TreeMaps

Agenda

- What is a Treemap?
- Treemap Basics
- Original Treemap Algorithm (Slice-and-dice layout)
- Issues for Treemaps
- Cushion Treemaps
- Squarified Treemaps
- Ordered Treemaps
- Quantum Treemaps
- Other Treemaps
- Application of Treemaps (PhotoMesa)
- Conclusion

What is a Treemap?

- Space filling technique to visualize hierarchical data

Basics

- Originally designed to **visualize files on a hard drive** and developed by Shneiderman and Johnson.
- Developed from a tree to **get over the limitation** that node and link diagrams use the display space inefficiently.
- The full display space is used to visualize the contents of the tree.
- Each node has a name (a letter) and an associated size (a number).
- **Each rectangle corresponds to an attribute of the data set.**
The size of leaves may represent for instance the size of individual files, the size of non-leaf nodes is the sum of the sizes of its children. The treemap is constructed via recursive subdivision of initial rectangle. The size of each sub-rectangle corresponds to the size of the node. Color and annotation can be used to give extra information about the leaves. The treemap is very effective when size is the most important feature to be displayed.

**Basics (cont.)**

**Aspect Ratio** (of a rectangle)
- Max (width/height, height/width).
  The lower the aspect ratio of a rectangle, the more nearly square it is.
  ie. a square has the ratio of 1.

**Layout of a treemap.**
- Set of rectangles that makes up the map.

**Original treemap algorithm** (Slice-and-dice layout)
- The simplest treemap algorithm.
- At each level, the orientation of lines switches. (vertical to horizontal or horizontal to vertical)
- Shading indicates the order.
- The size of each rectangle reflects the size of the leaf.
Issues for Treemaps

- Treemaps often fall short to **visualize the structure of tree.**
- **Aspect Ratio** difference. ie. Balanced tree. rectangles are difficult to compare and to select.
- Maintain **order**.
- Accommodate changes in the data.
- Difficult to find the data.

Cushion Treemaps Overview

- Compact, space-filling displays of hierarchical information based on recursive subdivision of a rectangular image space.
- **Take advantage of how human visual system is trained to interpret variations in shades as illuminated surface.**
- **Shading** is used to show insight in the hierarchical structure.
- During the subdivision **ridges are added per rectangle to form a surface** that consists of recursive cushion.
- Various coloring options are available to show the size, the level, and other attributes of the leaves.

Cushion Treemap Method to produce the surface.

1. Subdivide the interval and add a bump to each of the two subintervals.
2. Repeat the step above recursively. To each new sub-interval, add a bump again, with the same shape but half of the size of the previous one.
3. Use a parabola function to make the ridges have cushion-like shape.
4. If we interpret this curve as a side view of a bent strip, and render it as viewed from above, the bumps transform into a sequence of rides.

Cushion Treemap Algorithm

- Follows the original treemap algorithm.
- The main input consists of the root of the tree to be rendered, the initial rectangle to be used, and settings for height and parabola function.
- The main extension is that **during the generation of the rectangles, the surface is constructed.**
- The surface is bent in a direction.
- If the tree is a leaf, the cushion is rendered, else the direction is changed and its children are visited.
Cushion Treemap Summary

- Efficient: generation of image takes less than a second.
- Effective: the structure is visualized much more effective compared with original treemaps.
- Easy to implement: the algorithm is very simple.
- Emergence of thin, elongated rectangles.

Squarified Treemaps Overview

- For large hierarchical structure.
- Sub-rectangles have a lower aspect ratio.
- Display space is used more efficiently. The number of pixels to be used for the border is proportional to its circumference.
- Square items are easier to detect and point at.
- Comparison of the size of rectangle is easier when their aspect ratio are similar.
- The accuracy of presentation is improved.
Squarified Treemaps Method.

Squarification Key Ideas.

- Do not consider subdivision for all levels simultaneously.
  - Leads to an explosion in computation time.
- Produce square-like rectangles for a set of siblings, given the rectangle where they have to fit in, and apply the same method recursively.
  - The start point for the next level is close to square, which gives good subdivisions.

Squarification Treemap Algorithm

- **Split the initial rectangle.** Choose horizontal subdivision if the original rectangle is wider than high, vertical subdivision if the original rectangle is higher than wide.
- **Fill the left half by adding rectangles until we reach the optimum point of aspect ratio.**
- Start to process the right half based on above.
- Apply above recursively to work on the rest of rectangles.
  - The order in which the rectangles are processed is important. **Process large rectangles first.**
  - An optimal result can not be guaranteed.

Example of Squarification

- A rectangle with width 6 and height 4
- Subdivided into 7 rectangles (6,6,4,3,2,2,1)

Squarified Treemaps Frames

- Additional feature to improve the visualization of the structure.
- Derived from Nesting (maze-like images).
- **Filled in the borders with gray-shades based on a geometric model.**
Example of Squarification Treemap

Squarification Treemap VS Original Treemap

**Squarification Treemap Summary**

- Rectangles get forced to be **more square**.
- Represent sizes more precisely.
- **Frames** can improve the perception of structure.
- Changes in the data set can cause dramatic discontinuous changes in the layouts.
  - Hard to find items on the treemap by memory, decreasing efficacy for long-term users.
- **Orders are not preserved**.

**Ordered Treemap Overview**

- Ensures that **items near each other in the given order will be near each other** in the treemap layout.
- Maintaining relatively favorable aspect ratios of the constituent rectangles.
- Roughly **preserves the given ordering of the data (Index)**.
- Applicable to a situation where legibility, usability and smooth updating are important.
Ordered Treemap Algorithm

Inputs: Rectangle R and ordered list of items by index having specific areas

Recursive algorithm. At each step:

Select a pivot item (P)

If the width of R is greater than or equal to the height, divide R into four rectangles, $R_1$, $R_p$, $R_2$, and $R_3$ as shown below.

Ordered Treemap Algorithm

Selection of Pivot

- **Pivot-by-size**
  - The pivot is taken to be the item with the largest area since the largest item is the most difficult to place.

- **Pivot-by-middle**
  - The pivot is taken to be the middle item of the list since this is more likely to create a balanced layout.

- **Pivot-by-split-size**
  - Selects the pivot that will split $L_1$ and $L_3$ into approximately equal total areas.

Ordered Treemap Algorithm

(Cont.)

- Divide the list into 3 parts $L_1$, $L_2$, and $L_3$ to be laid out in $R_1$, $R_2$, and $R_3$.
- If the number of items is $\leq 4$, lay them out in either a pivot, quad, or snake layout,
- Apply recursively.
**Ordered Treemap Algorithm**

**Dividing the list L**

- Divide the items in the list, other than \( P \), into three lists, \( L_1, L_2, \) and \( L_3 \), such that \( L_1 \) consist of items whose index is less than \( P \) and \( L_2 \) have items having index less than those in \( L_3 \), and the aspect ratio of \( RP \) is as close to 1 as possible.

**Stopping condition**

- If number of items is \( \leq 4 \), lay them either in a **pivot**, **quad**, or **snake** layout.
- **Pick the best layout** whose average aspect ratio is closest to 1.

**Performance of the Ordered Treemap Algorithm**

- **Pivot-by-size**
  - \( n \log(n) \) average, \( n^2 \) worst case.

- **Pivot-by-middle**
  - \( n \log(n) \) worst case.
Example of Ordered Treemap

Ordered Treemap VS Original Treemap

Ordered Treemap Summary

- **Creates layouts that preserve order** and that update cleanly for dynamically changing data.
- Much **better aspect ratio than the original treemap** algorithm.
- **Left-to-right and top-to-bottom direction** in the layout.
- Tradeoff between aspect ratios and smoothness of layout changes.

Quantum Treemap

- Designed for laying out images or other objects of indivisible (quantum) size.
- ** Guarantees that every generated rectangle will have a width and height that are integral multiple of an input object size.**
- Always generates rectangles in which a grid of elements of the same size can be layed out.
- Applied for an image browsing software "PhotoMesa"
Quantum Treemap Algorithm

- Directly based on the Ordered Treemap Algorithm. ie. Uses pivot selection & stop condition
- Takes an elemental object dimension, and a list of numbers of objects. Returns a sequence of rectangles where each rectangle is large enough to contain a grid of the number of objects requested.
- Takes extra input that is the aspect ratio of the elements to be laid out in the Rectangle.
- Occasionally produces undesirable layouts due to too much wasted space.

Quantum Treemap VS Ordered Treemap

- Improved version of the ordered treemaps in terms of laying out images or objects.
- Generated rectangles will have a width and height that are integer multiple of an input object size.
- All the grid elements will align perfectly with rows and columns of elements running across the entire series of rectangles.

Other Treemaps

- Ordered Quantume Treemap
  - Combined the ideas of Quantum Treemaps and Ordered Treemaps.
- Bubblemaps
  - Lays out groups of quantum-sized objects in an ordered position with no wasted space per group, although there is a small amount of wasted space for the entire area.
- Strip Treemaps
  - Designed to produce highly readable displays. Layout has a consistently ordered set of rectangles while still maintaining good aspect ratios.
- StepTree
  - Extended into three dimensions by the simple expedient of stacking levels of the tree on top of each other in 3D space.
- Circular Treemaps
  - Don’t fill the available space completely.
  - Fill the available space to a varying degree which in the case of nested tree structures leads to the problem that circles of the same size could represent files (or folders) of a vastly different size.
Application of Treemaps

“PhotoMesa”

- A standalone application that supports **browsing of large sets of images**.
- Allows users to view multiple directories of **images in a zoomable environment**.
- Organizes images in a two-dimensional grid, where images with a shared attribute are grouped together.
- Apply the **quantum treemap, ordered treemap, and bubblemap** for images display.
- Originally written in Java, now written in .NET.
Conclusion

- Treemaps represent large hierarchical collections of two-dimensional quantitative data in a compact display.
  - One dimension is mapped to the area of the rectangles and the other is mapped to the color of a rectangle.
  - A label is placed in the rectangles.
- Each treemap algorithm has its merit and demerit.
  - Cushion Treemaps
  - Squarified Treemaps
  - Ordered Treemaps
  - Quantum Treemaps
  - Other Treemaps
- Application.
  - PhotoMesa
  - Stock Portfolio

References

- Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies
  - Benjamin B. Bederson
  - Ben Shneiderman
  - Martin Wattenberg
- Quantum Treemaps and Bubblemaps for a Zoomable Image Browser
  - Benjamin B. Bederson
- Ordered Treemap Layouts
  - Ben Shneiderman
  - Martin Wattenberg
  - http://www.ifs.tuwien.ac.at/~mlanzenberger/ps_infovis/ss03/stuff/auth/00963283.pdf

References (Cont.)

- Squarified Treemaps
  - Mark Bruls
  - Kees Huizing
  - Jarke J. van Wijk
- Cushion Treemaps: Visualization of Hierarchical Information
  - Jarke J. van Wijk
  - Huub van de Wetering
- Extending Tree-Maps to Three Dimensions: A Comparative Study
  - Thomas Bladh
  - David A. Carr
  - Jeremiah Scholl

References (Cont.)

- pebbles - using Circular Treemaps to visualize disk usage
  - Kai Wetzel