CSCI 4210 U – Information Visualization Lab
8. Wordle

OntarioTech UNIVERSITY

PURPOSE

To create your own layout algorithm for word clouds, following the strategy employed in the Wordle text visualization.

TASKS

Wordle is the most popular visualization of text data. It was developed by Jonathan Feinberg as an improvement over the tag clouds used in websites like Flickr and del.icio.us. Wordle was not created following a rigorous visualization research process; its design ethos valued aesthetics more than accuracy. Before Wordle, tag clouds used to look like this:

The tags were simply words in regular lines of text, each with a different color and font size. If one word was much bigger than the rest of the words in a line, it caused a lot of blank space overhead, which doesn't look right. Compare that with a Wordle:

Wordles are more tightly packed. Note how smaller words are placed even inside bigger letters. This is because Feinberg did not settle for mere bounding boxes; he leveraged knowledge of glyph boundaries in his placement strategy. Despite needing a fairly sophisticated collision detection approach (otherwise it's just too slow), the overall algorithm is surprisingly naive.

In this lab you're asked to implement a Wordle. The pseudocode is as follows. A breakdown of each step and tips are given in the next section. Read all tips before you begin coding.
The data provided is a list of the top 100 most frequent passwords found in the 2009 leak of the RockYou gaming website, along with their frequencies.

Pseudocode

```
1. define a preferred position for words (e.g., centre of the screen)
2. scale the word (font-size proportional to word frequency, you can use
 d3.scaleLinear)
```

```
3. place the word in the preferred position
4. while it intersects any of the previously placed words
5.    move it one step along an ever-increasing spiral
```

Do the procedure above for every word, in order of importance (high-frequency words should come first).

STARTER CODE HAS BEEN PROVIDED, BUT YOU DON'T NEED TO USE IT IF YOU'RE NOT COMFORTABLE WITH IT.

Tips

1. How do I put a word on the screen?

Define an SVG element in the HTML (or create one programmatically), then append a SVG Text element:

```
var myText = d3.select("svg")
               .append("text")
               .attr("x", x)
               .attr("y", y)
               .style("font-family", "sans-serif")
               .style("font-size", 15);
```

2. How do I *efficiently* check if a word collides with the others?

One way to efficiently do this is to maintain a quadtree. Fortunately, d3 has a pretty easy to use quadtree implementation. In a nutshell, a quadtree divides the space into four quadrants, then each quadrant into four quadrants, and so forth. These quadrants get organized as a tree where the leaf quadrants store actual points. So if we are looking for a point we don't need to serially search them all. We just start at the root of the tree and test in which quadrant the point should be, then do the same for the children of the target quadrants until we find the point. Divide and conquer. In the next snippet we show how to initialize and update a quadtree:

```
  var quadtree = d3.quadtree();
  quadtree.add([x, y]); // adding a point to the quadtree
  quadtree.cover(); // updates the boundaries of the quadtree,
                    // call this after you have added a batch of points
```

Here's how you test if a point is in the quadtree:

```
  // find any points within a .1 radius
  var matches = quadtree.find(x, y, .1);
```

SVG Text provides a bounding box, but it isn't tight. That is, it includes a lot of white

space. Like in the blue box below:

We need a more fine-grained solution. The first step is to load the font we will use with the opentype library. Below we are providing the URL to the Lato open font. You can use any font you like.

```
  var fontUrl =
'https://fontlibrary.org/assets/fonts/lato/29e379a6ecc1b86c96931fa6ce4b3b0c/1
233fdf19c04333c7f58af4eb8698452/LatoBlack.ttf';
  opentype.load(fontUrl, function(err, font) {
      if (err) {
          alert('Could not load font: ' + err);
      } else {
           // font is loaded
           // CONTINUE HERE
      }
  }
```

Then ask opentype for a path element that represents the contour of a word. Opentype needs the font size and location of the word.

```
var path = font.getPath(“Example”, x, y, fontSize);
```

The variable above, path, is an opentype object. It has a function that returns the *d* attribute of SVG Path. We have learned that *d* describes a path. Let’s get this value so we can create a Path in the page:

```
var d = path.toPathData();        // get d, a path descriptor
var pathNode = d3.select("#scene")
                 .append(‘path')  // create a path
                 .attr(‘d', d)    // set the value of d
                 .node();
```

Let’s recap. We would like to test if a word collides with another. SVG doesn’t give us very useful bounding boxes, so we need the word contours. Given a word, a font size, a position, and a font, opentype generates the contour that we need. It’s an SVG path. After running the code above, we should have the word path in the page. Now it’s a matter of testing every point of in a word’s contour agains every point in the other word’s contour. If there’s a match, the words collide. Sounds slow? That’s why we need the quadtree. It let’s us perform queries of the type “Does this point exist?” faster.

It’s worth noting there are much more clever methods to detect collisions. But if this already feels complicated, imagine the others.

To get every point in a path, we call path’s getPointAtLength:

```
var pathLength = pathNode.getTotalLength();
// yes, the next loop iterates every point along the path!
for (var j = 0; j < pathLength; j++) {
          var point = pathNode.getPointAtLength(j);
          point.x; // the x coordinate
          point.y; // the y coordinate
    }
```

3. The previous answer showed how to detect collisions. How do I put this all together?

You're free to choose any strategy. Here's a basic one:

1. Get the path for the position where you're trying to place the word.
2. Check if any of the points in this path can be found in the quadtree.
    1. If so, increment the position and try again. Don't forget to remove the path from the page before.
    2. If not, then the path doesn't collide and you can place the text there. Remember this is a path, not a text element. So remove the path, and create a text element in the same position, with the same size and font. Now update the quadtree with all points in this path. After all, it has been added to the screen. Remember, the quad tree store the contour points of all words that have been added to the Wordle.

4. For testing purposes, work with a subset of the data, like 5 words, before you're sure your algorithm works.

5. What's the equation of the spiral?

https://stackoverflow.com/a/6824451/1253334

Here's a jsfiddle. Change the values of angle, a, and b. To "travel" along the spiral you need to increment the value of angle. Do not choose a value too small, or your algorithm will be too slow.
http://jsfiddle.net/jingshaochen/xJc7M/