August 29, 2001
RT0427
Human-Computer Interaction 8 pages

Research Report

Data Jewelry-Box: A Graphics Showcase for Large-Scale Hierarchical Data Visualization

Takayuki ITOH, Yasumasa KAJINAGA, Yuko IKEHATA, Yumi YAMAGUCHI

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

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

Data Jewelry-Box: A Graphics Showcase for Large-Scale Hierarchical Data Visualization

Takayuki ITOH Yasumasa KAJINAGA Yuko IKEHATA Yumi YAMAGUCHI
IBM Research, Tokyo Research Laboratory
1623-14 Shimotsuruma, Yamato-shi, Kanagawa 242-8502 JAPAN
{itot, kajinaga, ikehata, yyumi}@jp.ibm.com

ABSTRACT

We see many kinds of large-scale hierarchical data in our daily lives, such as file systems of computers, company organizations, and category-based Web search sites such as Yahoo. Most of the GUIs for these data sources first represent a higher level of the data, and provide an interface so that users can select an interesting portion of the data and locally explore the lower levels. On the other hand, there are not many visualization techniques that give an overview of the data by placing all of the lower-level data onto a display region. How can we implement such a technique that represents all the data in one display, just like the showcases of a jewelry store displays all the jewels in the shop? That is the motivation of this research.

The report presents a visualization technique that places all of the lowest-level portions of a hierarchical data set on a display space. It first groups icons that denote the lowest-level data, and then generates rectangles that enclose each group of icons. It repeats the process of generating rectangles that enclose the lower-level rectangles, until the highest-level rectangles are enclosed by the largest rectangle. To use the display space most reasonably, our technique efficiently searches for gaps where rectangles can be located without overlapping adjacent rectangles. We use Delaunay triangular meshes that connect the centers of the rectangles to quickly find the gaps.

Keywords

Visualization, Hierarchical data, Delaunay triangular mesh, Rectangle packing.

1. INTRODUCTION

The report presents a new visualization technique for large-scale hierarchical data (i.e. data that contains more than thousands of data elements), which places all of the lowest-level data onto a display space. The technique is especially suitable to get a quick overview of all of the given hierarchical data.

There are various kinds of hierarchical data around us. For example, computer file systems, human resources and organizations in companies, financial or product data, processes of parallel computers, and category-based Web search sites such as Yahoo. Many user interfaces for such hierarchical data first display the top level of the data, and provide functions for manually exploring lower-level data. These interfaces are basically easy to understand, but are not always convenient for large-scale data. It often requires many clicks when the hierarchy is deep. In addition, it often requires a lot of scrolling when the data set is large and it is difficult to show all of it in one window.

For example, typical file system viewers first show the top-level directories, and then users manually select and display lower-level directories and files. While users are familiar with such tree-based user interfaces, the users may have troubles such as shown in Figure 1. Many clicks are required to explore deep directory structures. They also must scroll frequently when the data contains a very large number of files or directories, so that it is difficult to display all of them in one window.

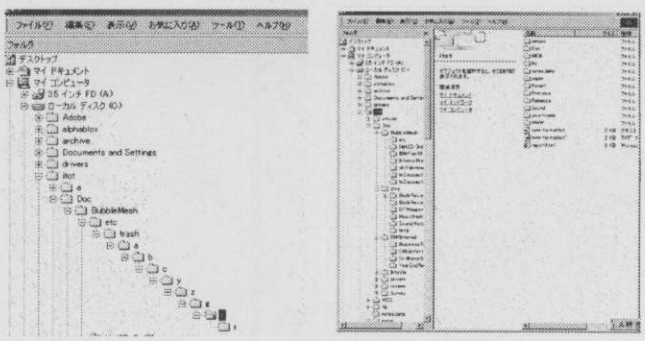

Figure 1. Example of tree-based file system viewers. (Left) It often requires many operations to explore a deep hierarchy. (Right) It is often difficult to display all files or directories of interest.

Similar problems may occur with other kinds of user interfaces for hierarchical data. For example, many clicks may be required to find interesting Web sites via category-based Web search sites.

How can we solve the above problems? Our solution is a visualization technique that displays large-scale and deep hierarchical data in one display area.

In contrast to many existing tree-based data visualization techniques, our visualization technique displays such hierarchical data in a manner like a Venn diagram. As shown in Figure 2, the technique represents hierarchical

data by using a set of nested rectangles. The technique first represents the lowest-level data as icons, and then encloses them in a rectangle. By repeating the process from the lowest level to the highest level, the technique places all of the data onto a display area.

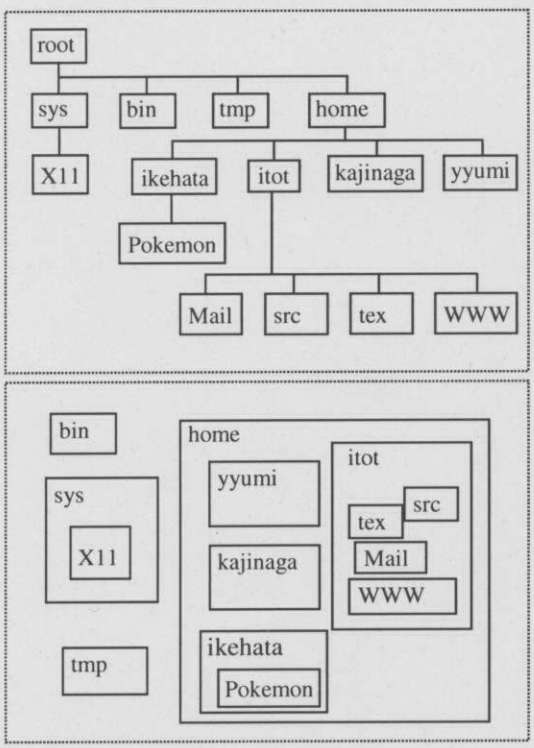

Figure 2. Comparison of existing and new visualizations. Both show the same data. (Top) Existing tree-based visualization. (Bottom) Our visualization.

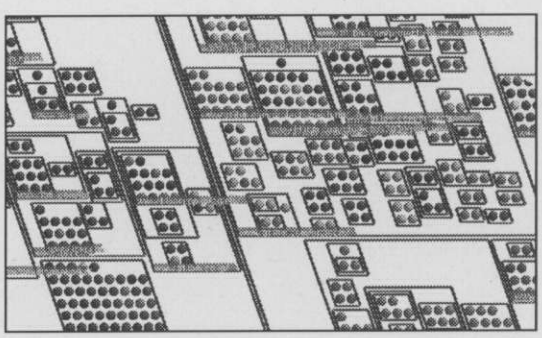

Figure 3. Example of our visualization.

Figure 3 is an example created by an implementation of our visualization technique. The example represents the lowest-level data by colored spherical icons. It represents hierarchies by rectangles enclosing a set of icons. Nested rectangles denote the depth of the hierarchies. The technique places all the data onto a display space at one time. This provides a useful overview of large-scale hierarchical data, just like the showcases in jewelry stores show all the jewels while arranging and categorizing the jewels. We think that our visualization is useful for analysis, retrieval, and monitoring of various kinds of hierarchical data.

The hierarchical data considered in this report is not assumed to have any geometric or geographic values. Data layout techniques are therefore necessary for visualization. In the case of our visualization style, a set of nested rectangles must be properly packed in a display region so that the display region is compact.

The packing problem is well known in the field of VLSI circuit design [Mur96], 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 generic 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 minimize the layout space, but only need to find a nearly optimized configuration within several seconds in order to provide an interactive visualization.

This report presents a new algorithm for efficiently packing a set of rectangles. The algorithm is used for our hierarchical data visualization technique. This report also shows results of the visualization technique, and finally compares the technique with previously reported hierarchical data visualization techniques.

2. OVERVIEW OF THE PROPOSED LARGE-SCALE HIERARCHICAL DATA VISUALIZATION

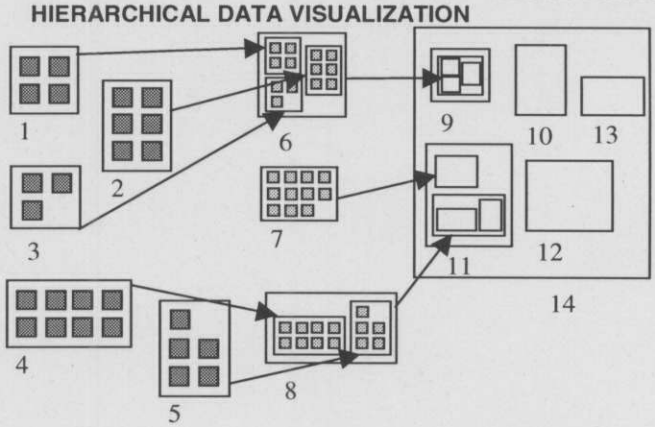

Figure 4. Algorithmic overview of the layout process for hierarchical data. Our algorithm first packs the lowest-level data, and then repeats a similar process for the higher levels. Numbers in this figure denote the order of the layout process.

The key technique of our hierarchical data visualization is the placement calculations for icons and rectangles.

Figure 4 shows an illustration of the order of the layout. Our algorithm first packs icons (square dots in Figure 4) that denote the lowest-level data, and then generates rectangles that enclose the icons. 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 algorithm places all of the data onto the layout area.

Figure 5 shows the tree structure of the hierarchical data we assume as input data, and the order of layout in our algorithm. To define the order of the layout process, the algorithm 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 the traversal.

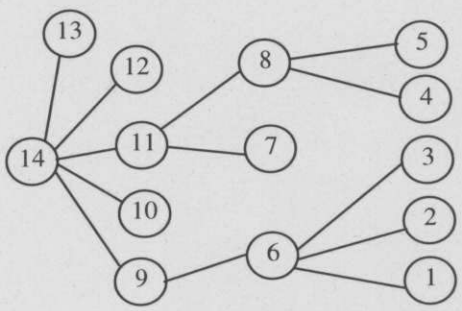

Figure 5. Definition of the order of the layout using a breadth-first traversal. A node of the tree structure in this figure denotes a rectangle in Figure 4. The hierarchal structure and the ordering in this figure correspond exactly to Figure 4.

3. ALGORITHM FOR EFFICIENTLY PACKING A SET OF TRIANGLES USING DELAUNAY TRIANGULAR MESHES

3.1 Algorithm overview

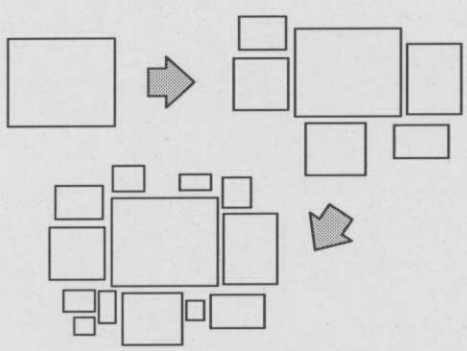

Figure 6. Illustration of layout of rectangles using our algorithm. The algorithm first places the largest rectangle, and then locates the others in the order of their areas, while it searches for gaps where they can be located.

Let us assume that n rectangles are given, and our algorithm positions all of them onto an xy-plane while it tries to minimize the layout area. Here, we also assume that all edges of the rectangles are parallel to the x-axis or y-axis. Under these assumptions, our algorithm places the rectangles by using the following procedure, as shown in Figure 6:

1. Sort the given rectangles in the order of their areas.
2. Repeat the following process for rectangles in the sorted order, starting from the largest rectangle:

(2a) Search for a region where the next rectangle can be placed without overlapping previously placed rectangles, and without enlarging the layout area.

(2b) If a position is found, place the rectangle there.

(2c) If no position is found, enlarge the layout area so the rectangle can be placed without overlapping any previously located rectangles, and place the rectangle there.

Since this report targets the development of an interactive visualization technique, our algorithm favors accelerating the rectangle packing process rather than minimizing the layout space. Following this strategy, this report presents an algorithm that efficiently finds gaps and places rectangles there. The algorithm generates a triangular mesh that connects the centers of the positioned rectangles, and uses the mesh to efficiently find gaps.

Here we describe the triangular mesh generated by the proposed algorithm. It first locates the largest rectangle at the center of the layout area. It then generates a rectangular region that entirely encloses the located rectangle. Let the region be R here. It then generates a triangular mesh that connects the center of the positioned rectangle and the four corners of R, as shown in Figure 7(left). The algorithm updates the triangular mesh whenever a new rectangle is placed in R, and updates the triangular mesh so that the mesh connects the centers of all the positioned rectangles. When k rectangles are positioned, the mesh consists of $(k+4)$ vertices, as shown in Figure 7(right).

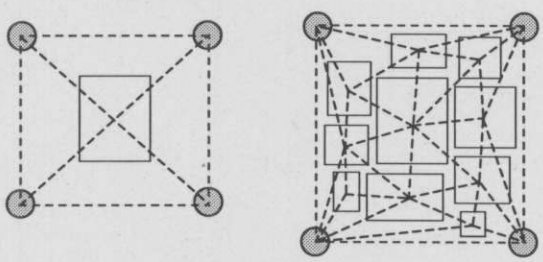

Figure 7. (Left) Initial configuration of the triangular mesh. (shown as dotted lines.) (Right) The triangular mesh that connects centers of positioned rectangles and four corners of R.

3.2 Search for gaps where rectangles can be placed

To search for gaps where rectangles can be placed, the proposed algorithm picks suitable triangles by using the following two strategies:

[Strategy 1] It first selects larger triangles, since it is easier to place rectangles inside larger triangles.

[Strategy 2] It next extracts interior triangles, since it is easier to place rectangles without enlarging the layout space if the rectangles are located inside interior triangles.

Following the above strategies, our algorithm scans the triangles one-by-one, and tries to place the current rectangle inside each of the triangles. The algorithm counts the number of corner vertices, c, for each triangle, and classifies all triangles according to c, where $c=0,1,2,3$. Here, corner vertices are the corners of R, the enclosing rectangle. Figure 8(left) shows an example of the distribution of c. This figure shows that interior triangles have the smaller c values.

The algorithm traverses the triangles grouped by c values, starting with c=0, in the order of their sizes. We calculate the radii of the inscribed circles of the triangles to estimate the sizes of triangles. The algorithm then places the rectangle when the current triangle satisfies the following two conditions:

[Condition 1] The rectangle can be placed inside the triangle without overlapping any previously placed rectangles.

[Condition 2] The rectangle can be placed inside the triangle without extending beyond the current layout area.

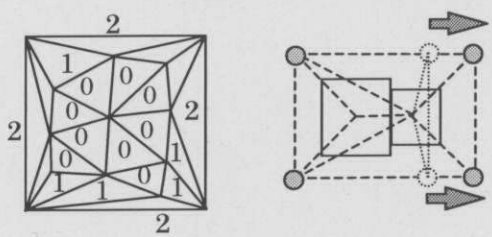

Figure 8. (left) Values of c for all triangles. (right) Moving the corners of R when the current rectangle placement requires enlarging the layout region R.

If the current triangle does not satisfy both conditions, the algorithm selects the next triangle and checks if it satisfies the conditions. The first triangle which satisfies only [Condition 1] is recorded for the later region enlargement procedure if no triangles satisfy both conditions. If no $c=0$ triangles satisfy both two conditions, the algorithm continues with the $c=1$ triangles, then the $c=2$ triangles, and finally the $c=3$ triangles. (Note that usually no $c=3$ triangles are produced.)

If no triangles satisfy the two conditions, the algorithm places the rectangle in the triangle mentioned above, the first one which satisfied [Condition 1], which causes the rectangle to extend beyond R. In this case the algorithm enlarges R by moving the corners, as shown in Figure 8(right).

3.3 Calculation of the position of a rectangle inside a triangle

When the algorithm tries to place a rectangle inside a triangle, it calculates the position of the rectangle using the following two heuristics:

[Heuristics 1] It is better that if rectangle is close to the center of the triangle, because it is more likely to overlap with other rectangle if it is located near edges or vertices of the triangle.

[Heuristics 2] It is better if the rectangle touches previously placed rectangles so that unnecessary gaps are avoided.

Based on the above heuristics, the algorithm first generates lines that connect the center of the triangle and three vertices. It then places the center of the rectangle so that it is on one of the generated lines, and the rectangle touches one of the previously placed rectangles, as shown in Figure 9. After these candidate positions are found, the algorithm checks if the rectangle overlaps with any other rectangles which have already been placed. If it does not overlap at one of the positions, the algorithm decides to place the rectangle there.

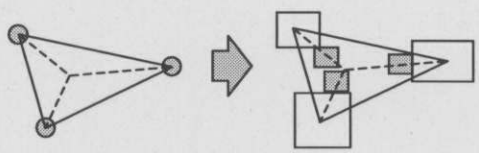

Figure 9. Location of a rectangle inside a triangle. The algorithm places the center of the rectangle on one of the lines which connects the center of the triangle and one of its vertices, and touching one of the previously placed rectangles. This figure shows three possible locations.

3.4 Local modification of triangular mesh after the addition of each rectangle

After the proposed algorithm places a rectangle by using the above process, it updates the triangular mesh by adding the center of the placed rectangle to the mesh. This process first connects the center of the rectangle to the three vertices of the triangle specified by the process described in Section 3.2, as shown in Figure 10(left). The process then locally modifies the mesh so that the mesh satisfies the Delaunay condition, as shown in Figure 10(right). The Delaunay condition is:

The circumscribing circle for each of the triangles in the mesh does not include a vertex from any other triangle of the mesh.

This process is quite similar to the Incremental Delaunay triangulation algorithm [Slo87].

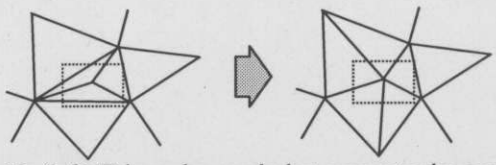

Figure 10. (left) Triangular mesh that connects the center of a rectangle and the three vertices of the triangle that encloses the center. (right) Locally modified mesh satisfying the Delaunay condition.

4. EXPERIMENTS

This section shows some examples using our visualization algorithm. We implemented the algorithm in a combination of C++ and Java, and performed the tests on an IBM IntelliStation Z-Pro (Pentium III 933 MHz) with Windows 2000.

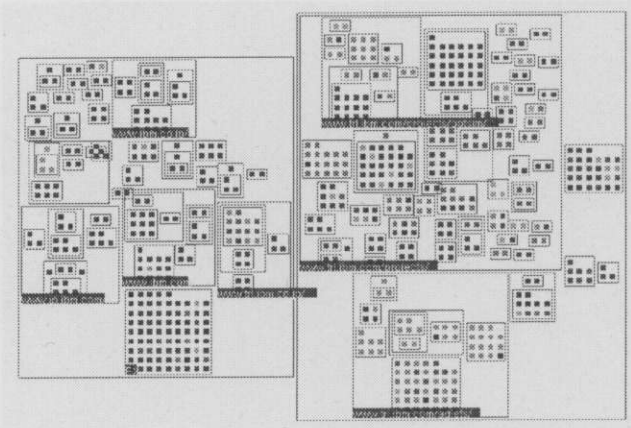

Figure 11. An example of our visualization technique.

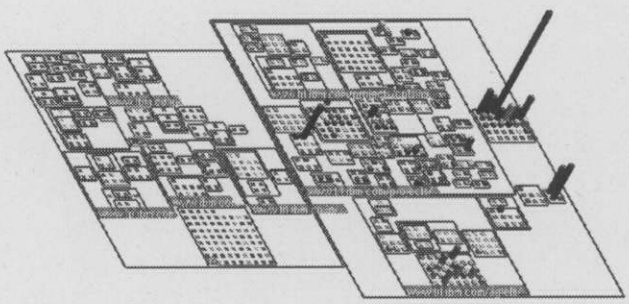

Figure 12. An example adding a third dimension of information to Figure 11.

Figure 11 shows an example of a Web sitemap as represented by our algorithm. In this example Web pages are clustered according to the URL structure and represented as square icons. Their colors denote an attribute of the Web pages, the date of last update. There are 835 icons and 160 rectangles. Our algorithm required 0.3 seconds to place all icons and rectangles.

Figure 12 shows an example in which our visualization algorithm represents another attribute by using the heights of the icons. In this case we mapped the access frequency of the Web pages to the heights of icons.

Figure 13 shows another example of a Web sitemap that includes 5,684 icons and 892 rectangles. Our algorithm required 2.5 seconds to find locations for all of the data.

Other examples are available at the following Web site:

http://www.trl.ibm.com/projects/webvis/

We are currently working on the development of a Web site visualization system using the proposed techniques. For example, the system represents which Web sites have hot information by highlighting the icons that denote the newest Web pages, frequently accessed Web pages, or Web pages highly scored by search engine or other Web applications. It can also monitor the load on Web servers, by highlighting icons that denote heavily accessed Web pages.

Our implementation can be also applied to various other kinds of hierarchical data. We think that it can be applied to collections of documents and images, personnel and organizations, financial and product data, scientific data, and so on.

5. COMPARISON

This section first describes the characteristics of the proposed technique as follows:

- It is suitable when users want an overview of the lowest-level data. On the other hand, it is not always suitable when users first want to see higher-level data.
- Since it locates all of the data onto an xy-plane, it is suitable for users who are not skilled with 3D computer graphics.
- It can represent multiple attribute values by varying the areas, heights, shapes, and colors of the icons.
- Our implementation only needs a few seconds to position thousands of data points onto a layout area. Computation times are usually lower than the times required to read the input data.

Reflecting the above characteristics, this section compares the new technique with previously reported hierarchical data visualization techniques.

[Tree-based visualization]

Tree representation is the most popular hierarchical data visualization technique, used in many programs such as Windows Explorer. Several variations, such as the Hyperbolic Tree [Lam96], Cone Tree [Car95], and Fractal Views [Koi95], have been reported for the interactive visualization of large-scale hierarchical data sets. Essentially tree-based techniques are most suitable for visualizing the higher-level data first, and then exploring the lower-level data according to users' operations. This approach is therefore different from our technique, since this new approach is suitable for providing an overview of the lowest-level data.

[Space-filling visualization]

Treemaps [Joh91], which represents hierarchical data as nested column charts, are a well-known space-filling approach for hierarchical data visualization. The space-filling approach has shape-related limitations for the lower-level data, and therefore it is difficult to clearly display or control the display of the lowest-level data.

[Semitransparent 3D visualization]

Information cubes [Rek93], which positions data in nested semitransparent hexahedrons, is a well-known 3D hierarchical data visualization approach. It has limitations in that it requires a graphics environment supporting 3D semitransparent representation, and that users' need 3D manipulation skills. In addition, it is unclear if this is a fully interactive visualization approach, since the computation times for 3D data layout are not described in the paper.

6. CONCLUSION

This report presents a visualization technique that places entire hierarchical data sets in a display area. It represents the lowest-level data by icons, and the hierarchy by nested rectangles. It positions all of the data by repeating a rectangle packing process starting from the lowest levels toward the highest level. This paper also presented an algorithm using Delaunay triangular meshes to efficiently place rectangles while avoiding overlapping rectangles and while reducing the layout space.

The following topics will be pursued in our future work:

- Implementation for even larger hierarchical data sets (i.e. more than million icons).
- Data layout with constraints so that user-specified pairs of clusters are located closer together. This can be done by modifying the order of packing rectangles, so that constrained pairs can be positioned at the same time.
- Seamless visualization of time-varying hierarchical data. Mesh modification techniques may be useful for the seamless update of data layout.
- Interactive exploration and layout of hierarchical data. The proposed technique places the data starting from the lowest level towards the highest level, however, it is also useful if the technique first places the highest-level data, and then places user-specified portions of lower-level data interactively.
- Algorithms for constructing hierarchies to allow applying these techniques for non-hierarchical data.
- Evaluations of the rectangle packing algorithm. For example, computation time and effective usage of the layout areas would be important metrics for evaluation.
- GUIs and system development using the proposed techniques. Currently we are working on developing a visualization system for Web sites.

ACKNOWLEDGEMENTS

We thank Kohichi Kajitani, Masaki Aono, Keisuke Inoue, Atsushi Yamada, Akira Tajima, and Jun Doi of the IBM Tokyo Research Laboratory for their encouragement and helpful advice.

REFERENCES

[Car95] Carriere J., et al., Research Report: Interacting with Huge Hierarchies Beyond Cone Trees, *IEEE Information Visualization '95*, pp. 74-81, 1995.

[Joh91] Johnson B., et al., Tree-Maps: A Space Filling Approach to the Visualization of Hierarchical Information Space, *IEEE Visualization '91*, pp. 275-282, 1991.

[Koi95] Koike H., Fractal Views: A Fractal-Based Method for Controlling Information Display, *ACM Trans. on Information Systems*, 13, 3, pp. 305-323, 1995.

[Lam96] Lamping J., Rao R., The Hyperbolic Browser: A Focus+context Technique for Visualizing Large Hierarchies, *J. Visual Languages and Computing*, 7, 1, 33-55, 1996.

[Mur96] Murata H., Fujiyoshi K., Shigetoshi N., Kajitani Y., VLSI Module Placement Based on Rectangle-Packing by the Sequence-Pair, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 15, 12, 1518-1524, 1996.

[Rek93] The Information Cube: Using Transparency in 3D Information Visualization, *Third Annual Workshop on Information Technologies & Systems*, pp. 125-132, 1993.

[Slo87] Sloan S.W., A Fast Algorithm for Constructing Delaunay Triangulation in the Plane, *Advances in Engineering Software*, 9, 34-55, 1987.

Figure 13. An example of large-scale data visualization.