Gheat

Release 0.2

Chad W. L. Whitacre

April 29, 2008

Zeta Design & Development
http://www.zetadev.com/software/gheat/
Email: chad@zetaweb.com

Abstract

Gheat is a heatmap tile server for Google Maps. Gheat solves the problem of presenting data on a Google map when there is too much data to be well-visualized by map markers.

CHAPTER

ONE

Introduction

Google Maps provides a JavaScript API for integrating their maps with your website. The API includes calls to add markers on the map for various data points, but this only works when the number of markers placed on any given map view is relatively small. Gheat solves the problem of presenting data on a Google map when there is too much data to be well-visualized by map markers. It does so by using another of the JavaScript API's Google makes available, whereby additional tilesets can be layered on top of the base map tileset. (Google Maps works by splitting map imagery into 256x256 pixel tiles and then knitting them together client-side.)

Gheat runs as a standalone web application under the Aspen webserver. Data points are stored in a SQLite database, with each data point uniquely identified and timestamped. You modify the data in this database using a bundled script, run from the command line. The script reads data you provide in a CSV file and updates the database accordingly.

Gheat only generates tiles containing data when requested, and it stores these tiles on the filesystem for future use. On subsequent requests, if no data relevant to the tile has changed, then the image is served straight from the filesystem. This saves considerable processing time. Most tiles will be empty, however, and Gheat precreates empty tiles for all zoom levels and serves the same file for all empty tiles at a given zoom level. This saves considerable disk space. These tile caches are organized on the filesystem under the name of each color scheme in the root of the gheat distribution. Under each such directory, there is a directory for each zoom level (named '0', '1', '2', etc.), and another called 'empties'. Gheat will create and recreate tile cache directories as necessary, and tile caches can be safely deleted at any time.

Because of the cost of generating heatmaps, Gheat is probably not suitable for realtime applications. But Gheat performs well when the underlying dataset evolves on a weekly or even daily basis.

CHAPTER

TWO

Installation

2.1 The Short Version

- Install Python (version 2.5 or newer).
- Install Aspen (version 0.8 or newer).
- Download the Gheat distribution.
- Install Pygame or PIL.
- Start **aspen** in the Gheat distribution.
- Add a GTileLayerOverlay to your embedded Google Map.

2.2 The Long Version

Gheat involves setting up a web application server, and then wiring up your web pages to make calls to that server via Google's JavaScript Maps API.

The server side of Gheat has a few dependencies. The first is Python. You need version 2.5 or newer, because Gheat requires the SQLite library, which was added to the standard library in 2.5. There is also some 2.5 syntax in Gheat. The second is Aspen, which is a Python webserver. Gheat is known to work with Aspen version 0.8. At this point you can download and unpack the Gheat distribution, or check it out or export it from the Subversion repository. The next dependency is an imaging library; Gheat supports both Pygame and PIL (if you install Pygame, you only need the imaging components). You can either install it locally in the '__' directory of the Gheat distribution (the virtualenv tool is a great way to manage this directory), or you can install it globally. The pygame backend is three or four times faster than PIL, but PIL is easier to install. The functionality is the same with both.

Once all dependencies are satisfied, run the **aspen** program within the Gheat distribution, which is an Aspen website root. The Gheat distribution includes a sample database out of the box, so at this point you should be able to hit the webserver now running on your machine and see a heatmap tile. Try this URL: http://localhost:8080/classic/4/4,6.png.

If you see an image like in Figure 2.1, then congratulations! You have successfully installed the Gheat server.

Once you get the server running, you need to set things up on the client side. First, you need to embed a Google Map on a web page. Then you need to define a new map layer using GTileLayer and GTileLayerOverlay. There is an example at 'example.html' in the Gheat distribution to get you started. With your Gheat server running, hit http://localhost:8080/example.html to see it in action (you may need to tweak the Google Maps API key). If it works, you will see the heatmap tiles loaded overtop of the underlying Google Map, and you will feel pumped.

See Also:

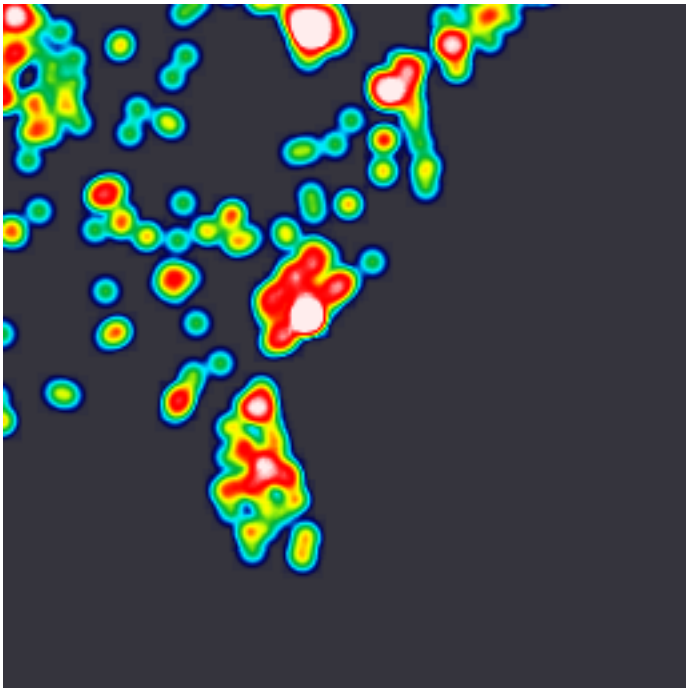

Figure 2.1: A heatmap tile

Aspen
: The homepage for the Aspen webserver

Google Maps
: The homepage for the Google Maps API

PIL
: The homepage for the Python Imaging Library (PIL)

Pygame
: The homepage for the Pygame library

Python
: The homepage for the Python programming language

virtualenv
: The homepage for virtualenv, a tool to create isolated Python environments

CHAPTER

THREE

Configuration

Gheat responds to three configuration options. These are located in the INI-formatted file '__/etc/aspen.conf', under a section named `[gheat]`. The three options are:

Option	Description
backend	The backend library to use for image generation; either `Pygame` or `PIL` (case-insensitive). `Pygame` is three or f
zoom_opaque	The zoom level at and below which the master opacity will be 100%
zoom_transparent	The zoom level at and above which the master opacity will be 0%

Here is an example 'aspen.conf' file:

```
[gheat]
backend=PIL
zoom_opaque=-1
zoom_transparent=11
```

3.1 Master Opacity

Google Maps' zoom level is zero-indexed, starting furthest out. As of this writing there are 20 zoom levels in use, so the highest available zoom level (closest in) is 19. The master opacity for Gheat will be scaled linearly between *zoom_opaque* and *zoom_transparent*. So, for example, with the default settings (-3 to 15), the master opacities for each available zoom level are as follows.

Zoom	Opacity
0	82.8%
1	77.3
2	71.9
3	66.4
4	60.5
5	55.1
6	49.6
7	44.1
8	38.7
9	33.2
10	27.3
11	21.9
12	16.4
13	10.9
14	5.5
15	0.0
16	0.0
17	0.0
18	0.0
19	0.0

If *zoom_transparent* is less than or equal to *zoom_opaque*, the opacity will be 100% for all zoom levels. The master opacity for each level is multiplied by any alpha channel in the color scheme (see below) to determine the final opacity of the image.

When you change the *zoom_opaque* and/or *zoom_transparent* settings, you need to manually delete the tile caches.

3.2 Color Scheme

When you wire up the client side of Gheat, you choose which color scheme you want to use (the process is described under Installation). The options bundled with Gheat are (the left image is the actual color scheme PNG; the right image is an example in use):

You can produce your own color scheme. A color scheme is defined by a PNG of height 256 pixels. The image may be as wide as you like; only the first column of pixels will be looked at. This first column is effectively a mapping of intensities to four-channel colors (red, green, blue, and alpha). To use a custom color scheme, place your PNG file in the '__/etc/color-schemes/' directory of your Gheat installation, and use your PNG's filename without the '.png' extension as the token to wire into your JavaScript.

The alpha channel of the pixels in the color scheme are multiplied with the master opacity setting for the given zoom level (see above) to determine the final opacity of a given pixel in each tile.

If and when you change the PNG file for a color scheme of a given name, you need to manually delete the tile cache for that color scheme.

See Also:

'aspen.conf'
: Documentation for the 'aspen.conf' file

`ConfigParser`
: This describes the 'INI' format used in 'aspen.conf' (the `RawConfigParser` is used)

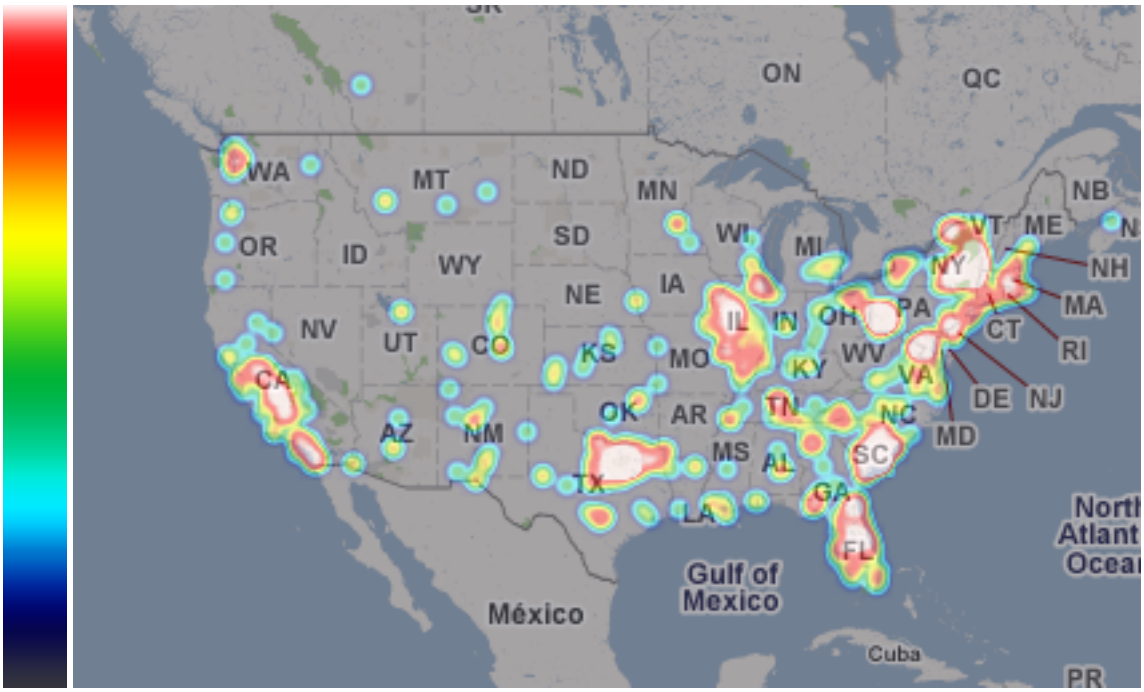

Figure 3.1: classic

Figure 3.2: fire

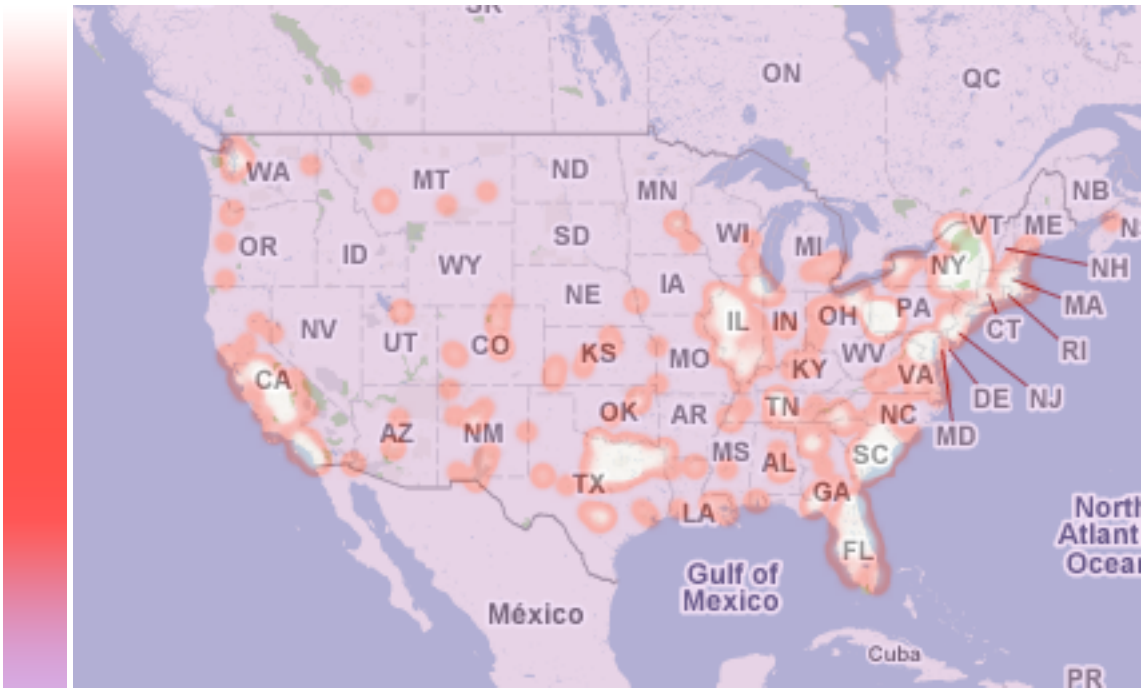

Figure 3.3: omg

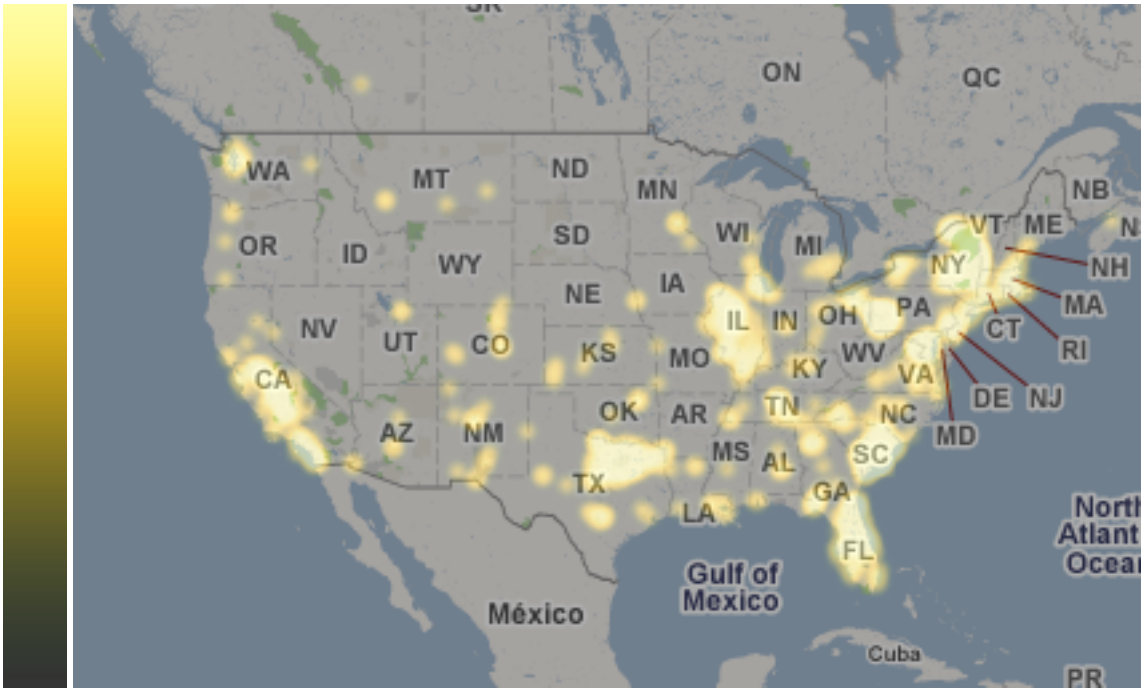

Figure 3.4: pgaitch

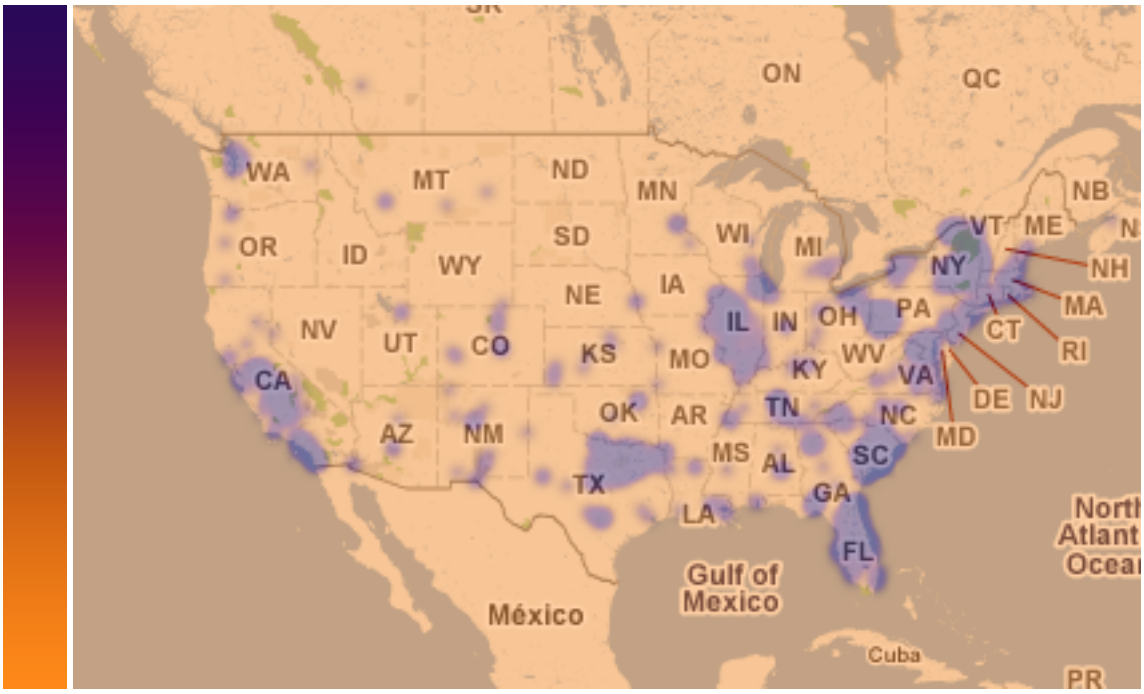

Figure 3.5: pbj

CHAPTER

FOUR

The db.py script

The script at '__/bin/db.py' is used to manage the database at '__/var/points.db'. It's available subcommands are:

Subcommand	Description
clear	Remove all points from the database
count	Print the number of points in the database to 'stdout'
delete	Delete the database from the filesystem
sync	Update the database from the file at '__/var/points.txt'

For the *clear*, *count*, and *sync* subcommands, the database will be created if it does not exist. The default subcommand is *sync*.

The '__/var/points.txt' file is a CSV-formatted file with three fields: *point id*, *latitude*, and *longitude*. The *point id* field can be any string. If it is not unique within the 'points.txt' file then the last point with a given *point id* will override any previous points with that same *point id*. Gheat uses the *point id* field to keep track of when points should be removed from the database, as well as when points are updated, in order to tell when the tile cache ought to be flushed. The *latitude* and *longitude* fields must be floats. Multiple data points may have the same latitude and longitude in order to increase the intensity of that geographic point on the heatmap.

The *sync* subcommand outputs a token for each point in the 'points.txt' file:

Token	Meaning
.	The point was not modified since the last time it was seen
o	The point was modified and has been updated
O	This is a new point

The number of points deleted can be determined by using the *count* subcommand before and after the *sync* subcommand.