Assignment 1: Squarified Treemaps

Due: 9-22, Monday, 2014, 11:59pm (midnight)

In this assignment, you will be using Processing to parse a dataset, create a hierarchy from this dataset, visualize the hierarchy by implementing the Squarified Treemap algorithm, and add interactive features such as zooming and the ability to re-arrange the hierarchy. The key points of this assignment are to (1) implement an appropriate data structure for storing and traversing a hierarchy, (2) practice good programming strategies such that your visualization code can be easily reused and integrated with each other, and (3) learn to read an academic paper pertaining to a visualization design and be able to execute the described algorithm.

This assignment will have a "two-stage" grading scheme. For implementing the "basic requirements", you will at most receive a B. For implementing the "additional requirements", you can receive up to an A (or A+) for your efforts.

Basic Requirements:

1. Read the paper "Squarified Treemap" by Bruls, Huizing, and van Wijk (2000).
2. In Processing, do the following:
    a. Parse the input file and create a hierarchical data structure. For the basic requirements, your visualization will need to parse the *.shf files (e.g. hierarchy.shf). See the "Data Formats" section for descriptions of this file format.
    b. Implement the Squarified Treemap algorithm as presented in the paper (section 3 in the paper) to visualize the dataset.
        i. Subdivision algorithm
            1. The algorithm by Bruls is a little confusing. If you need help, see the Appendix section at the bottom of this document.
        ii. Hierarchical data:
            1. For each cell in the tree map, consider each cell as a new canvas.
            2. Repeat the subdivision algorithm above.
        iii. Note that you do not have to implement "cushions" but will need to implement a frame (section 4 of the Bruls paper). That is, there should be some visually distinguishing feature between levels of hierarchy.
        iv. Lastly, for the sake of consistency, make sure that the largest cell/node is on the upper-left hand corner, and the order is left-to-right, top-to-bottom (same as the algorithm in the paper).
    c. Allow mouse-over to highlight each node. When a cell is highlighted, it should change color.
    d. Add zooming functionality (zoom in, zoom out):
        i. Allow left-clicking a cell to zoom-in on the Treemap. Whatever cell was clicked should temporarily become the "root" node of the visualization. Be sure to keep track of the parent node for step ii.
        ii. Allow right-clicking anywhere in the Treemap to zoom-out to the parent node.

Additional Requirements:

1. Use Treemaps to visualize non-hierarchical data.
    a. Create a hierarchical data structure from the School of Engineering funding dataset (soe-funding.csv). This hierarchy should have the following properties:

i. It can be considered as a tree, where each node has a parent (except for the root node). A node can either be a leaf node (a node without children) or a tree node (a node with children).
ii. For this assignment, we will assume that a leaf node must contain data, and a tree node cannot contain data. For instance, if the dataset is the hierarchy of the file system on your computer, a leaf node will be a document, and a tree node will be a folder.
iii. The data associated with a leaf node will be a single value, in this case the "Total" of funding. In the file system example, this value can be the size of the document (e.g., number of kb to store the document).
iv. The resulting data structure should have at least three levels: "Year", "PI Department", and "Sponsor".

2. Create a bar-chart using Treemaps as bars.
    a. For example, if "Year" is the top level of your hierarchy, each "bar" should be a Treemap containing data only from that year.
    b. Trigger this change via a button in the interface.
    c. Add faceted navigation to re-order the hierarchy:
        i. Add a visual representation of the hierarchy order (e.g. a horizontal set of buttons [Level1] – [Level2] – [Level3]).
        ii. Add interactions that allow a user to re-arrange the order of the hierarchy (e.g. drag Level2 to Level3 to get [Level1] – [Level3] – [Level2]). Drag-and-drop is not a requirement. In the simplest case, you can also use a single button that toggles through all 6 possibilities.
        iii. Note: feel free to get creative with the hierarchy representation and interaction methods. As an example, see http://cdnlarge.tableausoftware.com/sites/default/files/snippets/679x500_treemapsbarcharts.jpg

Submission:

1. Submit your work using "provide" by September 22, Monday, 2014, 11:59pm (midnight). The assignment is called "a1". The syntax for using provide is
```
provide comp150viz a1 file1 file2 file3  or
provide comp150viz a1 * (to submit all files in a directory)
```
2. Name your "main" pde file "a1.pde"
3. As part of your submission, provide a file named "team.txt" where you:
    a. Give the name of all the team members
    b. Provide a 1-2 sentence description regarding what each member did in the project.
4. Note that each team only needs to make one submission. We will keep track of the individuals' grades based on the names provided in the team.txt file.
5. Extra credit will be given for creative and novel designs. If you plan on implementing cool features to your visualization, please document them in a separate file called "extracredit.txt"
6. For grading, your team will demo the project to the instructor(s). Tentative date for grading is September 26 (Friday). We will put up a signup sheet later.

Additional Note:

1. When grading your assignment, the TA and I could be using a different input file that we will create.

2. For grading the "additional requirements", we will use a dataset that has the same properties as the soe-funding.csv file. That is, we will also assume 4 values in each row, and that the first two values are strings (like "Sponsor" and "PI Department") and the last two are integers (like "Year" and "Amount").
3. You can submit two different visualizations, one specifically for satisfying the basic requirements and the other for the additional requirements.
    a. If you choose to submit two different visualizations, please be sure to zip each visualization separately (as opposed to submitting all the pde files together). That way it is easier for us to keep track of which file belongs to which visualization.

Data Formats:

1. The *.shf file format. This is a file format that I made up. SHF stands for simple hierarchy format. This file has 4 parts:
    a. First line: a single number that describes the number of leaf nodes in the data (call this number i)
    b. Lines 2 to i+1: each line contains two numbers: first number is the node ID of the leaf node, the second number is the value of the node
    c. Line i+2: a single number that describes all parent-child relationships in the data (call this number j)
    d. Lines i+2 to i+2+j: each line contains two numbers: first number is the ID of the parent node, second number is the ID of the child node
    e. Note that the purpose of (a) and (c) is such that you can write this parser quickly using for loops…
2. The soe-funding.csv file is a simple CSV (comma separated value) file. It has two parts:
    a. First line: the headers – four strings each separated by a comma
    b. The rest of the file: four values: (1) the name of the department (string), (2) the sponsor (string), (3) year (integer), (4) amount (integer)

Need Help?

If you have questions about this assignment, please send an email to **BOTH** the TAs and instructors. Alternatively, you can just drop by during the TAs or my office hours.

Appendix: Squarified Treemap Subdivision Algorithm (by Remco)

1. Compute the area of the canvas (call it canvas_area).
2. Compute the total values of items (call it total_value)
3. Compute the ratio of total_value vs. canvas_area (call it VA_ratio)
4. Sort the values in a tree node.
5. Compare the width and height of the canvas, choose the side that is shorter (call that side short_side)
6. Choose the largest value from all the values
7. Represent this value as a rectangle and call this rectangle c1. The area of c1 is computed based on VA_ratio.
8. Place c1 on short_side.
9. Compute the aspect ratio of c1. Call this aspect ratio ratio_c1.

10. Choose the next largest value (call it c2)
11. Try adding c2 to short_side. Compute the aspect ratio of c2. Call this aspect ratio ratio_c2
12. If ratio_c2 is better than ratio_c1 (in that c2 is more square than c1), then we accept adding c2 on the short_side. Goto line 10 (but replace the old c1 with c2).
13. If ratio_c2 is worse than ratio_c1 (in that c1 is more square than c2), then go to line 5. The new canvas is the remaining empty space.