RC25132 (W1103-173) March 29, 2011
Computer Science

IBM Research Report

Dynamic Voronoi Treemaps: A Visualization Technique for Time-Varying Hierarchical Data

David Gotz
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Dynamic Voronoi Treemaps: A Visualization Technique for Time-Varying Hierarchical Data

David Gotz

Abstract— Treemaps are a widely used technique for the visualization of hierarchical data. In general, these techniques perform a space-filling recursive subdivision of a 2D space where the sizes of the created regions correspond to values of a particular data attribute. Several subdivision algorithms have been proposed to optimize specific criteria, such as region aspect ratio or stability. However, these goals are often contradictory. For example, existing layout algorithms that optimize for aspect ratio—important for legibility—are typically not stable. For this reason, Treemaps are rarely used in animated displays of time-variant data. When they are applied to dynamic data sets, unstable Treemap layout algorithms produce poorly animated transitions that include discontinuous jumps in region position when values change. This paper introduces a new technique called Dynamic Voronoi Treemaps. Our layout algorithm is specifically designed to support smooth, real-time animation of time varying hierarchical data while maintaining desirable aspect ratios. We describe our novel approach and outline how it overcomes key limitations of prior Voronoi-based Treemap work to enable the visualization of dynamic data. Results from an evaluation study of the technique are provided along with a brief use case highlighting a real-world application of Dynamic Voronoi Treemaps.

Index Terms—Information Visualization, Temporal Visualization, Treemaps, Voronoi Treemaps

1 Introduction

Hierarchical data structures are a widely used data abstraction for the organization and classification of information. Hierarchies have been utilized in a vast array of applications ranging from corporate organizational charts to computer file systems to the stock market. Given the ubiquitous nature of these data structures, hierarchical visualization techniques have received significant attention within the Information Visualization community.

One of the most well-known and widely used methods for visualizing hierarchical data is the Treemap [9]. First proposed by Johnson and Shneiderman in 1991, the Treemap is a space-filling visualization technique. A recursive spatial sub-division algorithm is used to create an arrangement of nested rectangles. The size of each rectangle encodes the value of an attribute of the data set (e.g., the size of a file or directory when visualizing a file system) while the nested rectangles represent children (e.g., files and sub-directories contained within a directory).

In subsequent years, several variations to the original Treemap algorithm have been proposed (e.g., [5, 17]). Many of these alternatives have attempted to reduce the often high aspect ratios of rectangular regions produced by the original algorithm. However, these improvements to aspect ratio typically come at the cost of increased instability [13]. None of these algorithms offer both full stability and ideal aspect ratios. As a result, Treemap applications have largely been applied to relatively static data sets where the sizes of cells do not change significantly over time. Developing a Treemap layout with both stability during animation and desirable aspect ratios remains an open problem.

To address this challenge, this paper presents the *Dynamic Voronoi Treemap* (DVT), a novel Treemap layout algorithm which provides both layout stability and desirable aspect ratios. As a result, DVTs enable smooth continuous animation between states when presented with dynamic data (see Figure 1). Unlike prior work in animating Treemaps for dynamic data (e.g., [8]), DVTs do not introduce holes or require overlapping regions during the animation sequence. In contrast, DVTs maintain a complete space-filling tessellation of the display space throughout all animated transitions.

The DVT technique is motivated by the work of Balzer and Deussen on Voronoi Treemaps [1]. Similar to Voronoi Treemaps, our algorithm uses an iterative optimization-based layout procedure and exploits the added layout flexibility made possible by using non-rectangular regions. However, the DVT algorithm approaches the hierarchical tessellation process differently.

In contrast to prior work, our novel layout algorithm produces a complete multi-level Voronoi tessellation at every optimization step. This is made possible by re-ordering the Voronoi computations and introducing a warping stage into the layout algorithm. The result is a Treemap layout that (1) is stable, (2) has regions that have very low aspect ratios, and (3) supports smooth animation where regions are easy to follow over time.

We have applied the DVT algorithm within a system administration application to visualize in real-time the job scheduling behavior of a massively parallel high-performance computing system. Results from our user study indicate that users found it significantly easier to perform two different tasks using the DVT-based visualization when compared to a similar tool based on the squarified Treemap layout algorithm.

• *David Gotz is with the IBM T.J. Watson Research Center, E-mail: dgotz@us.ibm.com.*

2 Related Work

Treemaps have a long history with research efforts focusing on several important aspects of the visualization technique. In this section, we review a sampling of related work to put our contributions in context.

2.1 Treemap Visualizations

Treemaps are a widely used technique for visualizing hierarchical data. Originally proposed by Johnson and Shneiderman in 1991 [9, 12], Treemaps use a recursive space-filling algorithm to subdivide a two-dimensional plane into nested regions. The size of each region corresponds to the value of a data element and the nesting depth corresponds to the depth of the data element in the corresponding hierarchical data structure. The original layout algorithm was successful at portraying the structure of complex hierarchies such as computer file systems.

Following up on this initial work, several alternative layout algorithms have been proposed to make Treemaps a more effective visualization technique. For example, Cushion Treemaps [15] introduced a shading techniques to make the structure of the data hierarchy more apparent.

Other work has focused on improving the shape or position of the rectangular cells produced during the subdivision process. For example, Squarified Treemaps [5] avoided the long and thin rectangles produced by the original Treemap algorithm that made it harder to judge the relative size of regions and obscured small areas. The squarified

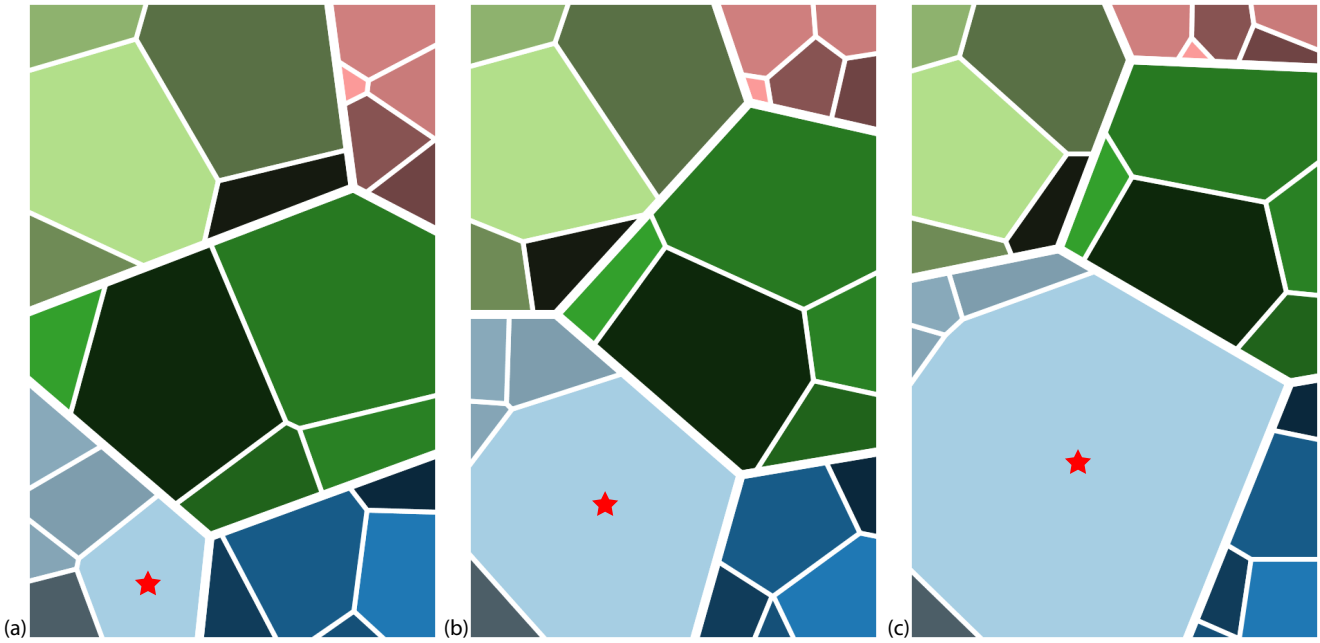

Fig. 1. This DVT sequence shows job resource utilization within a massively parallel high-performance computer system. Cell size indicates the fraction of system resources allocated to a job. Jobs are clustered by uniquely colored dispatch queues, with color intensity reflecting a key job performance metric. (a) Initially, the job marked with a star is using only a moderate amount of resources. (b) Over time, the job grows to become the most resource intensive on the system. (c) By the end of the sequence, the starred job has grown so large that its dispatch queue is using more than its share of system resources. The DVT's ability to smoothly animate this behavior allows a system technician to better understand the system's resource dynamics and diagnose problems.

algorithm is designed instead to create regions that have aspect ratios as close to one as possible.

Unfortunately, as observed by Shneiderman and Wattenberg [13], most algorithms that reduce aspect-ratio have the negative effect of introducing layout instability when used to render dynamic data that changes over time. They propose an ordered layout algorithm that strikes a balance between stability and aspect ratio. However, a measure of instability remains and aspect ratios are not ideal.

A limitation shared by all of the above Treemap techniques is that none of them produce both a stable layout (which is required for smooth animation of dynamic data) and ideal aspect ratios. The effect of the trade-off between these two desired properties can be seen online via a comparison tool hosted by the University of Maryland [18].

2.2 Non-Rectangular Treemaps

The vast majority of Treemap algorithms have focused on rectangular regions that are subdivided into smaller rectangular cells. However, non-rectangular regions have been examined. One approach is to combine smaller axis-aligned rectangles together into larger polygonal regions [11]. This produces a subdivision of interlocked concave regions.

More recently, Balzer and Deussen proposed Voronoi Treemaps [1] and used them to visualize software structures [2]. This technique uses an optimization-based layout algorithm to create Treemaps composed of nested convex polygonal shapes. The resulting regions have excellent aspect ratios. However, the algorithm does not produce layouts suitable for animation. This is in contrast to our novel DVT algorithm which is designed specifically to support animation. Because of the strong connection between Voronoi Treemaps and our own work, Section 3 discusses the original algorithm and its limitations in more detail.

2.3 Treemaps Applied to Dynamic Data

Any Treemap technique can be used to visualize dynamic data. However, the lack of stability that afflicts most Treemap layout algorithms manifests itself during animation as potentially large discontinuous jumps in the position and dimensions of individual cells. These jumps, most problematic for algorithms that produce the best aspect ratios, are highly disruptive to users. As a result, several techniques have been developed to allow improved temporal animations of Treemap visualizations.

In some cases, new algorithms have been developed that sacrifice aspect ratio in order to produce an ordered layout of cells (e.g., [3, 14]). Even with larger aspect ratios, however, these algorithms only improve stability. They do not eliminate it.

In other work, Fekete and Plaisant [8] propose a two-stage process for animating between two Treemap configurations. In the first stage, regions are linearly translated across the screen from old position to new position. In the second stage, region sizes are animated from the old size to the new size. This technique avoids discontinuous jumps during animation. However, the first animation stage produces both overlapping regions and holes in the layout which make the transition hard to follow. In contrast, DVTs support smooth single-stage animations that do not introduce any overlapping regions or holes.

3 Voronoi Treemaps

The Voronoi Treemap (VT) algorithm differs significantly from its predecessors. It produces non-rectangular spatial subdivisions that exhibit high quality aspect ratios that are close to one. However, it suffers from certain properties that make it unsuitable for animating the display of changing data values. In this section, we first review the basic Voronoi Treemap algorithm. We then describe in detail how dynamic data can be problematic for this approach.

3.1 Review of Voronoi Treemap Algorithm

VTs are composed of nested planar Voronoi tessellations. This section provides a brief overview of several key Voronoi tessellation concepts

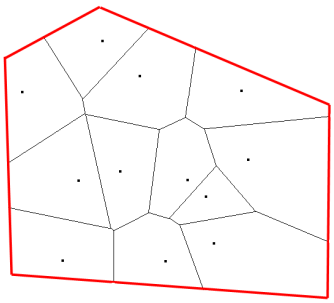

Fig. 2. A bounded planar Voronoi tessellation. The bounding region B is drawn in red. The inner polygons are the Voronoi regions R, each of which contains a single generator g_i.

and present a high-level summary of the VT algorithm. A more detailed discussion is beyond the scope of this paper and can be found in prior work [1].

3.1.1 Voronoi Tessellations

A Voronoi tessellation (also called a Voronoi diagram) is a partitioning of space into regions based on distances to a set of objects within the space. These objects are often called *generators*. In this paper, we consider only *planar Voronoi tessellations* which partition a 2D plane. Moreover, we limit our discussion to generators which are 2D points.

Planar Voronoi Tessellations. A planar Voronoi tessellation using a set of n generators $G := \{g_1, ..., g_n\}$ produces a corresponding set of n regions $R := \{r_1, ..., r_n\}$ such that (a) each region r_i contains exactly one generator g_i and (b) every point within a region r_i is closer to g_i than to any other generator in set G. Voronoi tessellations may be unbounded (a partition of the entire $\Re^2$ plane) or bounded (a partition of a region B defined within the plane $\Re^2$). In this paper, we focus on bounded planar Voronoi tessellations. An example of such a tessellation is shown in Figure 2.

Traditional Voronoi tessellations are calculated using a Euclidean distance metric. As a result, each generator is treated with equal weight, and region boundaries cross exactly midway between generators. This is captured by the following Euclidean metric where g_i is a generator and p is a point in $\Re^2$:

$$distance_e(g_i, p) := \|g_i - p\| = \sqrt{(x_{g_i} - x_p)^2 + (y_{g_i} - y_p)^2} \quad (1)$$

Weighted Voronoi Tessellations. Alternative distance metrics can be used when generating a Voronoi tessellation to influence the relative sizes of each region. Using a weighted metric, region boundaries will typically not cross midway between sites. Instead, boundaries will be closer to low-weight generators and further from high-weight generators. This property is critical for Voronoi-based Treemaps where each region's size must be scaled based on the data value that it represents.

In this paper, we utilize the *additively weighted power (AWP) distance metric* (Equation 2) which produces polygonal regions much like the classic Euclidean metric. Given a set of generators G, we define a corresponding set of weights $W := \{w_1, ..., w_n\}$. These weights are used within the AWP distance metric as follows:

$$distance_{awp}(g_i, w_i, p) := \|g_i - p\|^2 - w_i \quad (2)$$

Centroidal Voronoi Tessellations. Centroidal Voronoi tessellations (CVTs) are a special type of tessellation where the generator g_i for each region r_i is located at the center of mass of the region [6]. Such tessellations are relevant because they exhibit desirable aspect ratio properties. It is this property that has led to applications of CVTs within the visualization community [1, 7].

The iterative Lloyd's method [10] can be used to obtain a CVT for a set of generators G. The calculation begins with an initial set of generators G and the corresponding Voronoi regions R. On each iteration, the center of mass c_i for each region r_i is calculated and the generator g_i is moved to the position of c_i. The Voronoi tessellation is then recalculated. The iterative process continues until the distance by which each generator g_i is moved is below an error threshold.

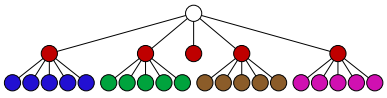

Fig. 3. The Voronoi Treemap algorithm performs a series of sequential tessellations as it recursively calculates its overall layout. In the data hierarchy shown here, five tessellations are required as indicated by the colors. The top level tessellation (for the red nodes) must iteratively converge prior to computing the Voronoi tessellation for the blue nodes. A complete tessellation is not available until the purple nodes are reached.

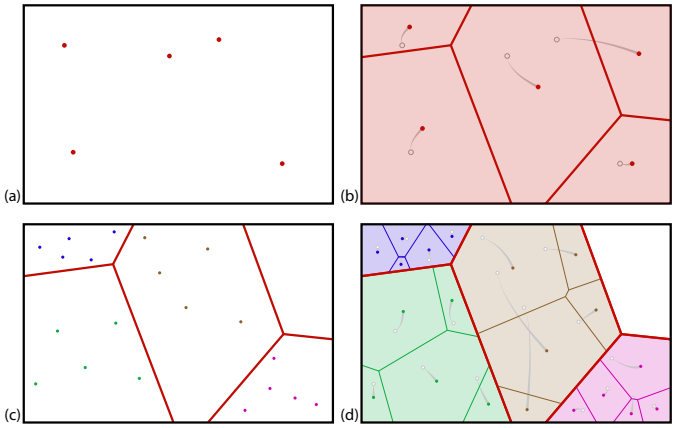

Fig. 4. A Voronoi Treemap for the data hierarchy in Figure 3. The layout begins by (a) choosing a set generators, one for each child of the top level data element (red nodes). (b) Lloyd's method is used to iteratively compute a weighted centroidal Voronoi tessellation. After the iteration converges, the process is applied recursively. (c) Generators are chosen within each sub-region and (d) new tessellations are computed.

3.1.2 The Voronoi Treemap Algorithm

The VT visualization technique [1] uses a recursive layout algorithm similar to that of traditional rectangle-based Treemap techniques. The top level is fully subdivided first. This is followed by recursive subdivisions as the corresponding data structure is traversed. However, in place of the normal procedural subdivision algorithm, VTs utilize a variant of Lloyd's algorithm to produce a CVT.

For example, consider the hierarchical data set shown in Figure 3. The construction of the corresponding VT visualization is illustrated in Figure 4. Beginning with the initial bounding region, a set of generators is selected such that there is one generator for each of the root data element's children in the data hierarchy. For example, Figure 4a shows one red generator for each red node in Figure 3. Each generator is assigned an initial weight which reflects the value of the generator's corresponding data element.

In the next stage, an iterative optimization algorithm computes a weighted CVT (Figure 4b). The process uses Lloyd's algorithm where generator positions are moved to a region's center of mass after each iteration. This produces a Voronoi tessellation with desirable aspect ratios. At the same time, the weight for each generator is adjusted on each iteration so that region sizes converge to an area that reflects the data value associated with the generator.

Once the iterative tessellation algorithm has converged to the desired weighted CVT, the process recurses to subdivide child regions. Within each region, generators are selected and assigned weights (Figure 4c). These generators are then used to produce the next level of

tessellations (Figure 4d). The process continues until the full data hierarchy is reflected within the VT visualization.

3.2 Complications for Dynamic Data

The original VT technique works well for static data and has several desirable properties. However, the layout algorithm as described above suffers from complications when applied to dynamic data. First, the calculation of lower level tessellations does not begin until after higher-level tessellations are finished. Second, generator positions can become invalid once bounding regions have been altered to reflect dynamic data. Both of these issues make animation problematic. Finally, the relatively long duration of the tessellation algorithm makes the animation of real-time dynamic data difficult.

As illustrated in Figure 4, the top level tessellation must be allowed to fully converge before any lower-level tessellations are performed. The result is that tessellations are performed serially. Unfortunately, this makes animation impossible because only a fraction of the Treemap's layout is determined until the layout algorithm reaches the final subdivision step. Drawing the visualization prior to this point would result in a Treemap with missing regions.

Moreover, because the top-level Voronoi regions are altered when data values change, lower-level generators used to compute the Voronoi regions prior to the data change may become invalid. Their positions, valid prior to the data change, may suddenly become located outside of the newly updated top-level Voronoi regions once the higher-level tessellation is performed. Choosing new generators, however, can result in wildly different layouts that prevent effective animation.

Finally, the VT layout algorithm can take a long time to converge. Several seconds are often required for relatively simple data structures and examples from the original paper take many minutes to complete. This is especially problematic given that complete tessellations are not generated during intermediate periods. As a result, animated VTs for dynamic data are not supported. Moreover, even if interpolation could be applied in place of intermediate tessellations, the long duration required to determine a final stable layout makes real-time animation impossible.

4 Dynamic Voronoi Treemaps

This section describes the Dynamic Voronoi Treemap (DVT) algorithm. Designed specifically to overcome the challenges outlined in Section 3.2, DVTs are capable of visualizing dynamic data in real-time using smoothly animated Voronoi-based Treemap layouts. This section first describes the basic DVT algorithm. It then addresses a number of additional design considerations.

4.1 Basic DVT Algorithm

The DVT algorithm has two distinct phases. First, there is an *initialization phase* which is performed only once to bootstrap the visualization regardless of how many times the data being visualized changes. After the initialization phase, an *iterative update phase* is performed continuously throughout the lifetime of the visualization to drive animated transitions when data changes.

4.1.1 DVT Initialization Phase

The initialization phase is responsible for generating the first complete multi-level Voronoi tessellation of the visualization space when a visualization is first constructed. The purpose of this phase is to bootstrap the iterative update procedure with a valid initial state. It therefore does not produce a CVT. Instead, the initial tessellation produced here will be optimized during the subsequent iterative update phase.

The initialization phase is a recursive procedure as illustrated in Figure 5. The algorithm recursively traverses the hierarchical data set starting from the root. At each recursive step, a bounding region B_i and a node D_i in the data hierarchy are passed in as input. The process begins with B_i equal to the overall visualization's bounding box B and D_i set to the root of the hierarchical data set being visualized. At each recursive step, generators are created by choosing a random point g_i within the bounding region B_i for each child of D_i. The generators are assigned an initial weight w_i. The value for this weight is not critical as it will be modified during the iterative update phase. However, we initialize each W_i to its corresponding data value.

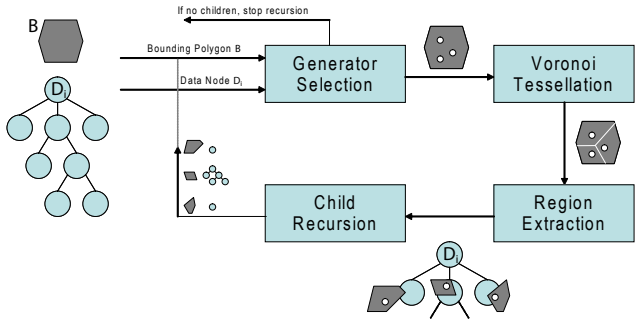

Fig. 5. The DVT initialization phase recursively computes a complete multi-level Voronoi tessellation. At each recursive step, random points are selected as generators. Using these generators, a Voronoi tessellation is performed. The regions produced by the tessellation are then mapped to corresponding sub-trees of the data set to prepare for the next round of recursion.

After the generators are selected, an AWP Voronoi tessellation is computed. Note that we do *not* compute a centroidal tessellation at this stage. Therefore, the iterative Lloyd's method is not required. Each region produced by the tessellation procedure is mapped to its corresponding sub-tree in the data set during the region extraction step. The algorithm is then recursively applied to these new regions and data sub-trees until the entire hierarchy has been traversed. We store with each node D_i its corresponding Voronoi region and the generator used to produce it.

This phase is extremely fast as no iterative optimization is performed during the tessellation step. For this reason, the sizes of the Voronoi regions produced in this phase do not yet correspond to the initial values of the data set. In addition, the regions are likely to have very poor aspect ratios. Optimization of these properties is accomplished during the iterative update phase.

4.1.2 DVT Iterative Update Phase

After the initialization phase completes, the DVT algorithm enters the iterative update stage. During this phase of the algorithm, the multi-level tessellation produced during initialization is iteratively improved to optimize both region size (to better correspond to possibly changing data values) and centroid location (to obtain desirable aspect ratios). Most important, each iterative step produces a complete hierarchical Voronoi tessellation that is suitable for animation.

High-Level Iterative Process. The high-level flow of the DVT iterative update algorithm is illustrated in Figure 6. The hierarchical Voronoi tessellation produced during initialization is passed to an incremental update module which performs a single optimization step over all levels of the Voronoi tessellation. Described in more detail below, this step (1) recursively evaluates and improves the aspect ratio and size of each Voronoi region in the tessellation, and (2) produces an overall error measure for the updated tessellation. Critically, the incremental update step produces a complete multi-level Voronoi tessellation for each iteration, enabling animated rendering. This overcomes the challenge described in Section 3.2.

The iterative update process is repeated until the error measure is reduced below a predefined threshold. Absent any changes to the underlying data set being visualized, the algorithm converges to a stable centroidal Voronoi-based Treemap layout. After convergence, any changes to underlying data values will trigger a new round of iterative updates. Data changes can also occur prior to convergence. In this case, the new values become active immediately during the incremental update step. For clarity, the discussion in this section is addresses only changes to data values for existing elements already in the data

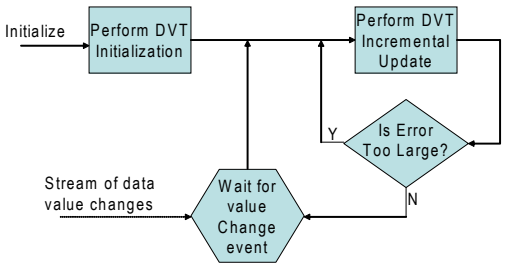

Fig. 6. The DVT iterative update phase performs a series of incremental layout updates that optimize both size and aspect ratio of the Voronoi regions. Each iteration produces a complete multi-level Voronoi tessellation. The process pauses when the layout error has been reduced below a threshold, and restarts whenever data values change.

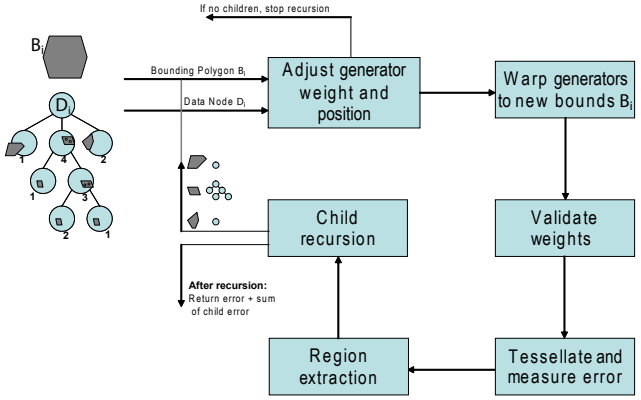

Fig. 7. The incremental update algorithm recursively traverses a multi-level Voronoi tessellation to measure and improve the layout. This approach is made possible by a warping stage that re-aligns old generator positions to new bounding regions for each recursive step. The result is a new complete hierarchical tessellation after each iteration.

hierarchy. The addition or removal of data elements themselves is discussed in Section 4.2.

Incremental Update Routine. The most critical stage of the DVT update phase is the incremental update routine. This step recursively traverses an existing hierarchical Voronoi tessellation, modifies generator positions and weights to improve the layout, and regenerates a new multi-level tessellation. Most critically, we introduce a warping step to this process to overcome the problem of invalidated generators created by changing bounding regions.

The incremental update algorithm is illustrated in Figure 7. It is a recursive algorithm that takes as input two pieces of data: (1) a data element D_i which is the root of a sub-tree in the hierarchical data set, and (2) a bounding polygon B_i within which the Voronoi tessellation for D_i must be embedded. As output, the algorithm produces an improved hierarchical tessellation for D_i and its children together with an overall error measure indicating the quality of the visualization layout.

The process begins by examining the generators and Voronoi regions associated with each child of D_i. Each of D_i's children D_j has associated with it both a generator g_j and a Voronoi region r_j. These values represent the Voronoi tessellation produced during the prior iteration (or during initialization when this stage is called for the first time). To improve region aspect ratio, the center of mass is determined for each r_j and the corresponding generator g_j is relocated to the center of mass. In addition, the area for each r_j is computed and compared to the area of the bounding region B_i. If the area is too small given the data value associated with D_j, the weight assigned to g_j is increased. If the area is too large, the weight is decreased. As in the original VT

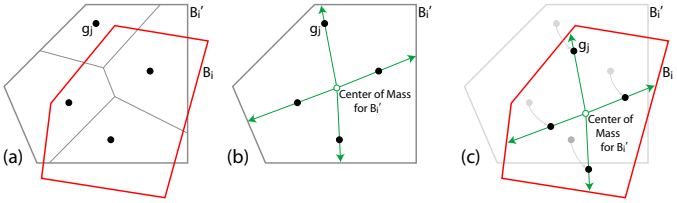

Fig. 8. (a) The weight and position for generator g_j are adjusted using Voronoi regions defined within the prior iteration's bounding region B_i' (shown in gray) and must be warped to corresponding locations within the current iterations bounds B_i (shown in red). (b) A ray casting technique is used to determine the position of each generator based on the B_i' region's center of mass. (c) Generators are then translated to corresponding positions within B_i.

algorithm, negative weights are allowed to ensure convergence when optimizing for region size.

Next, a warping step—unique to the DVT technique—is performed. As mentioned above, the adjustment of generators g_j is performed using Voronoi regions r_j produced during the prior iteration (or initialization). They are therefore positioned within the prior iteration's bounding region B_i'. However, bounding regions can change between iterations as the layout is optimized. This can lead to significant differences between the B_i' and the current iteration's bounding region B_i. Unfortunately, changes in bounding region can invalidate generator positions (if they fall outside of B_i) or place them far away from their ideal positions. For example, Figure 8a shows one invalid generator which falls outside of B_i. The other three generators remain valid, but are positioned in less desirable locations within B_i. The warping step is introduced to guarantee (1) that all generators remain valid even for dramatic changes in bounding region, and (2) that generator positions are translated to better reflect changes to bounding region.

The warping technique is illustrated in Figures 8b-c. First, we calculate the center of mass for B_i'. We then cast rays from the center of mass through each generator g_j to the edge of B_i'. For each ray, we calculate both the ray orientation and the position of the generator relative to the magnitude of the ray. To determine the new corresponding position of each generator within B_i, we first calculate the B_i's center of mass. Then, using the ray orientations from B_i' and and B_i's center of mass, new rays are extended to the edges of B_i. Finally, generators are positioned along these rays at the relative positions calculated using the old bounding region.

After the warping stage has completed, the weights assigned to each generator must be validated. During warping, it is possible that a high-weight generator g_h (with weight w_h) is positioned too close to a low-weight generator g_l (with weight w_l). In this case, the AWP distance (Equation 2) between the two generators g_h and g_l may be less than the AWP distance between g_l and itself. Mathematically, this occurs when the following inequality is violated:

$$\|g_h - g_l\|^2 > w_h - w_l \tag{3}$$

If a violation of Equation 3 is detected, the weight w_h is reduced so that $w_h = \|g_h - g_l\|^2 + w_l + \varepsilon$, where ε is a very small positive value.

After weight validation, the next step is to perform a Voronoi tessellation of B_i using the set off adjusted, warped, and validated generators. This produces a new tessellation for the children of D_i which is an incremental improvement from the prior iteration.

Following tessellation, we compute a layout error measure for the newly constructed Voronoi regions. The measure is calculated by summing the error associated with each individual region. For each region, the layout error is defined as a linear combination of *size error* and *aspect ratio error*. Size error measures how closely the size of a Voronoi region reflects the data value it represents, with a value of zero representing a region that is exactly the right size. Aspect ratio error is measured using the distance between a generator and the center of

mass of its corresponding Voronoi region. A region where the generator is located exactly at the center of mass has an aspect ratio value of zero.

Next, the region extraction step maps each new Voronoi region and its associated generator to the corresponding sub-tree in the data set. The algorithm then recurses to each of the children of D_i using the new Voronoi regions as bounding regions for the recursive tessellations.

Finally, the algorithm concludes by returning a sum of the layout error returned by each of the recursive calls added to its own measurement of layout error. The total overall layout error after the entire recursive process returns is used to determine if additional iterations are required to further improve the tessellation.

4.2 Additional Considerations

The basic DVT algorithm described above enables animated Voronoi-based Treemaps for data sets with changing values. In this section, we discuss several additional aspects of the DVT technique.

4.2.1 Removal and Insertion

The core DVT algorithm describes how to visualize data elements with changing values. However, there may also be data elements that are completely removed or added to the data set over time.

Animating the removal of a region from a DVT is straight forward. By setting the value of a data element to zero, the default algorithm will iteratively reduce the area assigned to the corresponding Voronoi region until it reaches zero and disappears. This produces an animated disappearance of a zero-valued region. Once the region has disappeared, the corresponding data element can be safely removed from the data structure.

Animating the insertion of a new data element is more complex. When a new data element is added to the data structure, it must be assigned an initial generator. This is required to allow the iterative update phase to proceed correctly.

The selection of generator position can follow one of two approaches: (1) randomized vertex selection, and (2) anchored vertex selection. Randomized vertex selection uses a random vertex from any peer Voronoi region in the data set as the generator's initial position. This technique produces a random insertion pattern where new nodes are spread out spatially. As a result, the DVT converges fairly quickly to a new layout even when several data elements are added at once. However, the spatial distribution of new regions makes it harder for users to recognize when new data elements appear.

Anchored vertex selection uses the same vertex for the insertion of all new nodes added to the same tier of the hierarchical data set. This approach has the benefit that all new regions first appear at the same place, producing a clear "flow" of new nodes into the visualization. However, because all new regions are inserted at the same location, they have further to travel before reaching a low-error layout. As a result, it takes longer to converge than the randomized approach.

Choosing the best approach for inserting new nodes is application dependent. In our use case described in Section 5, we employed anchored vertex selection because of the relatively slow pace of inserting new nodes and the importance of conveying the appearance of new nodes to the user.

After a generator position has been determined, a weight must be assigned. We calculate an initial weight using Equation 3, the same equation used for weight validation. We compare the new generator's position with it's peers and compute the lowest possible weight such that Equation 3 is satisfied. This initial weight is increased during the iterative update phase until the corresponding Voronoi region reaches the desired size.

4.2.2 Animation Throttling

The speed at which Voronoi regions move during animation can be highly variable. This is an artifact of the optimization algorithm which uses a greedy metric to improve the shape and size of each region independently. As a result, regions often "push" against each other for screen space. In some cases the movement of regions can become very slow as the optimization approaches a local equilibrium, then speed up quickly as the system "squeezes" past stable point on its way to the final optimized layout. This behavior is analogous to the geological concept of two tectonic plates rubbing against each other until the energy is so great that there is an earthquake.

These squeeze events are apparent during animation a DVT as suddenly fast movements which can distract from a user's ability to follow the motion of a region. To limit the visual impact of such events, we restrict the distance by which the centroid of any region can move in a single time step to a maximum threshold. This approach throttles the top speed at which a region can move during animation, making it easier for users to follow.

4.2.3 Landmarking

In the basic DVT algorithm, Voronoi regions can move freely around their corresponding bounding region during animation. As a result, a region that starts on the left side of the visualization can migrate over time to the right side of the display. However, there are times where it may be more desirable to keep an important region located at the same place within the visualization.

This can be accomplished by *landmarking*. This technique fixes the position of a landmarked region's generator to a static location within its bounding region. As a result, the landmarked region does not move during animation. Landmarking can be used to fix the location of one region or multiple regions. However, the use of too many landmarks can restrict layout flexibility and result in sub-optimal tessellations that take a longer time to converge.

4.2.4 Highlighting

Animation is a useful technique for visualizing the dynamic nature of a data set. However, our observations have shown that it can be difficult at times difficult to distinguish between regions undergoing small value changes and regions that are simply changing spatial positions with maintaining the same value. Highlighting can be used to mark regions with changing values to help users overcome this challenge. Our prototype implementation employs an optional time-decaying highlight around the border of any region that undergoes a change in value. The highlight appears with full intensity when a value change first occurs and fades over time.

5 Use Case: High-Performance Computing System Administration

Our work on developing a new Treemap algorithm for dynamic data is motivated by the real-world needs of system administrators who are tasked with managing a massively parallel high-performance computing system under development in our lab. The DVT algorithm is designed to give these administrative users a real-time overview of the machine's complex resource allocation behavior.

The parallel nature of the computer system means that a large number of jobs can run concurrently. Moreover, each job uses a time-varying portion of the system resources which is managed by an automated job scheduler. When jobs are submitted to the system, they are assigned to one of several weighted queues. The queue weights are then used by the job scheduler to dispatch new work as system resources become available. It is extremely difficult for administrators to understand and tune the behavior of the computer system due to its large scale, the complexity of the job scheduling algorithm, and the temporally dynamic system requirements of each job.

To assist these users, we have developed a web-based system administration application which provides access to both raw system status data and visualization-based tools. A key part of this application is a DVT-based visualization panel which shows a real-time animation of the system's job activity. The visualization is deployed as a Java applet and uses a DVT implementation built around an open-source implementation of the Bowyer-Watson [4, 16]. Voronoi tessellation algorithm.

The DVT tool is used to convey resource allocation to jobs. The tool provides flat views of jobs as shown in Figure 9, or hierarchical views similar to Figure 1 showing jobs organized by scheduling queue. The

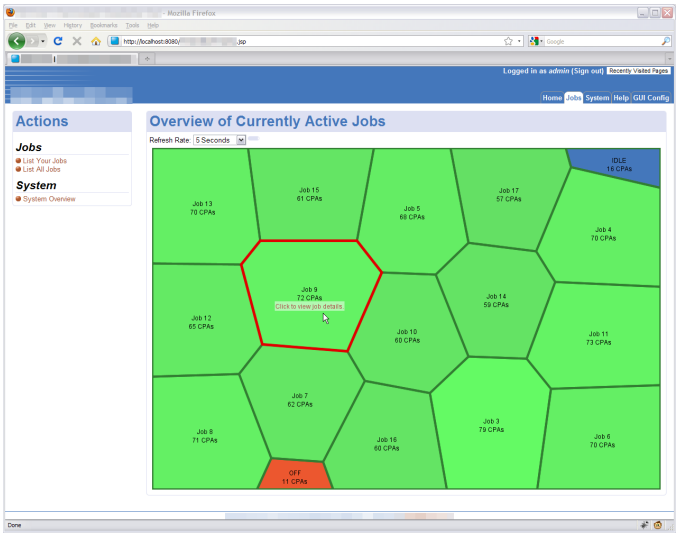

Fig. 9. A screen capture of our system administration application showing the DVT algorithm in use to view resource distribution across jobs. In the view shown here, the user is examining jobs independently (not hierarchically) with live resource allocation estimates updated every 5 seconds. The view shows 15 concurrent jobs with relatively even resource allocations. Users can click on individual regions to access detailed information about each job. Other views, similar in appearance to Figure 1, allow jobs to be grouped hierarchically based on job attributes.

DVT algorithm is required because of the dynamic nature of the system's jobs. New jobs appear as they are first scheduled; jobs disappear as they complete; and jobs use different amounts of system resources over time. The DVT algorithm is ideally suited for visualizing this sort of dynamic data.

The animated displays made possible by the DVT technique provide administrators with several key insights. They can observe how stable the job scheduler is at maintaining fairness between the various queues. If problems are seen, parameters can be adjusted to improve performance. Administrators can also use the DVT visualization to follow the lifetime of a single job. This can help identify problem jobs that may be running too long or obtaining too many system resources. The visualization also helps administrators understand how resources are re-allocated when a job terminates. Finally, the DVT algorithm helps administrators understand how failures impact overall system utilization to help maintain high availability.

These are just a few of the benefits that the DVT visualization provides to system administrators. By enabling a stable, real-time, animated display of system resource allocation, the DVT-based portion of our system administration application affords users a unique window into the dynamic nature of the computer system. As our evaluation in Section 6 shows, the unique properties of the DVT technique are rated very highly by users performing the visual tasks required to gain the insights outlined above.

6 Evaluation

This section presents a brief evaluation of the DVT technique. We first provide some statistics regarding the DVT's convergence behavior for data sets similar to those found in the use case described in Section 5. We then present results from a user study comparing DVT-based visualizations to animated Squarified Treemaps (STs).

6.1 Iterations and Convergence

To study the performance of the DVT algorithm, we gathered statistics for several test data sets patterned after the system administrator use case. Each test used a two-level tree of data with five top level nodes and a total of 30 children. We then used an instrumented version of our DVT application to visualize the data.

Test Case	Level 1 Iterations	Total Iterations
Test 1	11	84
Test 2	10	97
Test 3	15	29
Test 4	46	93
Test 5	11	49
Test 6	10	50
Test 7	11	30
Test 8	12	30
Test 9	16	51
Test 10	11	27
Averages	15.3	54

Fig. 10. Convergence statistics for the DVT layout algorithm. In our evaluation, an average of 15.3 iterations were required for the top level Voronoi tessellation to converge. Overall convergence required an average of 54 iterations.

We performed 10 independent experiments, each with randomly selected initial configurations and data values. For each experiment, we started with a fully converged DVT visualization. We then counted the number of iterations required to re-converge after data value changes were introduced that ranged from 1% to 2% of the total data set value (a typical amount for our use case application). When faced with this level of change, the DVT algorithm converged quickly to a new stable configuration. We The observed statistics are shown in Table 10.

On average, the layout algorithm required 54 iterations (ranging from 27-97) to converge with a rather small error tolerance ($e = 0.01$). The results show that the top level Voronoi regions converge far more quickly than the overall visualization. This behavior is expected given the nested nature of the Treemap. When top level Voronoi regions are still converging, the lower level tessellations are forced to adapt to changing bounding regions via the warping step. Only after the top level regions stabilize can lower level generators can be adjusted more accurately.

Also as expected, reducing the error threshold increases the iteration count. However, for the relatively small data sets in our use case application, convergence remains quick. Moreover, our informal probing of the error tolerance space showed that while layouts did improve statistically with lower error thresholds, the difference was not clearly perceivable to human users.

6.2 User Feedback

We performed a user study with 10 participants to compare DVTs with animated STs. We chose the ST visualization technique as a baseline in our study because of its similarly desirable aspect ratios and because the original VT algorithm does not support animation. In our study, each participant was shown both a DVT visualization and a ST visualization of the same dynamic hierarchical data set. Both visualizations used color and size to encode the same data properties.

The dynamic data used in the study was a two-level hierarchy with five top level nodes and 25 lower level nodes. Each top level node had five children of various sizes. During the experiment, the value assigned to a single data element was continuously increased from approximately 5% of the data set total value to approximately 50% of the total value. All other data elements were left with a static value. The transition from 5% to 50% took place over approximately 30 seconds.

For each visualization type, users were asked to perform two different visual tasks. First, users were asked to observe the overall data set and describe the changes that were taking place. This was the *global clarity* task. As part of that task, users were asked to rate the visualization on a scale of one to five on how well it helped them understand the changes taking place.

The second task asked users to follow a single region (not the one growing in size) throughout the animated time period and comment

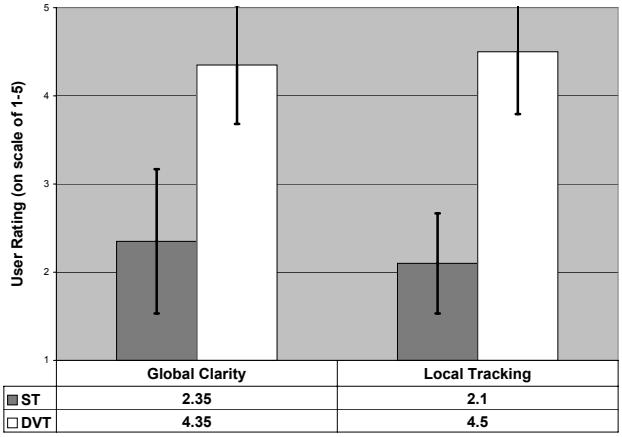

Fig. 11. Results from our user study show a statistically significant preference for DVT versus ST. Users rated DVT higher for both the *Global Clarity* and *Local Tracking* tasks.

on what changes were taking place to that specific value. This was the *local tracking* task. Once again, users were asked to rate the visualization on a scale of one to five on how well it helped them perform the task.

The results from our study are shown in Figure 11. In both tasks, the DVT visualization was rated significantly higher than the ST-based tool. In the global clarity task, the DVT tool scored an average of 4.35 compared to 2.35 for the ST visualization.[1] The higher rating for DVT is statistically significant with $p < 0.01$ using a one tailed paired t-test. Users responded that the sharp changes in layout in the ST visualization, caused by the changing data value, made it much harder to pick out the trend until late in the animation when the layout seemed to stabilize. In contrast, the DVT offered a smoothly animated view throughout the sequence.

The DVT visualization performed even more strongly in the local tracking task. DVT was given a 4.5 rating while ST scored 2.1. This statistically significant preference for the DVT technique ($p < 0.01$, one tailed paired t-test) reflects that following a single region using a Treemap technique like ST that lacks stability is nearly impossible. Each time a jump occurred in the layout, users had to perform a global search to re-located the region of interest. In contrast, the DVT visualization allowed users to follow the region easily throughout the duration of the animation.

7 Conclusion and Future Work

Treemaps are a powerful visualization technique for conveying the structure of hierarchical data structures. However, due to limitations associated with many layout algorithms, Treemaps are most often used to display static data sets. Specifically, Treemap layout algorithms typically sacrifice stability to support desirable aspect ratios (or vice versa). As a result, most techniques suffer from one of two problems: either (1) elongated high-aspect ratio regions, or (2) instability that creates large discreet jumps in region position when rendering dynamic data.

In this paper, we introduced Dynamic Voronoi Treemaps, or DVTs. Unlike prior work, DVTs provide both desirable aspect ratios and stable layouts that enable smoothly animated visualizations of dynamic data. We described our novel algorithm and discussed how the introduction of a warping step allowed our approach to produce a complete multi-level Voronoi tessellation for each iterative improvement to the layout. Moreover, we described how the DVT algorithm overcomes key limitations of the original Voronoi Treemap technique.

After presenting the technical details of our approach, we described a use case where DVTs have be used in a practical application. Motivated by that use case, we conducted a user study showing that DVTs were more effective at two different spatial tasks than animated squarified Treemaps. Our results provide statistically significant evidence that DVTs are preferred by users in both global clarity and local tracking tasks.

While our initial results are promising, there remain several areas for future study. For example, more effective alternatives to animation throttling would be very valuable. While throttling limits the top speed at which animation can occur, it doesn't smooth out less severe speed variations. A more effective approach to controlling variations in animation speed would be useful.

Another topic for future study is the perceptual impact of changes in error tolerance. Higher error thresholds produce animations that converge faster and produce less region movement. However, the resulting Voronoi regions are sized less accurately. While large discrepancies in size are problematic, moderate errors may be acceptable given users' difficulty in accurately judging the area of arbitrary polygons. Exploring this trade-off via user studies to determine the ideal error threshold is an important task.

[1] While users were asked to respond with integer ratings from 1-5, some participants provided fractional scores, such as 2.5. This led to averages that would not be possible had users restricted themselves to integer values.

References

[1] M. Balzer and O. Deussen. Voronoi treemaps. In *IEEE InfoVis*, page 7, Washington, DC, USA, 2005. IEEE Computer Society.

[2] M. Balzer, O. Deussen, and C. Lewerentz. Voronoi treemaps for the visualization of software structures. In *Proc of ACM Symposium on Software Visualization*, 2005.

[3] B. B. Bederson, B. Shneiderman, and M. Wattenberg. Ordered and quantum treemaps: Making effective use of 2d space to display hierarchies. *ACM Transactions on graphics*, 21(4):833–854, 2002.

[4] A. Bowyer. Computing dirichlet tessellations. *The Computer Journal*, 24(2):162–166, 1981.

[5] M. Bruls, K. Huizing, and J. J. van Wijk. Squarified treemaps. In *Proc. of Eurographics/IEEE TCVG Symposium on Visualization*, pages 33–42, 2000.

[6] Q. Du, V. Faber, and M. Gunzburger. Centroidal voronoi tessellations: Applications and algorithms. *SIAM Review*, 41(4):637–676, 1999.

[7] Q. Du and X. Wang. Centroidal voronoi tessellation ased algorithms for vector fields visualization and segmentation. In *Proc. of IEEE Visaulization*, 2004.

[8] J.-D. Fekete and C. Plaisant. Interactive information visualization of a million items. In *Proc. of IEEE InfoVis*, 2002.

[9] B. Johnson and B. Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *Proc of IEEE Visualizatoin*, 1991.

[10] S. P. Lloyd. Least square quantization in PCM. In *IEEE Transactions on Information Theory*, volume 28, pages 129–137, 1982.

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

[12] B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Transcations on Graphics*, 11:92–99, 1992.

[13] B. Shneiderman and M. Wattenberg. Ordered treemap layouts. In *Proc of IEEE InfoVis*, 2001.

[14] Y. Tu and H.-W. Shen. Visualizing changes of hierarchical data using treemaps. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1286–1293, 2007.

[15] J. J. van Wijk and H. van de Wetering. Cushion treemaps: Visualization of hierarchical information. In *Proc. ofIEEE InfoVis*, 1999.

[16] D. F. Watson. Computing the n-dimensional tessellation with application to voronoi polytopes. *The Computer Journal*, 24(2):167–172, 1981.

[17] M. Wattenberg. Visualizing the stock market. In *Proc. of ACM CHI Extended Abstracts*, 1999.

[18] M. Wattenberg and B. Bederson. Dynamic treemap layout comparison. http://www.cs.umd.edu/hcil/treemap-history/java_algorithms/LayoutApplet.html.