Hierarchical Data Visualization Using a Fast Rectangle-Packing Algorithm

Takayuki Itoh, *Member*, *IEEE Computer Society*, Yumi Yamaguchi, Yuko Ikehata, and Yasumasa Kajinaga

Abstract—This paper presents a technique for the representation of large-scale hierarchical data which aims to provide good overviews of complete structures and the content of the data in one display space. The technique represents the data by using nested rectangles. It first packs icons or thumbnails of the lowest-level data and then generates rectangular borders that enclose the packed data. It repeats the process of generating rectangles that enclose the lower-level rectangles until the highest-level rectangles are packed. This paper presents two rectangle-packing algorithms for placing items of hierarchical data onto display spaces. The algorithms refer to Delaunay triangular meshes connecting the centers of rectangles to find gaps where rectangles can be placed. The first algorithm places rectangles where they do not overlap each other and where the extension of the layout area is minimal. The second algorithm places rectangles by referring to templates describing the ideal positions for nodes of input data. It places rectangles where they do not overlap each other and where the combination of the layout area and the distances between the positions described in the template and the actual positions is minimal. It can smoothly represent time-varying data by referring to templates that describe previous layout results. It is also suitable for semantics-based or design-based data layout by generating templates according to the semantics or design.

Index Terms—Hierarchical data, Delaunay triangular mesh, rectangle packing.

1 Introduction

There are various kinds of hierarchical data around us and it is therefore important to provide visualization systems for such data. Some of them first represent higher-level portions of the hierarchical data and provide user interfaces to interactively explore lower-level portions. Others represent the overview of the data by placing all portions into display spaces.

Tree-based representation methods [5], [12], [13] provide good interfaces for exploring hierarchical data. These methods often first place a specific node (i.e., the node at the top of the hierarchy or a user-specified node) at the center of a display space and then place child nodes around the focal node. They do not always display all parts of the hierarchical data, but provide an interface to select and focus on the parts users are interested in. These interfaces are especially useful for navigation by end users.

On the other hand, some other visualization methods provide good views of entire hierarchical data sets. Treemap [1], [2], [4], [11], [18] is a well-known space-filling visualization technique that places all parts of the data onto one display space. The technique proposed in this paper is closer to the space-filling techniques rather than the tree-based techniques.

This paper presents a technique that represents hierarchical data in compact display spaces. The technique represents hierarchical data as a set of nested rectangles, in contrast to many existing tree-based data visualization techniques. Fig. 1 is an example of a representation using our technique. It first packs rectangular icons or thumbnails that denote leaf nodes under a nonleaf node without overlapping each other and encloses them by a rectangular border that denotes the nonleaf node. It then repeats the process, enclosing the sets of nodes by rectangles, from the lowest level to the highest level. It finally places all of the pieces of hierarchical data onto a display area.

We were motivated to develop the presented technique by the requirements for the visualization of Web sites. The main requirement was monitoring the attributes of thousands of Web pages, where the attributes include access frequency and last update time. We therefore aimed to pack thousands of icons representing Web pages into one display space. We avoided the overlap of icons so that all icons can be visualized without viewing operations. We represented all Web pages as equisized icons so that they can be evenly compared. We categorized the icons inside squarish subspaces according to the directory structure of the Web site so that Web pages under the same directory can be visually recognized as a group of icons. Fig. 1 shows how the presented technique represents thousands of Web pages in one display space without overlap.

Moreover, we aimed to roughly control the positions of the icons. It is convenient if icons of large-scale data are aligned according to user-specified semantics or placed according to users' designs because that makes it easier to search for interesting icons in thousands of displayed icons. Also, it is convenient if icons are placed to look similar to the layout results of similar data. These are reasons why we aimed to control their positions. It is an important requirement for the visualization of Web sites because the Web pages of Web sites often increase or decrease gradually. The hierarchical data of a Web site with small

• *The authors are with IBM Research, Tokyo Research Laboratory, 1623-14 Shimo-tsuruma, Yamato-shi, Kanagawa 242-8502 Japan. E-mail: {itot, yyumi, ikehata, kajinaga}@trl.ibm.com.*

Manuscript received 16 Mar. 2003; revised 22 Sept. 2003; accepted 14 Oct. 2003.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-0002-0103.

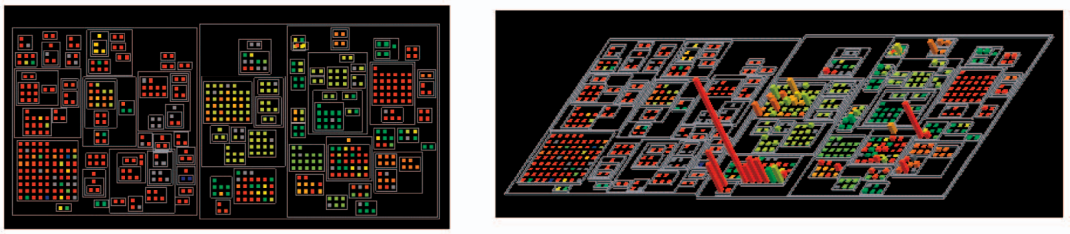

Fig. 1. Examples of the representation of hierarchical data. Colored dots denote leaf nodes and gray rectangular borders denote nonleaf nodes. Leaf nodes of this data denote the Web pages of a Web site and the nonleaf nodes of the data denote the directory hierarchy. The leaf nodes are colored according to their last update time and given heights according to their access frequencies.

time differences would therefore be similar and such similar data should be similarly represented.

According to these motivations, we developed a visualization technique that has the following features:

Feature 1: Efficient use of display spaces. It is often useful if visualization techniques pack all data items in a limited display space to provide a good overview. Traditional orthogonal tree-based systems, such as well-known file system viewers, have a bottleneck in that they may need a large display space if there are many nodes under one nonleaf node or if there is a deep hierarchy.

Feature 2: No overlaps between nodes. Some visualization methods may cause nodes to overlap in defocused regions. Our technique does not overlap them, so it provides a uniform overview of the data.

Feature 3: Aspect ratio of subspaces. When visualization techniques subdivide a display space to represent the parts of the given data, squarish subspaces are usually preferable over thin subspaces so that users can visually recognize the parts. Therefore, aspect ratios of subspaces should be considered.

Feature 4: Flexible placement of arbitrarily shaped nodes. When data items are represented as rectangular icons, we often assume that the aspect ratios and sizes of all icons should be entirely unified. On the other hand, we also assume that the aspect ratios and sizes of all the icons should be specifiable by users and they may even be varied. This makes it possible to visually emphasize important data items.

Feature 5: Similarity. It is desirable that similar data be similarly represented. For example, it is desirable that time-varying data be represented without drastically changing the layout of data items. To satisfy this feature, data items from similar data sets should be similarly placed onto the display spaces.

Feature 6: Semantics of placement. It is desirable that the positions of data items can be calculated according to user-specified semantics, e.g., alphabetical order, score of data items, or a user's design.

This paper presents an algorithm for quickly packing a set of arbitrarily sized rectangles in a small display space, which is called "mesh-edge-based rectangle packing." The mesh-edge-based algorithm is applied to the presented visualization technique so that it interactively displays large-scale hierarchical data, where icons, thumbnails, or borders are treated as arbitrarily sized rectangles. The mesh-edge-based algorithm places a set of rectangles one by one, while it looks for gaps between previously placed rectangles where the remaining rectangles can be placed without overlapping. It generates a Delaunay triangular mesh that connects the centers of the placed rectangles and refers to it to quickly find the gaps. It updates the mesh by connecting the centers of the placed rectangles one by one. This reduces the usage of display space, avoids overlaps between data items, and quickly places all of the data items. Using the algorithm, the visualization technique satisfies Features 1, 2, 3, and 4.

This paper also presents an extension of the rectangle-packing algorithm, which is called "template-based rectangle packing" in this paper, that refers to templates describing the ideal positions of the nodes of input data so that the presented visualization technique satisfies Features 5 and 6. The template-based algorithm places rectangles as close as possible to the ideal positions described in the templates, while it still reduces the usage of display space and avoids overlaps among the data items. Template positions can be created from semantics of nodes, rough design by users, or the positions of nodes in a previous time step.

We implemented the proposed technique, measured the computation time and layout results, and compared it with existing techniques. Section 6 describes the results of these measurements.

2 Related Work

2.1 Tree-Based Visualization

Tree representation is the most popular hierarchical data visualization technique, used in many programs such as well-known file system viewers. Several variations, such as the Hyperbolic Tree [13], Cone Tree [5], Fractal Views [12], and NicheWorks [22], have been described for the interactive visualization of large-scale tree data sets. These techniques are suitable for visualizing the higher-level data first and then exploring the lower-level data according to the users' choices. They are also suitable for visualizing the connectivity among nodes. Carriere and Kazman represented all parts of large-scale hierarchical data by using a Cone Tree in [5]. Wills also represented such data by using NicheWorks [22]. These provide good overviews of the data, but their results contain sparse regions.

2.2 Space-Filling Visualization

A Treemap [11], which represents hierarchical data in a manner like nested column charts, is a well-known space-filling approach for hierarchical data visualization. The space-filling approach is especially useful for representa-

tions appropriate to visualizing quantitative data attributes, as opposed to seeing connectivity, for which tree-based techniques may be more appropriate. This approach is also suitable for representing all of the data in a compact display space. The technique presented in this paper is close to Treemap in terms of providing overviews of the data.

Treemap satisfies Features 1 and 2 since it quickly places all parts of the data into one display space without overlap. However, it has shape-related limitations in that some of data items may be very thin. Squarified Treemap [4] improved the shapes of the rectangular subspaces so that it satisfies Feature 3. Ordered Treemap [18], [2] placed the data items in the order of user-specified semantics so that it satisfies Feature 6, in addition to improving the shapes of the rectangular subspaces. Such an ordered layout is also useful for dynamically updated data and, therefore, it satisfies Feature 5. However, they still do not guarantee the aspect ratios of the lowest-level data items. Quantum Treemap and Bubblemaps [1], [2] were recently proposed for the layout of icons or thumbnails when those aspect ratios are fixed, so that they satisfy Feature 4. As described in [2], the stable layout of dynamic data is a future research direction. The template-based algorithm presented in this paper addresses this issue.

Nested pie charts are also a space-filling approach applied to hierarchical data visualization [6], [21]. They also have shape-related limitations.

2.3 Semitransparent 3D Visualization

Information cubes [17] is a 3D hierarchical data visualization approach, which positions data in nested semitransparent hexahedrons. H-BLOBS [20] is also a 3D hierarchical data visualization approach, positioning data in nested semitransparent isosurfaces. These approaches have limitations in that they require a graphics environment supporting 3D semitransparent rendering and in that users' need 3D manipulation skills. Also, the computation times for data layout are not clearly mentioned in their papers.

2.4 Graph Layout

Data layout techniques are important not only for hierarchical data visualization but also for graph visualization. The force-directed algorithm is a famous graph layout approach which applies spring models to nodes and arcs. It is somewhat computationally unstable and expensive for iterative dynamics calculations, but, recently, many improved algorithms have been reported for large-scale and clustered graphs [9], [16]. Gansner et al. presented an improved force-directed approach [8] that forms a Voronoi diagram of nodes during each iterative calculation step. It moves nodes to the centers of the Voronoi polygons to refine the configuration. The heuristic used in this paper is similar to theirs. Freivalds et al. presented a space-efficient method for the representation of disconnected graphs [7]. It applies a rectangle packing technique to tightly place the disconnected parts of graphs.

2.5 Rectangle Packing

For our representation, a set of nested rectangles must be properly packed in a display region so that the display region is compact. A fast rectangle-packing algorithm is therefore a key technology for our representation. The packing problem is well-known in the field of VLSI circuit design [15], for the layout of mechanical parts onto sheet metal, and for the layout of parts of clothes. Many packing algorithms apply various optimization schemes such as genetic algorithms to minimize the layout areas. However, these algorithms often require minutes or even hours of computation time to find the optimized configurations and, therefore, they are not suitable for interactive visualization techniques. Our approach does not need to perfectly minimize the layout space, but only needs to find a reasonable configuration within a few seconds in order to provide an interactive visualization.

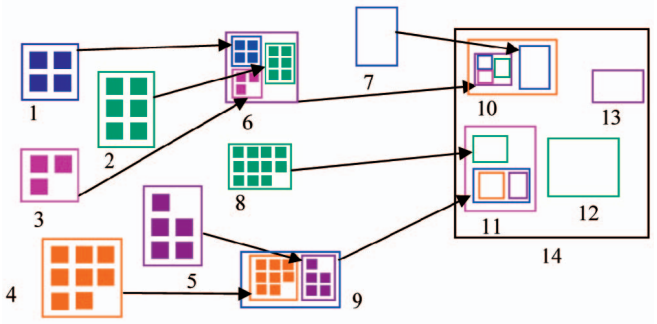

Fig. 2. Algorithmic overview of the layout process of the presented technique. Numbers in this figure denote the order of the process.

A good heuristic for fast object packing has been proposed for the purpose of texture generation [10]. While the method packs objects of semiuniform sizes as textures, our problem must deal with widely varied sizes of rectangles.

Space management has been studied in some visualization and user interface work. Design Galleries [14] tried to optimize the distribution of data items on display spaces, but it does not consider their overlap. The space manager [3] of Bell et al. tried to avoid the overlap of rectangular objects on display spaces during the addition and deletion of the objects, but it does not consider the minimization of the display spaces.

3 Overview of Hierarchical Data Layout

This section presents an overview of the proposed hierarchical data visualization technique, especially for providing an overview of large-scale, nonuniformly nested hierarchical data. The technique packs rectangular icons in rectangular regions and then repeats the packing of rectangular regions to represent the hierarchy.

Fig. 2 shows an illustration of the order of hierarchical data layout in the presented technique. The technique first packs icons (painted square dots in Fig. 2) that denote leaf nodes and then encloses them by rectangular borders that denote nonleaf nodes. Similarly, it packs a set of rectangles that belong to higher levels and generates the larger rectangles that enclose them. Repeating the process from the lowest level toward the highest level, the technique places all of the data onto the layout area.

Fig. 3 shows the tree structure of the hierarchical data we used as input data and the order of layout in the technique. To define the order of the layout process, the technique traverses the hierarchy using a breadth-first search algorithm, starting from the highest level of the data. It then places the data for each level in the reverse order of that traversal.

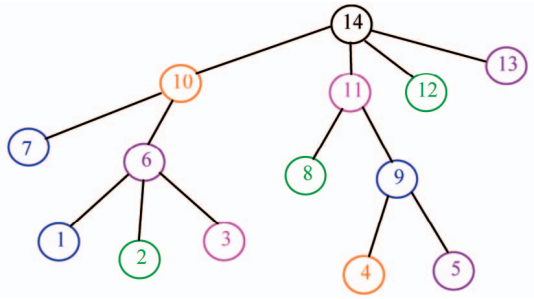

Fig. 3. Definition of the order of the layout using a breadth-first traversal. Each node of the tree structure in this figure denotes a rectangle in Fig. 2. The hierarchical structure and the ordering in this figure correspond exactly to Fig. 2.

Treating icons, thumbnails, and rectangular borders as a set of rectangles, the presented hierarchical data visualization technique applies a rectangle-packing algorithm to place them onto one display space. The packing algorithm places the rectangles tightly inside a small rectangular area and encloses them by a bigger rectangular border that denotes the parent nonleaf node.

If all nodes in a level are leaf-nodes and their sizes and aspect ratios are not specified, our implementation assumes that all nodes are represented by equal squares. In this case, the implementation places them onto an orthogonal regular grid, without using the rectangle packing algorithm. When the level contains n_l nodes, the implementation calculates the rounded value of $\sqrt{n_l}$, and sets the horizontal or vertical number of squares to that value. In Fig. 2, a set of nodes inside rectangular borders, labeled as 1, 2, 3, 4, 5, and 7, are placed onto the grid.

If the sizes and aspect ratios of the rectangles vary, the technique needs more robust rectangle packing algorithms. Sections 4 and 5 present the details of two rectangle-packing algorithms.

4 Mesh-Edge-Based Rectangle Packing Algorithm

This section presents the algorithm for quickly packing a set of rectangles, which is used for the presented hierarchical data visualization technique. Here, let us assume that n rectangles are given and the packing algorithm positions all of them onto an xy-plane while it tries to minimize the layout area. Let us also assume that all edges of the rectangles are parallel to the x-axis or y-axis. Under these assumptions, the algorithm places the rectangles one by one. Our implementation places rectangles in order of their areas, as shown in Fig. 4. It first places larger rectangles, and then searches for gaps to insert smaller rectangles.

4.1 Data Structures

This paper formalizes the problem as follows: Given rectangles $R = \{r_1, \ldots, r_n\}$, the algorithm places the set of rectangles one by one. Here, the algorithm favors accelerating the rectangle packing process rather than perfectly minimizing the layout space. We therefore did not apply optimization schemes to find the configuration of the rectangles, but used a heuristic to quickly find gaps and place the remaining rectangles in the gaps. The heuristic uses a triangular mesh connecting the centers of the previously placed rectangles, as shown in Fig. 5. Let us denote the mesh as $M(V, E, T)$ consisting of vertices $V = \{v_1 \ldots v_{n+4}\}$, edges $E = \{e_1 \ldots e_l\}$, and triangles $T = \{t_1 \ldots t_m\}$. The algorithm first positions the largest rectangle r_1 at the center of the layout area and generates a rectangular space that entirely encloses the positioned rectangle. Let the space be S, its four corner vertices be v_1 to v_4, and the center of r_i be v_{i+4}. We initially define the size of S as twice the size of r_1. The algorithm then generates four triangles t_1 to t_4, which connect the five vertices v_1 to v_5, as shown in Fig. 5a. Placing each new rectangle r_i one by one, the algorithm updates M as shown in Fig. 5, by connecting the new vertex v_{i+4} to several vertices and modifying several triangles. Note that M is independently generated for each set of rectangles under one nonleaf node.

While deciding on a position in which to place a rectangle, the algorithm calculates candidate positions on E and evaluates the candidate positions. Section 4.2 describes the order of visiting elements in E and Section 4.3 describes the evaluation of the candidate positions. Section 4.4 describes the modification of the triangular mesh. Section 4.5 describes the summary of the algorithm.

4.2 Order of Referring to the Mesh Edges

The algorithm places the rectangles one by one while it satisfies Features 1, 2, and 3. It searches for a position to place the rectangle, which satisfies the following conditions as much as possible:

Condition 1: No overlap between r_k and any previously placed rectangles.

Condition 2: Minimum extension of the layout area S and keep aspect ratio of S close to its preferred value.

Here, the preferred aspect ratio in Condition 2 is the same as the aspect ratio of the display or window space for

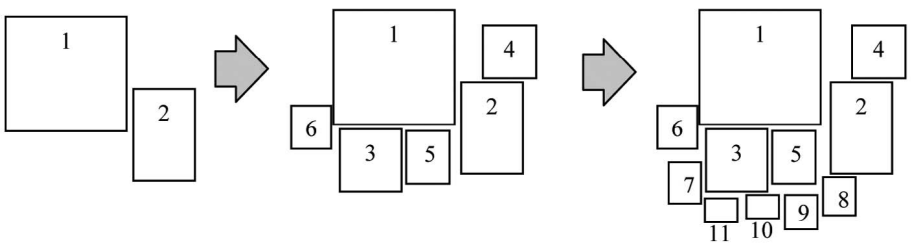

Fig. 4. Illustration of layout of rectangles using the presented algorithm. The algorithm first places the largest rectange and then places the others in the order of their areas while it searches for gaps where they can be placed without overlaps.

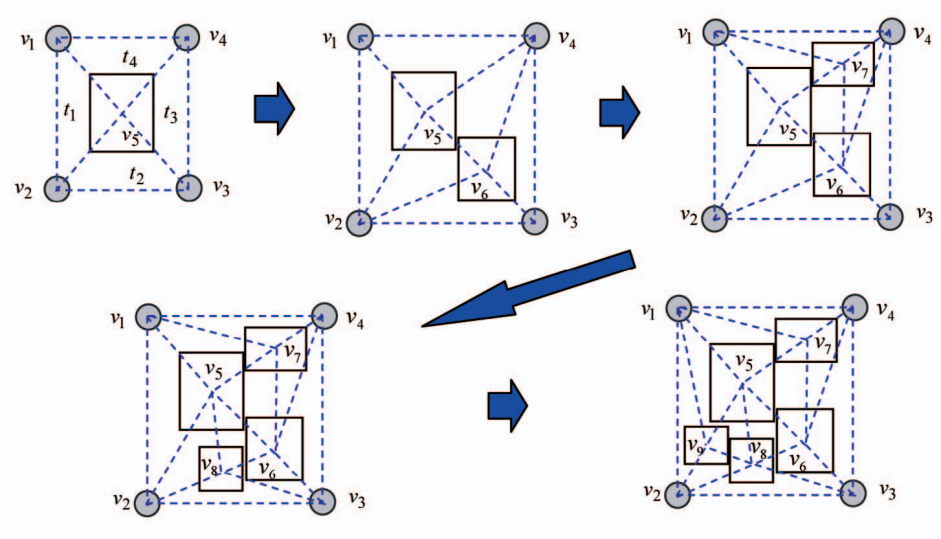

Fig. 5. Processing flow of rectangle placement and update of mesh M.

the highest level and 1 for the other levels based on the heuristics that squarish region is easier to visually recognize, as described in Feature 3.

To quickly search for positions where rectangles can be placed satisfying the above conditions, the algorithm picks suitable positions by using the following two strategies:

Strategy 1: It favors selecting sparsely populated regions since it is easier to place rectangles in such places without overlapping with other rectangles.

Strategy 2: It favors selecting interior positions since it is easier to place rectangles in such locations without enlarging the layout space.

Fig. 6a shows an example of a triangular mesh and rectangles. Here, let El be the length of a mesh edge, El_1 is the length of the part of the edge that is inside the rectangle whose center places it at an end of the edge, and El_2 is the length of the other part of the edge that is inside another rectangle, as shown in Fig. 6b. Our technique calculates the values of $El_r = El - (El_1 + El_2)$, the length of the remaining part of the edge lying outside the two rectangles. Here, the technique lets El_1 or El_2 take the value of zero when the ends of the edge are on $v_i (i = 1..4)$.This is because it is more likely that gaps will be found around mesh edges whose El_r values are larger. The numbers in Fig. 6c denote that the edges are ordered from the largest El_r to the smallest. The algorithm searches for gaps on the edges in this order so that it satisfies Strategy 1.

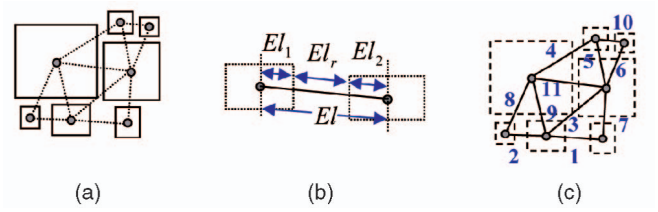

Fig. 6. (a) Delaunay triangular mesh connecting centers of previously placed rectangles. (b) Calculation of values of EL_r. (c) Order of the EL_r values.

We use a Delaunay triangular mesh for the algorithm according to our heuristics that Delaunay triangulation makes candidate positions well-distributed. Since the definition of Delaunay triangulation is the triangulation that maximizes the minimum angle of M, it avoids making closer candidate positions by adjacent edges connected with a narrow angle.

At the same time, the algorithm counts c_{e_j}, the number of corner vertices v_1 to v_4 touching the edge e_j. Fig. 7a shows an example of the distribution of c_{e_j}. This figure shows that interior mesh edges have the smaller c_{e_j} values.The algorithm then groups the mesh edges according to their c_{e_j} values. The algorithm starts the trial placement of rectangles on the edges. It first extracts edges from the $c_{e_j} = 0$ group, then the $c_{e_j} = 1$ group, and, finally, the $c_{e_j} = 2$ group, so that it satisfies Strategy 2.

The algorithm extracts edges in each group in the sorted order, starting from the edge that has the largest El_r value. The algorithm calculates at most two candidate positions where r_k touches the rectangles previously placed at the ends of e_j, as shown by the two dotted rectangles in Fig. 7b, and tries to place r_k at each of these positions. For $c_{e_j} = 1$ edges, the technique tries to place r_k at a position where r_k touches the rectangle previously placed located at one of the ends of e_j. In the case of $c_{e_j} = 2$ edges, the technique tries to place r_k at the center of e_j because there are no rectangles at the ends of $c_{e_j} = 2$ edges.

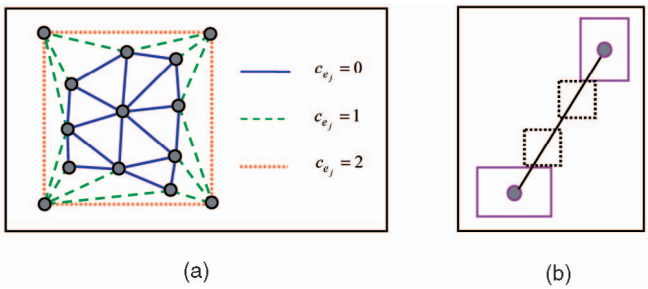

Fig. 7. (a) Values of c_{e_j} for edges. (b) Two positions to try to place the current rectangle.

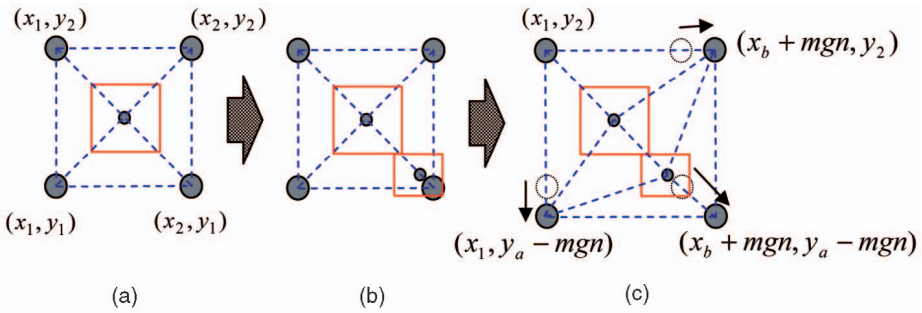

Fig. 8. (a) A rectangle and triangular mesh. (b) One more rectangle is placed on a mesh edge. (c) Corners of S are moved when the placement of the current rectangle requires enlarging the layout region S.

4.3 Evaluation of Candidate Positions

Given a candidate position, the algorithm checks if the point satisfies the two conditions described in Section 4.2. Starting from the $c_{e_j} = 0$ edges, the algorithm refers to edges in the sorted order and calculates candidate positions on the edges. The algorithm then attempts to place r_k at the candidate positions. It checks overlaps between r_k and previously placed rectangles and calculates enlargement of S. If the algorithm finds that one of the candidate positions satisfies both conditions, it decides to place r_k there and selects the next rectangle. Otherwise, the algorithm selects the next edge to check to see if it satisfies both conditions. If no $c_{e_j} = 0$ edge satisfies both conditions, the algorithm continues with the $c_{e_j} = 1$ edges and, finally, the $c_{e_j} = 2$ edges.

Even if the candidate position satisfies only Condition 1, the algorithm can place r_k after enlarging S. In this case, the algorithm evaluates the point. Our implementation evaluates points using a combination of layout area and layout aspect ratio. It calculates the value $aA + rR$, where a and r are user-defined positive values. A is calculated as follows:

$$A = A_{after}/A_{before},$$

where A_{after} is the area of S after the placement of r_k and A_{before} is the area of S before the placement. A_{after} can be calculated after the enlargement of S described in Section 4.4. R is calculated as follows:

$$R = \begin{cases} R_s/R_{best} (R_s > R_{best}) \\ R_{best}/R_s (R_{best} > R_s), \end{cases}$$

where R_s is the aspect ratio of S after the placement of r_k and R_{best} is the preferred aspect ratio, the aspect ratio of the display or window space for the highest-level of the hierarchy and 1 for the other levels. Again, R_s can be calculated after the enlargement of S described in Section 4.4. We usually define the value of a and r as $a = r = 1$, but it depends on the requirements of users: If the efficiency of the use of display spaces is important, a should be larger than r.

If $aA + rR$ calculated on the candidate position is smaller than the smallest in the values of previously processed candidate positions, the algorithm saves the candidate position with this $aA + rR$ value.If no candidate positions satisfy both Conditions 1 and 2, the algorithm places r_k at the most recently saved candidate position because this was evaluated as the best position.

4.4 Local Modification of the Triangular Mesh after the Addition of Each Rectangle

If it is decided to place the rectangle r_i at a candidate position that does not satisfy Condition 2, the algorithm enlarges S by moving some of the $v_i (i = 1..4)$, as shown in Fig. 8. Here, let the positions of $v_i (i = 1..4)$ are $(x_1, y_1), (x_2, y_1), (x_2, y_2), (x_1, y_2)$, where $x_1 x_2, y_1 y_2$. Also, we position the four corners of r_i at $(x_a, y_a), (x_b, y_a), (x_b, y_b), (x_a, y_b)$, where $x_a x_b, y_a y_b$. The algorithm enlarges S by recalculating the position of $v_i (i = 1..4)$ as follows:

$$\begin{aligned} &\text{if } x_a < x_1 \text{ then } x_1 = x_a - mgn, \\ &\text{if } y_a < y_1 \text{ then } y_1 = y_a - mgn, \\ &\text{if } x_b > x_2 \text{ then } x_2 = x_b + mgn, \text{ and} \\ &\text{if } y_b > y_2 \text{ then } y_2 = y_b + mgn. \end{aligned}$$

Here, mgn is a constant positive value. Our implementation applies $mgn = 0.1 S_w$ if $S_w > S_h$; otherwise, $mgn = 0.1 S_h$, where S_w is the width of S and S_h is the height of S.

After the algorithm places r_i by using the above steps, as shown in Fig. 9a, it updates the triangular mesh by adding the center of the placed rectangle v_{i+4} to the mesh. This process first connects v_{i+4} to the two other vertices of the triangles that share the edge e_j and divides each of the triangles into two new triangles, as shown in Fig. 9b. The process then locally modifies the mesh, starting from the triangles that share the newly added edges. It selects an adjacent triangle to modify and swaps their shared edge to improve the triangles, as shown in Fig. 9c. The modification is recursively repeated between the modified triangles and their adjacent triangles until no triangles are modified. The detailed algorithm of the mesh modification is described in [19].

4.5 Algorithm Summary

Fig. 10 shows the pseudocode of the algorithm. Here, we consider the complexity in terms of the number of rectangles n and assume that the numbers of vertices, edges, and triangles of a mesh are all proportional to n. To place all of the rectangles, the algorithm requires at least $O(n \log n)$ computation time on average for the creation and modification of the Delaunay mesh described in Section 4.4. It also requires $O(n \log n)$ for sorting the rectangles according to their areas and the mesh edges according to their El_r values. In addition, the algorithm requires $O(n^2)$ computation time when it checks for the overlap among the rectangles currently being placed and all of the previously placed rectangles. Managing the previously placed rectangles by using additional data

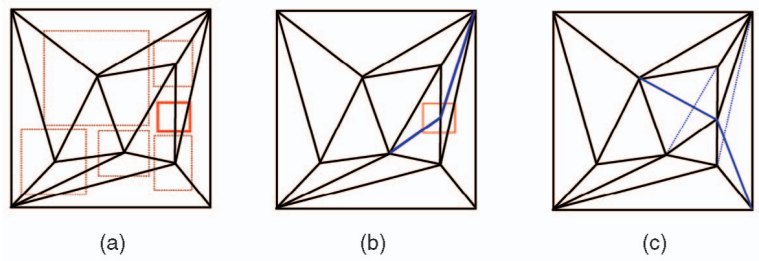

Fig. 9. (a) Triangular mesh that connects the center of a rectangle and the currently placed rectangle (with thick red border). (b) Two (thick blue) edges connecting the center of the current rectangle. (c) Locally modified mesh.

structures can reduce the cost. If the rectangles are managed by using a k-d tree, the computation time will be $O(n^{3/2})$ for the overlap check. Also, if the near rectangles are grouped into $\sqrt{n}$ groups, the computation time will also be $O(n^{3/2})$.

5 Template-Based Rectangle-Packing Algorithm

The mesh-edge-based algorithm presented in Section 4 does not explicitly control the positions of the nodes. It is possible that the algorithm will yield very different layout results for very similar hierarchical data sets. This section presents an extension of the rectangle-packing algorithm, a template-based algorithm, addressing this problem. The template-based algorithm refers to a template describing the ideal positions of nodes while it places the rectangles. It places them as closely as possible to the positions described in the template, while it still tries to minimize the layout area. Not only does this solve the variability problem, but the template-based algorithm can also be applied for various other purposes, as described in Section 1.

Similar to the application of the mesh-edge-based algorithm to hierarchical data visualization, the template-based algorithm is also applied to place nodes in the lowest level first and repeats a similar layout process from lower levels to higher levels. Let us denote a template as $Tpl(I, P)$, consisting of indices of rectangles $I = \{i_1 \dots i_k\}$, and ideal positions of the corresponding rectangles $P = \{p_1 \dots p_k\}$. The template-based algorithm supposes that templates are provided for each nonleaf node, but this is not necessary. If there are nonleaf nodes for which templates are not provided, the mesh-edge-based algorithm is used for the layout of the nodes under the nonleaf-nodes. The template-based algorithm also supposes that a template does not have to specify the positions of all nodes. If there are nodes whose positions are not described in the template, the template-based algorithm first places the other nodes whose indices and positions are described and then places the skipped nodes by using the mesh-edge-based algorithm.

It is possible that the scale between positions in a template and the specified sizes of the rectangles will be very different. The template-based algorithm therefore normalizes the positions in the templates between

```
for each rectangle r_i {
  for each edge e_j { // Section 4.2 describes the order of processing edges
    Calculate one or two candidate positions to place r_k on e_j;
    For (each candidate position cp) {
      If (cp satisfies both Conditions 1 and 2) {
        P = cp; goto PlaceNow;  // P is the final position of r_i
      }
      if (cp satisfies only Condition 1) {
        Calculate aA+rR; // Section 4.3 describes the calculation of aA+rR
        If (aA+rR is less than the previous candidate positions) {
          P=cp;
        }
      }
    } // end of for (cp)
  } // end of for (e_j)
PlaceNow:
  Place r_i at P; Modify M; // Section 4.4 describes the modification of the mesh
} // end of for (r_i)
```

Fig. 10. Pseudocode of mesh-edge-based rectangle-packing algorithm.

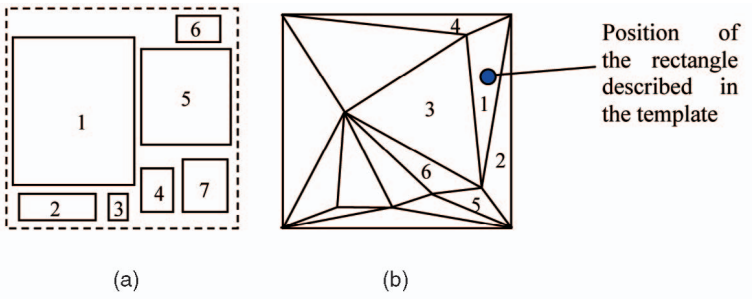

Fig. 11. Template-based rectangle-packing algorithm. (a) Order of placing rectangles. The algorithm first places the largest rectangle and then places the others in the order of their distances from the largest rectangle. This "adjacency-based order" is effective to tightly pack the rectangles next to each other. (b) Order of referring to triangles.

$(-1,-1)$ and $(1,1)$ and also normalizes the sizes of the placed rectangles. After placing all the given rectangles, it calculates the actual positions of the rectangles.

The template-based algorithm is different from the mesh-edge-based algorithm on the following points:

- One more condition is applied in deciding where to place the rectangles. See Section 5.1.
- The order of placing the rectangles is different. See Section 5.2.
- The template-based algorithm refers to the triangles of M, not to the edges. See Section 5.3.
- The equation for the evaluation of candidate positions is different. See Section 5.4.

After placing a rectangle, the template-based algorithm applies the same local mesh modification technique described in Section 4.4.

5.1 Conditions for the Placement of Rectangles

In addition to the two conditions described in Section 4.2, the template-based algorithm applies one more condition so that the rectangles are placed very close to the positions described in templates as follows:

Condition 3: Rectangle r_k should be placed as close as possible to the position described in the template.

The mesh-edge-based algorithm evaluates the satisfaction of Condition 2 by calculating $aA + rR$. The template-based algorithm calculates $aA + rR + dD$ to evaluate the combined satisfaction of Conditions 2 and 3, where d is a user-defined positive value and D is the distance between the candidate position in the normalized coordinates and the ideal position of the rectangle described in the template. The template-based algorithm tries to place rectangles at several candidate positions, calculates $aA + rR + dD$ at each of the positions, and, finally, places the rectangle where $aA + rR + dD$ is smallest.

5.2 Order of Placing Rectangles

While the mesh-edge-based algorithm places rectangles in the order of their area, the template-based algorithm places them using the following steps:

1. The largest rectangle is positioned first.
2. The other rectangles are placed one by one, in the order of their closeness to the largest rectangle, as calculated from the positions described in the template. See Fig. 11a. Here, closeness is evaluated according to the distance between the positions of each pair of rectangles as described in the template.

5.3 Order of Referring to Triangles

While the mesh-edge-based algorithm calculates candidate positions on the mesh edges, the template-based algorithm needs more candidate positions because of the additional conditions. Therefore, while the mesh-edge-based algorithm refers to the mesh edges to calculate the candidate positions, the template-based algorithm refers to the triangles and calculates the candidate positions inside the triangles. The template-based algorithm first visits the triangle that encloses the position of the rectangle currently being positioned as described in the template and then recursively visits adjacent triangles, as shown in Fig. 11b. The order makes it possible to quickly find a suitable position for the rectangle. Also, the algorithm skips visiting a triangle if all of the distances between the position of the rectangle and the three vertices of the triangle are not far enough apart so that it can reduce unnecessary computation. It calculates the candidate positions on the lines connecting the vertices and the edges of the triangle, where the lines divide the angles of the vertices in regular parts. The candidate positions are set where the rectangle currently being placed can touch the previously placed rectangles without creating unnecessary gaps, as shown in Fig. 12. The number of candidate positions is defined so that intervals between adjacent candidate positions are constant.

5.4 Evaluation of Candidate Positions

While examining the trial placement of a rectangle at a candidate position, the template-based algorithm decides that:

1. If the rectangle would overlap with any of the previously placed rectangles, the rectangle cannot be placed there.
2. Otherwise, the template-based algorithm calculates the value $aA + rR + dD$. After trying the placement for all of the candidate positions, it places the rectangle where $aA + rR + dD$ is smallest.

The $aA + rR + dD$ values will increase while referring to triangles in the above order. This means that the extension can skip examining many of the distant triangles.

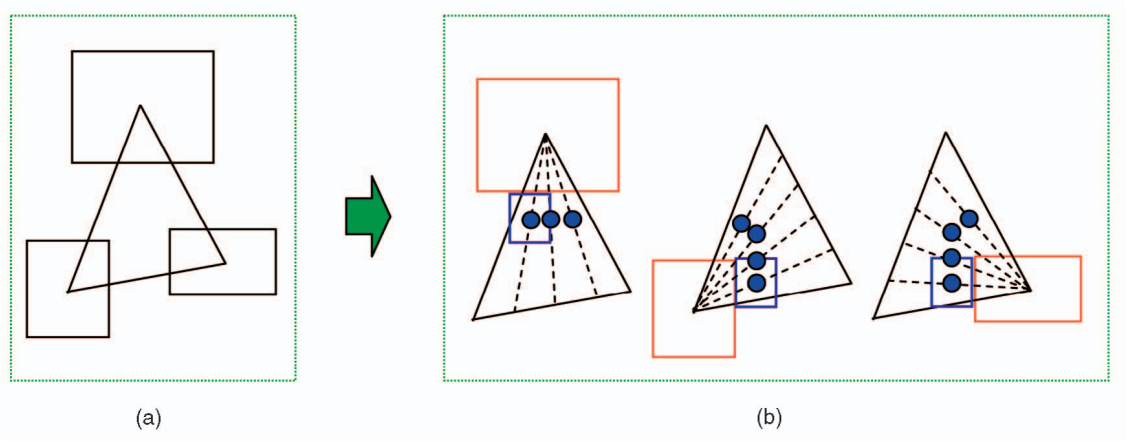

Fig. 12. Calculation of candidate positions. (a) A triangle and three rectangles on the vertices of the triangle. (b) Candidate positions that are generated where the rectangle currently being placed touches one of the previously placed rectangles.

6 Experiments

We implemented the proposed algorithms in Java and performed tests on an IBM IntelliStation Z-Pro (Pentium III 933 MHz) with Windows 2000.

The proposed technique aimed to represent data items as equishaped icons without overlapping each other. Quantum Treemap [1], [2] therefore seems one of the most similar approaches. We compared the results of our algorithms with those of Quantum Treemaps. We implemented Quantum Treemaps by applying the Squarified Treemap [4] and Strip Treemap [2] before the quantization of rectangular subspaces. Here, the proposed algorithms allow leaf nodes and nonleaf nodes to be mixed under the same nonleaf nodes. It is unclear whether Quantum Treemaps allow such data structures and, therefore, we inserted nonleaf nodes between the parent nonleaf node and leaf nodes.

We used four data sets generated from directory hierarchies of the following free software:

1. Java Development Kit 1.3.1.04 (http://java.sun.com/j2se/),
2. Apache Tomcat 4.1.12 (http://jakarta.apache.org/tomcat/),
3. Apache AXIS 1.0 (http://ws.apache.org/axis/), and
4. IBM Web Services Toolkit 3.3 (http://www.alphaworks.ibm.com/tech/webservicestoolkit/).

We generated tar files of the above software under the installed directory and then converted the table of contents in the tar files into our hierarchical data format and, finally, represented them by placing the data items onto displays. Table 1 shows the sizes of the four data sets.

We displayed several hierarchical data sets using the mesh-edge-based algorithm and measured the computation times, average aspect ratios, and average wasted space ratios. Here, an aspect ratio is calculated as the ratio between the lengths of the horizontal and vertical edges of a rectangle. The wasted space ratio is calculated as the ratio between the area of the rectangle and the sum of the areas of its child rectangles and icons. We compared the results of the proposed algorithm with those of Quantum Treemaps.

Table 2 shows the computation times. It shows that our technique places hierarchical data including thousands of nodes in reasonable computation times. However, it is much slower than Treemaps. Our implementation of the mesh-edge-based packing algorithm is still a greedy one that requires $O(n^2)$ computation for overlap check, but, as mentioned in Section 4.5, the complexity of the proposed algorithm can be reduced to $O(n^{3/2})$. Speed up of the packing algorithm is one of our future goals.

Table 3 shows the average aspect ratios of the rectangular subspaces. The results of the proposed algorithm are even better than those of Squarified Treemap. Table 4 shows the average wasted space ratios which are calculated in each rectangular subspace. The results of the proposed algorithm are fair, but, in most cases, Quantum Treemaps has the better results. Here, [2] does not seem to mention the quantization of deeply nested data; therefore, our implementation might independently quantize them under nonleaf nodes. This may cause different sizes of the leaf nodes, but we did not unify these sizes in our experiments. When we entirely unified the sizes of all leaf nodes as the smallest size, the wasted space ratios of Quantum Treemaps occasionally became more than 1 in our experiments.

TABLE 1
Sizes of the Four Data Sets

Data set	(1)	(2)	(3)	(4)
Number of leaf nodes	694	2,742	4,080	10,341
Number of non-leaf nodes	116	395	537	1,534

TABLE 2
Computation Times

	method	(1)	(2)	(3)	(4)
Time (sec.)	Mesh-edge-based packing	0.160	0.351	0.390	0.662
Time (sec.)	Quantum Squarified Treemap	0.025	0.045	0.050	0.070
Time (sec.)	Quantum Strip Treemap	0.005	0.015	0.055	0.051

TABLE 3
Average Aspect Ratios

	method	(1)	(2)	(3)	(4)
Aspect ratio	Mesh-edge-based packing	1.245	1.237	1.209	1.242
Aspect ratio	Quantum Squarified Treemap	2.327	2.876	3.189	2.928
Aspect ratio	Quantum Strip Treemap	2.522	7.848	3.824	4.792

TABLE 4
Average Wasted Space Ratios

	method	(1)	(2)	(3)	(4)
Wasted space	Mesh-edge-based packing	0.484	0.408	0.456	0.486
Wasted space	Quantum Squarified Treemap	0.424	0.366	0.332	0.386
Wasted space	Quantum Strip Treemap	0.479	0.461	0.369	0.398

Fig. 13 shows a simpler example of representation of the same data using the three techniques. This example also shows that Quantum Treemaps bring better space usage and our technique brings better aspect ratios. Here, the colors of the leaf nodes are simply calculated from their sequential numbers.

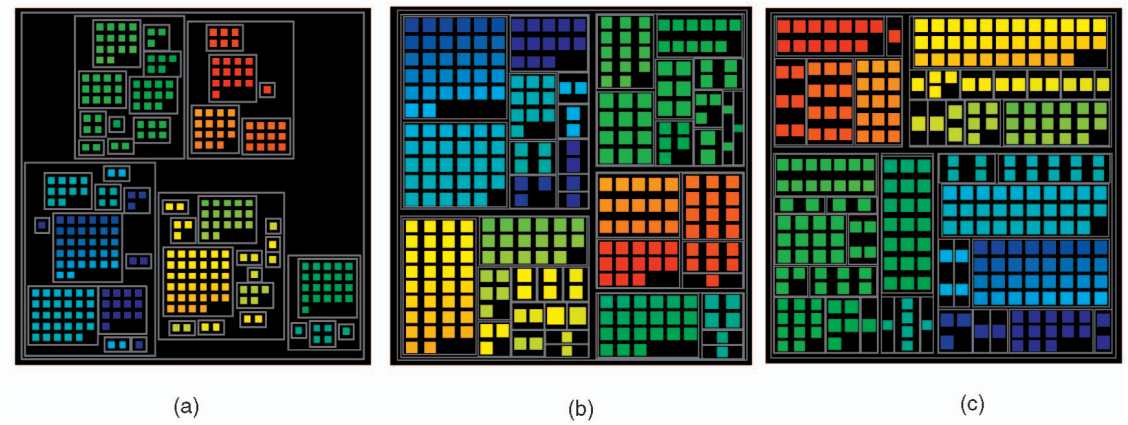

Fig. 13. Representation of the same hierarchical data using the three techniques. (a) Mesh-edge-based rectangle packing. (b) Quantum Squarified Treemap. (c) Strip Squarified Treemap.

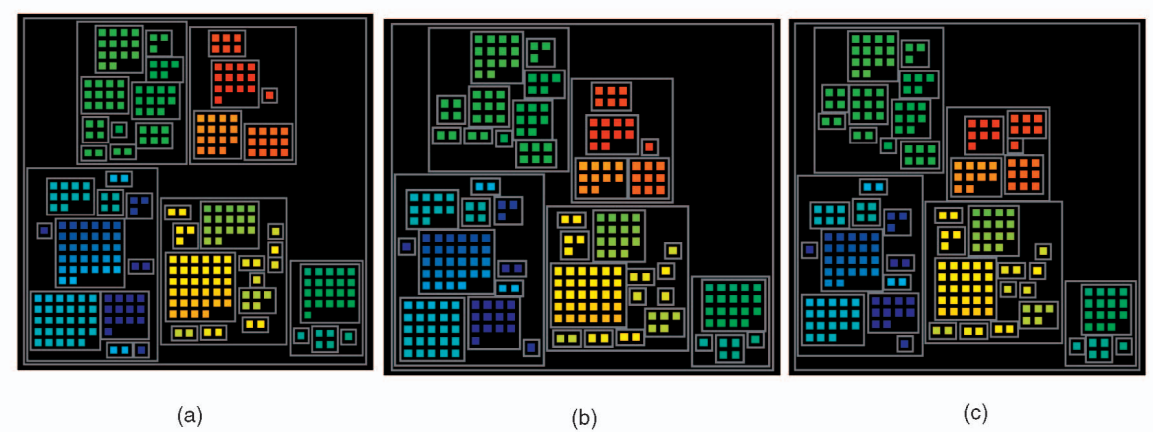

Fig. 14. Representation of a sequence of hierarchical data using the template-based algorithm. (a) First layout by mesh-edge-based algorithm. (b) Second layout by the template-based algorithm, referring to the template that describes the first layout. (c) Third layout by the template-based algorithm, referring to the template that describes the second layout.

TABLE 5
Average and Worst Values of Distances D of Corresponding Nodes between Previous and Next Layouts

method	Average (between first and second layout)	Worst (between first and second layout)	Average (between second and third layout)	Worst (between second and third layout)
Template-based packing	0.1209	0.8137	0.1176	0.8721
Quantum Squarified Treemap	0.4824	1.3638	0.4827	1.3831
Quantum Strip Tree map	0.4756	1.3655	0.4729	1.3255

Next, we intentionally generated time series data by repeatedly removing some leaf nodes from the original hierarchical data and represented the sequence of the data by using the template-based algorithm. Fig. 14 shows an example of the sequence of hierarchical data represented by template-based packing algorithm. Again, the colors of the leaf nodes are simply calculated from their sequential numbers. This example shows that the template-based packing algorithm achieves the stable display layout of time series data.

We also measured the stability of the display layout using the three techniques, with intentionally generated time series data by repeatedly removing some leaf nodes from the original hierarchical data, which was data set (2) shown in Table 1. Table 5 shows the average and worst values of the distances of the rectangles between the previous and next layouts of the time-series hierarchical data. Here, our implementation of Quantum Treemaps calculated the central positions of the subspaces in the normalized spaces, $(-1,-1)$ to $(1,1)$, and the distances of the corresponding subspaces between the previous and the next layout. This result shows that the template-based algorithm achieved more stable layouts.

7 Conclusion and Future Work

This paper presented a new hierarchical data visualization technique that represents the hierarchy by using nested rectangles. To place the rectangles in one display space, the technique applies two rectangle-packing algorithms that refer to triangular meshes connecting the centers of the rectangles to find suitable positions at which to place rectangles. Experimental results show that the technique packs rectangles with good aspect ratios and achieves stable layouts for time-varying data.

The following issues are area of our possible future work:

Various sizes and aspect ratios of leaf nodes. Since one feature of the proposed technique is even representation of leaf nodes, our implementation fixes their sizes and aspect ratios. However, our technique has the capability of representing leaf nodes with arbitrary sizes and aspect ratios by applying the rectangle-packing algorithm for all nodes without using any orthogonal grids. We would like to extend our implementation so that the sizes and aspect ratios of leaf nodes denote additional attributes.

Speed up. Our implementation of the packing algorithms is still too greedy and requires $O(n^2)$ computation for the overlap check, though, as mentioned in Section 4.5,

the complexity of the proposed algorithm can be reduced to $O(n^{3/2})$. We would like to implement the faster algorithm and measure the computation times again.

Comparison of triangulation methods. We have no theoretical justification that Delaunay triangulation is the best method for our purpose. We would like to implement other triangulation methods for the purpose of rectangle packing and compare them with Delaunay triangulation.

User study. It is interesting that Strip Treemap obtained good user study results [2] though other Treemaps occasionally obtained better numerical evaluations. We would like to show our technique to various users and measure the usability.

Applications. Though we did not describe it in this paper, we have applied the technique for the visualization of Web accesses [23] and real-time monitoring of distributed processes [24]. We would like to explore other applications that the proposed technique can be applied to. We think that semantics-based and design-based layout, which we have not tested with the template-based algorithm, may allow for other applications.

REFERENCES

[1] B. Bederson, "PhotoMesa: A Zoomable Image Browser Using Quantum Treemaps and Bubblemaps," *Proc. UIST 2001,* pp. 71-80, 2001.

[2] B. Bederson and B. Schneiderman, "Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies," *ACM Trans. Graphics,* vol. 21, no. 4, pp. 833-854, 2002.

[3] B.A. Bell and S.K. Feiner, "Dynamic Space Management for User Interfaces," *Proc. UIST 2000,* pp. 238-248, 2000.

[4] D.M. Bruls, K. Huizing, and J.J. Wijk, "Squarified Treemaps," *Proc. Data Visualization 2000 (Joint Eurographics and IEEE TCVG Symp. Visualization),* pp. 33-42, 2000.

[5] J. Carriere and R. Kazman, "Research Paper: Interacting with Huge Hierarchies beyond Cone Trees," *Proc. IEEE Information Visualization '95,* pp. 74-81, 1995.

[6] M. Chuah, "Dynamic Aggregation with Circular Visual Designs," *Proc. IEEE Information Visualization '98,* pp. 35-43, 1998.

[7] K. Freivalds, U. Dogrusoz, and P. Kikusts, "Disconnected Graph Layout and the Polyomino Packing Approach," *Proc. Graph Drawing 2001,* pp. 378-391, 2001.

[8] E. Gansner et al., "Improved Force-Directed Layouts," *Proc. Graph Drawing '98,* pp. 364-373, 1998.

[9] M.L. Huang et al., "A Fully Animated Interactive System for Clustering and Navigating Huge Graphs," *Proc. Graph Drawing '98,* pp. 374-383, 1998.

[10] T. Igarashi et al., "Adaptive Unwrapping for Interactive Texture Painting," *Proc. Symp. Interactive 3D Graphics 2001,* pp. 209-216, 2001.

[11] B. Johnson et al., "Tree-Maps: A Space Filling Approach to the Visualization of Hierarchical Information Space," *Proc. IEEE Visualization '91,* pp. 275-282, 1991.

[12] H. Koike, "Fractal Views: A Fractal-Based Method for Controlling Information Display," *ACM Trans. Information Systems,* vol. 13, no. 3, pp. 305-323, 1995.

[13] J. Lamping, R. Rao, and P. Pirolli, "The Hyperbolic Browser: A Focus+Context Technique for Visualizing Large Hierarchies," *J. Visual Languages and Computing,* vol. 7, no. 1, pp. 33-55, 1996.

[14] J. Marks et al., "Design Galleries: A General Approach to Setting Parameters for Computer Graphics and Animation," *Proc. ACM SIGGRAPH '97,* pp. 389-400, 1997.

[15] H. Murata et al., "VLSI Module Placement Based on Rectangle-Packing by the Sequence-Pair," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 15, no. 12, pp. 1518-1524, 1996.

[16] A.J. Quigley et al., "FADE: Graph Drawing, Clustering and Visual Abstraction," *Proc. Graph Drawing 2000,* pp. 197-210, 2000.

[17] J. Rekimoto, "The Information Cube: Using Transparency in 3D Information Visualization," *Proc. Third Ann. Workshop Information Technologies & Systems,* pp. 125-132, 1993.

[18] B. Shneiderman and M. Wattenberg, "Ordered Treemap Layouts," *Proc. IEEE Information Visualization Symp. 2001,* pp. 73-78, 2001.

[19] S.W. Sloan, "A Fast Algorithm for Constructing Delaunay Triangulation in the Plane," *Advances in Eng. Software,* vol. 9, pp. 34-55, 1987.

[20] T.C. Sprenger et al., "H-BLOB: A Hierarchical Visual Clustering Method Using Implicit Surfaces," *Proc. IEEE Visualization 2000,* pp. 61-68, 2000.

[21] J. Stasko and E. Zhang, "Focus+Context Display and Navigation Techniques for Enhancing Radial, Space-Filling Hierarchy Visualizations," *Proc. IEEE Information Visualization Symp. 2000,* pp. 57-65, 2000.

[22] G.J. Wills, "NicheWorks—Interactive Visualization of Very Large Graphs," *J. Computational and Graphical Statistics,* vol. 8, pp 190-212, 1999.

[23] Y. Yamaguchi, T. Itoh, Y. Ikehata, and Y. Kajinaga, "Interactive Poster: Web Site Visualization Using a Hierarchical Rectangle Packing Technique," *Proc. IEEE Symp. Information Visualization 2002 Interactive Poster Session,* 2002.

[24] Y. Yamaguchi and T. Itoh, "Visualization of Distributed Processes Using 'Data Jewelry Box II' Algorithm," *Proc. Computer Graphics Int'l 2003,* 2003.

Takayuki Itoh received the BS, MS, and PhD degrees from Waseda University in 1990, 1992, and 1997. He is a research staff member at IBM Research, Tokyo Research Laboratory. He is also a visiting assistant professor at Kyoto University, Academic Center for Computing and Media Studies. His research interests are in the area of geometric modeling, computer graphics, CAD/CAE, scientific and information visualization, Web Services, and XML security. He is a member of the ACM and the IEEE Computer Society.

Yumi Yamaguchi received the BS and MS degree from the Department of Information Sciences at Ochanomizu University in 2000 and 2001. She is a researcher at IBM Research, Tokyo Research Laboratory. Her research interests are in the area of scientific and information visualization and Web Services.

Yuko Ikehata received the BS degree from the Department of Information Technology at Ritsumeikan University in 1997 and the MS degree from the Graduate School of Computer Science at Keio University in 2000. She is a researcher at IBM Research, Tokyo Research Laboratory. Her research interests are in the area of collaboration systems, education systems, graphical user interfaces, and information visualization.

Yasumasa Kajinaga received the BS, MS, and PhD degrees from the University of Tokyo in 1994, 1996, and 1999, respectively. He is a reseacher at IBM Research, Tokyo Research Laboratory. His work ranges over various areas in recent computer engineering, such as optimization, information visualization, and performance of Web services and its security.