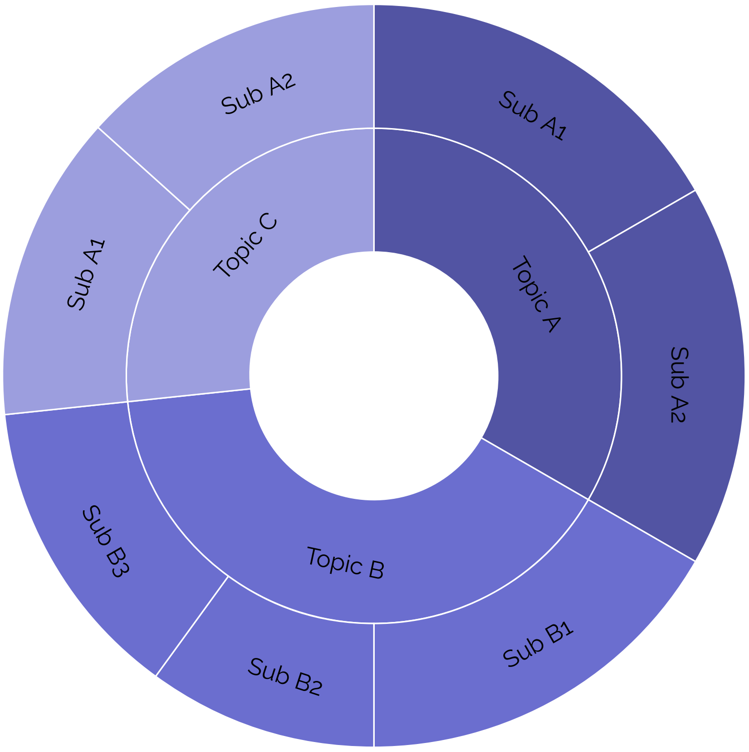
David Richards's Block f059c1f78f9c39d922b1c208815d18af
Updated July 7, 2017

Popular / About

Sunburst Tutorial (d3 v4), Part 2



Open

Sunburst Labels & an External json File

In this tutorial, we'll begin with the "no frills" sunburst from Tutorial 1. But we'll limit our detailed walk-through to the 2 new features:

1. properly-rotated labels
2. data loaded from external json file

On the bl.ocks.org page, scroll to the bottom to see the uninterrupted code of a Sunburst visual, based on d3 version 4. This tutorial builds on Part 1.

Do good! —David Richards

I'm not a fan of the default font for our sunburst labels. We'll add a style block to our html to clean it up.

```
<style>
@import url('https://fonts.googleapis.com/css?family=Raleway');

body {
  font-family: "Raleway", "Helvetica Neue", Helvetica, Arial, sans-serif;
}
</style>
```

The `<style>` block creates boasts a surprisingly powerful set of tools for to enhance our user-experience. The above lines import a Google Font for this page and then tell the page what to do if that font (or other fonts that we like) are not available. * `@import url('https://fonts.googleapis.com/css?family=Raleway')` gets the Raleway font from Google (fonts.google.com). * `body { font-family: "Raleway", "Helvetica Neue", Helvetica, Arial, sans-serif; }` tells our page to use Raleway first, but provides alternatives if it's not available.

Get the data

In the first tutorial we began with the web page and variable definitions. This time, we'll skip all of that and go right for the first new code: getting data from our *.json file.

```
d3.json("data.json", function(error, nodeData) {
        if (error) throw error;

    // Put the code that works with our data here.

});
```

d3.json is a super-simple d3 function that allows us to pull our data from a json file (d3 has other similar functions for csv, tsv, etc. files). We include the filename 2 arguments:

1. "data.json" (since we don't have any folder references, it assumes that this file is in the same directory as the current file)
2. A special *anonymous* function that returns either an error or the data as a variable, "nodeData". All of our code that processes and presents the data typically fits within this anonymous function block.

If you inspect the data.json file, you'll note that we've added some attributes (text, sentiment, source). You can ignore those for this tutorial. We'll use them in the future.

Calculate Each Arc

In this section we (1) find the root node of our data, and begin sizing process. "nodeData" is the data from our external json file. (2) Then we format the data in a way that our d3 Sunburst visualization expects, using the partition command. (3) Last, we'll calculate the size of each arc (we'll draw them later).

```
var root = d3.hierarchy(nodeData)
    .sum(function (d) { return d.size});

partition(root);
var arc = d3.arc()
    .startAngle(function (d) { return d.x0 })
    .endAngle(function (d) { return d.x1 })
    .innerRadius(function (d) { return d.y0 })
    .outerRadius(function (d) { return d.y1 });
```

The above code is identical to Tutorial 1. We've simply moved inside of our new data call, `d3.json()`.

Draw a Slice for Each Node

In this section, we append a "container" g element for each node in our data. Then we add draw the path element based on the arc variable that we calculated above. Last, we color the lines and the newly drawn slices.

```
g.selectAll('g')  // <-- 1
    .data(root.descendants())
    .enter().append('g').attr("class", "node")  // <-- 2
    .append('path')  // <-- 2
    .attr("display", function (d) { return d.depth ? null : "none"; })
    .attr("d", arc)
    .style('stroke', '#fff')
    .style("fill", function (d) { return color((d.children ? d : d.parent).
```

In the previous tutorial, we selected our non-existent `<path>` elements below and used the d3 Update Pattern to add a `<path>` for each node in our data. This time, we also want to create a visible label, as a `<text>` element in each slice. Sadly, `<path>` cannot contain `<text>` elements. So, instead of beginning with the element, we'll our container <g> element. Then we'll add both the `<path>` and the `<text>` elements to the <g> element.

To be precise, we've made the 2 changes to our code from Tutorial 1:

1. `g.selectAll('path')` --> `g.selectAll('g')`
2. `.enter().append('path')` --> `.enter().append('g').attr("class", "node").append('path')`

In step 2, we now append a <g> element for each data node. To that, we a class attribute ("node"). This will let us get a hold of just these elements later. *Now* we're ready to append the `<path>` element to the new `<g class='node'>` elements. In the end, each data node has an entry that looks like this:

```
<g class="node">
    <path d="..." style="..."></path>
</g>
```

The other lines of this block remains the same as Tutorial 1.

Add a Label for Each Node

Now we add our labels to each of the g elements that we created above. Then we rotate and position them properly.

```
g.selectAll(".node")  // <-- 1
    .append("text")  // <-- 2
    .attr("transform", function(d) {
        return "translate(" + arc.centroid(d) + ")rotate(" + computeTextRot
    .attr("dx", "-20")  // <-- 4
    .attr("dy", ".5em")  // <-- 5
    .text(function(d) { return d.parent ? d.data.name : "" });  // <-- 6
```

Now we'll add and populate the `<text>` elements with our data-driven titles.

1. `g.selectAll(".node")` like the last block, starts with the g variable that we created way above. This variable will select for us the single <g> element that we originally appended to our <svg> element. Then it looks for any child elements that have "node" as a class, and selects them. These are the newly created `<g class='node'>` elements from the block above.

2. `.append("text")` appends an empty `<text>` element to each `<g class='node'>` element

3. `.attr("transform", function(d) { return ...; })` adds a transform attribute to each our newly created `<text>` elements.

    1. `"translate(" + arc.centroid(d) + ")"` moves the reference point for this `<text>` element to the center of each arc (the variable we defined above). The centroid command from d3 computes the midpoint [x, y] of each arc.
    2. `"rotate(" + computeTextRotation(d) + ")"` then we'll rotate our `<text>` element a specified number of degrees. We'll do that calc in a separate function below.

4. `.attr("dx", "-20")` // Moves the text element to the left, which makes our labels look centered.

5. `.attr("dy", ".5em")` // Pulls our text element in closer to the center of the Sunburst, which makes our labels look centered.

6. `.text(function(d) { return d.parent ? d.data.name : "" })` returns the "name" attribute for each node, unless that particular node has no parents (the "root" node). In that case, it returns an empty string.

    1. This `d.parent` check is an alternative to the `d.depth` check that we did in Tutorial 1 for finding our *root* node.

Putting this block all together, each data node has an entry that looks like:

```
<g class="node">
    <path d="..." style="..."></path>
    <text transform="translate(105.5409906877519,-66.97834937237457)rotate(
        dx="-20" dy=".35em">Topic A</text>
</g>
```

This is the end of the `d3.json()` block.

computeTextRotation Function

The computeTextRotation function calculates the correct distance to rotate each label based on its location in the sunburst. It also avoids upside down labels. It takes a single argument, "d", which represents a single d3 Node (this function is called one time for each slice / node / label).

```
function computeTextRotation(d) {
    var angle = (d.x0 + d.x1) / Math.PI * 90;  // <-- 1

    // Avoid upside-down labels
    return (angle < 90 || angle > 270) ? angle : angle + 180;  // <--2 "lab

    // Alternate label formatting
    //return (angle < 180) ? angle - 90 : angle + 90;  // <-- 3 "labels as
}
```

Three math-rich lines in this function:

1. `var angle = (d.x0 + d.x1) / Math.PI * 90` calculate the angle (in degrees) of the label that will work for 1/2 of the labels.

    1. d.x0 = the beginning angle of this node / slice (in radians).
    2. d.x1 = the end angle of this node / slice (in radians).

2. `return (angle < 90 || angle > 270) ? angle : angle + 180` handles rotation to avoid any upside-down labels: If rotation angle is in the 1st or 2nd quadrants (top 1/2), leave the already calc'd angle. Otherwise, flip the text over so that the text appears right-side-up.

3. `return (angle < 180) ? angle - 90 : angle + 90` Alternatively, this line rotates the labels so that they appear in the traditional "spoke" formation. And it avoids any upside-down labels. It's currently commented out.

Nice! You've made it through 2 tutorials (or maybe you wandered directly into this one). Either way, great job on making it this far. You're now able to add labels to your sunburst. We've still just scratched the surface. If you're ready, join me for Tutorial 3.

index.html

```
<!DOCTYPE html>
<head>
    <title>Sunburst Tutorial (d3 v4), Part 2</title>
    <script src="https://d3js.org/d3.v4.min.js"></script>
</head>
```

```
<style>
@import url('https://fonts.googleapis.com/css?family=Raleway');

body {
  font-family: "Raleway", "Helvetica Neue", Helvetica, Arial, sans-serif;
}
</style>
<body>
    <svg></svg>
</body>

<script>

    // Variables
    var width = 500;
    var height = 500;
    var radius = Math.min(width, height) / 2;
    var color = d3.scaleOrdinal(d3.schemeCategory20b);

    // Size our <svg> element, add a <g> element, and move translate 0,0 to the center of the eleme
    var g = d3.select('svg')
        .attr('width', width)
        .attr('height', height)
        .append('g')
        .attr('transform', 'translate(' + width / 2 + ',' + height / 2 + ')');

    // Create our sunburst data structure and size it.
    var partition = d3.partition()
        .size([2 * Math.PI, radius]);

    // Get the data from our JSON file
    d3.json("data.json", function(error, nodeData) {
        if (error) throw error;

        // Find the root node of our data, and begin sizing process.
        var root = d3.hierarchy(nodeData)
            .sum(function (d) { return d.size});

        // Calculate the sizes of each arc that we'll draw later.
        partition(root);
        var arc = d3.arc()
            .startAngle(function (d) { return d.x0 })
            .endAngle(function (d) { return d.x1 })
            .innerRadius(function (d) { return d.y0 })
            .outerRadius(function (d) { return d.y1 });


        // Add a <g> element for each node in thd data, then append <path> elements and draw lines
        // variable calculations. Last, color the lines and the slices.
        g.selectAll('g')
            .data(root.descendants())
            .enter().append('g').attr("class", "node").append('path')
            .attr("display", function (d) { return d.depth ? null : "none"; })
            .attr("d", arc)
            .style('stroke', '#fff')
            .style("fill", function (d) { return color((d.children ? d : d.parent).data.name); });


        // Populate the <text> elements with our data-driven titles.
        g.selectAll(".node")
            .append("text")
            .attr("transform", function(d) {
                return "translate(" + arc.centroid(d) + ")rotate(" + computeTextRotation(d) + ")";
            .attr("dx", "-20") // radius margin
            .attr("dy", ".5em") // rotation align
            .text(function(d) { return d.parent ? d.data.name : "" });

    });


    /**
     * Calculate the correct distance to rotate each label based on its location in the sunburst.
     * @param {Node} d
     * @return {Number}
     */
    function computeTextRotation(d) {
        var angle = (d.x0 + d.x1) / Math.PI * 90;

        // Avoid upside-down labels
        return (angle < 120 || angle > 270) ? angle : angle + 180;  // labels as rims
        //return (angle < 180) ? angle - 90 : angle + 90;  // labels as spokes
    }

</script>
```

data.json

```
{
    "name": "TOPICS", "children": [{
        "name": "Topic A",
        "children": [{"name": "Sub A1", "size": 5, "text": "A story", "sentiment": 0.8, "source": "
            {"name": "Sub A2", "size": 5, "text": "A note", "sentiment": 0.3, "source": "dictionary
    }, {
        "name": "Topic B",
        "children": [{"name": "Sub B1", "size": 5, "text": "A vignette", "sentiment": 0.5, "source"
            {"name": "Sub B2", "size": 3, "text": "A tall-tale", "sentiment": 0.2, "source": "frier
            {"name": "Sub B3", "size": 4, "text": "A joke", "sentiment": 0.8, "source": "email"}]
    }, {
        "name": "Topic C",
        "children": [{"name": "Sub A1", "size": 4, "text": "A narrative", "sentiment": 0.2, "source
            {"name": "Sub A2", "size": 4, "text": "A chronology", "sentiment": 0.3, "source": "emai
    }]
}
```

LICENSE