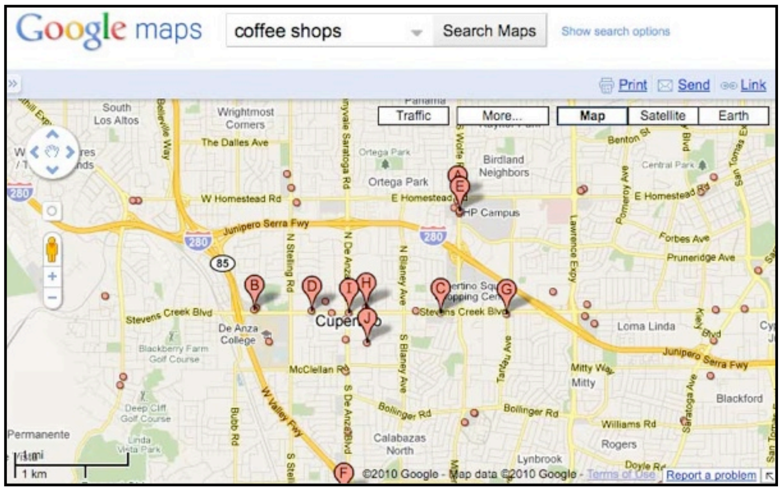
Simulating Markers with a Tile Layer

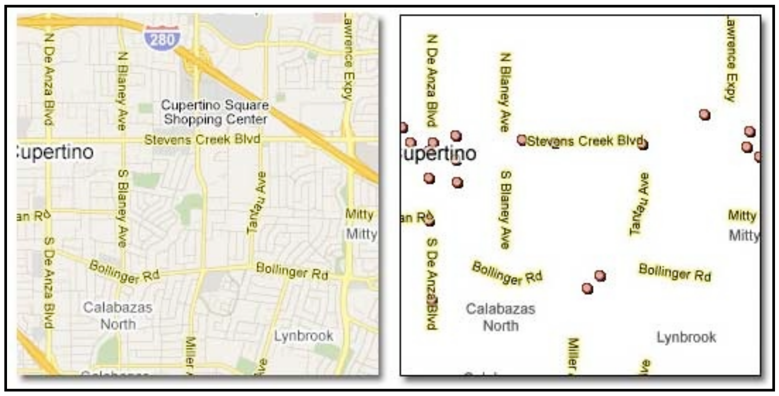
John Coryat, USNaviguide LLC, August 3rd, 2010
Companion website: http://www.usnaviguide.com/ws-2008-08

A lot of custom tile layer requirements can be met using image cutting utilities, they are fast and efficient yet fail when image size becomes too large for the average system to handle. Once this point is reached, the best solution is to create each tile as an individual image instead.

This article will focus on one implementation, icon images embedded into a tile layer to simulate markers.



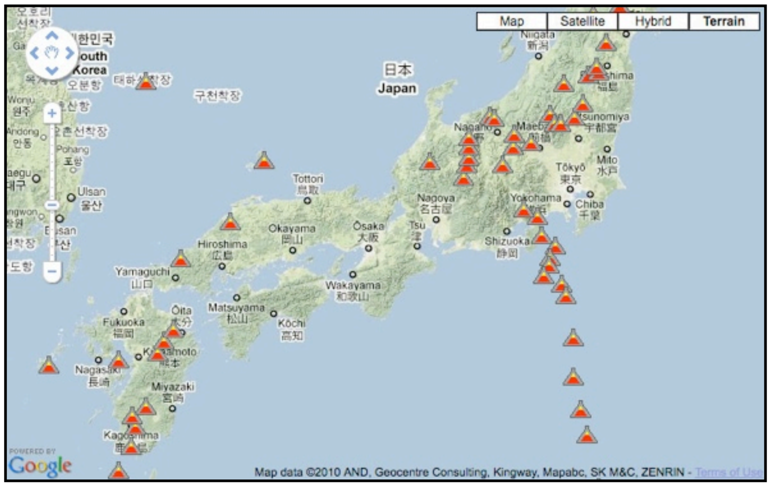
Google uses this technology in their standard search results, note in this image how the primary results are lettered markers while the secondary results are little red dots.



The secondary results, while also selectable to reveal the details of that point, are in fact not marker objects but part of a tile layer. Using this method, many more results can be displayed then if marker objects alone were used.

The main advantage of using a separate layer to display simulated markers centers on the ability to show more markers than if JavaScript objects were solely used. In addition, using a tile layer is fast and efficient, doesn't reveal data in the way markers do and since it's created on the server, more processing can be done like clustering, changing the icon depending on the zoom along with others. Some disadvantages include increased bandwidth to transmit tile images and some delay in displaying the infowindow when the marker is clicked.

Simulating Markers with a Tile Layer



In this example, the tile layer shows volcanoes from the Smithsonian Global Volcanism Program. There are about 1,300 volcanoes in the database and it requires over 4,000 tiles to display the entire world at zoom levels three to ten. At zoom level seven, the icon changes from a small circle to the more elaborate volcano image. This is easily accomplished using a tile layer, JavaScript marker objects would be quite a bit more difficult. We'll be using this example (link: http://www.usnaviguide.com/ws-2010-08/volcano.htm) throughout this article.

Useful Perl Module

In order to do many of the calculation involved with transforming points into drawn tiles, we're providing a handy Perl module that calculates just about everything that might be needed in the process. Two helpful functions include converting coordinates into pixels, and calculating a tile name from a pixel location and zoom.

USNaviguide_Google_Tiles.pm

Calculate:

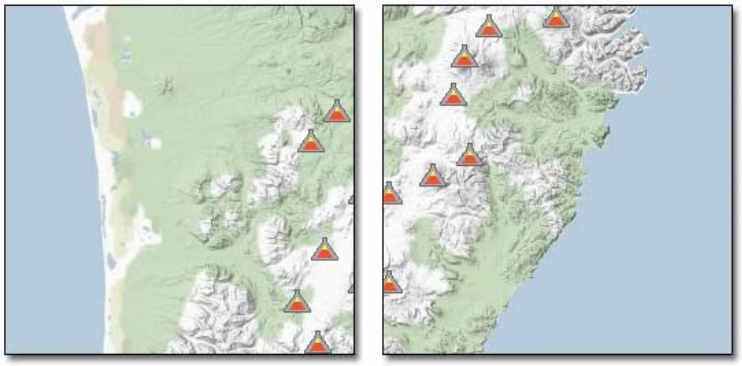
- All tiles for a bounding box and zoom
- Factors needed (Zoom, Tilesize)
- Tile features from a tile name and zoom
- Tile name to pixel
- Coordinate to pixel
- Pixel to coordinate
- Tile name from a pixel location and zoom

Calculating Tiles

The first step in constructing a tile layer is figuring out what tiles will be required. One method to do this is to create a database table that has zoom, X and Y tile names and the X and Y coordinates of the top left position of the icon image. The top left position of the icon is what will be needed to place it on the tile.

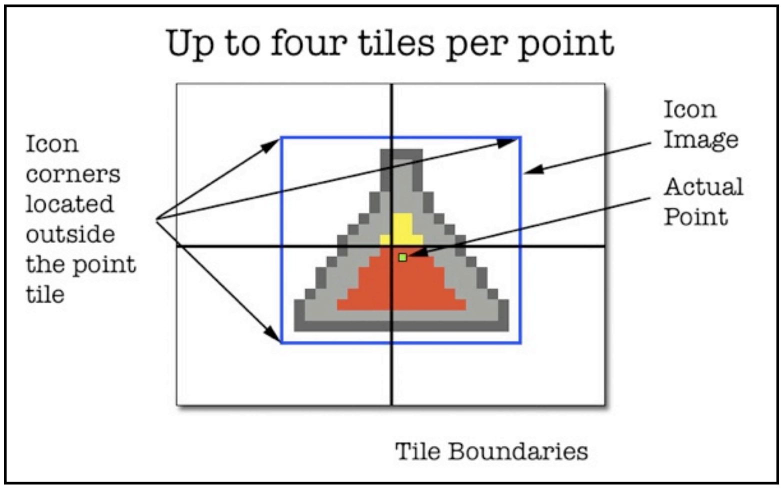
Simulating Markers with a Tile Layer

Icons can overlap Tiles



A complication arrises when trying to merge the icon image into the tile, often, icon images overlap from one tile to another. A point will always reside on a single tile (per zoom), however, once a point becomes an icon image, if the point is close to the edge of the tile, the associated icon image may cross tile boundaries. This has to be handled in order to maintain a seamless view when the map is displayed.

Overlapping Problem


Up to four tiles per point
Icon corners located outside the point tile
Icon Image
Actual Point
Tile Boundaries

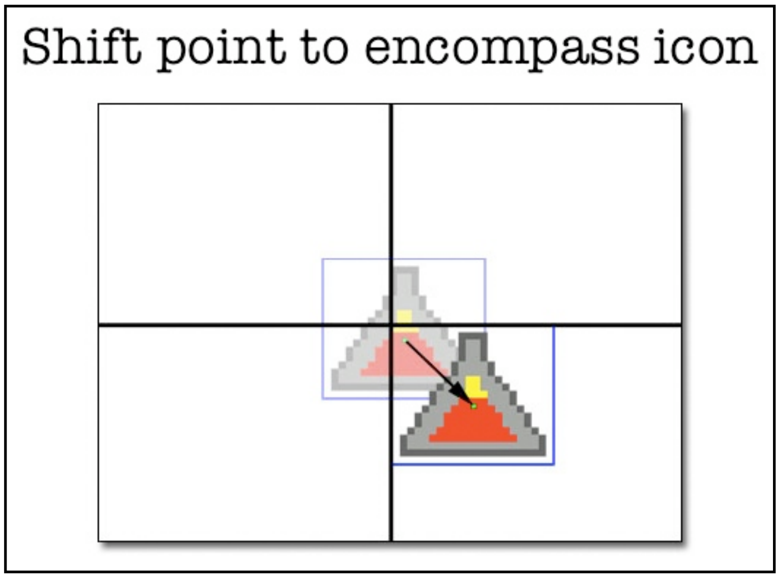
Here we see a worst case scenario where an icon image has crossed into four tiles. In order to draw a set of perfect tiles, either the icon image has to be partially drawn on four different tiles or the point has to be moved slightly to allow the image to fit on the tile.

The two methods for handing this overlap I call “Fudging” and “Exact.”

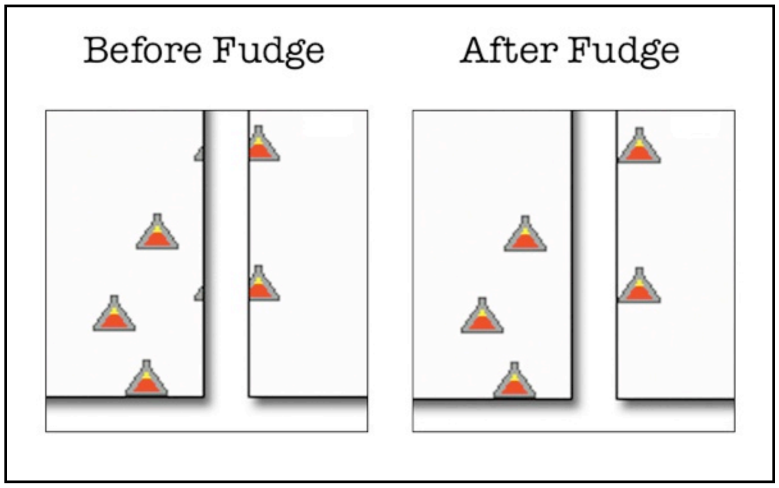
Simulating Markers with a Tile Layer

Fudging Method

In the "Fudging" method, the point is shifted enough so that the icon image resides completely in one tile. This method works well when the icon is relatively small, there are a limited number of points, the points represent an areal feature such as a city, lake, park or other feature and where shifting the point a few pixels won't detract from the accuracy of the map. This method allows for faster tile drawing and the process is simplified.

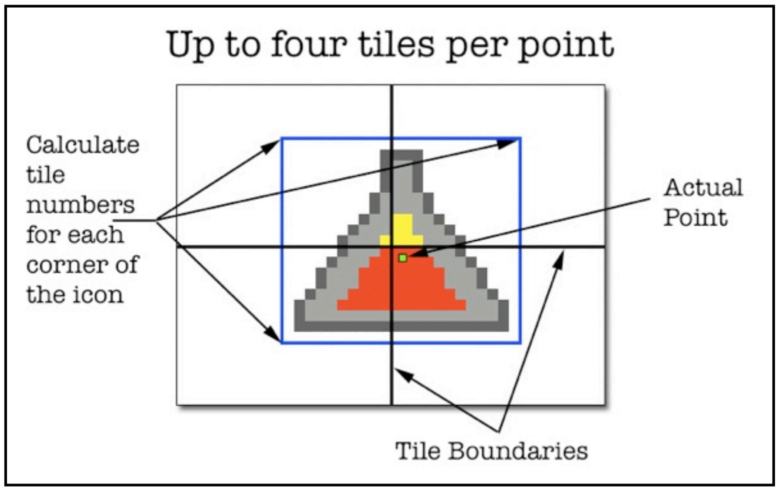


"Fudging" fails when there's a good possibility two points will be "fudged" to occupy the same coordinate, if the point represents a small feature like a building, transit station or where even a slight shift will detract from the accuracy of the map. In those cases, using the "Exact" method is the best bet.



Exact Method

With the "Exact" method, the tile locations of each corner of the icon image are calculated and stored, along with the top left hand corner of the image. When tile drawing, icons with overlap will have just a piece of the image but when the tiles are assembled the resulting image will look seamless.


Up to four tiles per point
Calculate tile numbers for each corner of the icon
Actual Point
Tile Boundaries

Simulating Markers with a Tile Layer

Drawing Tiles

Once the points and tiles required have been calculated, it's time to draw. Select "distinct" rows by the tile name, since there could be many rows with duplicate tile names as a result of the calculations done previously. Cycle through the table, selecting all points for each tile and merge the icon images onto the tile using the top left corner pixel location, once all points are processed, the tile can be created.

Drawing Tile Steps:

1. Select list of tiles required

2. Gather points for the tile

3. Merge icon images into tile

4. Output tile

Creates a "sparse" tile set

It will be advantageous to separate zoom levels into their own directory. This will reduce the time it takes the server to find a tile and deliver it to the client. Since a "sparse" tile set is created, or put another way, only tiles with data are drawn, there will occasionally be a request for a tile that doesn't exist, resulting in a server 404 message. The API will handle these errors correctly and display a blank tile, it will be faster and more efficient to handle these missing tiles in another manner.

Simulating Marker Clicks

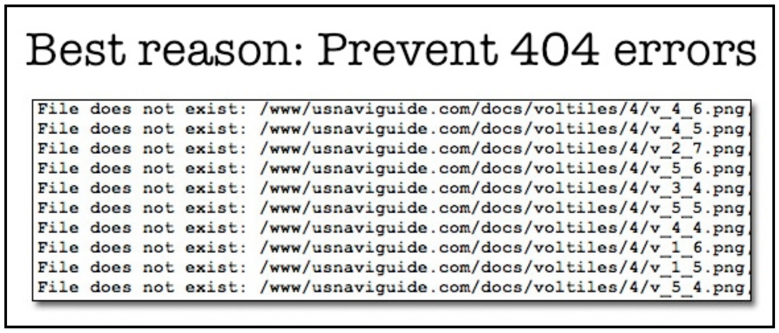
The final step in the simulation of markers is to handle the click event as if it was a marker. The usual result of a marker click reveals the "infoWindow" to the user. The same can be done by using AJAX. A click listener sends the server a message with the zoom level and coordinate, the server selects from the point table the closest point, which is examined to determine if the click was within a reasonable range of pixels to be considered a hit and not just a random click. This is done using a relatively simple distance formula, since the values are all pixels, the curvature of the Earth can be ignored. The results are transmitted back the the client as XML and the client displays the infoWindow, just as a marker click with a JavaScript object.

Simulating Marker Clicks

- Client sends coordinate and zoom
- Server selects the closest point to the click
- Server checks tolerance
- Server sends XML to client
- Client displays results

Simulating Markers with a Tile Layer

Using a Tile Server



Best reason: Prevent 404 errors

```
File does not exist: /www/usnaviguide.com/docs/voltiles/4/v_4_6.png
File does not exist: /www/usnaviguide.com/docs/voltiles/4/v_4_5.png
File does not exist: /www/usnaviguide.com/docs/voltiles/4/v_2_7.png
File does not exist: /www/usnaviguide.com/docs/voltiles/4/v_5_6.png
File does not exist: /www/usnaviguide.com/docs/voltiles/4/v_3_4.png
File does not exist: /www/usnaviguide.com/docs/voltiles/4/v_5_5.png
File does not exist: /www/usnaviguide.com/docs/voltiles/4/v_4_4.png
File does not exist: /www/usnaviguide.com/docs/voltiles/4/v_1_6.png
File does not exist: /www/usnaviguide.com/docs/voltiles/4/v_1_5.png
File does not exist: /www/usnaviguide.com/docs/voltiles/4/v_5_4.png
```

A "tile server" handles the image request from the client, instead of the default file handler. The best reason is to avoid server "404 not found" errors. 404 errors take more effort for the server to handle as an error message has to be logged and the error reporting to the client consumes more bandwidth than a blank tile. The tile server checks for the existence of the image file and if not found, returns a blank image instead, bypassing the 404 error.

There are other advantages to using a tile server, such as being able to hide the tile directory from public access and allowing a security system to prevent unauthorized web sites from using tile images.

Resources

The following resources are available for download. Please feel free to use any of them as a project base, convert into any language and modify any way that makes sense.

- tiles.pl - Calculate and draw tiles using the "exact" method
- volcano.htm - v3 page used in this presentation
- volcano.pl - Click handler for volcano.htm
- tileserver.pl - Tile server used in this presentation
- volcano.sql - PostgreSQL dump for data used in this presentation
- USNaviguide_Google_Tiles.pm - Calculate tile factors (perl module)
- ws-2010-08-article.pdf - This article
- download.zip - All the above in a zipped format

All the above materials are available under the Apache license.

Link: http://www.usnaviguide.com/ws-2010-08