CHAPTER THREE

Wordle

Jonathan Feinberg

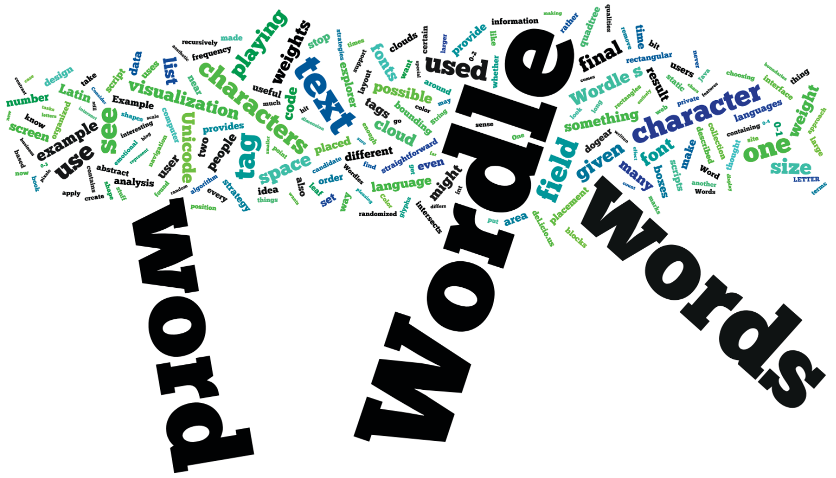


Figure 3-1. *A Wordle of this chapter*

BY NOW, EVEN PEOPLE who have never heard of "information visualization" are familiar with the colorful word collage known as *Wordle*, "the gateway drug to textual analysis."[*] Like any such drug, Wordle was designed for pleasure, although its roots lie in the utilitarian tag clouds popularized by such sites as del.icio.us and Flickr.

* See *http://www.profhacker.com/2009/10/21/wordles-or-the-gateway-drug-to-textual-analysis/*.

Wordle's Origins

In 2004, my colleague Bernard Kerr and I made a social bookmarking application, which Bernard named "dogear" (Millen, Feinberg, and Kerr, 2006). Any application that lets users tag content is bound to provide a *tag cloud*, a vaguely rectangular collection of clickable keywords. So, when we designed dogear, we made sure to feature a prominent tag cloud on every page (see Figure 3-2).

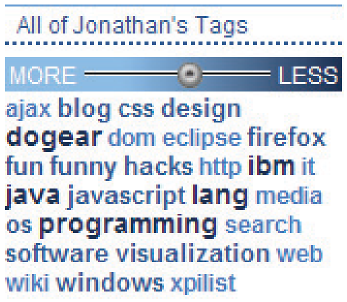

All of Jonathan's Tags
MORE
LESS
ajax blog css design
dogear dom eclipse firefox
fun funny hacks http ibm it
java javascript lang media
os programming search
software visualization web
wiki windows xpilist

Figure 3-2. *The author's tags as they appeared in dogear*

I never found tag clouds to be particularly interesting or satisfying, visually. There's not much evidence that they're all that useful for navigation or for other interaction tasks, either.* But when blogger Matt Jones† posted his del.icio.us tags as a beautiful, typographically lively image (see Figure 3-3), I was thrilled. I thought that there was no reason why a computer program couldn't create something similar. At the very least, I wanted to end up with something that could—like Jones's cloud—put the dot of an "i" into the lower counter of a "g", something well beyond what tag clouds could do at the time.

* See *http://doi.acm.org/10.1145/1240624.1240775*.

† See *http://magicalnihilism.com/2004/07/04/my-delicious-tags-july-2004/*.

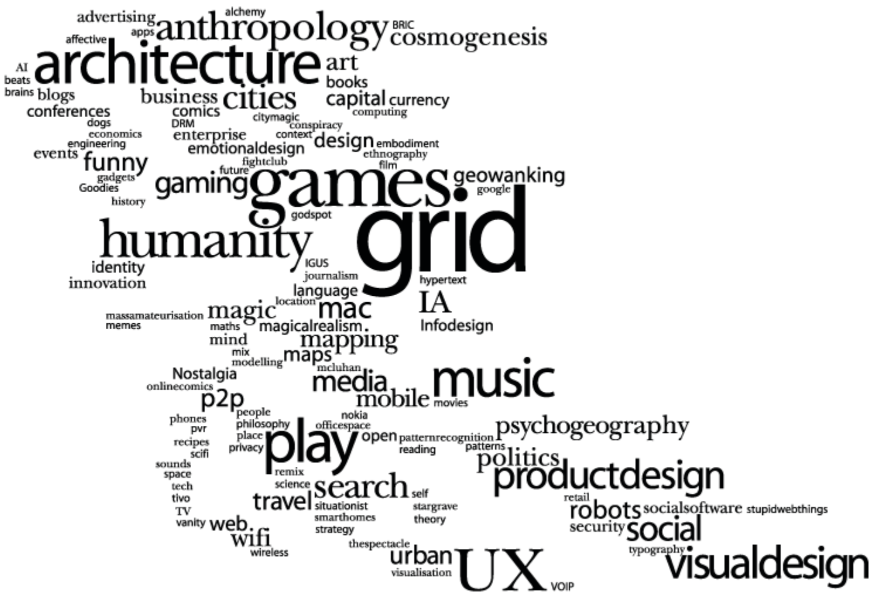


Figure 3-3. *Matt Jones's typographically aware tag cloud*

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.

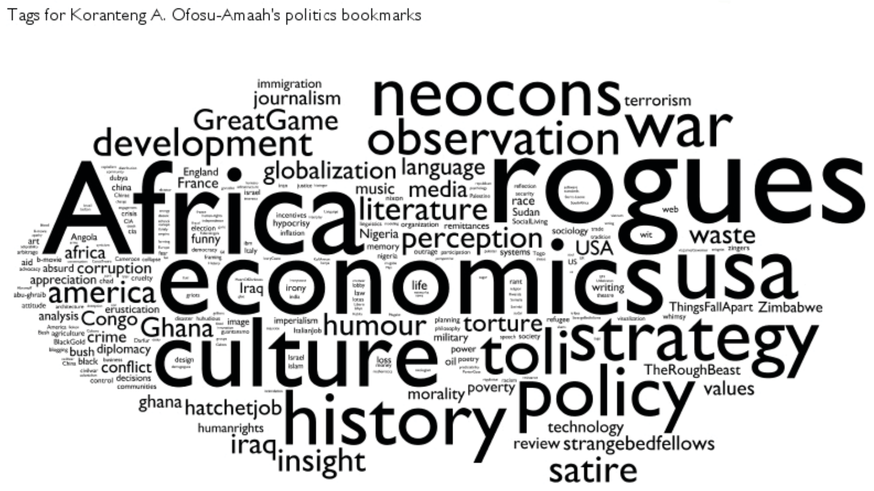
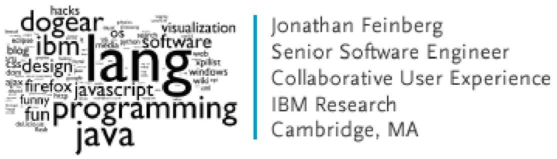


Figure 3-4. *The dogear tag explorer*[*]

* See *http://www.flickr.com/photos/koranteng/526642309/in/set-72157600300569893*.

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).



Jonathan Feinberg
Senior Software Engineer
Collaborative User Experience
IBM Research
Cambridge, MA

Figure 3-5. *The author's 2006 work email signature*

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).

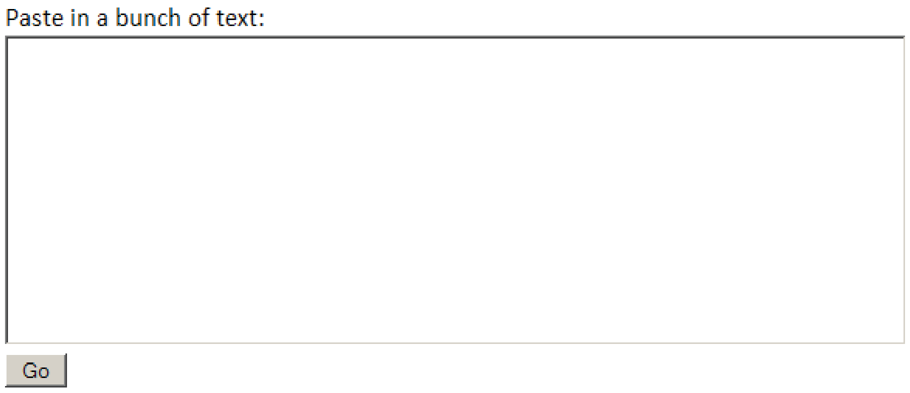


Figure 3-6. *Wordle's text-analytics user interface*

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).

[*] See *http://www-01.ibm.com/software/lotus/products/connections/bookmarks.html*.

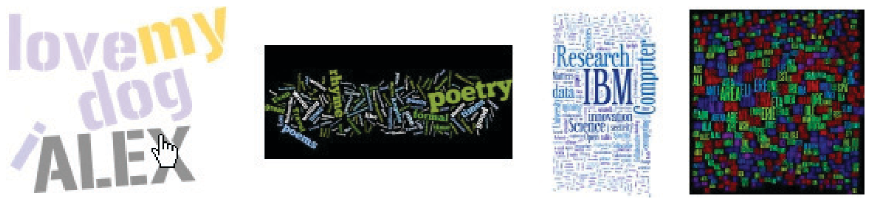


Figure 3-7. *Wordle provides varied palettes, fonts, and layouts*

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 disproportionate 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".

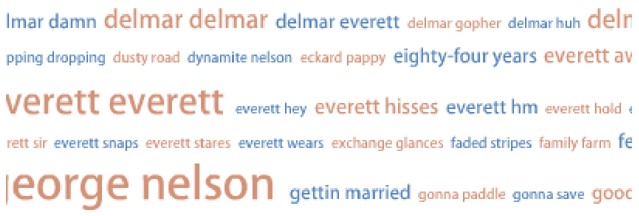

lmar damn delmar delmar delmar everett delmar gopher delmar huh delr
pping dropping dusty road dynamite nelson eckard pappy eighty-four years everett av
verett everett everett hey everett hisses everett hm everett hold
rett sir everett snaps everett stares everett wears exchange glances faded stripes family farm fe
eorge nelson gettin married gonna paddle gonna save gooc

Figure 3-8. *Lost in White Space*[†]

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.

* For a thorough survey of tag cloud designs, with thoughtful commentary, see *http://www.smashingmagazine.com/2007/11/07/tag-clouds-gallery-examples-and-good-practices/*.

† See *http://manyeyes.alphaworks.ibm.com/manyeyes/page/Tag_Cloud.html*.

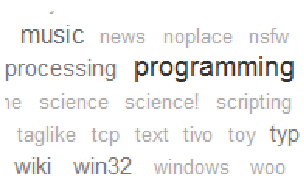


Figure 3-9. *Squashing the scale of differences between word weights*

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).

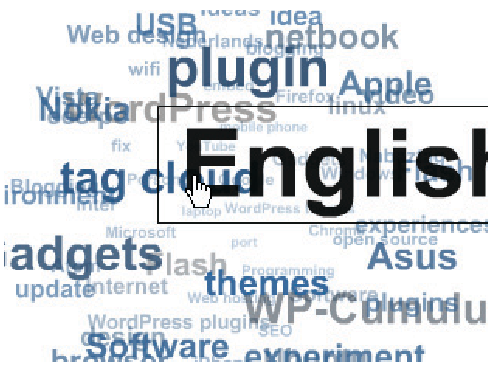


Figure 3-10. *WP-Cumulus: can't…quite…click on "tag cloud"…*

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/*.

‡ See *http://en.wikipedia.org/wiki/Bin_packing_problem*.

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).

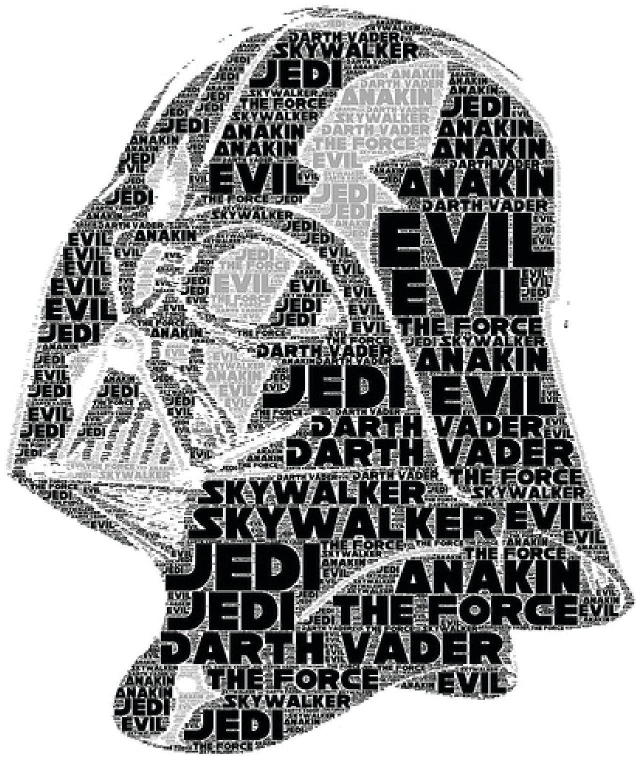


Figure 3-11. *Do not underestimate the power of the randomized greedy algorithm*

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.

* See *http://levitated.net/daily/levEmotionFractal.html*.

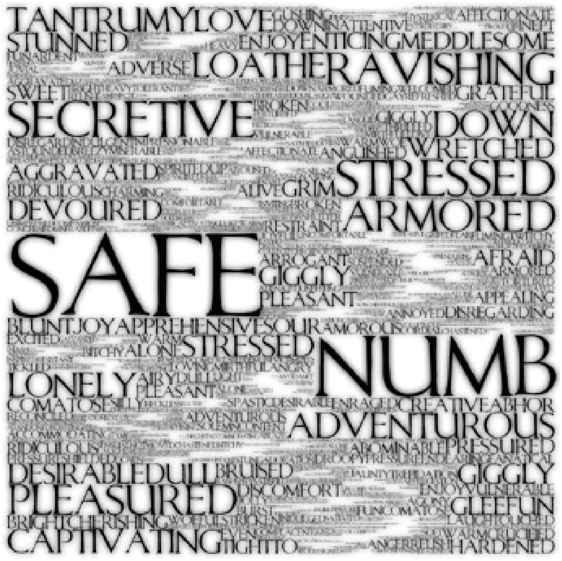


Figure 3-12. *Jared Tarbell's Emotion Fractal*

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.

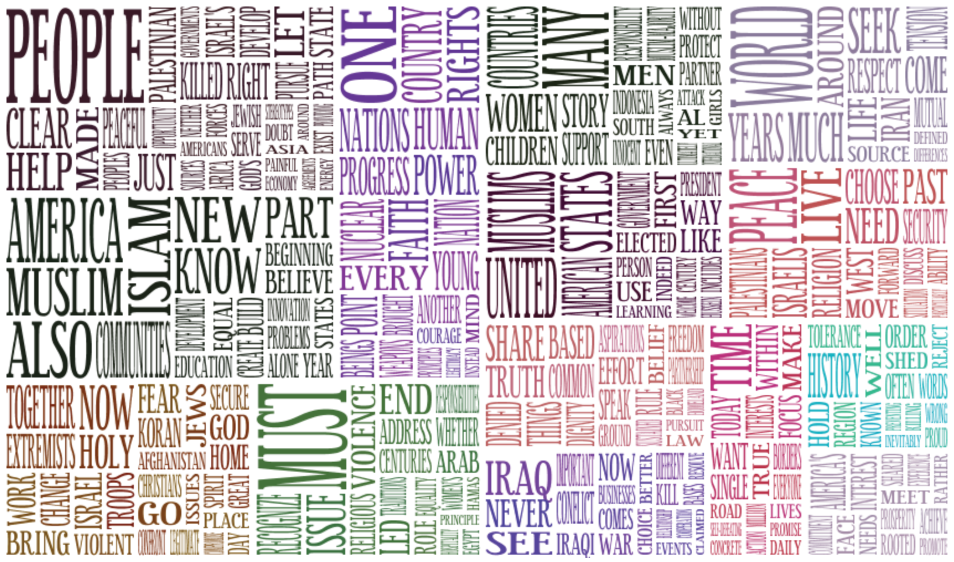


Figure 3-13. *Word treemap of an Obama speech*

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

It must be said that long before there were Processing sketches and Flash applets, people were exploring these sorts of typographical constructions in mass media and in fine art (Figure 3-14); we have been probing the boundary between letters as forms and letters as signs for a long time (Figure 3-15). The goal of these algorithmic explorations is to allow the wit and elegance of such examples to *influence* the representation of textual data.

Given this rather brief tour of the technical and aesthetic environment in which Wordle evolved, we're now ready to look at Wordle's technical and aesthetic choices in a bit more detail.

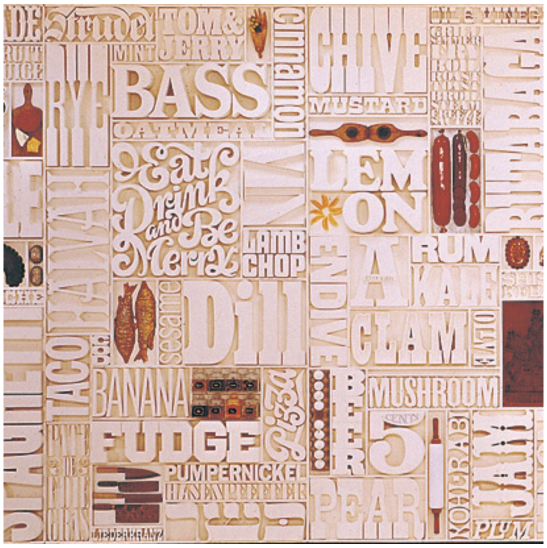


Figure 3-14. *Herb Lubalin and Lou Dorfsman's Typographicalassemblage (courtesy of the Center for Design Study)*

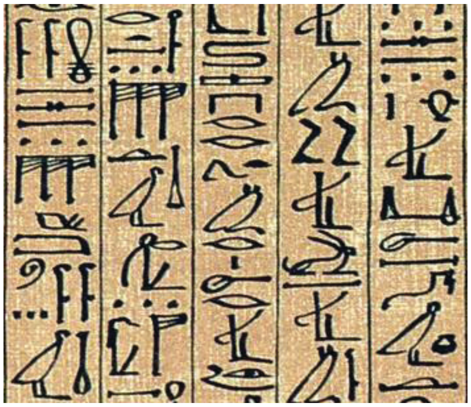


Figure 3-15. *Before we made pictures with words, we made words with pictures*

How Wordle Works

Wordle is implemented as a Java applet. Some of the technical details I provide here will be in terms of Java-specific features. Nothing described here is impossible in other languages, using other libraries and frameworks, but Java's strong support for Unicode text processing and 2D graphics (via the Java2D API) makes these things pretty straightforward.

Text Analysis

We'll now take a step back and consider some of the fundamental assumptions that determine Wordle's character. In particular, we have to examine what "text" is, as far as Wordle is concerned.

While this kind of text analysis is crude compared to what's required for some natural-language processing, it can still be tedious to implement. If you work in Java, you might find my `cue.language` library* useful for the kinds of tasks described in this section. It's small enough, it's fast enough, and thousands use it each day as part of Wordle.

Remember that natural-language analysis is as much craft as science,† and even given state-of-the-art computational tools, you have to apply judgment and taste.

Finding words

Wordle is in the business of drawing pictures of *words*, each having some *weight*, which determines its size. What does Wordle consider to be a "word"?

Wordle builds a regular expression (regex) that recognizes what it considers to be words in a variety of scripts, and then iteratively applies that regex to the given text, as illustrated in Example 3-1. The result is a list of words.

Example 3-1. *How to recognize "words"*

```
private static final String LETTER = "[@+\\p{javaLetter}\\p{javaDigit}]";
private static final String JOINER = "[-.:/''\\p{M}\\u2032\\u00A0\\u200C\\u200D~]";
/*
A word is:
    one or more "letters" followed by
    zero or more sections of
        one or more "joiners" followed by one or more "letters"
*/
private static final Pattern WORD =
    Pattern.compile(LETTER + "+(" + JOINER + "+" + LETTER + "+)*");
```

* See *http://github.com/vcl/cue.language*.

† For an illuminating demonstration of this craft, see Peter Norvig's chapter on natural-language processing in the sister O'Reilly book *Beautiful Data*.

A *letter* is any character that the Java `Character` class considers to be either a "letter" or a "digit," along with `@` (at sign) and `+` (plus sign). *Joiners* include the Unicode `M` class, which matches a variety of nonspacing and combining marks, other pieces of punctuation commonly found in URLs (since Wordle users expect to see URLs preserved as "words"), the apostrophe, and several characters used as apostrophes in the wild (such as U+2032, the PRIME character). Wordle accepts the tilde (~) as a word joiner but replaces it with a space in the output, thereby giving users an easy way to say "keep these words together" without having to find the magical key combination for a real nonbreaking space character.

Determining the script

Having extracted a list of words (whatever we take "word" to mean), we need to know how to display those words to the viewer. We first need to know what characters we'll be expected to display, so that we can choose a font that supports those characters.

Wordle's collection of fonts is organized in terms of what *scripts* each can support, where a *script* is what you might think of as an alphabet: a collection of *glyphs* that can be used to visually represent sequences of *characters* in one or more languages. A given script, in Unicode, is organized into one or more *blocks*. So, the task now is to determine which fonts the user might want to use by sniffing out which blocks are represented in the given text.

Java provides the static method `UnicodeBlock.of(int codePoint)` to determine which block a given code point belongs to. Wordle takes the most frequent words in a text and looks at the first character in each of those words. In the rather common case that the character is in the Latin block, we further check the rest of the word to see if it contains any Latin-1 Supplement characters (which would remove certain fonts from consideration) or any of the Latin Extended blocks (which would bar even more fonts). The most frequently seen block is the winner.

To keep it responsive and limit its use of network resources, Wordle is designed to permit the use of only one font at a time. A more full-featured word cloud might choose different fonts for different words; this could provide another visual dimension to represent, for example, different source texts.

As of this writing, Wordle supports the Latin, Cyrillic, Devanagari, Hebrew, Arabic, and Greek scripts. By design, Wordle does not support the so-called CJKV scripts, the scripts containing Chinese, Japanese, Korean, and Vietnamese ideographs. CJKV fonts are quite large and would take too long for the average Wordle user to download (and would cost a great deal in bandwidth charges). Also, determining word breaks for ideographic languages requires sophisticated machine-learning algorithms and large runtime data structures, which Wordle cannot afford.

KNOW THY DATA

Unicode in a Nutshell

Since Wordle understands text in Unicode terms, here's what you have to know in order to understand some of the terms and notations you'll see here.

The Unicode* standard provides a universal coded character set and a few specifications for representing its characters in computers (as sequences of bytes).

A *character* is an abstract concept, meant to serve as an atom of written language. It is not the same thing as a "letter"—for example, some Unicode characters (accents, umlauts, zero-width joiners) are only meaningful in combination with other characters. Each character has a name (such as GREEK CAPITAL LETTER ALPHA) and a number of properties, such as whether it is a digit, whether it is an uppercase letter, whether it is rendered right-to-left, whether it is a diacritic, and so on.

A *character set* or *character repertoire* is another abstraction: it is an unordered collection of characters. A given character is either in, or not in, a given character set. Unicode attempts to provide a universal character set—one that contains every character from every written language in current and historical use—and the standard is constantly revised to bring it closer to that ideal.

A *coded character set* uniquely assigns an integer—a *code point*—to each character. Once you've assigned code points to the characters, you may then refer to those characters by their numbers. The convention used is an uppercase U, a plus sign, and a hexadecimal number. For example, the PRIME character mentioned earlier in this chapter has the code point U+2032.

Coded characters are organized according to the *scripts* in which they appear, and scripts are further organized into *blocks* of strongly related characters. For example, the Latin script (in which most European languages are written) is given in such blocks as Basic Latin (containing sufficient characters to represent Latin and English), Latin-1 Supplement (containing certain diacritics and combining controls), Latin Extended A, Latin Extended B, and so on.

When it comes time to actually put pixels onto a screen, a computer program interprets a sequence of characters and uses a *font* to generate *glyphs* in the order and location demanded by the context.

* See *http://unicode.org*.

Guessing the language and removing stop words

It would be neither interesting nor surprising to see that your text consists mostly of the words "the," "it," and "to." To avoid a universe of boring Wordles, all alike, such *stop words* need to be removed for each recognized language. To know which list of stop words to remove for a given text, though, we have to guess what language that text is in.

Knowing the script is not the same as knowing the language, since many languages might use the same script (e.g., French and Italian, which share the Latin script).

Wordle takes a straightforward approach to guessing a text's language: it selects the 50 most frequent words from the text and counts how many of them appear in each language's list of stop words. Whichever stop word list has the highest hit count is considered to be the text's language.

How do you create a list of stop words? As with the definition of a "word," described earlier, this kind of thing is a matter of taste, not science. You typically start by counting all of the words in a large corpus and selecting the most frequently used words. However, you might find that certain high-frequency words add a desirable flavor to the output while other, lower-frequency words just seem to add noise, so you may want to tweak the list a bit.

Many of Wordle's stop word lists came from users who wanted better support for their own languages. Those kind folks are credited on the Wordle website.

By default Wordle strips the chosen language's stop words from the word list before proceeding to the next steps, but Wordle users can override this setting via a menu checkbox.

Assigning weights to words

Wordle takes the straight path in assigning a numeric *weight* to each word. The formula is: weight = word count.

Layout

Once you've analyzed your text, you're left with a list of words, each of which has some numeric *weight* based on its frequency in the text. Wordle normalizes the weights to an arbitrary scale, which determines the magnitude of various constants that affect the resulting image (such as the minimum size of a hierarchical bounding box leaf, as described later in this chapter). You're now ready to turn words into graphical objects and to position those objects in space.

Weighted words into shapes

For each word, Wordle constructs a font with a point size equal to the word's scaled weight, then uses the font to generate a Java2D `Shape` (see Example 3-2).

Example 3-2. *How to turn a String into a Shape*

```
private static final FontRenderContext FRC
    = new FontRenderContext(null, true, true);

public Shape generate(final Font font, final double weight, final String word,
            final double orientation) {
    final Font sizedFont = font.deriveFont((float) weight);
    final char[] chars = word.toCharArray();
    final int direction = Bidi.requiresBidi(chars, 0, chars.length) ?
        Font.LAYOUT_RIGHT_TO_LEFT : Font.LAYOUT_LEFT_TO_RIGHT;
    final GlyphVector gv =
        sizedFont.layoutGlyphVector(FRC, chars, 0, chars.length, direction);
    Shape result = gv.getOutline();
    if (orientation != 0.0){
        result = AffineTransform.getRotateInstance(orientation)
                    .createTransformedShape(result);
    }
    return result;
}
```

The playing field

Wordle estimates the total area to be covered by the finished word cloud by examining the bounding box for each word, summing the areas, and adjusting the sum to account for the close packing of smaller words in and near larger words. The resulting area is proportioned to match the target aspect ratio (which is, in turn, given by the dimensions of the Wordle applet at the moment of layout).

The constants used to adjust the size of the *playing field*, the area in which Wordles are laid out, were determined by the time-honored tradition of futzing around with different numbers until things looked "good" and worked "well." As it happens, the precise size of the playing field is rather important, because the field boundaries are used as constraints during layout. If your playing field is too small, your placement will run slowly and most words will fall outside the field, leaving you with a circle (because once a word can't be placed on the field, Wordle relaxes that constraint and you wind up with everything randomly distributed around some initial position). If it's too large, you'll get an incoherent blob (because every nonintersecting position is acceptable).

One "gotcha" to look out for is an especially long word, which could have a dimension far larger than the calculated width or height based on area. You must make sure that your playing field is big enough to contain the largest word, at least.

Remember that the playing field is an abstract space, a coordinate system not corresponding to pixels, inches, or any other unit of measurement. In this abstract space, you can lay out the word shapes and check for intersections. When it comes time to actually put pixels on the screen, you can do some scaling into screen units.

Placement

Having created a place to put words, it's time to position the words in that space. The overall placement strategy is a randomized greedy algorithm in which words are placed, one at a time, on the playing field. Once a word is placed, its position does not change.

Wordle offers the user a choice of placement strategies. These strategies influence the shape and texture of the completed Wordle, by determining where each word "wants" to go. On the Wordle website, the choices are center-line and alphabetical center-line. Both strategies place words near the horizontal center-line of the playing field (not necessarily upon it, but scattered away from it by a random distribution). The alphabetical strategy sorts the words alphabetically and then distributes their preferred *x* coordinates across the playing field.

Interesting effects are possible through the use of smarter placement strategies. For example, given clustering data—information about which words tend to be used near each other—the placement strategy can make sure that each word tries to appear near the last word from its cluster that was placed on the field (see Figure 3-16).

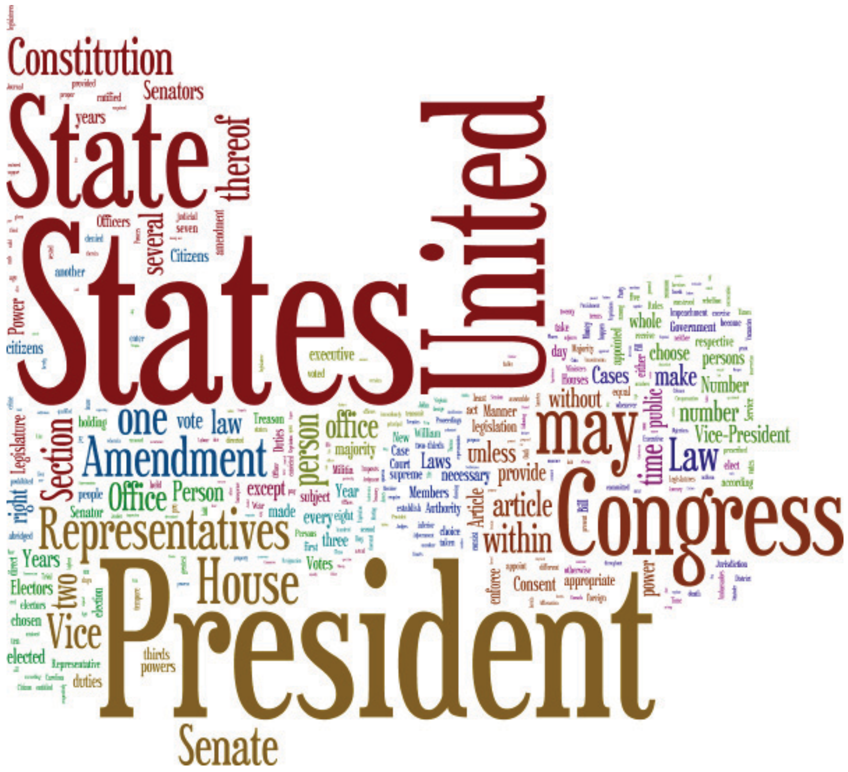


Figure 3-16. *The result of a clustering placement strategy*

The word shapes are sorted by their respective weights, in descending order. Layout proceeds as in Example 3-3, with the result as illustrated in Figure 3-17.

Example 3-3. *The secret Wordle algorithm revealed at last!*

```
For each word w in sorted words:
    placementStrategy.place(w)
    while w intersects any previously placed words:
        move w a little bit along a spiral path
```

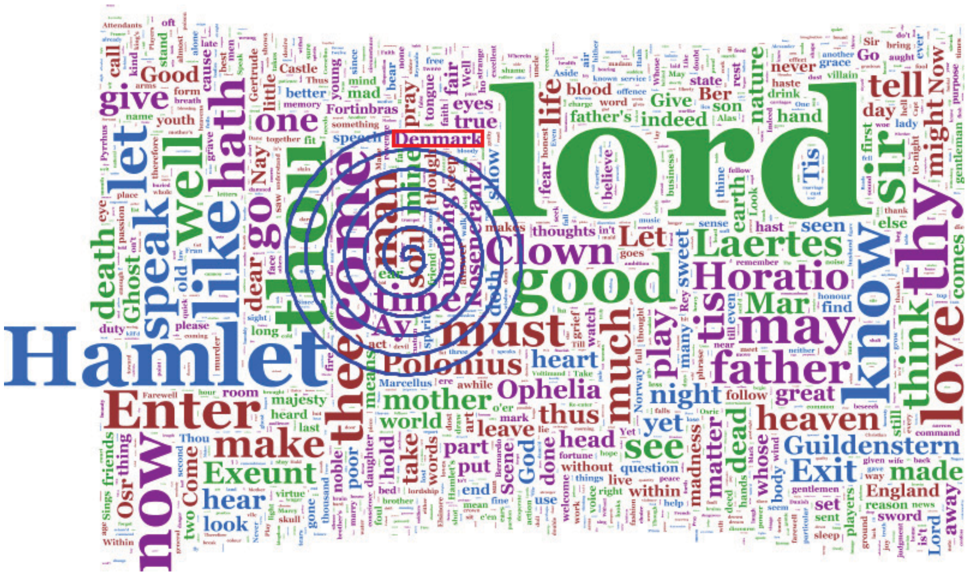


Figure 3-17. *The path taken by the word "Denmark"*

To make matters a bit more complicated, Wordle optionally tries to get the words to fit entirely within the rectangular boundaries of the playing field—this is why it's important to guess how big the whole thing is going to be. If the rectangular constraint is turned on, the intersection-handling routine looks like Example 3-4.

Example 3-4. *Constraining words to the playing field*

```
while w intersects any previously placed words:
    do {
        move w a little bit along a spiral path
    } while any part of w is outside the playing field and
            the spiral radius is still smallish
```

Intersection testing

The pseudocode in Example 3-4 breezily suggests that you move a word while it intersects other words, but it does not suggest how you'd go about determining such a thing. Testing spline-based shapes for intersection is expensive, and a naïve approach to choosing pairs for comparison is completely unaffordable. Here are the techniques that Wordle currently uses to make things fast enough:

Hierarchical bounding boxes

The first step is to reduce the cost of testing two words for intersection. A simple method for detecting misses is to compare the bounding boxes of two words, but it's not uncommon for two such boxes to intersect when the word glyphs do not. Wordle exploits the cheapness of rectangle comparisons by recursively dividing a word's bounding box into ever-smaller boxes, creating a tree of rectangles whose leaf nodes contain chunks of the word shape (see Figure 3-18). Although it's expensive to construct these hierarchical bounding boxes, the cost is recovered by an order of magnitude during the layout. To test for collision, you recursively descend into mutually intersecting boxes, terminating either when two leaf nodes intersect (a hit) or when all possible intersecting branches are excluded (a miss). By taking care with the minimum size of leaf rectangles and by "swelling" the leaf boxes a bit, the layout gets a pleasing distance between words "for free."

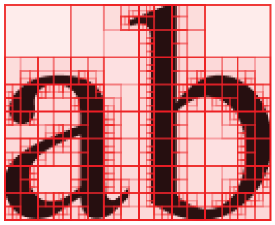


Figure 3-18. *Hierarchical bounding boxes*

Broadphase collision detection

In choosing pairs of words to test for intersection, the simplest approach is to test the current candidate word against all of the already-placed words. This approach results in a hit test count around the order of N^2, which is far too slow once you get up to 100 words or so. Therefore, Wordle does some extra work to avoid as much collision testing as possible.

Caching

One very simple improvement stems from the observation that if a word A intersects some other word B, it's very likely that A will still intersect B if A is moved slightly. Therefore, Wordle caches a candidate word's most recently intersected word and tests it first.

Spatial indexing

To further reduce the number of hit tests, Wordle borrows from computational geometry the *region quadtree*, which recursively divides a two-dimensional space (in this case, the Wordle playing field) into four rectangular regions. Here, a quadtree serves as a *spatial index* to efficiently cull shapes from the list of words to be compared to some candidate shape. Once a word is placed on the playing field, Wordle searches for the smallest quadtree node that entirely contains the word, and adds the word to that node. Then, when placing the next word, many already-placed words can be *culled* from collision testing by querying the quadtree.

There's an entire research field around efficient collision detection, much of which is very well summarized in Christer Ericson's (2005) book *Real-Time Collision Detection*. I recommend that book to anyone who wants to play with randomized graphics algorithms like Wordle's; my own quadtree implementation is based on my understanding of its discussion.

Is Wordle Good Information Visualization?

If you consider Wordle strictly as an information visualization tool, certain aspects of its design could be criticized for their potential to mislead or distract its users. Here are some of my own Wordle caveats.

Word Sizing Is Naïve

Wordle does not take into account the length of a word, or the glyphs with which it's drawn, when calculating its font size. The result is that, given two words used the same number of times, the word with more letters will take up more space on the screen, which *may* lead to the impression of the longer word being more frequent.

On the other hand, I don't know of any studies on how relative word size corresponds to perceived relative weight. What's more, the commonly used trick of scaling by the square root of the word's weight (to compensate for the fact that words have *area*, and not mere length) simply makes a Wordle look boring.

Color Is Meaningless

In a medium—your computer screen—that provides precious few dimensions, Wordle is shockingly free with its use of color. Color means absolutely nothing in Wordle; it is used merely to provide contrast between word boundaries and for aesthetic appeal.

Color could be used to code various dimensions, such as clustering (indicating which words tend to be used near each other) or statistical significance (as in the inaugural address word clouds—see Figure 3-19). Wordle could also use color to let two or more different texts be represented in the same space.

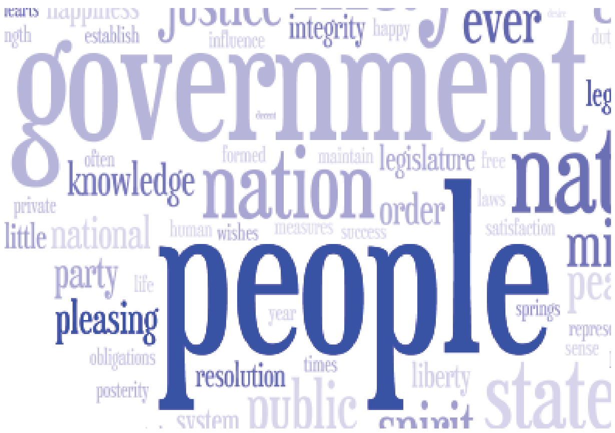


Figure 3-19. *"Government" was used a lot in this speech, but not much more than in the other speeches; "pleasing" was used only a couple of times but is an unusual word in the corpus; "people" was used a lot and is unusually frequent*

It should also be mentioned that Wordle makes no provision for colorblind users, although one can always create a custom palette via the applet's Color menu.

Fonts Are Fanciful

Many of Wordle's fonts strongly favor aesthetics and expressiveness over legibility. This has to do, partly, with the design of the Wordle website—the gallery pages would be monotonous without fairly broad letter-form diversity. Most importantly, a font has to look good *in a Wordle*, which may mean that it wouldn't necessarily work well for body text.

For applications where legibility is paramount, Wordle provides Ray Larabie's Expressway font, which is modeled on the U.S. Department of Transportation's Standard Alphabets.

Word Count Is Not Specific Enough

I have seen Wordle used to summarize each book of the New Testament, leading to one page after another of "Lord," which tells you nothing about how the chapters are *distinct* from one another. Merely counting words does not permit meaningful comparisons of like texts. Consider, for example, a blog post. It might be most revealing to emphasize how the post differs from other blog posts by the same author, or to show how it differs from posts on the same topic by other bloggers, or even to show how it differs from the language of newspaper reporting.

There are plenty of statistical measures that one may apply to a "specimen" text versus some "normative" body of text to reveal the specific character of the specimen, with proper attention paid to whether some word use is statistically significant. Given a more nuanced idea of word weight, beyond mere frequency, one could then apply the Wordle layout algorithm to display the results.

I explored this idea in an analysis of every presidential inaugural address,* in which each speech was compared to the 5 speeches nearest to it in time, the 10 nearest speeches, and all other inaugural addresses. Such an analysis has the advantage of revealing the unexpected *absence* of certain words. For example, Figure 3-20 is a visualization of Harry Truman's 1948 inaugural address. On the left is a Wordle-like representation of the words he used, and on the right are the words that his contemporaries used more than he did. This visualization reveals Truman's emphasis on foreign policy.

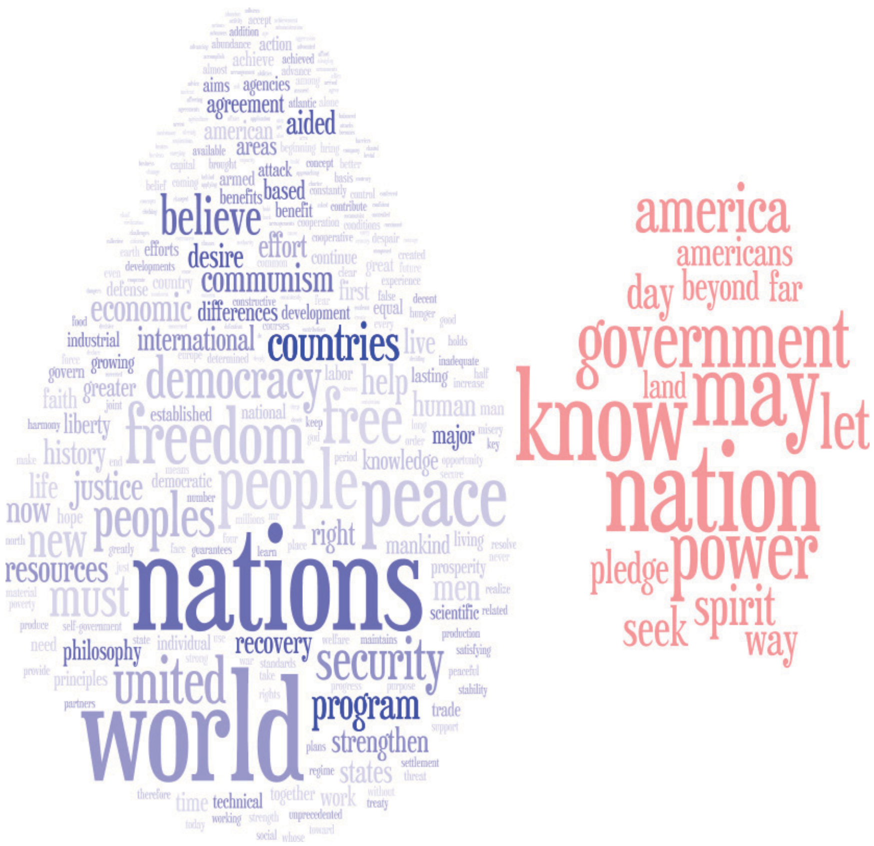


Figure 3-20. *Harry Truman's 1948 inaugural address: the words in red were conspicuously absent from Harry Truman's speech, relative to those of his contemporaries*

* See *http://researchweb.watson.ibm.com/visual/inaugurals/*.

How Wordle Is Actually Used

Wordle was not designed for visualization experts, text analysis experts, or even experienced computer users. I tried to make Wordle as appliance-like as possible.

As of this writing, people have created and saved over 1,400,000 word clouds in the Wordle gallery. They have been used to summarize and decorate business presentations and PhD theses, to illustrate news articles and television news broadcasts, and to distill and abstract personal and painful memories for victims of abuse. Wordle has also found an enthusiastic community in teachers of all stripes, who use Wordles to present spelling lists, to summarize topics, and to engage preliterate youngsters in the enjoyment of text.

As the survey results in Table 3-1 (Viégas, Wattenberg, and Feinberg, 2009) illustrate, when people use Wordle they feel *creative*, as though they're making something.

Table 3-1. *How people feel when they make a Wordle*

	Agree %	**Neutral %**	**Disagree %**
I felt creative	**88**	9	4
I felt an emotional reaction	**66**	22	12
I learned something new about the text	**63**	24	13
It confirmed my understanding of the text	**57**	33	10
It jogged my memory	**50**	35	15
The Wordle confused me	5	9	**86**

So, by one traditional academic measure of a visualization's efficacy—"I learned something new about the text"—Wordle can at least be considered moderately successful. But where Wordle shines is in the creation of *communicative artifacts*. People who use Wordle feel as though they have created something, that the created thing succeeds in representing something meaningful, and that it accurately reflects or intensifies the source text. This sense of meaningfulness seems to be mostly intuitive, in that many people do not realize that word size is related to word frequency (guessing, instead, that the size indicates "emotional importance" or even "word meaning").

The special qualities of Wordle are due to the special qualities of text. Simply putting a single word on the screen, in some font that either complements or contrasts with the sense of the word, immediately resonates with the viewer (indeed, there have been many thousands of single-word Wordles saved to the public gallery). When you juxtapose two or more words, you begin to exploit the tendency of a literate person to make sense of words in sequence. Wordle's serendipitous word combinations create delight, surprise, and perhaps some of the same sense of recognition and insight that poetry evokes intentionally.

Using Wordle for Traditional Infovis

Notwithstanding Wordle's special emotional and communicative properties, the analytic uses of information visualization are certainly available to the expert user. To serve those who want to use Wordle as a visualization for their own weighted text, where the weights are not necessarily based on word frequency, the Wordle website provides an "advanced" interface, where one can enter tabular data containing arbitrarily weighted words or phrases, with (optional) colors.

Still more advanced use is possible through the "Word Cloud Generator" console application, available through IBM's alphaWorks website.*

The ManyEyes collaborative data visualization site also provides Wordle as a text-visualization option beside its innovative Phrase Net and Word Tree visualizations (and a more traditional tag cloud).†

Conclusion

People often want to preserve and share the Wordles they make; they use Wordles to *communicate*. A beautiful visualization gives pleasure as it reveals something essential.

Acknowledgments

I would like to thank Martin Wattenberg and Irene Greif at IBM CUE for making possible my participation in this book. I am very grateful to Ben Fry, Katherine McVety, Fernanda Viégas, and Martin Wattenberg, who each read this chapter with great care and suggested many improvements. Please see *http://www.wordle.net/credits* for information about the many people who have helped me create and improve Wordle.

References

Ericson, Christer. 2005. *Real-Time Collision Detection*. San Francisco, CA: Morgan Kaufmann.

Millen, D. R., J. Feinberg, and B. Kerr. 2006. "Dogear: Social bookmarking in the enterprise." *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Montréal, Québec, Canada, April 22–27, 2006). *http://doi.acm.org/10.1145/1124772.1124792.*

Viégas, Fernanda B., Martin Wattenberg, and Jonathan Feinberg. 2009. "Participatory visualization with Wordle." *IEEE Transactions on Visualization and Computer Graphics* 15, no. 6 (Nov/Dec 2009): 1137–1144. doi:10.1109/TVCG.2009.171.

* See *http://www.alphaworks.ibm.com/tech/wordcloud*.
† See *http://manyeyes.alphaworks.ibm.com/manyeyes/page/Visualization_Options.html*.