronoi tessellations [5]. This method is widely-used for energy minimizations in many domains of application, for example data compression, image processing, mesh refinement, resource planning, scientific visualization [6]. In the domain of information visualization, the underlying Voronoi tessellation is commonly used [20, 22, 18, 9], whereas the enhancement to centroidal Voronoi tessellations is uncommon.

In the following Section 2, fundamental aspects of Treemaps and corresponding layout algorithms are discussed. Section 3 introduces the concept of Voronoi Treemaps by explaining the theory of Voronoi tessellations, their utilization for Treemaps, and the resulting layout algorithm. A discussion of the achieved results is given in Section 4.

2 BACKGROUND

The common approach for visualizing hierarchical structures, so called trees, are graphs. They consist of nodes representing the objects in the hierarchy, and edges that illustrate the containment. This method is ideal for small hierarchies, but not appropriate for hierarchical structures with hundreds or thousands of nodes. The main reason for this is the under-utilization of the available display area. Unlike graphs, Treemaps [11] remedy this problem by subdividing the given rectangular display area according to an attributed hierarchy without producing holes or overlappings. Thereby the term ‘attributed’ signifies that each node in the hierarchy has a value which represents its size relating to a given measurement.

The construction of Treemaps is exemplified in Figure 1. Each node in the hierarchy has a name and an associated size, whereas the size of an internal node is the sum of the sizes of its contained leaf nodes. The Treemap is constructed via recursive subdivision of the initial rectangle. The direction of each one-dimensional subdivision step alternates per level: first horizontally, next vertically, again horizontally, etc. The area size of each sub-rectangle corresponds to the size of the represented node. As a result of its construction, the Treemap reflects the structure of the tree and the sizes of its nodes. This original Treemap layout is called Slice-and-Dice.

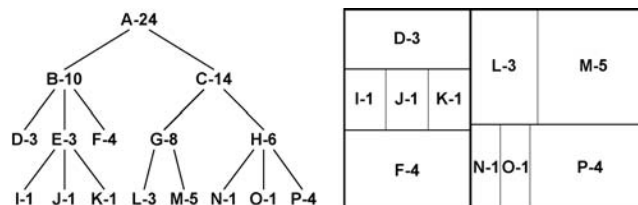


Figure 1: Tree and corresponding Treemap—each node is labeled with its name and size; the area sizes in the Treemap correspond to the node sizes

A substantial problem of this initial layout algorithm is the restriction of subdividing the plane in each step solely in one dimension. As a 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 [7, 23, 3]. Figure 2 presents an example of a real-world data set containing 698 nodes at 5 hierarchy levels. The left image visualizes this hierarchy with nodes of different sizes, whereby small nodes are hard to recognize. The same hierarchy with nodes of equal size is presented in the right image, illustrating the divergence of actual and perceived node sizes in Slice-and-Dice layouts. The reason for both problems is unerringly the high aspect ratio of the rectangles.

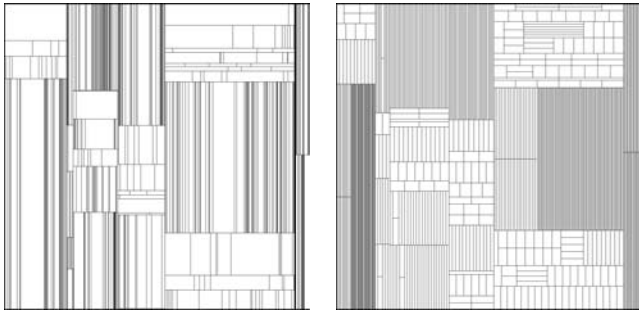


Figure 2: Slice-and-Dice Treemap layouts of 698 nodes at 5 hierarchy levels with nodes of different sizes (left) and nodes of equal size (right)—the high aspect ratio between width and height causes the hard recognition of small nodes and the bad perception of node sizes

Consequently, in the further developments of Treemap layout algorithms mainly the issue of the high aspect ratio was addressed, for instance in Clustered Treemaps [27], Squarified Treemaps [3], Ordered Treemaps [21], and Modifiable Treemaps [25]. 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 sub-rectangles to the shape of a square, so that the overall aspect ratio between width and height of the sub-rectangles is minimized. Aside from the aspect ratio criterion, other criteria have also been considered. For example, the order of nodes or the nearness of two or more nodes, whereby these other criteria are mostly related to the respective application domain. A demonstrative member of this group of advanced Treemap layout algorithms is presented in Figure 3, visualizing the same data set as in Figure 2 with a Squarified Treemap layout. Similarly, nodes of different sizes are shown in the left image and nodes of equal size in the right image. Obviously, this layout algorithm maintains a much better aspect ratio, resulting in a better separation of the nodes and perception of their sizes.

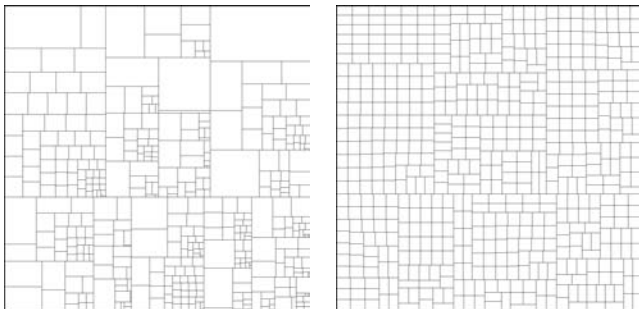


Figure 3: Squarified Treemap layouts of the same data set as in Figure 2—the interpretation of the hierarchical structure is ambiguous and often edges seemingly run into each other

Disadvantageous in the mentioned sophisticated algorithms is the unclear representation of the hierarchical structure, whereas it is preserved by Slice-and-Dice Treemap layouts. For example, Figure 2 clearly represents a hierarchy with a root node containing six child nodes, which is conform to the data set. The indication therefor is the alternating horizontal and vertical subdivision. In contrast, this observation can not be made in Figure 3. Here the number of child nodes of the root node is ambiguous. A variant is a root node with two child nodes *A* and *B*, whereby *A* contains two and *B* four of the original six child nodes. Aside from this example, more delude interpretations are possible. 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. For example, in the Squarified Treemap layout shown in Figure 3 the two largest nodes are vertically divided from the other four smaller nodes due to the functioning of the algorithm. This ‘internal’ division step may be interpreted as branching in the hierarchical structure.

Another problem of the existent algorithms are edges that seemingly run into each other, whereby a distinction of nodes between and within different hierarchy levels becomes even more difficult. The reason is the still limited degree of freedom in each subdivision step. By aligning each edge only horizontally or vertically, the probability that the end point of an edge is near the starting point of another equally aligned edge is quite 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 visualization.

The representation of the hierarchical structure is a crucial requirement to Treemaps. Already Shneiderman and Johnson enhanced their initial Treemap layouts by assigning borders in every subdivision step, called Nested Treemaps [11]. The approach of Cushion Treemaps [24] employs shading to improve the perception of structure by feigning specular reflection and thereby simulating a curved surface. This method is also adapted for the enhancement of Nested Treemaps to Framed Treemaps [3], resulting in quasi-three-dimensional borders. Another option is to use transparencies [2] to hide uninteresting or unwanted parts of the hierarchy. These layout-independent methods can be applied to the existing Treemap algorithms and may reduce, but do not prevent, misinterpretations concerning the hierarchical structure.

So far, all existent Treemap layout algorithms have one thing in common: they are based on and are thereby restricted to axis-aligned rectangles. A slight exception are ET-Maps [19] that generate shapes composed of axis-aligned rectangles, but their general appearance is identical. 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, this 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. Although, these shapes may not be necessary if Treemap visualizations are used independently. However, by embedding Treemap layouts within more complex visualizations, a better adaptability is quite useful or even necessary.

Hence, the approach of Voronoi Treemaps is presented, enabling a polygon-based two-dimensional subdivision following the Treemap paradigm. It offers low aspect ratios, better interpretability of hierarchical structures, and flexible adaptability regarding the enclosing shape.

3 VORONOI TREEMAPS

Global application-independent constraints and optimization criteria for a Treemap layout regarding the shape of the Treemap subareas are:

1. *Constraint:* The division in subareas must fully utilize the given overall area, thereby avoiding holes and overlappings.
2. *Optimization criterion:* Subareas should have an overall aspect ratio between width and height that converges to one.
3. *Optimization criterion:* Siblings in the hierarchy should not be grouped during the layout process, thereby making the identification of the hierarchical structure non-ambiguous.
4. *Optimization criterion:* Subareas of the Treemap should have non-regular shape, so that edges between the subareas do not seemingly run into each other.

Obviously, polygons can be used for the division of a given area into subareas. A polygon is defined as a closed plane figure with n sides. Each polygon can be subdivided into smaller polygons, hereby satisfying the constraint number 1. Polygons can have arbitrary shapes, and polygons with many sides can approximate curves. Both refer to the optimization criteria number 2, 3, and 4.

The principle structure of the layout algorithm is similar to the original Treemap layout algorithm. The first step is to create a polygonal subdivision of the given display area according to the top hierarchy level. The output is a set of polygons representing the nodes of the top hierarchy level. For the next hierarchy level, this procedure is performed recursively for all top level nodes within the respective polygons. When the recursion ends, a complete Treemap layout is obtained.

The basic concept for generating polygonal subdivisions is to utilize Voronoi tessellations. They enable an iteratively computation of good layouts in accordance to the given constraint and optimization criteria. Finding an optimal solution is a NP-complete problem—even for layouts based on axis-aligned rectangles. However, a good approximation is sufficient for Treemaps. To clearly convey the idea of Voronoi Treemaps, necessary theoretical knowledge of Voronoi tessellations is given in the subsequent Section 3.1. The layout algorithm for computing Voronoi Treemaps is explained in detail in Section 3.2.

3.1 Voronoi Tessellations

Voronoi tessellations enable the partitioning of a m -dimensional space without producing holes or overlappings. For an easier understanding of their underlying theory, the explanations in this section are restricted to relevant aspects regarding their application for Treemap layouts. Here only *planar Voronoi tessellations* in the two-dimensional Euclidian space are considered. All definitions and denotations are according to [17, 5].

Basic Properties: Let $P := \{p_1, \dots, p_n\}$ be a set of n distinct points in \mathbb{R}^2 with the coordinates $(x_1, y_1), \dots, (x_n, y_n)$. These points are the *generators*. The subdivision of \mathbb{R}^2 into n *Voronoi regions* $V(p_i)$, with the property that a point $q(x, y)$ lies in the region $V(p_i)$ if and only if $distance(p_i, q) < distance(p_j, q)$ for each $p_i, p_j \in P$ with $i \neq j$, is defined as the *Voronoi tessellation* $\mathcal{V}(P) := \{V(p_1), \dots, V(p_n)\}$. The denotation $distance(p_i, q)$ represents a specified distance function between the generator p_i and the point q . In general, a Voronoi tessellation is defined in an unbounded space. Having a bounded space S , the set $\mathcal{V}_{\cap S}(P) := \{V(p_1) \cap S, \dots, V(p_n) \cap S\}$ is called a *bounded Voronoi tessellation*

of P by S . An *ordinary Voronoi tessellation* $\mathcal{V}_\epsilon(P)$ is a Voronoi tessellation using the Euclidian metric, defined by

$$distance_\epsilon(p_i, q) := \|p_i - q\| = \sqrt{(x_i - x)^2 + (y_i - y)^2}, \quad (1)$$

as their distance function. The bisector of two regions $V_\epsilon(p_i)$ and $V_\epsilon(p_j)$ of an ordinary Voronoi tessellations is the perpendicular bisector of the generators p_i and p_j . Examples for an unbounded and a bounded planar ordinary Voronoi tessellation are given in Figure 4.

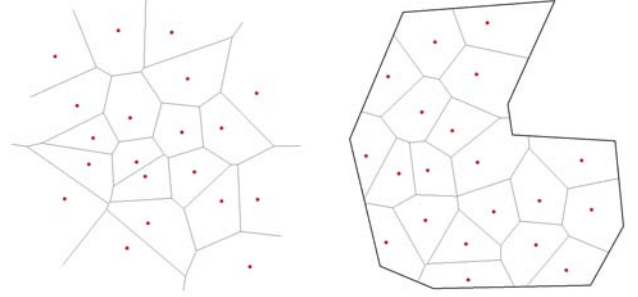


Figure 4: Unbounded and bounded planar ordinary Voronoi tessellation

Weighted Voronoi Tessellations: In the basic Voronoi tessellation $\mathcal{V}(P)$ it is implicitly assumed that each generator has the same weight. As an extension, a set of parameters W may be given, and to each generator $p_i \in P$ a parameter $w_i \in W$ is assigned. These parameters are the weights. By using weighted generators, it is possible to define weighted distance functions, generating *weighted Voronoi tessellations* $\mathcal{V}(P, W)$.

An *additively weighted Voronoi tessellation* $\mathcal{V}_{aw}(P, W)$, briefly the AW Voronoi tessellation, uses the following distance function between a generator $p_i \in P$ with its assigned weight $w_i \in W$ and a point q :

$$distance_{aw}(p_i, w_i, q) := \|p_i - q\| - w_i. \quad (2)$$

The bisector of two regions $V_{aw}(p_i, w_i)$ and $V_{aw}(p_j, w_j)$ of an AW Voronoi tessellations forms a hyperbolic curve with foci p_i and p_j . The left image in Figure 5 presents an example for an AW Voronoi tessellation.

The distance function for the *additively weighted power Voronoi tessellations* $\mathcal{V}_{pw}(P, W)$, briefly PW Voronoi tessellation, is:

$$distance_{pw}(p_i, w_i, q) := \|p_i - q\|^2 - w_i. \quad (3)$$

This distance function yields a bisector of two regions $V_{pw}(p_i, w_i)$ and $V_{pw}(p_j, w_j)$ that is a straight line. The bisector corresponds to the perpendicular bisector of p_i and p_j moved away from their midpoint depending on their weights w_i and w_j . The right image in Figure 5 presents an example for a PW Voronoi tessellation.

Both, the AW and the PW Voronoi tessellation, may be illustrated as Voronoi tessellations that are using circles as generators. This is obvious, if circles are considered as points with a size or weight parameter. Thereby, the weight w_i in the AW distance function represents directly the radius of the circle, whereas in the PW distance function, w_i is the square of the radius. To satisfy the continuity of AW Voronoi tessellations, the circles are not allowed to overlap. Additionally, the abstraction of circles with negative radii has to be made in order to achieve the complete spectrum of possible Voronoi tessellations for the AW and the PW distance function.

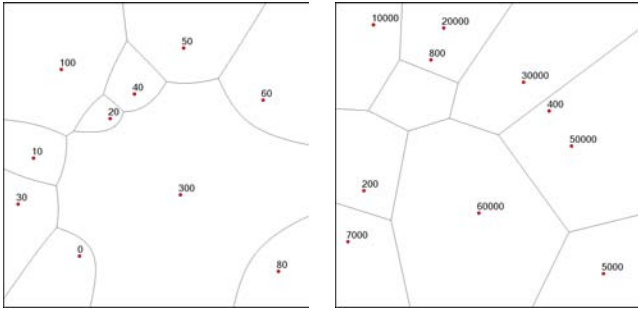


Figure 5: Weighted Voronoi tessellations using the AW and the PW distance function

Centroidal Voronoi Tessellations: The center of mass, or centroid, c_i of a Voronoi region $V(p_i)$ within the Euclidian space is calculated by $c_i = \int_{V(p_i)} x dx$. A *centroidal Voronoi tessellation*, briefly CVT, is a special Voronoi tessellation with the property that each generator p_i is itself the center of mass c_i of the corresponding Voronoi region $V(p_i)$. Obviously, there exist many different CVTs for a given number of generators.

The mathematical importance of the CVT is founded by its relationship to the energy function

$$\mathcal{K}(P, \mathcal{V}(P)) = \sum_i \int_{V(p_i)} \|x - p_i\|^2 dx. \quad (4)$$

It is proven that a necessary condition for $\mathcal{K}(P, \mathcal{V}(P))$ to be minimized is that $\mathcal{V}(P)$ is a CVT [5]. Since to find a CVT of a given number of generators with $\mathcal{K}(P, \mathcal{V}(P))$ in a global minimum is NP-complete [5], approximations of CVTs that are located in local minima of $\mathcal{K}(P, \mathcal{V}(P))$ are used. CVTs can be iteratively computed with the Lloyd's method [16]: By starting with an initial distribution of generators P within a bounded plane S , in each iteration step each generator $p_i \in P$ is moved into the center of mass c_i of its Voronoi region $V_{\cap S}(p_i) \in \mathcal{V}_{\cap S}(P)$. This iterative computation stops when the difference between each generator p_i and its corresponding center of mass c_i is below a chosen error threshold ϵ . Figure 6 exemplifies this method.

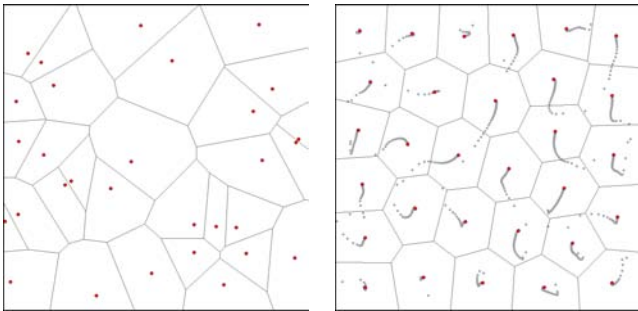


Figure 6: Voronoi tessellation of 20 random points and an associated CVT—traces illustrate the movements of the points during the computation of the CVT

The concept of CVTs can be generalized to non-Euclidian distance functions and spaces, and can also be extended to weighted Voronoi tessellations. The only prerequisite is that the existence of a however defined center of mass for each Voronoi region is guaranteed.

Computational Complexity and Algorithms: A lower bound of the worst-case time complexity for constructing the ordinary Voronoi tessellation $\mathcal{V}_\epsilon(P)$ of n generators is $O(n \log n)$. A

lower bound of the space complexity for computing $\mathcal{V}_\epsilon(P)$ is $O(n)$ in the worst case. Both lower bounds for $\mathcal{V}_\epsilon(P)$ are tight in the sense that actually there exist optimal algorithms having these lower bounds [17].

The computations of the AW Voronoi tessellation $\mathcal{V}_{aw}(P, W)$ and the PW Voronoi tessellation $\mathcal{V}_{pw}(P, W)$ have the same time and space complexity as $\mathcal{V}_\epsilon(P)$ in the worst case. Optimal algorithms are existent, for instance [8] for $\mathcal{V}_{aw}(P, W)$, and [1, 14, 15] for $\mathcal{V}_{pw}(P, W)$.

The time complexity of the iterative computation of an approximation of a CVT is $O(dO_\gamma)$ with d as the number of iterations and O_γ as the complexity of the used Voronoi tessellation. Thereby d is not directly related to the number of generators, but rather to the minimal approximation error that should be achieved, and it therefore does not affect the overall complexity. The complexity of the calculation of the centers of mass of a Voronoi region does also not affect the overall complexity, because it is only $O(n)$, and each Voronoi tessellation has at least a time complexity of $O(n)$. Thus, the overall time complexity for computing the approximation of a CVT is equal to the time complexity of the used Voronoi tessellation. Nevertheless, CVT algorithms are very time-consuming, since often many iterations are necessary for obtaining a good approximation. By taking advantage of the fact that it is not necessary to calculate the Voronoi tessellations itself for obtaining a CVT, it is possible to use distributed computing environments for their computation [12]. The idea is to calculate only the centers of mass in each iteration step by using large random sets of discrete sample points. This method achieves a nearly perfect linear speed-up with the number of processors used, and has the same time complexity as the respective Voronoi tessellation. The space complexity of the iterative computation of an approximation of a CVT is $O(n)$.

3.2 Voronoi Treemap Algorithm

The principle structure of the algorithm is similar to the recursive structure of the original Treemap layout algorithm. The modification for Voronoi Treemaps is applied to the subdivision in each recursion step. Therefore the computation of the centroidal Voronoi tessellation is utilized.

CVTs enable the subdivision of a given area without producing holes and overlappings, which satisfies the constraint for Treemap layouts. CVTs minimize the overall energy of the Voronoi tessellation. The energy of the CVT is thereby equivalent to the overall aspect ratio of the subareas of the Treemap layout [5], which refers to optimization criterion number 2. In a Voronoi tessellation each generator is treated separately from the other generators. CVTs are special cases of Voronoi tessellations, and therefore also do not group their generators. This refers to the optimization criterion number 3. Non-degenerated vertices within Voronoi tessellations arise from three generators that have the same distance to a point, whereby each non-degenerated vertex is assigned to three edges. The energy minimization in CVTs simultaneously minimizes the number of degenerated vertices, and maximizes the distances between vertices and the angles between the edges of a vertex [5]. In combination with the general non-regular topology of CVTs, this refers to the optimization criterion number 4. Concluding it can be stated that CVTs are suitable for generating good Treemap layouts in reference to the given constraint and optimization criteria.

The essential characteristic of Treemaps is that their subarea sizes correspond to the sizes of the nodes in the hierarchy. In standard CVTs using the Euclidian distance function, the sizes of the Voronoi regions are roughly the same, which disqualifies them for the generation of Treemap layouts. CVTs with weighted distance functions have the property to produce divisions with variable sized subareas. The problem is that during the computation of a CVT with an arbitrarily defined distance function, the area sizes of the

Voronoi regions are not observed. For that reason, an extension of the standard CVT computation method is necessary in order to control the size of every Voronoi region. The idea is to adaptively alter the weight parameter of each generator in the next iteration step according to the size of the dedicated Voronoi region in the current iteration step. For example, if the size of a node in the hierarchy is 20% of the size of the parent node, and in the current iteration step the dedicated Voronoi region covers only 16% of the overall area, it is tried to increase the area size of this Voronoi region in the next iteration step from 16% to 20% by increasing the weight of the generator by 25%. Due to the facts that the relation between the weight of a generator and the area size of the dedicated Voronoi region is not a linear dependency, and the weight and positions of the other generators are changing at the same time, the correct area size is most likely not achieved in the next iteration step. Rather it is presumed that a better approximation is obtained. By performing these adjustments of the generator weights in each iteration step, the computation stops in a stable state, whereby the error between the designated relative size of each node in the hierarchy and the relative area size of the dedicated Voronoi region is below a chosen maximum error ε . This extended computation method for weighted CVTs for the generation of Treemap layouts is further outlined by means of Algorithm 1.

Algorithm 1 Voronoi Treemap subdivision

Input: bounded plane S in \mathbb{R}^2 ; set of n values $A_{desired} := \{a_{1_{desired}}, \dots, a_{n_{desired}}\}$ with $0 < a_{i_{desired}} \leq 1$ and $\sum a_{i_{desired}} = 1$; error threshold ε

Output: subdivision of S in n disjoint subareas $s_i \subset S$ with $\left| \frac{AreaSize(s_i)}{AreaSize(S)} - a_{i_{desired}} \right| < \varepsilon$

- 1: initialize a set of n points $P := \{p_1, \dots, p_n\}$ with $p_i \in S, p_i \neq p_j$
- 2: initialize a set of n weights $W := \{w_1, \dots, w_n\}$ with $w_i = 1$
- 3: initialize a data structure for the Voronoi tessellation $\mathcal{V}_{\cap S}(P, W)$
- 4: **repeat**
- 5: *ComputeVoronoiTessellation*($\mathcal{V}_{\cap S}(P, W)$)
- 6: *stable* = *true*
- 7: initialize a set of n values $A := \{a_1, \dots, a_n\}$
- 8: **for** each $a_i \in A$ **do**
- 9: $a_i = \frac{AreaSize(\mathcal{V}(p_i, w_i))}{AreaSize(S)}$ with Voronoi region $V(p_i, w_i) \in \mathcal{V}_{\cap S}(P, W)$
- 10: **if** $|a_i - a_{i_{desired}}| \geq \varepsilon$ **then**
- 11: *stable* = *false*
- 12: **end if**
- 13: **end for**
- 14: **for** each $w_i \in W$ **do**
- 15: *AdjustWeight*($w_i, a_i, a_{i_{desired}}$)
- 16: **end for**
- 17: *MoveGenerators*($P, W, \mathcal{V}_{\cap S}(P, W)$)
- 18: **until** *stable* == *true*
- 19: *ExtractSubareas*($\mathcal{V}_{\cap S}(P, W)$)

In general, arbitrary weighted distance functions may be used for the described subdivision algorithm. However for the application of Treemaps and the given optimization criteria, especially the additively weighted distance function—with restriction to distributions equivalent to non-overlapping circles—and the additively weighted power distance function are qualified. This is caused by the continuous topology of their resulting Voronoi regions. The difference between the AW and the PW Voronoi tessellation is the shape of the bisector of two regions. AW Voronoi tessellations create hyperbolic curves, and PW Voronoi tessellations create straight lines. Thus, it has to be distinguished between *AW Voronoi Treemaps* and *PW Voronoi Treemaps*. Additionally, the different characteristics of

the AW and PW distance functions necessitate a different handling of the auxiliary functions *AdjustWeight*() and *MoveGenerators*() in Algorithm 1.

During the computation of AW Voronoi Treemap subdivisions, it is necessary that the weight parameter w_i can have negative values. If not, it may be possible that the area size error of small regions will not get below the chosen error threshold ε . Furthermore, the special case has to be observed that the weight w_i in AW tessellations does not become zero or nearby zero. Therefore, it has to be checked against a very small value δ . In contrast, according to experience, the convergence of the algorithm for PW Voronoi Treemap subdivisions is improved, if each $w_i \geq 1$. The resulting algorithms for the *AdjustWeight*() function are outlined in Algorithm 2 for AW Voronoi Treemaps and in Algorithm 3 for PW Voronoi Treemaps.

Algorithm 2 *AdjustWeight*() for AW Voronoi Treemaps

Input: weight value w_i ; area size value a_i ; desired area size value $a_{i_{desired}} \neq 0$

Output: adjusted weight value w_i

- 1: $0 < \delta \lll 1$
- 2: **if** $|w_i| < \delta$ **then**
- 3: $w_i = \text{sign}(w_i) \cdot \delta$
- 4: **end if**
- 5: $w_i = w_i + |w_i| \cdot \frac{a_{i_{desired}} - a_i}{a_{i_{desired}}}$

Algorithm 3 *AdjustWeight*() for PW Voronoi Treemaps

Input: weight value w_i ; area size value a_i ; desired area size value $a_{i_{desired}} \neq 0$

Output: adjusted weight value w_i with $w_i \geq 1$

- 1: $w_i = w_i \cdot \left(1 + \frac{a_{i_{desired}} - a_i}{a_{i_{desired}}}\right)$
- 2: **if** $w_i < 1$ **then**
- 3: $w_i = 1$
- 4: **end if**

For granting continuous AW Voronoi regions during the subdivision algorithm, it is required that the distribution of generators in combination with the according weights is equivalent to a distribution of non-overlapping circles. This can be achieved by again adjusting the weights after the generators have been moved into the centers of mass of the dedicated Voronoi regions, while keeping the ratio between the weights. Thereby all weights are multiplied by a maximum factor, so that for each subset of generators $\{p_i, p_j\} \subset P$, with $i \neq j$, their Euclidian distance is not smaller than the sum of the assigned weights w_i and w_j . Algorithm 4 outlines the implementation of the function *MoveGenerators*() for AW Voronoi Treemaps. For PW Voronoi Treemaps, no additional conditions must be satisfied. Thus, the function *MoveGenerators*() for PW Voronoi Treemaps is reduced to moving each generator p_i into the center of mass of the dedicated Voronoi region $V_{pw}(p_i, w_i) \in \mathcal{V}_{pw \cap S}(P, W)$. The formal declaration of an algorithm is set aside.

For the standard CVT computation method with constant weights, it has been shown that the energy $\mathcal{K}_i(P, \mathcal{V}(P))$ in iteration i is lower than the energy $\mathcal{K}_{i-1}(P, \mathcal{V}(P))$ in iteration $i-1$, and therefore the computation stops in a local minimum [5]. This cannot be proven for the extended CVT computation method because of the permanently changing weights of the generators. Nevertheless, from experience it can be stated that in most cases this extended CVT computation method also stops in a local minimum. If not, this problem can be remedied by a re-initialization of the generators with new random positions. Aside from the energy of the

Algorithm 4 *MoveGenerators()* for AW Voronoi Treemaps

Input: set of n points $P := \{p_1, \dots, p_n\}$; set of n weights $W := \{w_1, \dots, w_n\}$; AW Voronoi tessellation $\mathcal{V}_{aw \cap S}(P, W)$

Output: set of n points $P := \{p_1, \dots, p_n\}$ with $p_i = \text{CenterOfMass}(V_{aw}(p_i, w_i))$ and $V_{aw}(p_i, w_i) \in \mathcal{V}_{aw \cap S}(P, W)$; set of n weights $W := \{w_1, \dots, w_n\}$ with $\|p_i - p_j\|^2 - (w_i + w_j) \geq 0$ for $\{p_i, p_j\} \subset P, i \neq j$

```
1: for each  $p_i \in P$  do
2:    $p_i = \text{CenterOfMass}(V_{aw}(p_i, w_i))$  with  $V_{aw}(p_i, w_i) \in \mathcal{V}_{aw \cap S}(P, W)$ 
3: end for
4:  $factorWeight = \infty$ 
5: for each  $\{p_i, p_j\} \subset P$  with  $i \neq j$  do
6:    $f = \frac{\|p_i - p_j\|^2}{w_i + w_j}$ 
7:   if  $0 < f < factorWeight$  then
8:      $factorWeight = f$ 
9:   end if
10: end for
11: if  $factorWeight < 1$  then
12:   for each  $w_i \in W$  do
13:      $w_i = w_i \cdot factorWeight$ 
14:   end for
15: end if
```

Voronoi tessellation, the area size error of each Voronoi region and the overall area size error is also reduced during the computation. Indeed, at the end of the computation, this error does not reside in a local minimum, but rather below a chosen threshold ε . Figure 7 illustrates a typical convergence of the maximum area size error of a Voronoi region and of the overall area size error of the Voronoi tessellation during the computation of a CVT with an AW and a PW distance function. The data set consisted of ten generators with different sizes. The area size error of each Voronoi region was less than 0.1% at iteration 185 for the AW distance function, and at iteration 206 for the PW distance function.

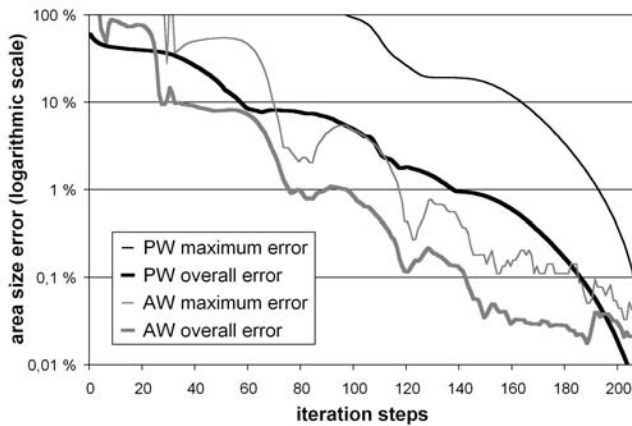


Figure 7: Typical convergence of the maximum area size error of a Voronoi region and of the overall area size error of the Voronoi tessellation during the computation of a CVT with an AW and a PW distance function

The entire AW and PW Voronoi Treemap layouts are generated by using the described computation of CVTs in the subdivision step of the Treemap recursion. Figure 8 presents the result for a layout with the AW distance function, and Figure 9 presents the result for a layout with the PW distance function. All four images use the data

set with 698 nodes at 5 hierarchy levels, as in the examples of Slice-and-Dice and Squarified Treemap layouts in Section 2. Again, the left images visualize the hierarchy with nodes of different sizes, and the right images with nodes of equal size.

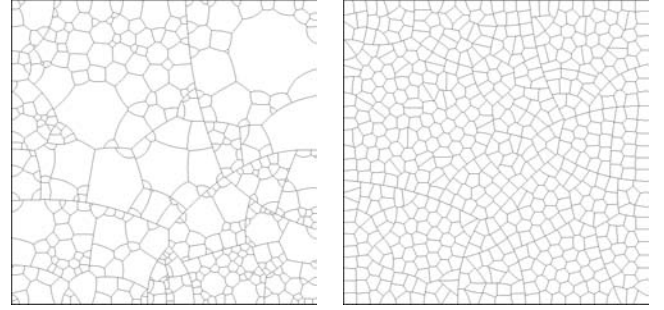


Figure 8: AW Voronoi Treemap layouts of the same data set as in Figure 2 with nodes of different sizes (left) and nodes of equal size (right)

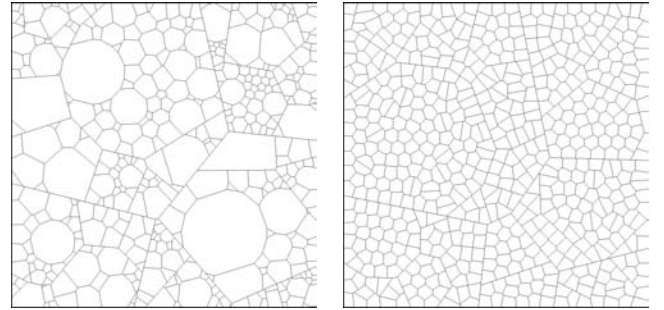


Figure 9: PW Voronoi Treemap layouts of the same data set as in Figure 2 with nodes of different sizes (left) and nodes of equal size (right)

Similarly to the other Treemap layout algorithms, enhancements like borders, adaptive edge sizes, cushions, coloring, etc., may also be applied to the described layout method. This additionally supports the user in the perception and interpretation of the Treemap visualization. Examples for such enhanced Voronoi Treemap layouts are presented in Figures 10–12.

4 DISCUSSION

This paper presents a new approach for the generation of Treemap layouts. Contrary to existent layout algorithms that are based on the subdivision in rectangles, this new layout algorithm enables the subdivision in arbitrary polygons. This also allows to create Treemap visualizations within areas of arbitrary shape, such as circles or triangles. The evidence for the suitability of the introduced layout method has been produced by the constraints and global application-independent optimization criteria for Treemaps. Especially, the minimization of the overall aspect ratio of the subareas is explicitly attributed to the proven energy minimization in centroidal Voronoi tessellations. This minimization does thereby not entail ambiguities of the interpretability of the hierarchical structure in the visualization, such as observed in other Treemap layout algorithms.

Because of the intricacy of the expanded degree of freedom for these layouts, the presented method iteratively computes an approximation of a layout with an error below a desired threshold. Depending on this threshold, the number of iterations required may

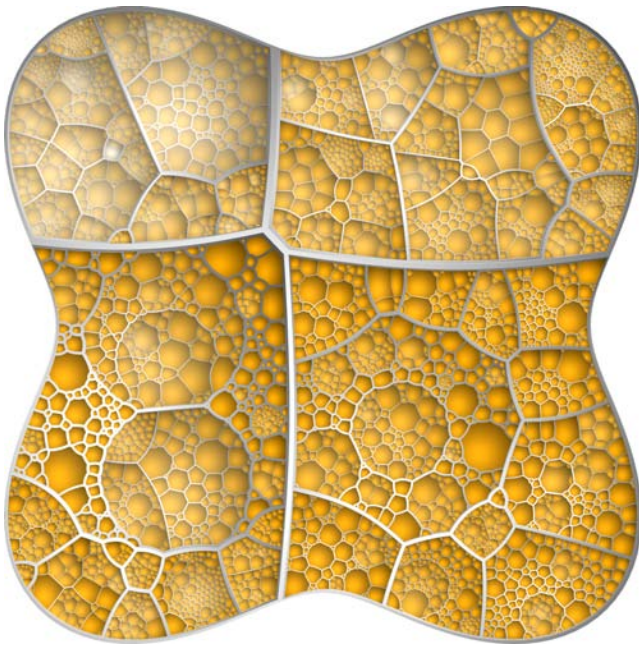


Figure 10: Enhanced AW Voronoi Treemap layout of 4075 nodes at 10 hierarchy levels (a brighter color indicates a lower hierarchy level)

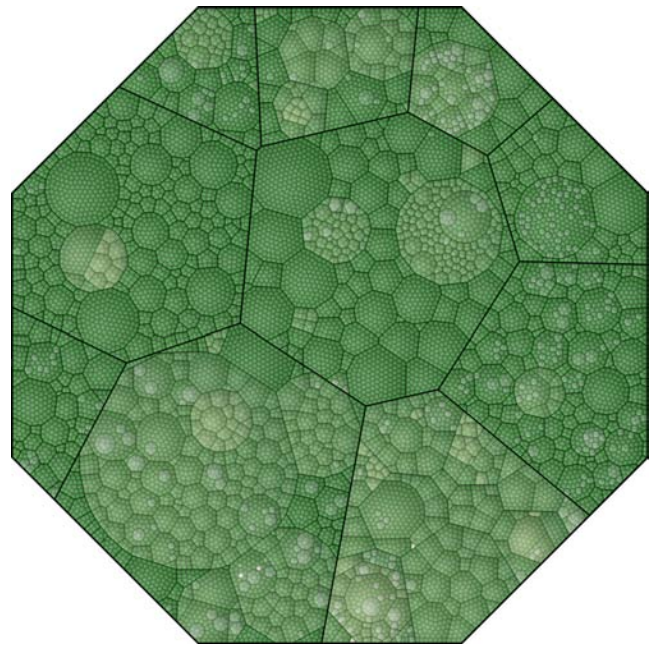


Figure 11: Enhanced PW Voronoi Treemap layout of 16288 nodes at 7 hierarchy levels (a brighter color indicates a lower hierarchy level)

become very large. Thus, with regard to computation time, other Treemap layout algorithms outperform this method by far. This problem is diminished by using distributed computing environments. The recursive structure of the Treemap algorithm and the ability to massively parallelize the computation of the CVTs, enable an almost perfect linear scalability with the number of processors used. In the existing prototype implementation of the layout algorithm, a variable number of compute servers is utilized to generate even large Treemap layouts within a reasonable time—using eight Intel Xeon CPUs each with 2.4 GHz, the computation of Figure 10 required 7:13 minutes, and that of Figure 11 required 5:48 minutes. Indeed, this method is not appropriate for real time calculation.

In future work, the properties and abilities of the presented layout method will be studied extensively. For example, the restriction of the movement of the generators, and the temporal coherence of the subareas, will be investigated. That will permit ordered layouts, and the visualization of time-variant data sets respectively. Also, adaptations of the presented method to other layout problems outside the Treemap domain will be addressed, such as the visualization of georeferenced statistical data.

REFERENCES

- [1] Franz Aurenhammer. Power diagrams: Properties, algorithms, and applications. *SIAM Journal on Computing*, 16(1):78–96, 1987.
- [2] Michael Balzer, Oliver Deussen, and Claus Lewerentz. Voronoi treemaps for the visualization of software structures. In *Proceedings of the ACM Symposium on Software Visualization*. ACM, 2005.
- [3] Mark Bruls, Kees Huizing, and Jarke J. van Wijk. Squarified treemaps. In *Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization*, pages 33–42. IEEE Computer Society, 2000.
- [4] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd rev. edition, 2000.
- [5] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal voronoi tessellations: Applications and algorithms. *SIAM Review*, 41(4):637–676, 1999.
- [6] Qiang Du and Xiaoqiang Wang. Centroidal voronoi tessellation based algorithms for vector fields visualization and segmentation. In *Proceedings of the IEEE Visualization*, pages 43–50. IEEE Computer Society, 2004.
- [7] Paul M. Fitts. The information capacity of the human motor system in controlling the amplitude of movements. *Journal of Experimental Psychology*, 47(6):381–391, 1954.
- [8] Steven J. Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2:153–174, 1987.
- [9] Michael Granitzer, Wolfgang Kienreich, Vedran Sabol, Keith Andrews, and Werner Klieber. Evaluating a system for interactive exploration of large, hierarchically structured document repositories. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 127–134. IEEE Computer Society, 2004.
- [10] Liquin Jin and David C. Banks. Tennisviewer: A browser for competition trees. *IEEE Computer Graphics and Applications*, 17(4):63–65, 1997.
- [11] 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. IEEE Computer Society, 1991.
- [12] Lili Ju, Qiang Du, and Max Gunzburger. Probabilistic methods for centroidal voronoi tessellations and their parallel implementations. *Parallel Computing*, 28(10):1477–1500, 2002.
- [13] Walter-Alexander Jungmeister and David Turo. Adapting treemaps to stock portfolio visualization. Technical Report UMCP-CSD CS-TR-2996, University of Maryland, College Park, Maryland 20742, USA, 1992.
- [14] Deok-Soo Kim, Donguk Kim, and Kokichi Sugihara. Voronoi diagram of a circle set from voronoi diagram of a point set: I. Topology. *Computer Aided Geometric Design*, 18(6):541–562, 2001.
- [15] Deok-Soo Kim, Donguk Kim, and Kokichi Sugihara. Voronoi diagram of a circle set from voronoi diagram of a point set: II. Geometry. *Computer Aided Geometric Design*, 18(6):563–585, 2001.
- [16] Stuart P. Lloyd. Least square quantization in PCM. In *IEEE Transactions on Information Theory*, volume 28, pages 129–137. IEEE Computer Society, 1982.
- [17] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Dia-*

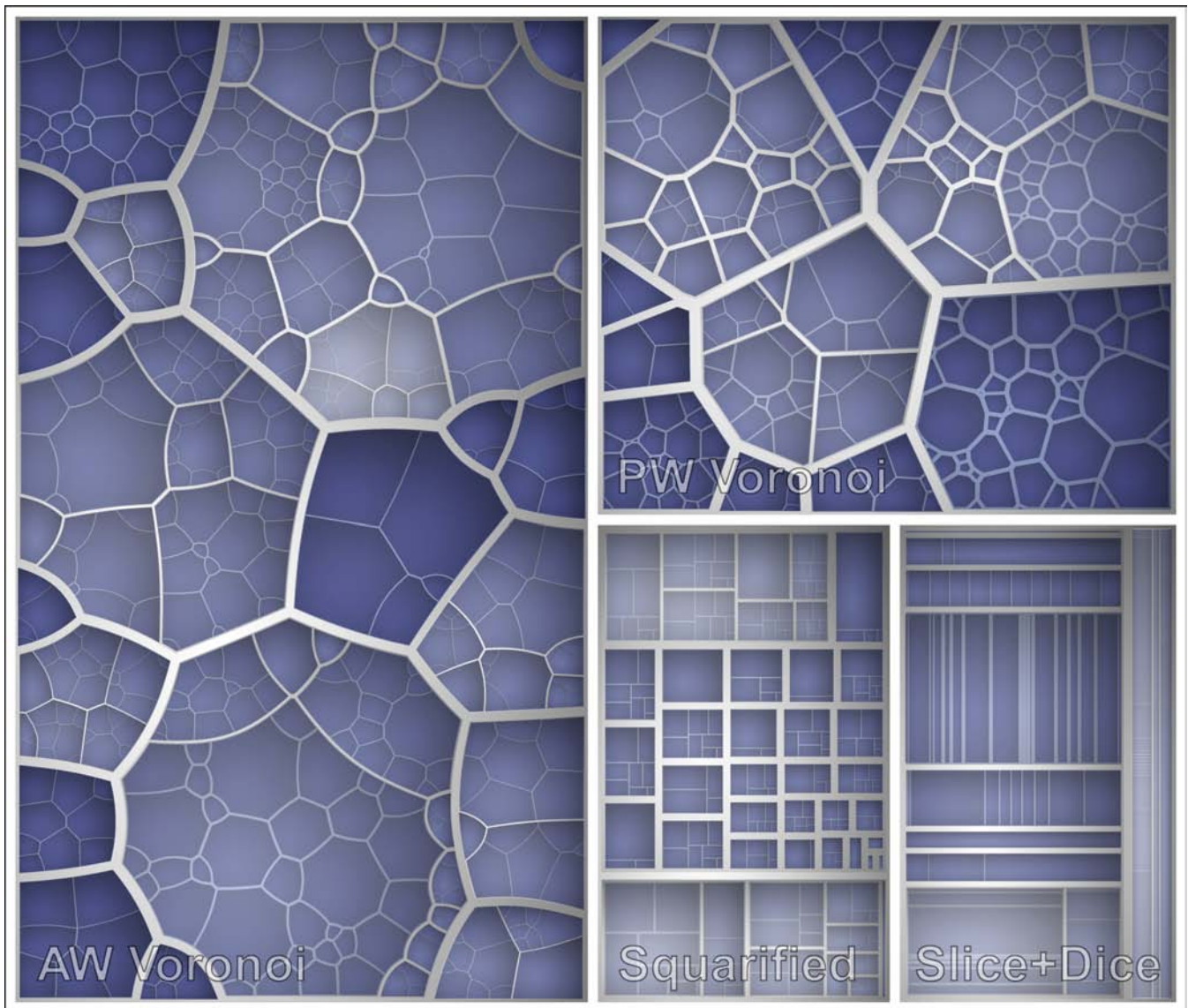


Figure 12: Comparison of four Treemap layout algorithms—at first, the top hierarchy level was subdivided with the Squarified Treemap algorithm, then for each of the four subareas according to its label a different layout algorithm was used (a brighter color indicates a lower hierarchy level)

- grams. John Wiley and Sons Ltd., 2nd edition, 2000.
- [18] Rene Reitsma, Stanislav Trubin, and Saurabh Sethia. Information space regionalization using adaptive multiplicatively weighted voronoi diagrams. In *Proceedings of the 8th International Conference on Information Visualisation*, pages 290–294. IEEE Computer Society, 2004.
- [19] Dmitri Roussinov and Hsinchun Chen. A scalable self-organizing map algorithm for textual classification: A neural network approach to the saurus generation. *Communication and Cognition – Artificial Intelligence*, 15(1-2):81–112, 1998.
- [20] Shirley Schussman, Martin Bertram, Bernd Hamann, and Kenneth I. Joy. Hierarchical data representations based on planar voronoi diagrams. In *Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization*, pages 63–72. Eurographics Association, 2000.
- [21] Ben Shneiderman and Martin Wattenberg. Ordered treemap layouts. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 73–78. IEEE Computer Society, 2001.
- [22] Robert Strzodka and Alexandru Telea. Generalized distance trans- forms and skeletons in graphics hardware. In *Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization*, pages 221–230. Eurographics Association, 2004.
- [23] 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. IEEE Computer Society, 1992.
- [24] 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. IEEE Computer Society, 1999.
- [25] Frederic Vernier and Laurence Nigay. Modifiable treemaps containing variable-shaped units. In *Extended Abstracts of the IEEE Symposium on Information Visualization*. IEEE Computer Society, 2000.
- [26] Martin Wattenberg. Map of the market, 1998. SmartMoney.com, <http://smartmoney.com/marketmap>.
- [27] Martin Wattenberg. Visualizing the stock market. In *Extended Abstracts on Human Factors in Computing Systems*, pages 188–189. ACM Press, 1999.