I spent a week or so creating the code for what I called the “tag explorer” (see Figure 3-4), a Java applet that permitted users to navigate through dogear by clicking on tags related to the current context.

It was immediately clear that the tag explorer was useful as a *portrait* of a person’s interests, as when a number of my fellow IBMers used screenshots of the tag explorer to illustrate their résumés and email signatures (see Figure 3-5).

![Figure 3-5. The author’s 2006 work email signature](image)

When dogear became an IBM product,* the tag explorer did not go with it, and I forgot all about it. When I found the tag explorer code by chance a couple of years later, I thought it was worth developing.

The original tag explorer was intimately tied to dogear, and to the idea of tag clouds in general. I wanted to find a way to decouple the word-cloud effect from the whole idea of “tags,” since the pleasing and amusing qualities of the word cloud seemed generally accessible, while “tags” were familiar only to a technologically sophisticated crowd. This led to the idea of simply counting words. Once I had decided to build a system for viewing text, rather than tags, it seemed superfluous to have the words do anything other than merely exist on the page. I decided that I would design something primarily for pleasure, in the spirit of Charles Eames’s remark, “Who would say that pleasure is not useful?” This decision, in turn, made it easy to decide which features to keep, which features to reject, and how to design the interface (shown in Figure 3-6).

![Figure 3-6. Wordle’s text-analytics user interface](image)

Since Wordle (as it was now called) was meant to be pleasing, I had to give some thought to the expressive qualities of fonts and color palettes (see Figure 3-7).

I believe that my efforts to simplify Wordle, and to emphasize pleasure over business, have been paid for many times over. Wordle has been used in ways I’d never anticipated, by far more people than I’d dared to expect. Some of Wordle’s success is due to the design of the web application itself, with its one-paste/one-click instant gratification. However, to the extent that the design of the Wordle visualization itself has contributed to its ubiquity, it might be worth looking at what Wordle is not before we examine in detail what it is and how it works.

**Anatomy of a Tag Cloud**

The typical tag cloud is organized around lines of text.* If one word on a line is larger than another, the smaller word will have a disproportinate amount of whitespace overhead, which can look awkward. For example, see Figure 3-8, where “everett hey” has an enormous expanse of white above, because the line height is determined by its neighbor “everett everett”.

One way to mitigate the ragged whitespace caused by such extreme contrasts in size is to squash different word weights into a small number of bins, as del.icio.us does. In Figure 3-9, the “programming” tag has been used 55 times and “scripting” only once, but the font for the more frequently used word is only 50% larger. Notice also the use of font weight (boldness) to enhance the contrast between different word weights.

Figure 3-9. *Squashing the scale of differences between word weights*

In effect, del.icio.us is scaling the word weights—roughly—by logarithm. It’s sensible to scale weights using logarithms or square roots when the source data follows a power-law distribution, as tags seem to do.*

Somewhere between these earnest, useful designs and the fanciful world that Wordle inhabits, there are other, more experimental interfaces. The WP-Cumulus† blog plug-in, for example, provides a rotating, three-dimensional sphere of tags (see Figure 3-10).

Figure 3-10. *WP-Cumulus: can’t…quite…click on “tag cloud”…*

The desire to combine navigation with visualization imposes certain constraints on the design of a word cloud. But once we are liberated from any pretense of “utility”—once we’re no longer providing navigation—we can start to play with space.

**Filling a Two-Dimensional Space**

There are lots of computer science PhDs to be garnered in finding incremental improvements to so-called *bin-packing* problems.‡ Luckily, the easy way has a respectable name: a randomized greedy algorithm. It’s *randomized* in that you throw stuff on the screen somewhere near where you want it to be, and if that stuff intersects with other stuff, you try again. It’s *greedy* in that big words get first pick.

* See http://www.citeulike.org/user/andreacapocci/article/1326856.
† See http://wordpress.org/extend/plugins/wp-cumulus/.
Wordle’s specific character depends on a couple of constraints. First, we are given a list of words, with associated (presumably meaningful) weights. We can’t show any word more than once, and we don’t want to distort the shape of the word beyond choosing its font size. If we remove those constraints, though, many other interesting and beautiful effects are possible.

For example, you can use a randomized greedy strategy to fill almost any region (not just a rectangle) as long as you have a set of words as a palette, from which you can arbitrarily choose any word, at any size, any number of times (see Figure 3-11).

Figure 3-11. Do not underestimate the power of the randomized greedy algorithm

Consider Jared Tarbell’s exquisite Emotion Fractal* (see Figure 3-12), which recursively subdivides a space into ever-smaller random rectangles, filling the space with ever-smaller words. This effect depends on a large set of candidate words, chosen at random, with arbitrary weights.

If you don’t mind distorting your fonts by elongating or squashing the words as needed, other effects are possible. For example, Figure 3-13 shows a variation on the venerable treemap,* which uses text, rather than rectangles, to fill space. Each word fills an area proportional to its frequency; each rectangular area contains words strongly associated with each other in the source text.

* See http://www.cs.umd.edu/hcil/treemap-history/.