Table of Contents

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Chapter 7. Extreme Google Maps Hacks

7.1. Hacks 62–70: Introduction

So far, we've seen the power and ease with which Google Maps can show nearly anything on a map, from your vacation photos to a hypothetical nuclear explosion. Now it's time for us to take Google Maps hacks to the next level by adding whole new feature sets and APIs. In this chapter, we'll supplement the missing features of Google Maps in various ways, by turning street addresses into map coordinates, generating dynamic icons using Google's polyline service, adding new controls to bring in background imagery from around the 'Net, and even creating our own custom background tiles! These hacks go far above and beyond anything that even Google could have imagined or intended for Google Maps—that's why we call them "Extreme Google Maps Hacks."

Hack 62. Find the Latitude and Longitude of a Street Address

The Google Maps API won't do it for you, but there are other ways to find the coordinates of a given street address.

As we've seen all through this book, Google Maps makes it easy to make custom maps of anything for which you have a latitude and a longitude. However, people don't tend to think of places in terms of geographic coordinates; more often, people commonly know and refer to places by a street or mailing address. In order to find and show these places on a map, we need to be able to turn a given address into the corresponding latitude/longitude coordinates. The process of turning addresses into map coordinates is generally referred to as *geocoding*.

Unfortunately, for all the great things that the Google Maps API offers, the very common task of geocoding street addresses simply isn't among them. Although the Google Maps webUI will show you the location of an address, often with great accuracy, the problem is that Google doesn't own this data and has been unable to negotiate permission with its data providers to offer address lookups as a service in the API. Furthermore, screen-scraping the Google Maps results page is strictly a no-no, according to the Terms of Use. If you want to stay legit—and not risk angry takedown letters from you-know-who—how can you get the lat/long coordinates for a given street address?

7.2.1. The Hack

Fortunately, within the United States, the Census Bureau collects street address information as part of its constitutionally assigned duties of enumerating the populace of the country every ten years. What's more, the Census Bureau publishes this information in the public domain, in the form of the TIGER/Line data set. This data is freely available from the Census Bureau web site at http://www.census.gov/geo/www/tiger/. As of 2004, updated versions are published twice a year.

The problem is that the TIGER/Line data set is composed of 3,233 separate ZIP files, one for each county in the entire United States. The entire data set is 4.3 GB compressed and runs to almost 16 GB uncompressed. That's a lot of data to struggle with if you just want to look up a few lousy addresses. This is where *Geocoder.us* comes in.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Geocoder.us offers a web service (http://geocoder.us/) for geocoding U.S. street addresses from TIGER/Line data. Actually, several styles of web service are offered, including SOAP, XML-RPC, and REST. All of these services have but one goal, which is to take a street address or intersection in the U.S. and turn it into latitude and longitude coordinates that can be displayed on a map.

Figure 7-1 shows the working demo application of the Geocoder.us service at http://geocoder.us/demo.cgi. The latitude and longitude returned from the lookup can be trivially turned into a marker on a map, using the Google Maps API. You can use this URL to test out the service manually, but if you want to do address lookups in your program, you should *definitely* use the web service interfaces instead.

7.2.2. The Code

7.2.2.1. XML-RPC.

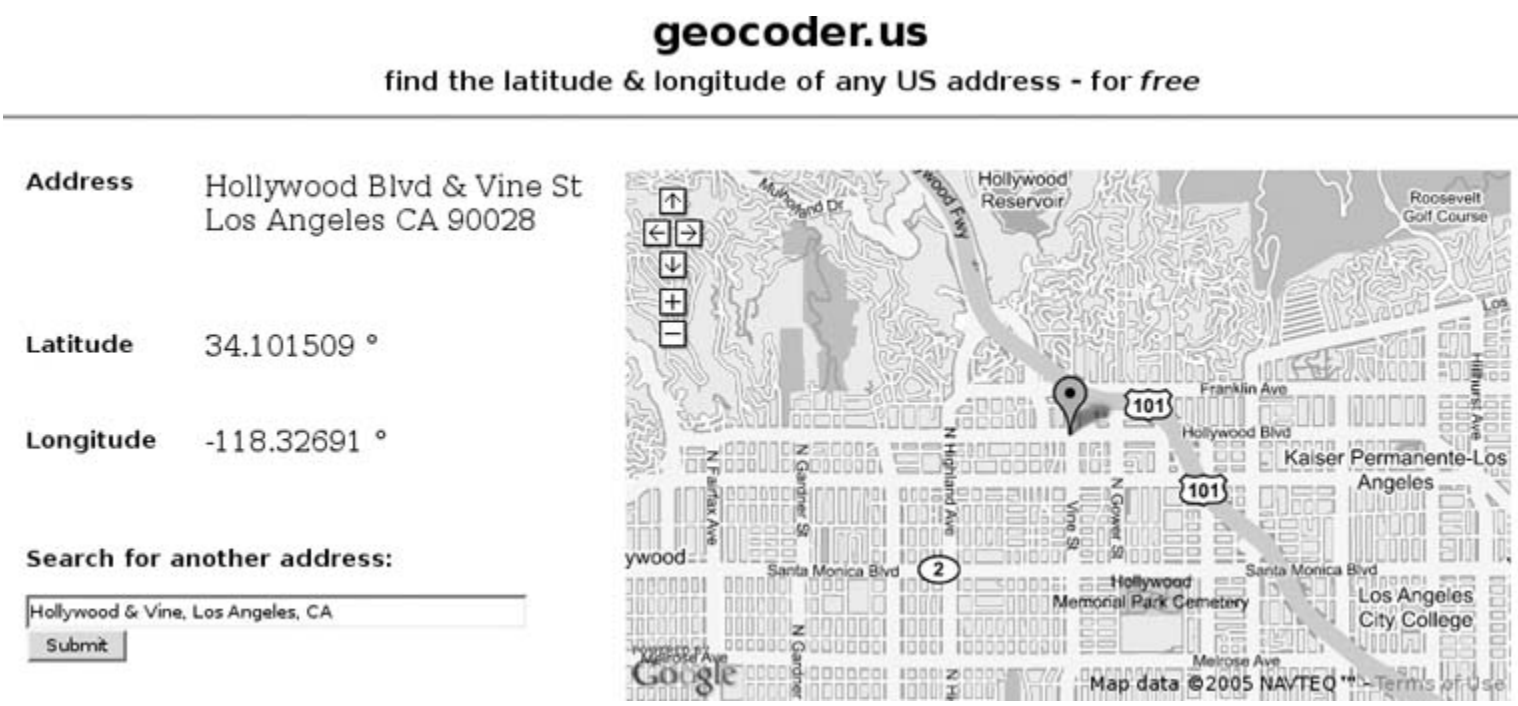
The easiest way we know for accessing the Geocoder.us web service is to use the XML-RPC interface from within Perl. The outline of the code looks like this:

```
use XMLRPC::Lite;

my $result = XMLRPC::Lite
  -> proxy( 'http://rpc.geocoder.us/service/xmlrpc' )
  -> geocode( '1005 Gravenstein Hwy, Sebastopol, CA 95472' )
  -> result;
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 7-1. The geocoder.us site at work



Either a properly formatted U.S. street address or an intersection of the form "Hollywood Blvd & Vine St, Hollywood, CA" will be accepted by the web service. You must supply *either* a city and state *or* a ZIP Code, though providing both doesn't usually hurt.

If the lookup succeeds, then the `$result` variable will contain a reference to an array. Each item in the array is a hash, or associative array, that contains key/value pairs describing the results of the lookup. The following outlines the structure of the data returned by the XML-RPC request:

```
$result = [
    {
            'number' => '1005',
            'prefix' => ''
            'street' => 'Gravenstein',
            'type' => 'Hwy',
            'suffix' => 'N',
            'city' => 'Sebastopol',
            'state' => 'CA',
            'zip' => '95472',
            'lat' => '38.411908',
            'long' => '-122.842232',
    }
];
```

As you can see, the Geocoder.us web service attempts to break an address into its components and then normalizes those components before doing the lookup, and then returns the normalized components along with the coordinates. Here's a bit of code that prints the latitude/longitude pairs returned:

```
if ($result) {
    for my $address (@$result) {
        if ($address->{lat}) {
            print "Address found: $address->{lat} $address->{long}\n";
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
        } else {
            print "Couldn't locate the address!\n";
        }
    }
} else {
    print "Couldn't parse the address!\n";
}
```

The `$result` variable should be tested for truth before accessing its contents, because the service will return an undefined value if the address can't be parsed. If the address *can* be parsed, but no match is found in the database, the result will be an array containing a single hash, with an empty string in place of the latitude and longitude values. Finally, if the address given matches multiple addresses in the database, the array will contain a separate hash for each match found.

The `XMLRPC::Lite` Perl module is part of the `SOAP::Lite` distribution. If you don't have either module installed on your system, download them from the CPAN at http://search.cpan.org/ or use the CPAN shell to install them as follows:

```
# perl -MCPAN -e 'install XMLRPC::Lite'
```

XML-RPC client libraries are also available for virtually every programming language out there, so you shouldn't have too much trouble finding one for your language of choice.

7.2.2.2. REST.

If, for whatever reason, you prefer not to use XML-RPC, you can always use the Geocoder.us REST service, by sending an `HTTP GET` request to `http://rpc.geocoder.us/service/rest/geocode?address=`*`[your address here]`*. (Don't forget to URI-escape the address—e.g., turn whitespace to `%20`—before passing it to your HTTP client, if your client library doesn't do it for you. The `URI::Escape` module from the CPAN can help with this.)

The REST interface returns an RDF/XML document with `geo:Point` elements for each address match. Here's an example:

```
<rdf:RDF
     xmlns:dc="http://purl.org/dc/elements/1.1/"
     xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

<geo:Point rdf:nodeID="aid34279220">
       <dc:description>1005 Gravenstein Hwy N, Sebastopol CA 95472</dc:
description>
       <geo:long>-122.842232</geo:long>
       <geo:lat>38.411908</geo:lat>
</geo:Point>
</rdf:RDF>
```

Multiple `geo:Point` elements will be returned in the document if multiple matches for the requested address are found.

Here's a bit of Perl code that makes use of the REST service. You will need the `XML::Simple`, `LWP::Simple`, and `URI::Escape` modules from the CPAN.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
use XML::Simple;
use LWP::Simple;
use URI::Escape;
use strict;

sub geocode_rest {
    my $xml = get( "http://rpc.geocoder.us/service/rest/geocode" .
                   uri_escape( $address );
    if ($xml) {
        my $result = eval { XMLin( $xml, ForceArray => ['geo:Point'] ) };
        if ($result) {
            my $points = $result->{'geo:Point'};
            for my $point (@$points) {
                my $lat = $point->{'geo:lat'};
                my $lon = $point->{'geo:long'};
                if ($lat and $lon) {
                    ### Success! Do something with the coordinates.
                } else {
                    ### Couldn't find a match for the address.
                }
            }
        } else {
            ### Couldn't parse the XML, so the service spit
            ### out an error message, meaning it couldn't parse the address.
    } else {
        ### The HTTP GET failed, indicating a network error.
    }
}
```

7.2.2.3. SOAP.

The SOAP interface to the Geocoder.us web service is probably the most difficult to use, entirely owing to the subtle complexities of SOAP itself. However, if you're a Java or C# user, it might actually be easier for you to use SOAP than the other web service interfaces, because you can usually autogenerate result classes from the WSDL description. Accordingly, you may be pleased to know that a WSDL file exists for the Geocoder.us SOAP interface at http://geocoder.us/dist/eg/clients/GeoCoder.wsdl.

7.2.3. The Caveats

In principle, you should be able to access the Geocoder.us web services directly from the browser by using JavaScript, but in practice, the browser security model of the better web browsers out there will prevent you from using JavaScript to access Internet domains aside from the one that your page originates from. One way around this, which many sites are currently using, is to set up a server-side script that accesses the Geocoder.us site, as shown above. You can then use the `GXmlHttp` class from the Google Maps API to request address lookups via your site's geocoding proxy, and get the results back in JavaScript as either XML or plain text, depending on your preferences.

One caveat you should be aware of is that the TIGER/Line data set is less complete and/or less accurate in some areas than the commercial data sources that Google Maps uses to locate addresses. The flip side, of course, is that TIGER/Line is freely available. It's a trade-off! In general, residential addresses are more likely to be accurate than addresses in industrial areas or commercial office parks.

Also, you *need* to examine the terms and conditions of service for Geocoder. us at http://geocoder.us/terms.shtml, before using their service. In particular, use of the free web services for for-profit commercial ends is strictly prohibited. If you are a commercial user, you will need to subscribe to the commercial service instead.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

The final caveat is that the Geocoder.us web services are throttled by IP address, to keep the service from being hammered by a single user. As of this writing, a delay of 15 seconds between requests is in place, but Locative Technologies, the maintainers of the Geocoder.us service, reserves the right to adjust this upwards or downwards as necessary. If the service seems unduly slow, it's probably because your requests aren't spaced more than 15 seconds apart. Of course, commercial account users don't suffer this restriction.

If the throttling or the expense are a problem for you, you can always set up your own local Geocoder.us database, by downloading the `Geo::Coder::US` module and its prerequisites from the CPAN. The documentation for the `Geo::Coder::US::Import` module, which comes with the distribution, contains all the instructions you need for getting the TIGER/Line data from the Census Bureau, and using it to build your own local database. Although the Geocoder.us database for the whole country runs to almost 800 MB, you can simplify things for yourself by constructing a database composed only of the counties you're interested in.

7.2.4. Geocoding Addresses Outside the U.S.

Geocoding street addresses outside the U.S. for free is considerably harder, because the data sets simply aren't freely available. (Denmark is one notable exception.) In Canada, you may be able to use the Geocoder.ca web site at http://geocoder.ca/, but that service is based on government data that itself is not freely available. For Europe, Japan, and the rest of the world, you may have to purchase access to a commercial data set in order to geocode addresses in your country. The alternative is to petition your government's legislators to change its geospatial data access policies, so that citizens in your country can access information whose collection has already been subsidized with your tax money!

7.2.5. See Also

- If you plan to use the Geocoder.us web services, you should definitely read the developer documentation at http://geocoder.us/help/, as well as the terms and conditions of service at http://geocoder.us/terms.shtml.
- If you want to set up your own U.S. address geocoder, you can start by getting the `Geo::Coder::US` backend code from http://search.cpan.org/~sderle/Geo-Coder-US/, or by using the CPAN shell to install the module and its prerequisites automatically. Be sure to read the documentation for the `Geo::Coder::US::Import` module.
- The `Geo::StreetAddress::US` module contains all of the rules for how street addresses are parsed. We always welcome patches to this module, to help improve our hit rate!
- The Census Bureau's TIGER/Line data set lives on the Webat http://www.census.gov/geo/www/tiger/.

Hack 63. Read and Write Markers from a MySQL Database

Keep track of almost anything with Google Maps and a relational database.

Adding a map of something new to Google Maps is good fun, but there is a lot of data already in SQL databases that is just begging to be mapped. This tutorial describes the way the Subfinder application, at http://www.map-server.com/googlemaps/subfinder.php, is integrated with a MySQL database using PHP. The Subfinder is itself an extension of the Who Locations site at http://www.map-server.com/googlemaps/

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

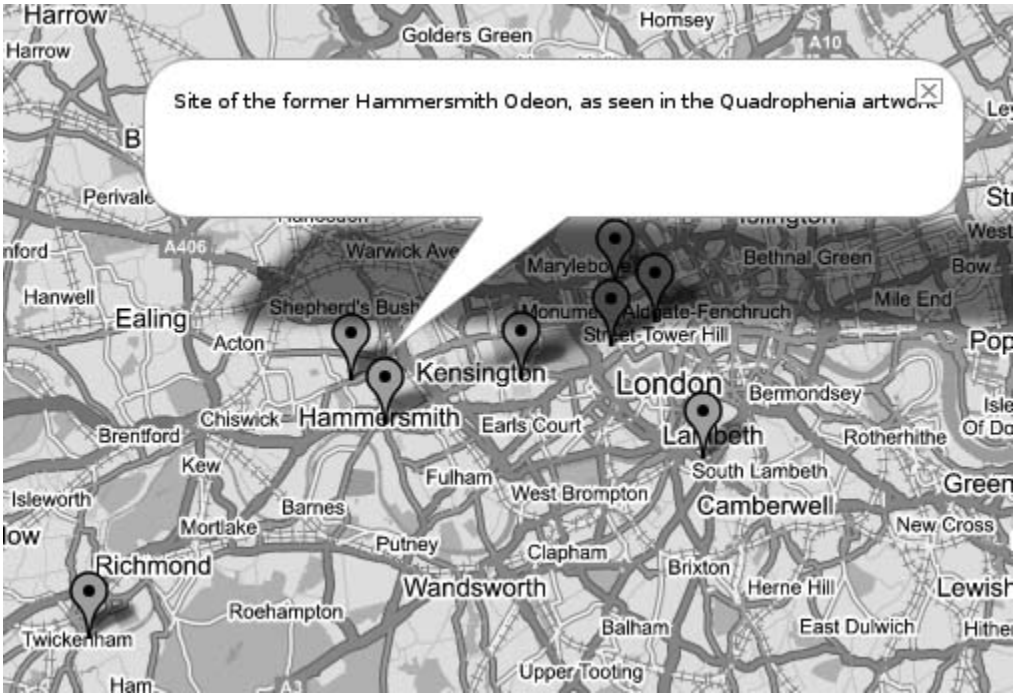
wholocations2.php, which is described in a tutorial at http://www.map-server.com/googlemaps/tutorial.html. Figure 7-2 shows a map of the sites that any fan of The Who must know about.

There are easier ways to get points on a Google Map, but to integrate with a MySQL database you will need:

1. An Apache web server running PHP and MySQL. (Other web server software with PHP and MySQL will probably do as well.)
2. A table in your database with `lat`, `long`, and `description` columns.
3. Some basic HTML and PHP knowledge.

We are going to use PHP to dynamically create an HTML document with the appropriate Google Maps JavaScript code. One of the functions shown in this sample is the option for users to add their own locations, with additional attributes, into the database. The value of one of the attribute determines the type of marker placed.

Figure 7-2. A map for Who fans



7.3.1. Structuring Your Database

The database can be recreated with this snippet of SQL:

```
CREATE TABLE subfinder (
  id int(11) NOT NULL auto_increment,
  lat decimal(10,6) NOT NULL default '0.000000',
  lon decimal(10,6) NOT NULL default '0.000000',
  desc varchar(255) NOT NULL default '',
  url varchar(255) NOT NULL default '',
  marker char(1) NOT NULL default '',
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
  PRIMARY KEY (id)
) TYPE=MyISAM;
```

If you save this into the file *create_subfinder.sql*, you can create a new database and this table with the following commands:

```
$ mysqladmin create sub_db
$ mysql sub_db < create_subfinder.sql
```

As you can see, there are fields for the latitude and longitude values, a description, URL (which will of course be displayed as a link), and a marker field. The value of the marker field will determine the type of marker placed. This application, which lists submarines visible on the Google Maps photos, makes a distinction between submarines that are in active duty (*A*), museum ships (*M*), places where submarines have sunk (*S*), and places where important events have taken place (*E*). The appropriate letter is displayed in the marker.

7.3.2. The Code

This system uses two PHP files to do its work. The main file, *subfinder.php*, is used to read the data from the database and display it on the map. The second file, *subfinder_load_db.php*, is called by the first to write data into the database. A standard HTML form is used for this.

The *subfinder.php* file imports the Google Maps API, and sets up the map in the usual way. The following bit of JavaScript creates the custom markers:

```
subfinder.php: setting up the custom markers
<script type="text/javascript">
//<![CDATA[

var baseIcon = new GIcon();
baseIcon.shadow = "http://www.map-server.com/img/shadow50.png";
baseIcon.iconSize = new GSize(20, 34);
baseIcon.shadowSize = new GSize(37, 34);
baseIcon.iconAnchor = new GPoint(9, 34);
baseIcon.infoWindowAnchor = new GPoint(9, 2);
baseIcon.infoShadowAnchor = new GPoint(18, 25);

// Creates a marker whose info window displays the given number
function createMarker(point, text, markerstyle)
{
  var icon = new GIcon(baseIcon);
  icon.image = "http://www.map-server.com/img/marker" + markerstyle + ".png";
  var marker = new GMarker(point, icon);
  // Show this markers index in the info window when it is clicked
  var html = text;
  GEvent.addListener(marker, "click", function() {
     marker.openInfoWindowHtml(html);
  });
  return marker;
}
```

In the code above, I've copied the PNG image used by the official Google sample to my own server for this. I then made four different PNG files, one for each letter used (A, E, M, S). This letter is being passed on through the `markerstyle` variable.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Next, the map is defined, some controls are added, and the initial map view is set through the `map.centerAndZoom` function. A relatively new addition to the Google Maps API (undocumented at the time of this writing) is the `GScaleControl` class, which adds a scale bar in the lower-left corner:

```
var map = new GMap(document.getElementById("map"));
map.addControl(new GLargeMapControl());
map.addControl(new GMapTypeControl());
map.addControl(new GScaleControl());
map.centerAndZoom(new GPoint(0, 0), 16);
```

Next, we use some PHP code to read the information from the database, parse it, and write it into the document. I've used an external configuration file (*conf.php*) to declare the variables that hold the server, username, and password for my database connection. A link to the database is set up and a query is passed to it.

```
<?php

     include_once("conf.php");
     $link = mysql_connect($dbserver, $username, $password)
       or die("Could not connect: " . mysql_error());
     mysql_select_db("dbmapserver",$link)
       or die ("Can\'t use dbmapserver : " . mysql_error());

     $result = mysql_query("SELECT * FROM subfinder",$link);
     if (!$result)
     {
       echo "no results";
     }
```

The result of this query is then put into an array. For every element in the array, the function `createMarker` (which was defined above) is called with the lat, long, and description of that point. Note that this will generate the code for the browser to interpret. If you look at the source code of the page in your browser, you'll see the result of this PHP code.

```
while($row = mysql_fetch_array($result))
{
  if ($row['marker'] == 'A') $marker_type = 'Active';
  if ($row['marker'] == 'M') $marker_type = 'Museum';
  if ($row['marker'] == 'S') $marker_type = 'Sunk';
  if ($row['marker'] == 'E') $marker_type = 'Event';
  $info_text = "<b>" . $marker_type . "</b><br>" .
        $row['desc'] . "<br><br><a href=" . $row[url] .
        " target=_blank>Link</a>";
  echo "var point = new GPoint(" . $row['lon'] . ","
                          . $row['lat'] . ");\n";
  echo "var marker = createMarker(point, '" . $info_text
                            . "','" . $row['marker'] . "');\n";
  echo "map.addOverlay(marker);\n";
  echo "\n";
}

mysql_close($link);
?>
```

The next piece of code from *subfinder.php* sets up the HTML form to add new information to the database:

```
GEvent.addListener(map, 'click', function(overlay, point) {
  if (overlay) {
    //map.removeOverlay(overlay);
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
  } else if (point) {
    var form;
    form = '<form name="form1" method="post" action="subfinder_load_db.php">';
    form += '<table border="0"><tr><td>Lat:</td>';
    form += '<td align="left"><input name="new_lat" type=text id=new_lat
value="
          + point.y + '"></td>';
    form += '<td rowspan="4">';
    form += '<input name="new_marker" type="radio"
       value="A" checked="1"> Active';
    form += '<br><input name="new_marker" type="radio" value="M">Museum';
    form += '<br><input name="new_marker" type="radio" value="S">Sunk';
    form += '<br><input name="new_marker" type="radio"
       value="E">Event</td></tr>';
    form += '<tr><td>Lon:</td>';
    form += '<td align="left">';
    form += '<input name="new_lon" type="text" id="new_lon" value="
         + point.x + '"></td></tr>';
    form += '<tr><td>Text:</td>';
    form += '<td align="left">';
    form += '<input name="new_desc" type="text" id="new_desc" size="60">
    form += '</td></tr>';
    form += '<tr><td>URL:</td><td align="left">';
    form += '<input name="new_url" type="text" id="new_url" size="60">';
    form += '</td></tr>';
    form += '<tr></td><td align="left">';
    form += '<input type="submit" name="Submit" value="Submit"></td></tr>';
    form += '</table></form>';
    output.innerHTML = form

    if (map.getZoomLevel() >= zoomToLevel) {
        map.centerAndZoom(point, zoomToLevel);
    }
    map.addOverlay(new GMarker(point));
  }
});
//]]>
</script>
<div id="output"></div>
```

The block that contains all the HTML is generated when the user clicks somewhere on the map to add a location, which sets up a form to allow the user to add more information. This form is stored in `output.innerHTML`. The `point.x` and `point.y` variables are being supplied by the listener.

The PHP file that gets called by the form in *subfinder.php* is a lot simpler. The code, stored in *subfinder_load_db.php*, is just your basic "take the parameters and stuff them in the database" code:

```
<?php

include_once("conf.php");

echo "<html><head><link href=\"/style/hans.css\" rel=\"stylesheet\" type=\
"text/css\">";
echo "</head><body>";
echo "Lat: " . $new_lat . "<br>";
echo "Lon: " . $new_lon . "<br>";
echo "Text: " . $new_desc . "<br>";
echo "URL: " . $new_url . "<br>";
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
echo "Marker: " . $new_marker . "<br>";
echo "Will now be added to the database…";

$new_desc = addslashes($new_desc);
$link = mysql_connect($dbserver, $username, $password)
    or die("Could not connect: " . mysql_error());
mysql_select_db("dbmapserver",$link)
    or die ("Can\'t use dbmapserver : " . mysql_error());
$sql = "INSERT INTO subfinder ";
$sql = $sql . "VALUES('','"
    .$new_lat."','".$new_lon."','".$new_desc."','".$new_url."','"
    .$new_marker ."')";
$result = mysql_query($sql ,$link);
if (!$result)
{
  echo "<p>Due to an error (" . mysql_error() . ")<br>, your entry could "
    . "not be loaded into the database. Please return to "
    . "<a href=\"subfinder.php\">Subfinder</a>.";
} else {
  echo "<p>Your entry has been loaded into the database. "
    . "Please return to <a href=\"subfinder.php\">SubFinder</a>.";
}

echo "</body></html>";
?>
```

Once again, the *conf.php* with the access variables is read. One important thing to note here is the `addslashes()` function. As we mentioned earlier, JavaScript can be quite fussy about quotes and special characters. If there are any of those characters in the texts you want to display in the info windows, it may result in either the map not being drawn at all, or just your markers not being drawn. The `addslashes()` function properly escapes those characters before putting the text into the database so that your users don't have to worry about that themselves.

7.3.3. See Also

- Red Geographics Map-Server page: http://www.map-server.com/

—Hans van der Maarel

Hack 64. Build Custom Icons on the Fly

Using libraries that extend the Google Maps API in various ways, you can quickly achieve powerful results.

The XMaps library is a set of extensions to the Google Maps API. As of this writing, the library builds on the Google Maps API to add many capabilities that the API itself does not provide. As of this writing, the additional features provided by XMaps include filled polygons, text along polylines, and custom icons and markers. For this hack, we will see how to easily create new icons by making use of the `XIcon` and `XMarker` classes from XMaps. The XMaps library lives on the Web at http://xmaps.busmonster.com/.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

7.4.1. The Hack

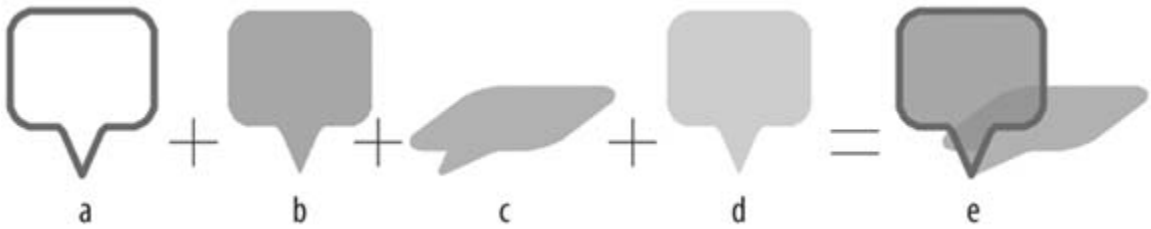
The `XIcon` class is similar to Google's `GIcon` class, but it automates some of the steps necessary to create icons, making it simpler to make new icons on the fly. You could do the same thing that the `XIcon` class does by using a server-side script, but here we are using something that does not require any extra server horsepower or bandwidth from you. `XIcon` objects are limited in that they are a filled concave polygon (e.g., the outline of a car, lion, or circle), but you have the advantage of being able to specify the color, size, opacity, and any text on it on the fly.

The `GIcon` class makes use of a set of pre-rendered PNG images stored on a server somewhere. The default `GIcon` uses three images from Google Ride Finder to create its resulting image: the reddish-orange marker image, a nearly transparent version of that same image (for click detection on IE), and a mostly transparent shadow image. Figure 7-3 shows the parts of this pre-rendered icon.

Figure 7-3. Google Maps pushpin icon parts: (a) the pushpin image, (b) the shadow, (c) the click target for IE, and (d) the resulting icon

Instead of using pre-rendered images, an `XIcon`'s images are created on the fly by using the existing (but unofficial) service that Google Maps already uses to draw polylines. Using this service, we can actually draw images of any color and opacity. By specifying the outline of the icon shape, the `XIcon` class can create the necessary images for you (including the shadow image). For example, if we wanted to create a rounded box icon, we could come up with a list of points for the outline, give them to `XIcon`, and get the set of images shown in Figure 7-4 for the icon.

Figure 7-4. Custom XMaps icon parts: (a) the box outline, (b) the filled box, (c) the shadow, (d) the click target, and (e) the final rounded box icon



Because we are creating an additional image in the set of images used for the icon, we cannot use the existing `GMarker` class in the Google Maps API, which can only use the images in a `GIcon`. Instead, we use the `XMarker` class, which can handle either `GIcon` or `XIcon` objects. `XMarker` gives us an additional ability that `GMarker` does not provide, namely the ability to place text or other HTML on top of the icon. When we create our icon, we will specify the bounds of any content that might be displayed on top of the icon, and an `XMarker` will use these bounds to place text or a generic HTML DOM element on top of the icon.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

7.4.2. The Code

Suppose, for example, that we wanted to create a web site all about the Bermuda Triangle. What better icon to use for our map than a custom-made triangle pointing at our map! I'll assume you are familiar with the general layout of a Google Maps page. The fragments of code here require that there is an existing `GMap` object already created and assigned to a JavaScript variable called `map`.

The first thing we'll have to do is include the XMaps library along with the Google Maps API JavaScript files. Here's the code that should appear at the beginning of our page. Note that you will want to replace the API key with one for your site, and you will want to make sure you are using the most recent version of the XMaps library from http://xmaps.busmonster.com/.

```
<script src="http://maps.google.com/maps?file=api&v=1&key=abcdefg"
       type="text/javascript"></script>
<script src="http://xmaps.busmonster.com/script/xmaps.1b.js"
       type="text/javascript"></script>
```

Next, let's define the icon's shape. Here I've specified a triangle approximately 30 pixels to a side.

```
var triShape = {
    iconAnchor: new GPoint(15, 26),
    infoWindowAnchor: new GPoint(15, 0),
    shadow: true,
    contentAnchor: new GPoint(0, 0),
    contentSize: new GSize(31, 20),
    points: [0,0, 15,26, 30,0]
};
```

As you'll see, we can scale this later, so the initial size does not matter too much. The shape description is contained in a simple associative array, with the following properties:

`iconAnchor`
: This is the pixel of the icon's image, which should be tied to the point on the map. In this case, it is the bottom middle pixel (the triangle's bottom point).

`infoWindowAnchor`
: This is the pixel on the icon's image, which should be tied to the bottom point of the info window.

`shadow`
: This determines if the icon will have a shadow or not (the shadow is created for us automatically).

`contentAnchor T`
: This is the top left pixel on the icon's image that should be used for any HTML content on top of the icon (e.g., text).

`contentSize`
: This is the size, in pixels, of the box in which HTML content is allowed.

`points`
: This is an array of x/y pairs that define the shape of the icon. In this case, we have the corners of an upside-down triangle.

You can see the purpose of some of these properties in Figure 7-5.

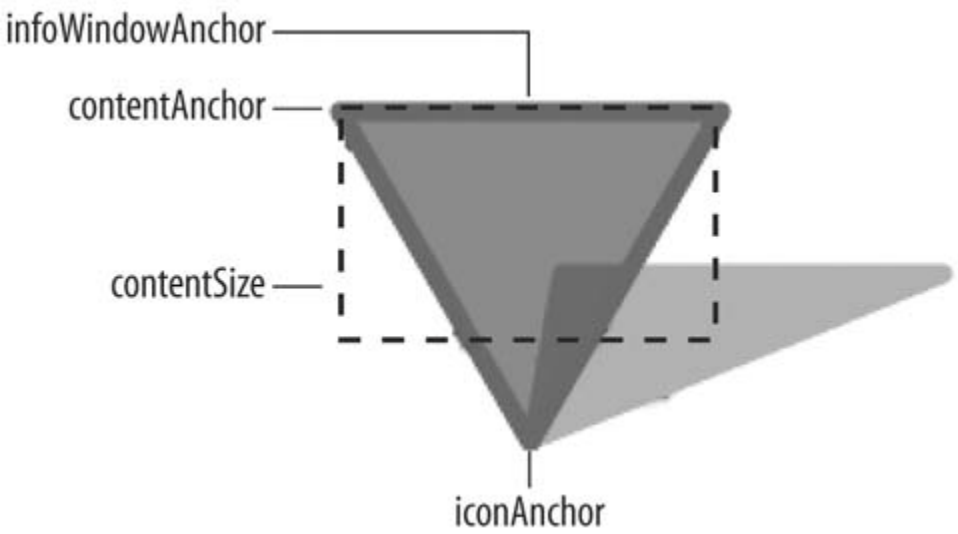
Now we are ready to give our shape a name. We do this so that XMaps can cache the URLs that are used to generate these shapes for better performance.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
XIcon.shapes['triangle'] = triShape;
```

Our shape is now registered with the Xmaps API, and we are ready to use it. When we create an `XIcon`, we have a number of style properties we can specify. Each of these has defaults, so you will not always have to specify all of them, but I will do so here to show you an example of each property:

Figure 7-5. Triangle icon properties




```
var triStyle = {
    scale: 1.5,
    outlineColor: '#ffffff',
    outlineWeight: 2,
    fillColor: '#7f00ff',
    fillOpacity: 0.5
};
```

Once again, the icon's style is defined with a simple associative array. Here's what each property does:

`scale`

Specifies how big we want our icon. Since I specified 1.5, we will end up with a triangle with approximately 30 * 1.5 = 45 pixels per side.

`outlineColor`

Specifies the color for the outline of the icon. Here, we are using a white outline.

`outlineWeight`

Specifies the width, in pixels, of the icon's outline.

`fillColor`

Specifies the color of the interior of the icon.

`fillOpacity`

Specifies the opacity of the interior of the icon; must be in the range 0 to 1.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

We are ready to create our icon. The call to the constructor looks like this:

```
var triIcon = new XIcon('triangle', triStyle);
```

Now we have an icon we can use with any `XMarker`. Here, we'll plot one:

```
var bermudaTriangle = new GPoint(-70.532227, 25.878994);
var iconText = "?";
var hoverText = "The Bermuda Triangle";
var triMarker = new XMarker(bermudaTriangle, triIcon, iconText, hoverText);
map.addOverlay(triMarker);
```

The `XMarker` constructor takes a point and a few optional arguments. We pass the icon we created as well as some text to be displayed on the icon and when the mouse hovers over the icon. The XMaps API allows us to pass the `XMarker` object directly to `map.addOverlay ( )`, just as if it were an ordinary `GMarker`.

And we're done! With a minimum of effort, we have a new marker with a custom icon that has its own shadow, is scaled and colored like we want, and has the clickable region specified by the icon outline. To see the results, browse over to http://xmaps.busmonster.com/triangle.html and check it out.

We can follow these same steps to create lots of reusable icons. To give you a head start, XMaps comes with a few handy premade icons, including the rounded box you saw in Figure 7-4. You can play around with the web pages listed below to see more examples of custom icons with XMaps.

7.4.3. See Also

- The XMaps Library at http://xmaps.busmonster.com/ has links to the newest version of the library, full documentation, and many examples.
- The Bermuda Triangle example at http://xmaps.busmonster.com/triangle.html shows a completed version of our code fragments from above.
- The Icon Color Shades example at http://xmaps.busmonster.com/shades.html shows different icons of different colors.
- The Bermuda Triangle entry at http://en.wikipedia.org/wiki/Bermuda_triangle in Wikipedia has plenty of information about how and where to get lost at sea!

—*Chris Smoak*

Hack 65. Add More Imagery with a WMS Interface

What's a pretty face like you doing on the geospatial web?

The Google Maps service is cool, the user interface is wonderful, the API is a joy, and the maps are pretty, but there is more to the world of mapping than what fits on one company's massive server farm.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

7.5.1. The Hack

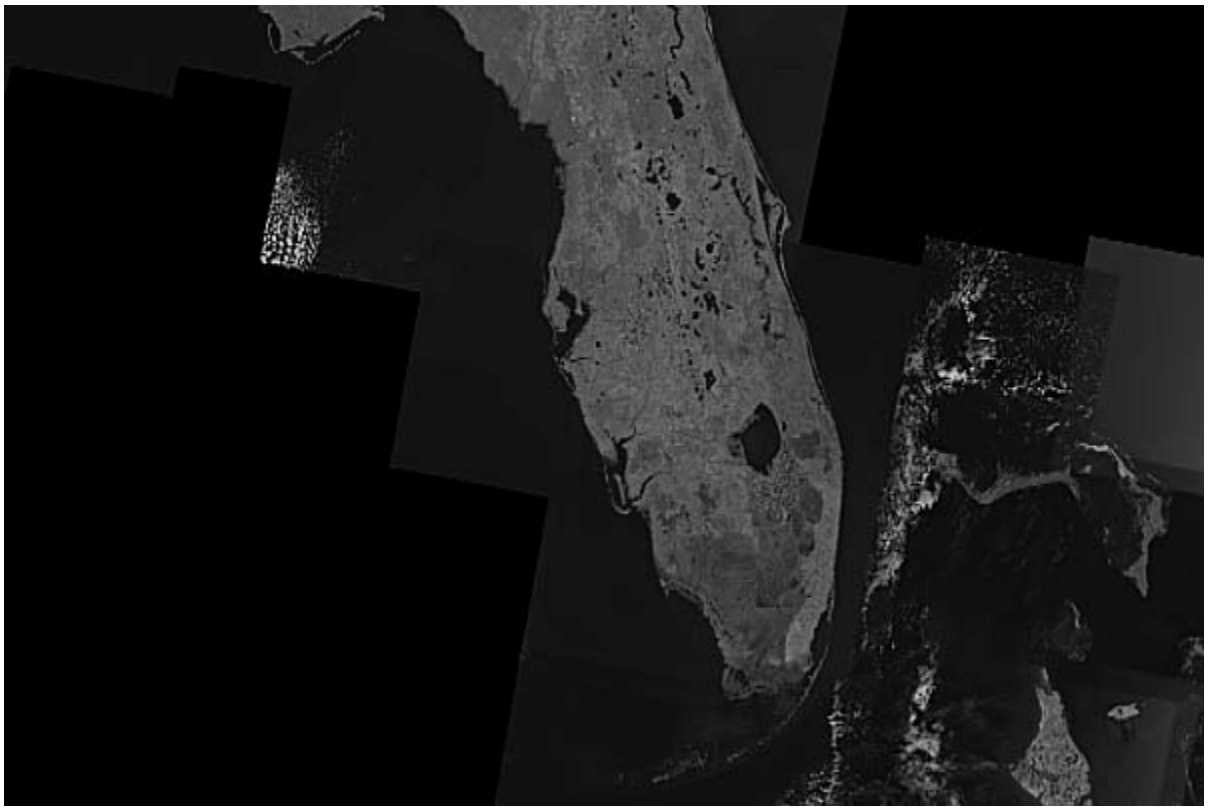
The Open Geospatial Consortium (OGC) has defined a number of interoperability standards for web mapping applications. The simplest to understand is the Web Mapping Service (WMS) specification. In Google Maps terms, you would ask a WMS server for a tile that contains something like a Google Map or Satellite image. The key is that the WMS is not limited to the data that Google Maps maintains. There are WMS servers with things from Landsat imagery to topographic (elevation) data to habitat data. WMS requests are normally made by programs, but you can assemble a request by hand.

If you put this URL into your browser:

```
http://wms.jpl.nasa.gov/wms.cgi?request=GetMap&layers
   =global_mosaic_base&srs=EPSG:4326&bbox=-87.90,24.38,-76.65,30.01
   &FORMAT=image/png&width=600&height=400&styles=pseudo
```

you will get the image of Florida shown in Figure 7-6.

Figure 7-6. Florida as shown by a WMS request



This looks a lot like a Google Maps tile. Even without understanding the format of a WMS request, you can look at that URL and get some information. For example, it has a `bbox` (or *bounding box*) that covers the area from 87.9 W to 76.65 W and 24.38 N to 30.01 N, which happens to be most of Florida.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

You can make a request to a WMS server for a map that covers a certain area. But what is a map? WMS returns a raster (an image) for the selected area and for selected features. WMS Servers report what their capabilities are, what sorts of information they contain, and WMS clients can then select the layers they want to show.

David Knight has written a JavaScript extension library that you can insert into your own Google Maps page to let you browse through tiles from a WMS server using the Google Maps user interface. For an example of this, visit the Global Coordinate page at http://globalcoordinate.com/. Figure 7-7 shows Landsat imagery of South Florida within a Google Maps interface. The Global Coordinate page also demonstrates near real-time updating of the map from RSS feeds.

Figure 7-7. Florida from a WMS Server

7.5.2. The Code

We've created a simple example of a WMS-enabled Google Map at http://mappinghacks.com/projects/gmaps/wms.html. Sure you have the standard map and satellite views, but it includes a little extra kick in the data layer department. It also includes Landsat and VMAP0 data for the whole world and elevation and landcover layers for the United States. It will produce a display like Figure 7-8.

Assuming you have created the Hello World map **[Hack #10]**, you can add these WMS layers with this code:

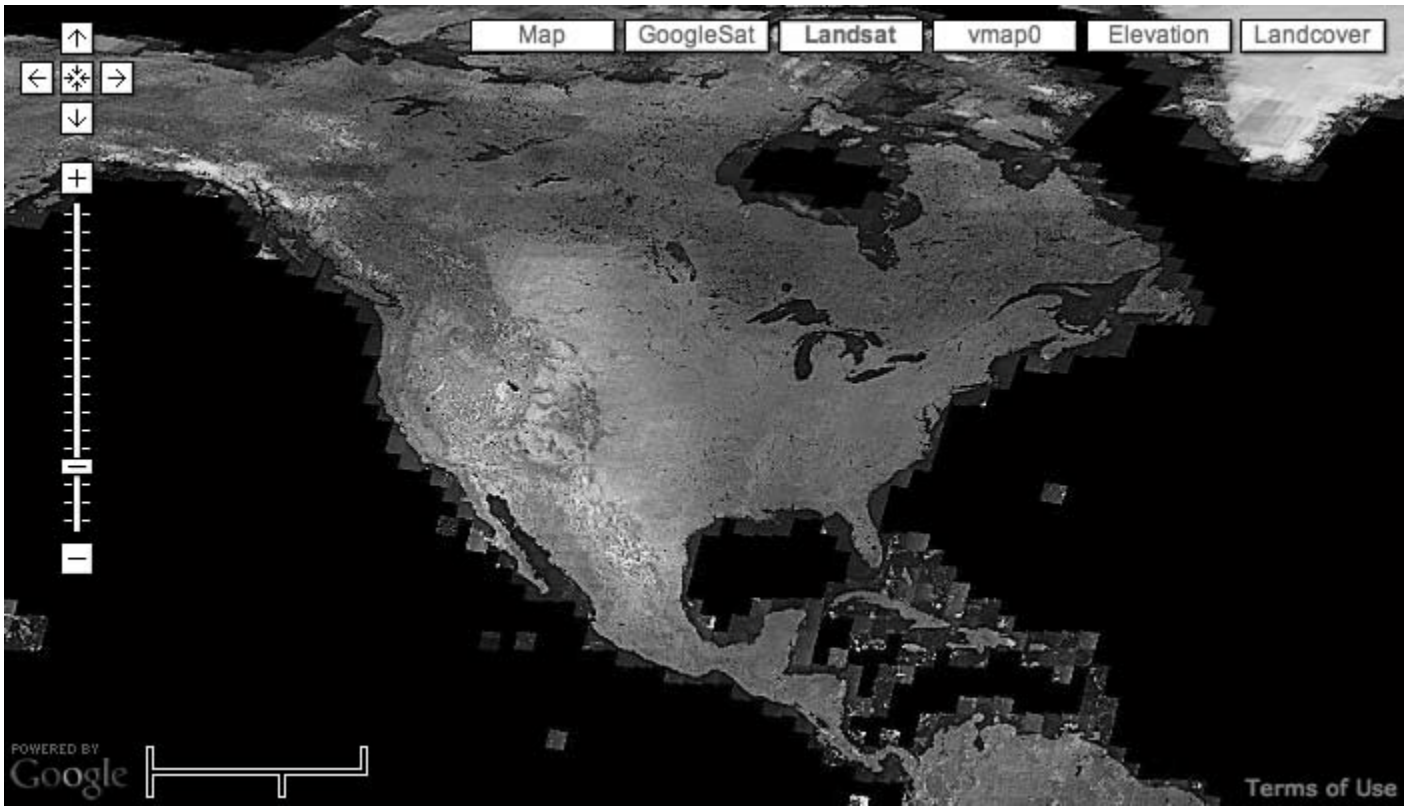
```
<script src="http://www.globalcoordinate.com/jscript/WMSTiles.js"
      type="text/javascript"></script>
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

This includes David's library. You can look at it by pointing your browser to this URL: http://www.globalcoordinate.com/jscript/WMSTiles.js.

Using the WMS tiles library is fairly straightforward. In the following JavaScript example, the `mapSpecs` variable holds an array of objects that is used to load parameters needed for the WMS fetch and display.

Figure 7-8. WMS layers on a Google Map




```
var mapSpecs = [];
createMapSpecs();

_SATELLITE_TYPE.getLinkText = function() {
    return "GoogleSat";
}

mapSpecs.push(_GOOGLE_MAP_TYPE);
mapSpecs.push(_SATELLITE_TYPE);
```

First, we create a new WMS map specification and then push it onto `mapSpecs`. To create a layer, pass the base WMS URL and the name that you want displayed on your map. There are four layers here: Landsat, VMAP0, National Elevation Dataset (NED), and National Landcover Dataset. These four layers come from three different servers. The Geospatial web is alive, and with a few lines of code you can add more, different, and sometimes better layers to your Google Maps applications!

```
wmsSpec =new _WMSMapSpec ("http://wms.jpl.nasa.gov/wms.cgi?request
   =GetMap&layers=global_mosaic_base&srs=EPSG:4326","Landsat");
mapSpecs.push(wmsSpec);
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
wmsSpec =new _WMSMapSpec ("http://geocoder.us/surfer/wms.cgi?service\
   =wms\&version\=1.1.1\&request=GetMap&layers=vmap0&srs
   =EPSG:4326", "vmap0");
mapSpecs.push(wmsSpec);

wmsSpec =new _WMSMapSpec
   ("http://gisdata.usgs.net/servlet/com.esri.wms.Esrimap?servicename
   =USGS_WMS_NED&request=GetMap&layers=US_NED_Shaded_Relief&srs
   =EPSG:4326","Elevation NED");
mapSpecs.push(wmsSpec);

wmsSpec =new _WMSMapSpec
   ("http://gisdata.usgs.net/servlet/com.esri.wms.Esrimap?servicename
   =USGS_WMS_NLCD&request=GetMap&layers=US_NLCD&srs=EPSG:4326","Landcover");
mapSpecs.push(wmsSpec);
```

Next, we pass the array of `mapSpecs` to the `GMap` constructor and set normal attributes with the API calls.

```
var map = new GMap(document.getElementById("map"),mapSpecs);

map.setMapType( _SATELLITE_TYPE );
map.addControl(new GLargeMapControl());
map.addControl(new GScaleControl());
```

This puts the map layer choices on the map. Without this, the only way to select between Map, Satellite, Landsat, VMAP0, NED, and Landcover would be with custom navigation—say a button that triggers JavaScript to set the appropriate map type. (Map "type" is Google-speak for "raster layer".)

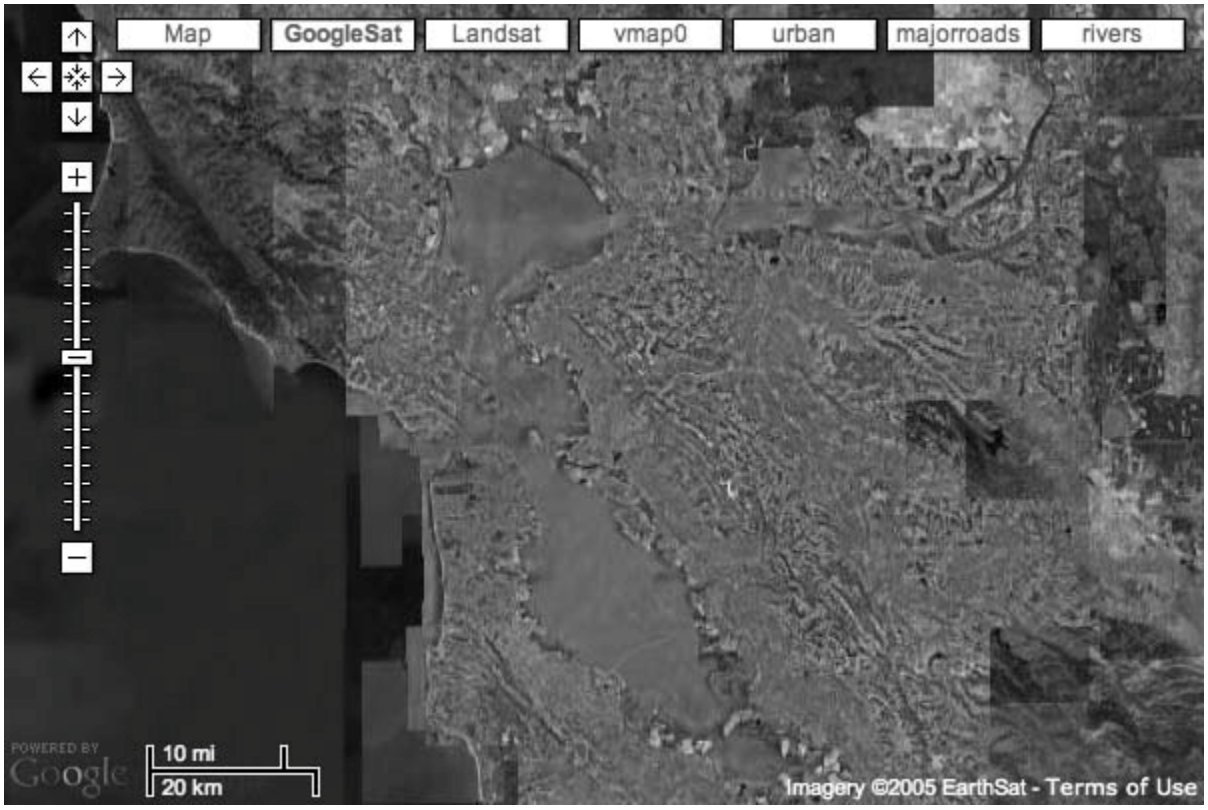
```
map.addControl(new GMapTypeControl());
map.centerAndZoom(new GPoint(-120, 38), 4);
```

7.5.3. The Virtues of Additional Map Layers

Different layers make more sense for different uses, and at different zoom levels. The San Francisco Bay area looks sort of patchy in the Google Satellite image in Figure 7-9.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

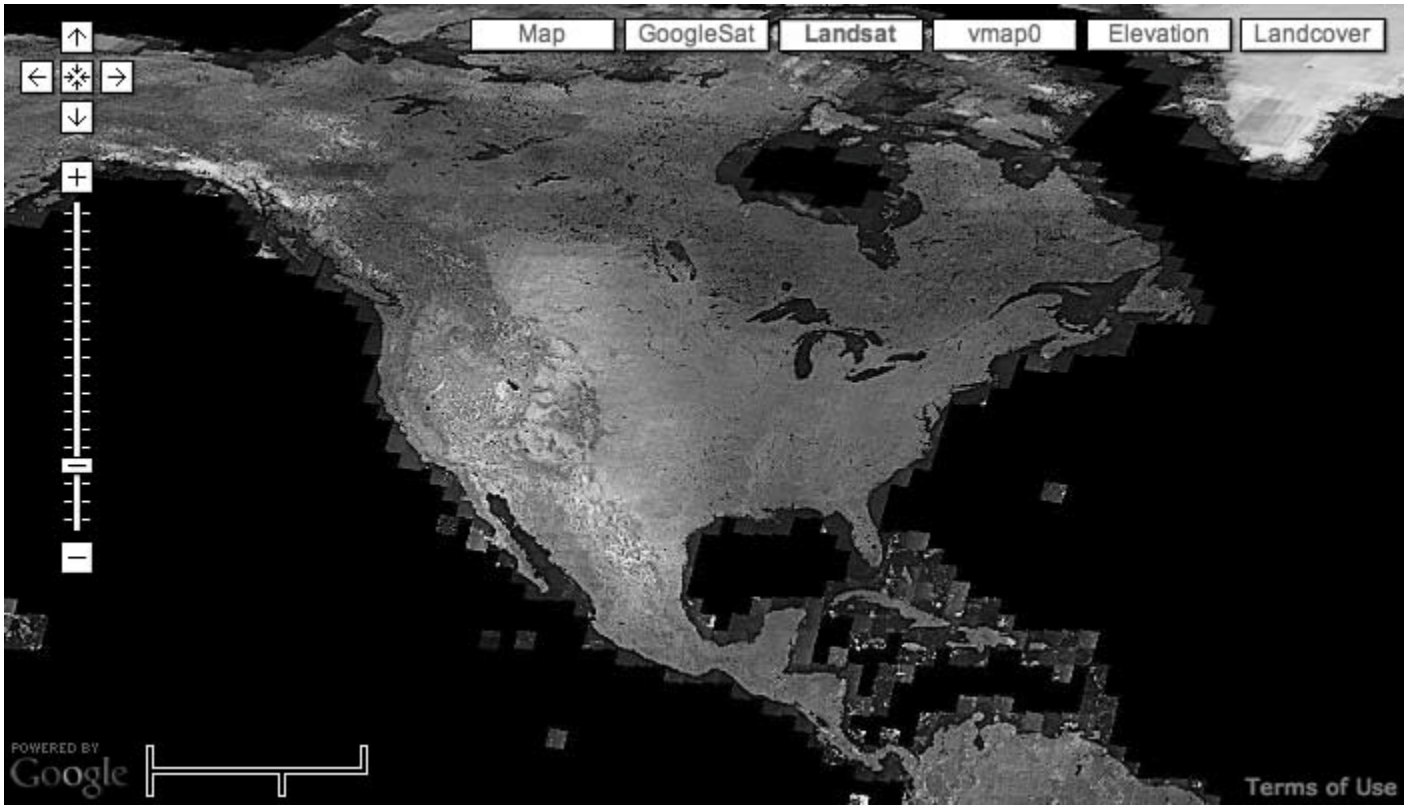
Figure 7-9. Google Satellite image of the Bay Area—pretty choppy



In Figure 7-10, the Landsat imagery is smoother and more informative at this scale.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

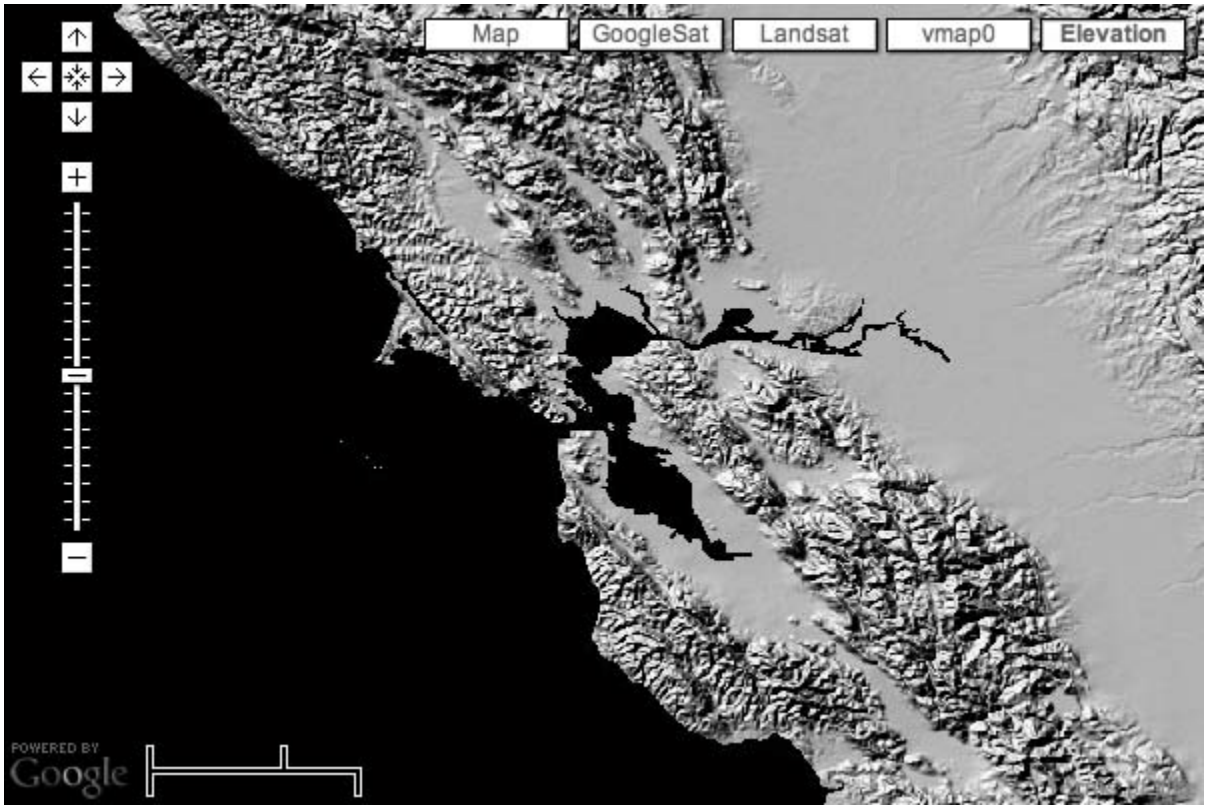
Figure 7-10. The Landsat image looks nice at this scale



The National Elevation Dataset covering the Bay Area, shown in Figure 7-11 is just beautiful.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

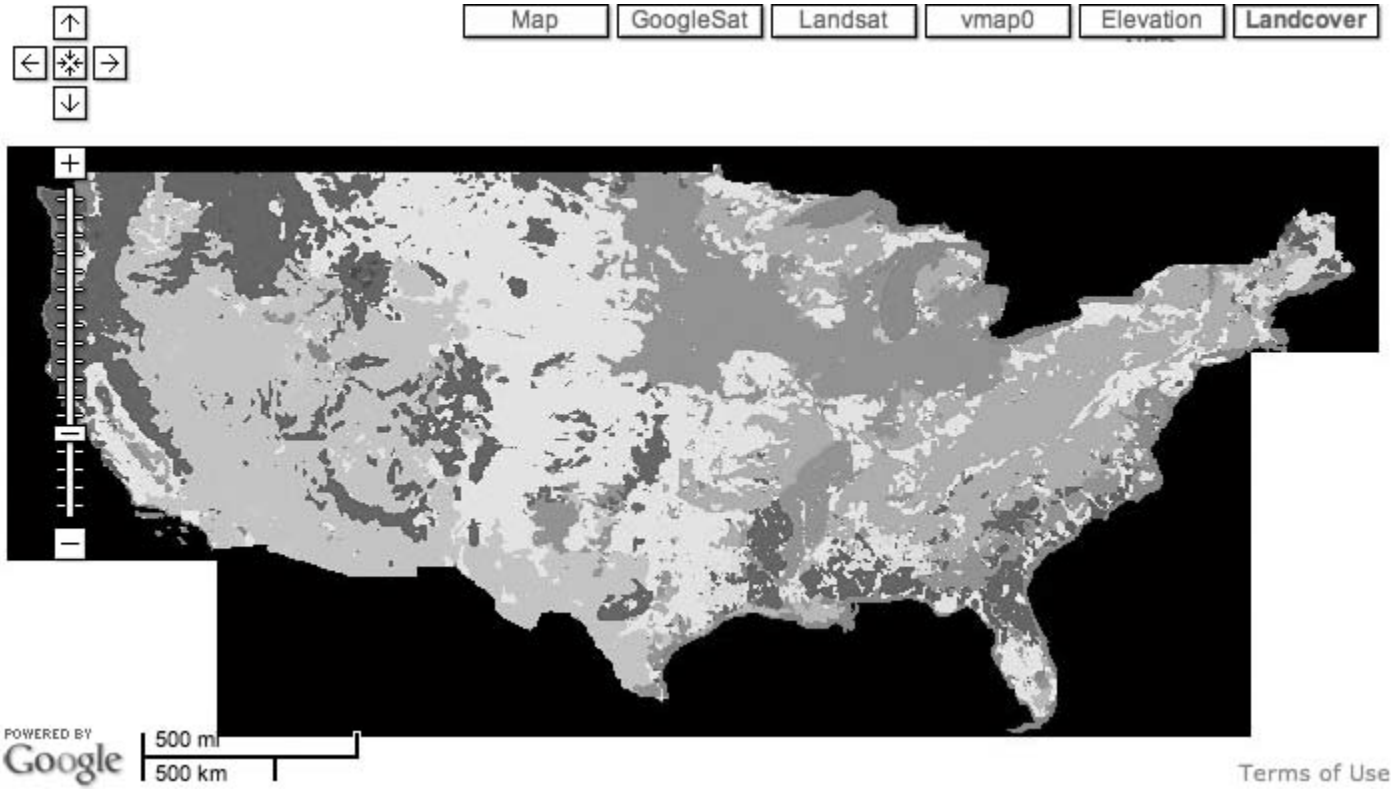
Figure 7-11. The National Elevation Dataset for the Bay Area



At a larger level, the NED and Landcover images are useful in understanding the natural history of the United States, as shown in Figures 7-12 and 7-13, respectively.

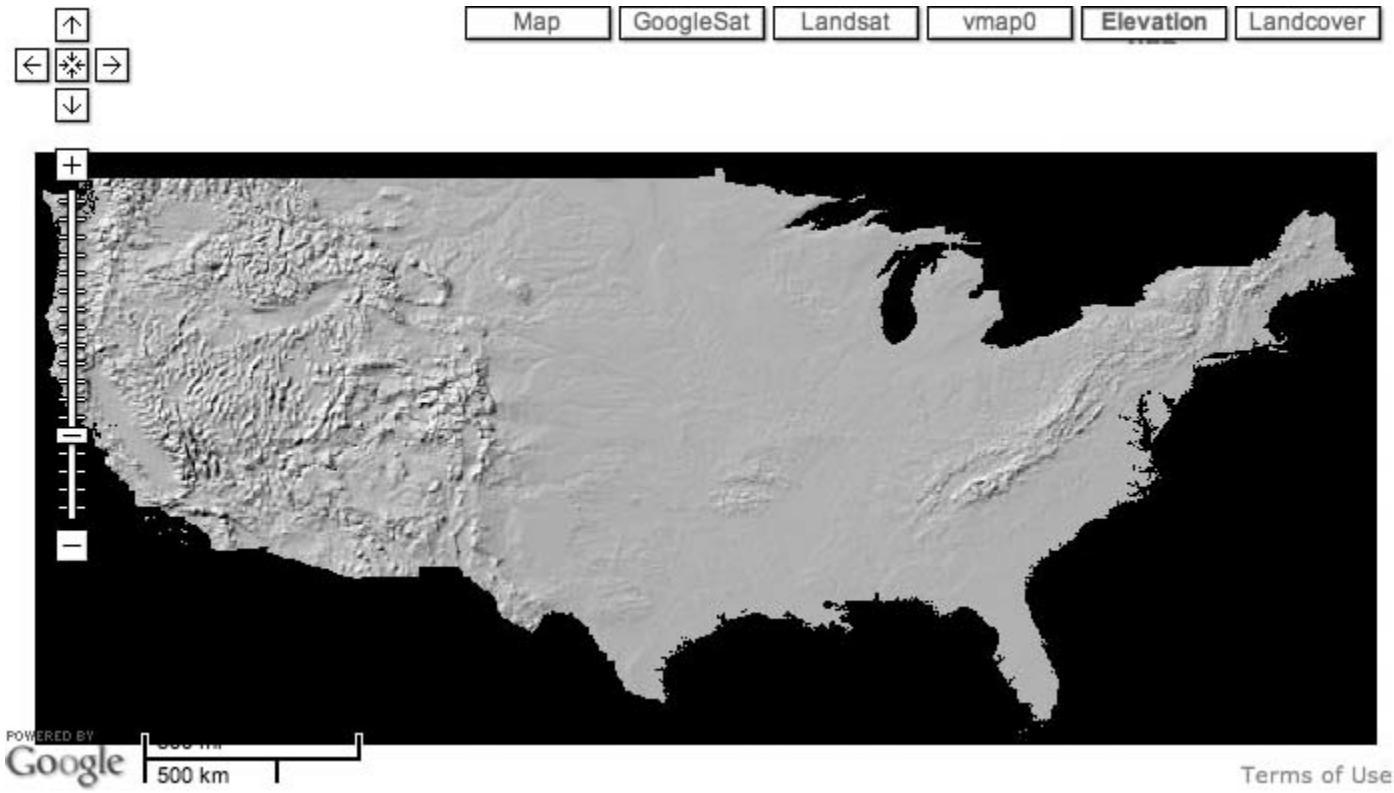
Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 7-12. U.S. Landcover



Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 7-13. U.S. National Elevation Dataset



7.5.4. Finding and Using Other WMS Servers

We can find WMS servers that have a lot of different data! And all those WMS servers are kind enough to tell us what information they have available. Ionic Software provides the AskTheSpider service at http://askthespider.com/ to serve as a central search engine for the Geospatial Web. Each result from AskTheSpider includes the links to the GetCapabilities page for each server. You can also search for "WMS Servers" on Google and be provided with a lot of choices.

To see which layers a WMS server has available, issue a `GetCapabilities` request. This is an HTTP `GET` call to a URL, probably a script, on the WMS server. For example, ask the WMS server at the Jet Propulsion Laboratory (JPL) what it can deliver. Load http://wms.jpl.nasa.gov/wms.cgi?request=GetCapabilities into your browser, and an XML file is returned. Depending on your browser, the capabilities will be displayed, or you may be given the opportunity to save the page to your local drive. Technically, a proper capabilities request needs the `service` and `version` parameters; for example:

```
http://wms.jpl.nasa.gov/wms.cgi?request=GetCapabilities&service
   =WMS&version=1.0.0
```

However, many WMS servers are forgiving about this.

The capabilities document from the JPL WMS server is a 1,696-line XML document that includes lots of detail to specify what is available. The capabilities document will contain one or more `Layer` elements. Here is part of one of the shorter `Layer` sections from the JPL capability document:

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
<Layer queryable="0">
        <Name>us_colordem</Name>
        <Title>Digital Elevation Map of the United States, DTED dataset, 3
second resolution, hue mapped</Title>
        <Abstract>
             The DTED Level 3 US elevation, mapped to a color image using the
full spectrum.
        This result is not achievable by using SLD, so it is presented as a
different layer.
      </Abstract>
      <LatLonBoundingBox minx="-127" miny="23" maxx="-66" maxy="50"/>
      <Style>
           <Name>default</Name>
    <Title>Default Color Elevation</Title>
         </Style>
         <ScaleHint min="45" max="5000" />
         <MinScaleDenominator>20000</MinScaleDenominator>
    </Layer>
```

Even with no more documentation on WMS, you can guess that this is the Digital Elevation Map of the United States, and you know that it has data for latitudes from 23 N to 50 N, and 127 W to 66 W—i.e., the continental United States.

The original long request that generated an image of Florida starts to make more sense when you look at the full XML `GetCapabilities` document. Every element in the URL is defined by the Capabilities document.

```
http://wms.jpl.nasa.gov/wms.cgi?request=GetMap&layers
   =global_mosaic_base&srs=EPSG:4326&bbox=
   -87.90,24.38,76.65,30.01&FORMAT=image/png&width=600&height=400&styles=pseudo
```

The base URL is defined in the `OnlineResource` element. The available layers are in the `Layer` elements. The *spatial reference system*, or SRS, corresponds to the exact cartographic projection of the layer and is often defined by European Petroleum Survey Group, or EPSG, codes (there is a long story behind this). These codes are sometimes referred to as *spatial reference identifiers*, or SRIDs. An SRID defines the specific assumptions that are being made about the shape of the earth, how you mark location, and where zero is. In this case, EPSG 4326 refers to the latitude/longitude geographic coordinate system, referenced to the WGS-84 standard ellipsoid and datum.

If you have the PROJ.4 cartographic projections library (http://proj.maptools.org), you can examine the list of EPSG codes stored in */usr/share/proj/epsg*. If you have PostGIS (http://postgis.refractions.net/) installed you can see what SRID 4326 means by looking in the `spatial_ref_sys` system table. If you don't have PostGIS—or don't want to bother—you can mostly assume that everything uses EPSG code 4326. This isn't always true, but it is true often enough to be a useful simplifying assumption.

```
=
> select * from spatial_ref_sys where srid=4326;

GEOGCS["GCS_WGS_1984",DATUM["D_WGS_1984",SPHEROID["WGS_1984",6378137,298.
257223563]],PRIMEM["Greenwich",0],UNIT["Degree",0.01745329251994295]] |
+proj=longlat +ellps=WGS84 +datum=WGS84
```

The bounding box used in the `bbox` parameter defines the minimum and maximum latitude and longitude that will be included in the returned map. The `format` parameter is the image format to use (PNG, JPG, PNG, PDF), the height and width are the number of pixels in the returned image, and the `styles` parameter is defined in the Layers section and means whatever that particular server says that it means.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

With this knowledge, we can make almost any WMS server a source for imagery that will appear in a Google Maps interface on our own maps. We need to figure out the base URL to generate maps from a particular WMS server. Here is the `GetCapabilities` line for the JPL WMS server compared with the `_WMSMapSpec` that is needed by David's JavaScript:

```
http://wms.jpl.nasa.gov/wms.cgi?request=GetCapabilities
http://wms.jpl.nasa.gov/wms.cgi?request=GetMap&layers=global_mosaic_
base&srs=EPSG:4326
```

To get a map you replace `GetCapabilities` with `GetMap`. Then you must specify the name of one or more layers. These are from the `Name` element within the `Layers` section of the document returned by the `GetCapabilities` request. You also need to specify the Spatial Reference System, which is in the `SRS` element within the `Layer` element.

7.5.5. See Also

- Open Geospatial Consortium's Directory of Services at http://www.ogcnetwork.org/. Includes links to the WMS Cookbook at http://www.opengeospatial.org/resources/?page=cookbooks, as well as other documents, links to sample applications, and support mailing lists.
- Another WMS Cookbook maintained by Allan Doyle, the Kevin Bacon of the Open Geospatial world: http://www.intl-interfaces.com/cookbook/WMS/.

Hack 66. Add Your Own Custom Map

This time, it's personal.

Well, as everyone else has written in this book, those Google Maps sure are pretty. Then again, there is always room for improvement right? And sometimes Google's maps just don't highlight the right information. To really spice things up, you may want to add your own map…a map that looks the way you want it to look. This off-API hack shows you how to add your own custom map. In "Serve Custom Map Imagery" **[Hack #67]**, you'll get a script to serve custom map tiles, and finally in "Automatically Cut and Name Custom Map Tiles" **[Hack #68]**, there is a script to quickly make the thousands of little images you will need to display your map.

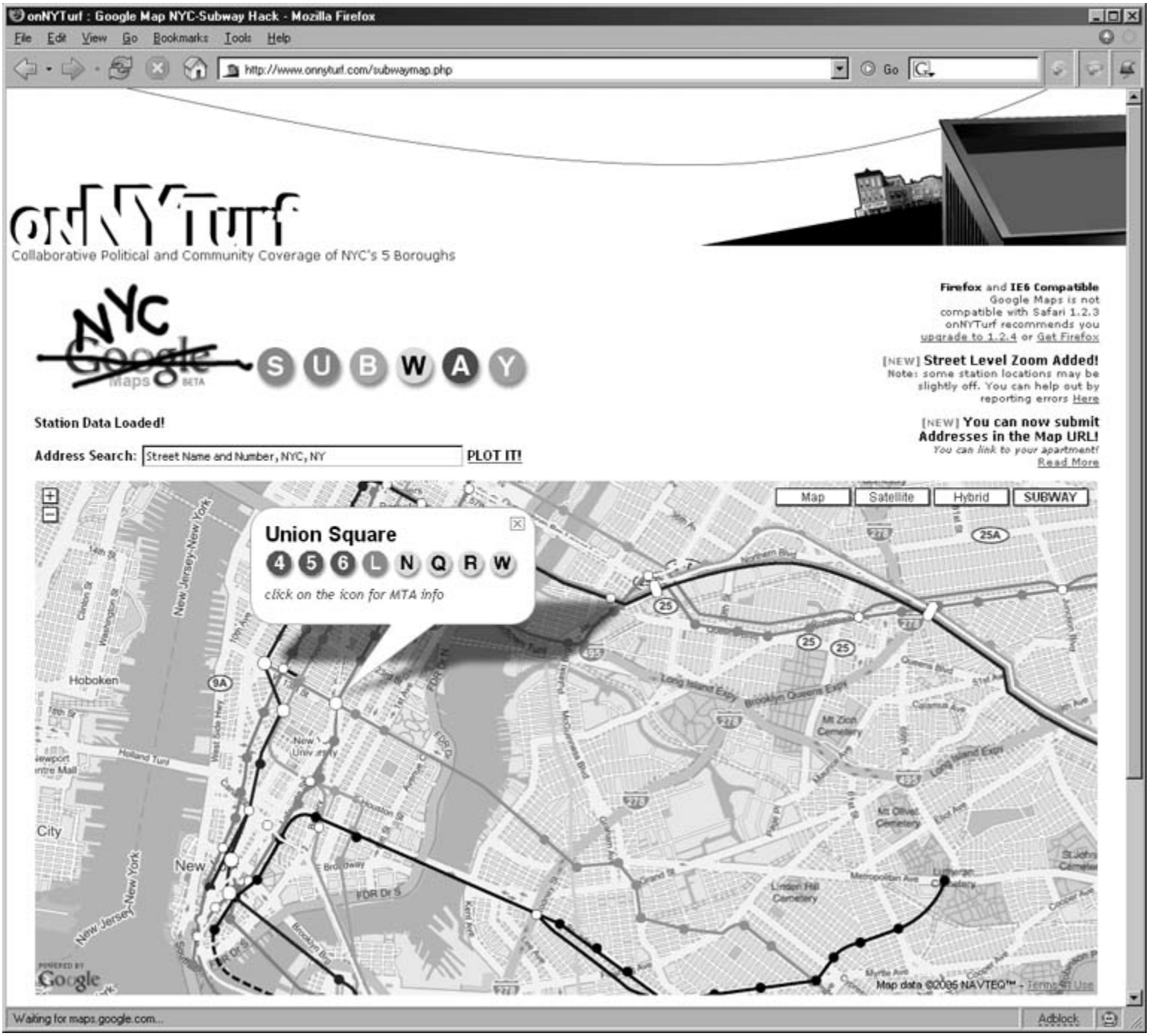
There are plenty of cool reasons to add your own map images. In creating the NYC Subway Google Map hack (http://www.onnyturf.com/subwaymap.php), I felt that Google's polylines were not elegant enough, and they were hard to see on top of Google's orange/green default map. To make the subway lines easier to see against the streetscape and to make the lines look nicer, I decided to add a custom map type that I illustrated, as you can see from Figure 7-14.

Another great use of adding your own map is using it as a semi-transparent overlay on the other map images. Figure 7-15 shows the street map layered on top of the satellite map, with a semi-transparent setting allowing you to see the relationship between the two. You can do this sort of thing with your own custom map too.

Creating this transparency effect is outside the scope of this tutorial. You can learn how to add transparency by looking at the source of the example above, which can be found at http://www.kokogiak.com/gmaps-transparencies.html. You will need to know these basics on adding a custom map in order to add transparency in that way.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 7-14. Custom map for NYC subway map hack



Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

7.6.1. Four Steps to Add a Custom Map

There are four simple things you will need to do to add your own map.

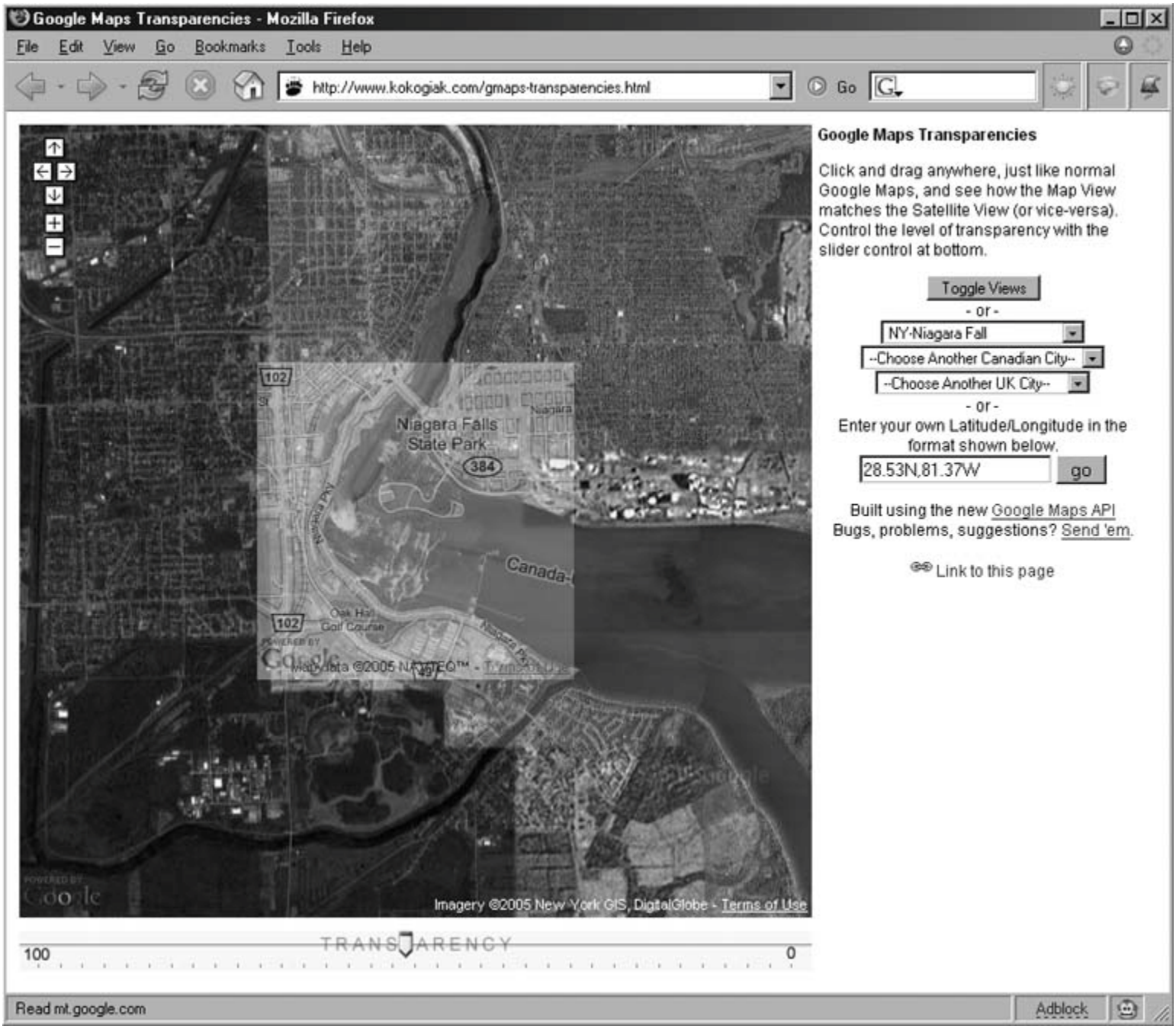
- Create a new map type.
- Tell the map where it can find the images for your new type.
- Customize your map's type button.
- Add your new map control to Google Maps' list of map types.

7.6.2. Create a New Map Type

The Google Map comes with three default maps, which you are probably already familiar with, namely, map, satellite, and hybrid. These are known in Google Maps as *map types*, and they are accessed from an array called `mapTypes`, which is a property of our `GMap` object. To add your own map, we need to create a new map type and then add it to this array.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 7-15. Transparent road map overlaying the satellite map



Each map type Google provides has many properties. We need to make our custom map type just like these. But we really don't want to have to build a whole map type from scratch, setting all the properties anew. We can save a lot of time by duplicating an existing map type and switching just a few of the values to suit our needs.

For the NYC Subway Google Map hack, I made my map type match Google's default map type. This default map is the first map stored in the `mapTypes` array. So we are just going to copy that first map type and store it in a variable temporarily for editing (here it is named

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

yourMapType, but you can name this variable what ever you like). There is one special thing we need to be aware of here; since JavaScript assigns object values by reference, we can't just dupe the map type object like this:

```
yourMapType = map.mapTypes[0];
```

We need to make sure we copy all the properties of the map type object, so we have a special little function we are going to include and call to copy all the properties of the default map type:

```
function copy_obj(o) {
    var c = new Object();
    for (var e in o)
        c[e] = o[e];
    return c;
}
```

Using our copy_obj function, we can now duplicate the default map type:

```
yourMapType = copy_obj(map.mapTypes[0]);
```

Now we change a few of the properties to customize our map type.

7.6.3. Set the Path to Your Map Images

Each Google map type is made up of thousands of image tiles. When you see a Google map in a window, you are really seeing about 12 tiles at a time, depending on your screen size, as suggested by Figure 7-16.

As you drag the map around in the window, your browser downloads the tiles it needs in order to show more of the map. This happens in the background and is one of the more innovative features of Google Maps. Each map type has a path to its associated image tiles. This property is called baseURL. The default maps point to Google's server. To load our own image tiles, we need this path to point at our own server. We actually don't point right at a directory of images. The map sends values to the server and asks for an image in return. So we use a PHP script to serve up the tiles based on the values the map sends (this image serving script is discussed in more detail later in "Serve Custom Map Imagery" **[Hack #67]**). We set the baseURL to point at this script.

```
yourMapType.baseUrl = "http://www.yoursite.com/googlemap/images/index.php?";
```

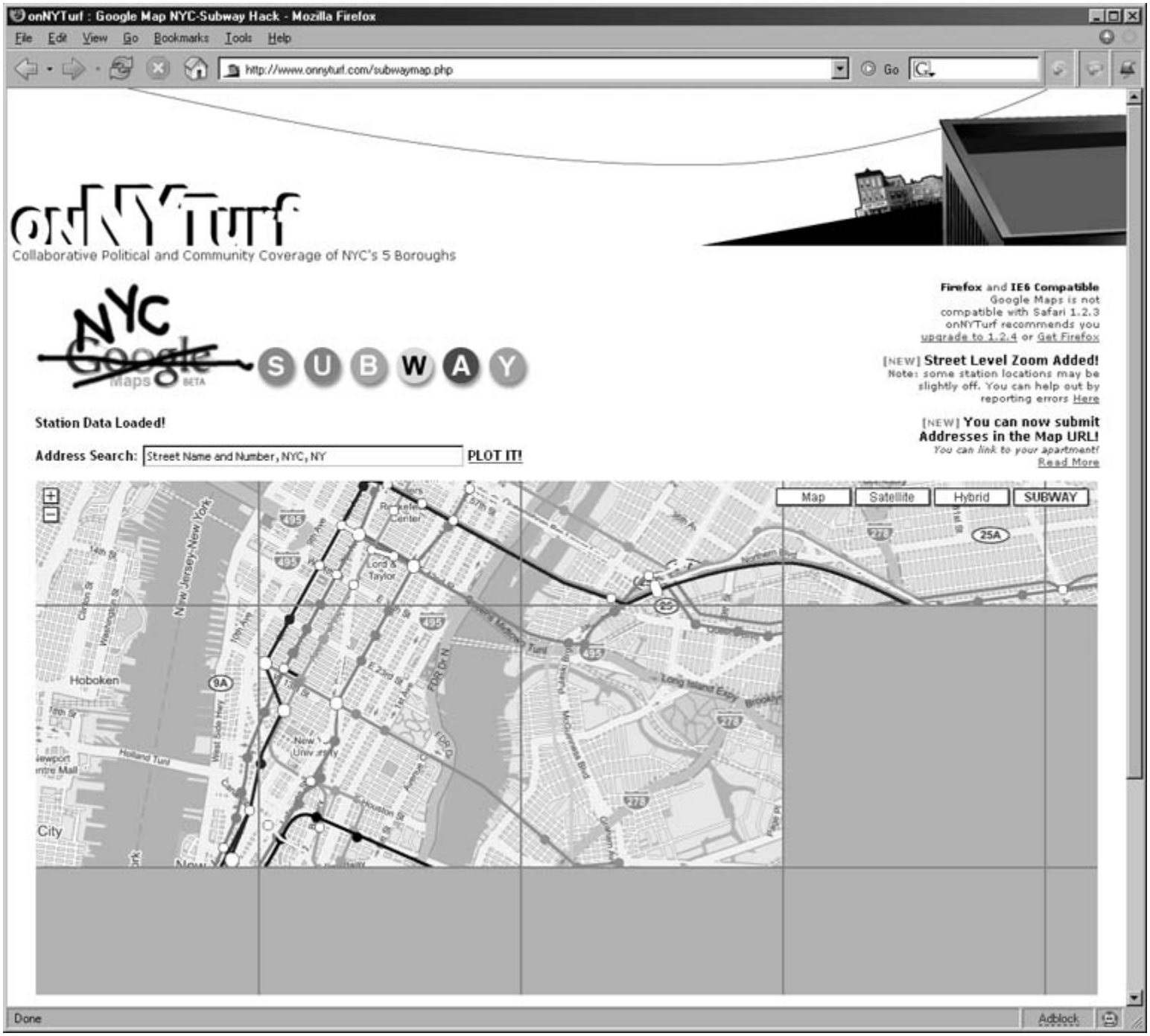
You'll notice we include a ? at the end of our path. The Google Maps application adds CGI parameters at the end of the path when requesting images, so the ? is necessary for posting those values. With that path configured, our images will load when our custom map type is selected.

7.6.4. Customize Your Map's Button Name

Although the Google Map API does not automatically add the map types buttons that appear in the upper-right corner of their map, most people add them using the API addControl method. If you call this method in your map, a button will automatically be generated for your map type too, as you can see on the Subway map shown in Figure 7-17.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 7-16. Google Maps tile grid



However, that button did not just magically get the name "Subway." We set the name of our map type's button by setting the `getLinkText` property. The Google Map application calls it as a function, so we want to return the name we want to give it, like so:

```
yourMapType.getLinkText = function() { return 'SUBWAY'; }
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

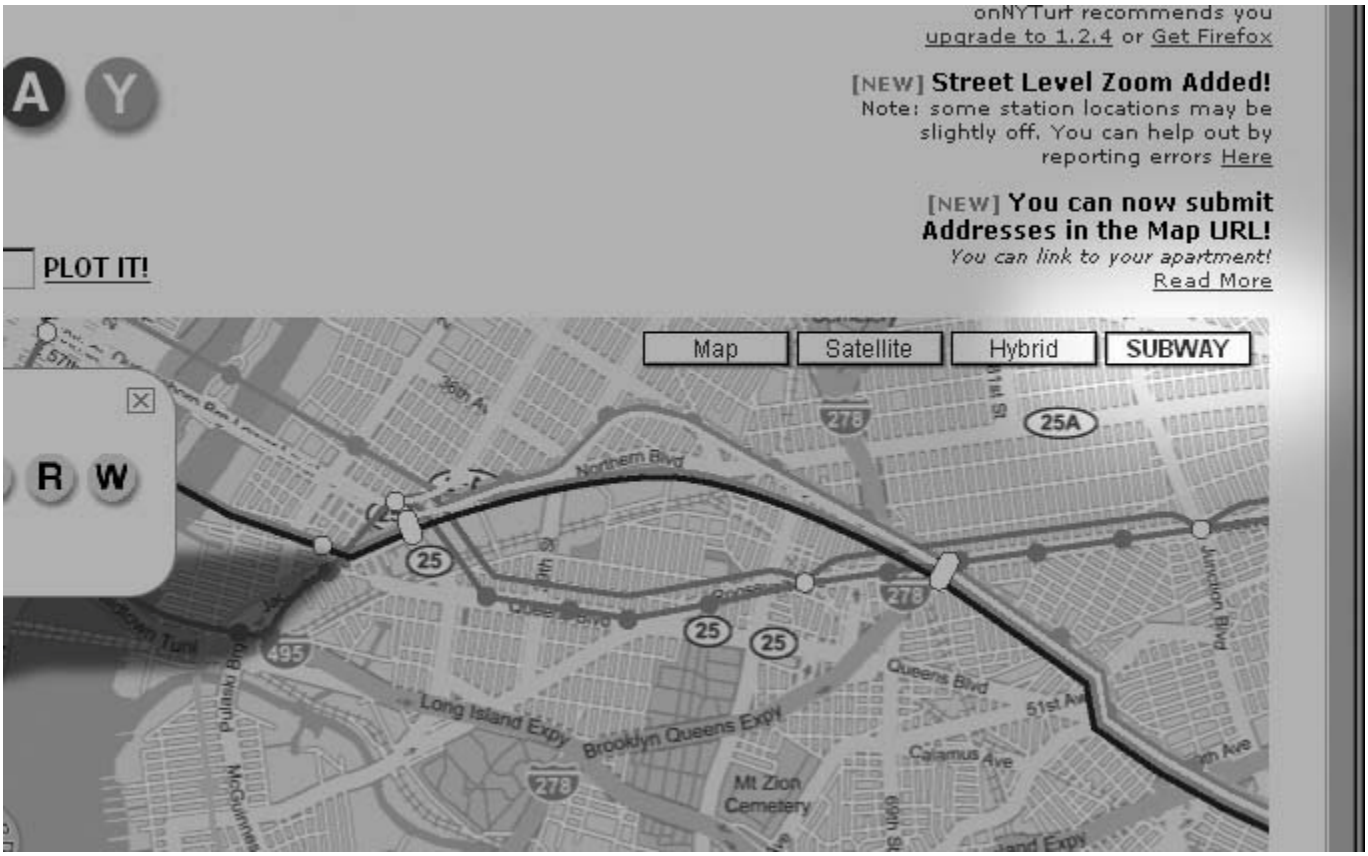
You'll note that `getLinkText` is set to a function. This means you can have that function return different values depending on external conditions.

Nice and simple eh? So that's all we have to do to make a custom map type. Now we add it to the array `mapTypes`.

7.6.5. Add Your Map to the List of Map Types

We will add our map to the end of the `mapTypes` array, and then, because we love our map more than any other map in the world, we set it to be the default map when our map page first loads. To add our custom map to the list, we just add the map type object that we have been working on to the end of the array. We use the API's `setMapType( )` method to make our custom map type load first.

Figure 7-17. Subway map type button




```
map.mapTypes[map.mapTypes.length] = yourMapType;
map.setMapType(map.mapTypes[map.mapTypes.length-1]);
```

Finally let's put it all in the context of the rest of our typical map initializing function. This function is called from `onLoad` attribute of the `body` element in our HTML:

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
function onLoad() {

     //Create a new map and load it to the map at the html object id="map"
      map = new GMap(document.getElementById('map'));

      // Copy Object Function
      function copy_obj(o) {var c = new Object(); for (var e in o) { c[e] =
o[e]; } return c;}

      //Copy mapType
      yourMapType = copy_obj(map.mapTypes[0]);

      // Set Path to Our Custom Map Image Tiles
      yourMapType.baseUrl = "http://www.yoursite.com/googlemap/images/index.
php?";
      // Map Display name in the auto-generated maplink in the top right
corner.
      yourMapType.getLinkText = function() { return 'SUBWAY'; }

      // Register the new mapType with the running google map.

      map.mapTypes[map.mapTypes.length] = yourMapType;

      //Set the onload view to our new map
      map.setMapType(map.mapTypes[map.mapTypes.length-1]);

      //Add Map Type buttons in the upper right corner
      map.addControl(new GMapTypeControl());

      //Add Small zoom controls
      map.addControl(new GSmallZoomControl());
}
```

Even though we added our map to the list of map types, we still need to serve up the images when requested **[Hack #67]**.

7.6.6. See Also

- "Add More Imagery with a WMS Interface" **[Hack #65]**
- "Serve Custom Map Imagery" **[Hack #67]**
- "Automatically Cut and Name Custom Map Tiles" **[Hack #68]**

—Will James

Hack 67. Serve Custom Map Imagery

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

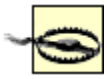
You can serve your own custom tiles to Google Maps with a simple script.

If you've created a custom map layer **[Hack #66]**, you need a way to serve requests for tiles. This hack describes a PHP script to serve Google Maps compatible imagery.

Every time you drag the map around in the window, the Google Map application requests whatever tiles it needs in order to show more of the map. It does this by passing three values to the server for each tile it needs: an X coordinate value, a Y coordinate value, and a Zoom value. For each tile it needs, the map application sends a set of these three values.

Our image serving script uses the three values sent by the Google Map application to dynamically create paths to the appropriate image required to fill in a space in the map window. Our script is kept in a file named *index.php*. It assumes that tiles have been created and named by the naming standards described in "Automatically Cut and Name Custom Map Tiles" **[Hack #68]**.

With a little fancy footwork we can also use our script to send generic filler tiles to the map when we don't have custom tiles for a requested area. In the most common case, you will probably want to create a custom map that covers a small area, and not the whole globe. Therefore you will probably need to use some filler tiles where your map does not cover an area or zoom level. We can use our image serving script to determine when we don't have an appropriate custom tile and need to send a filler tile.



If you decide that you *do* need to make tiles for the whole planet, be sure to read "How Big Is the World?" **[Hack #16]** first!

To understand this script, we'll start with the whole thing and then break it down. Here is the full script that goes in our file *index.php*:

```
<?php
        define("NO_DATA", "./no_data.gif");
        define("ZOOM_IN", "./zoom_in.gif");
        define("ZOOM_OUT", "./zoom_out.gif");

        $x = $_GET["x"];
        $y = $_GET["y"];
        $z = $_GET["zoom"];

        $filename = "./nycmaps/${x}_${y}_${z}.gif";
        if ( $z < 2 ) {
            $content = file_get_contents( ZOOM_OUT );
        }else if ( $z > 5 ){
            $content = file_get_contents( ZOOM_IN );
        }else if ( is_numeric($x) && is_numeric($y) && is_numeric($z)
          && file_exists( $filename ) ){
                    $content = file_get_contents( $filename );
        }else{
            $content = file_get_contents( NO_DATA );
        }
        header("Content-type: image/gif");
        echo $content;
?>
```

OK, there it is. Now let's break it down.

The first three lines define our filler tiles. We have one for each of three conditions. If we are in an area for which we have data, but not at the current zoom level, we use the *zoom_in.gif* and *zoom_out.gif* images. If we have no data, we send the *no_data.gif* image.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
<?php

        define("ZOOM_IN", "./maptiles/zoom_in.gif");
        define("ZOOM_OUT", "./maptiles/zoom_out.gif");
        define("NO_DATA", "./maptiles/no_data.gif");
```

You'll notice that above we also set the path to these tiles to a directory called *maptiles/*. This is where we will put all our image tiles. This directory sits inside the *images/* folder, where this script also resides, as established in "Add Your Own Custom Map" **[Hack #66]**.

Next we get the *x, y*, and zoom values from the map and store them in similarly named variables.

```
$x = $_GET["x"];
$y = $_GET["y"];
$z = $_GET["zoom"];
```

Using these three values, we create a path to a tile file, here a GIF, and we store it in the variable `$filename`. You'll note that the *x, y*, and zoom (*z*) values are part of the tile name. This is the basic structure we use to name all our tiles. This is how we differentiate the hundreds or thousands of custom tiles we likely end up with when creating a custom map. Now that's a lot of custom names, but don't worry about it too much, because later we'll automate the cutting and naming of your tiles.

```
$filename = "./maptiles/${x}_${y}_${z}.gif";
```

Now we have a path appropriate to the data sent, but that does not mean the tile actually exists. `$filename`, whatever its value, is just a fourth image option, in addition to `NO_DATA`, `ZOOM_IN`, and `ZOOM_OUT`. Before we send any image back to the browser, we want to check if the request is within our zoom range and map area. This next chunk of the script uses several `if/else` statements to filter though the possibilities and determine the appropriate image. The winner will be stored in variable `$content`. The specific checks we make are explained by the in-line comments:

```
// if the zoom level requested is too close, we send images that
// say zoom out
if ( $z < 2 ) {
    $content = file_get_contents( ZOOM_OUT );

// if the zoom level requested is too far out, we send images that
// say zoom in
} else if ( $z > 5 ){
    $content = file_get_contents( ZOOM_IN );

// here we make sure values were sent for x, y, and zoom, and that
// the image tile actually exists. if it all checks out we send
// the file we defined in $filename
} else if ( is_numeric($x) && is_numeric($y) && is_numeric($z)
            && file_exists( $filename ) ){
    $content = file_get_contents( $filename );

// otherwise, if one of the values was not sent, or the tile does
// not exist we send a NO_DATA image
} else {
    $content = file_get_contents( NO_DATA );
}
```

Now that we know which image we are sending back, we are ready to do so. The reason we didn't just send it previously is that we first need to tell the browser that we are sending an image. We tell it this by sending it some header information:

```
header("Content-type: image/gif");
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Now we can pass it the image path with a simple echo of `$content`

```
    echo $content;
?>
```

7.7.1. Conclusion

That's all there is to it! Save it as *index.php* and upload it to your image folder.

There are two things I really like about this script that make it a lifesaver. The first is the `NO_DATA` condition. For the subway map, I made only tiles that covered the greater NYC area. Outside that, there are no custom tiles. Without the `NO_DATA` image, in these places where I don't have tiles to match Google's, we would normally see broken image links or the grey background of the map window. But thanks to the `NO_DATA` image, I can create one blue tile that matches my border and extends infinitely anywhere I don't have custom tiles for a location. Figures 7-18, 7-19, and 7-20 show what my zoom-in, zoom-out, and no-data files look like.

Figure 7-18. Zoom-in image

Figure 7-19. Zoom-out image

Figure 7-20. No-data image

Alternatively for those more advanced, you could extend this script to relay the *x, y*, and zoom values back to Google; then you could request their tiles for places where you do not have any.

The other thing I like about this script is that you can quickly adjust which zoom levels are available. You just set min and max thresholds in those `if/else` statements and you get alternative tiles sent to the map for anything outside your zoom range. In creating the NYC Subway Map, I tweaked the map art at each level. This is a bit time consuming, but gives me better-looking maps. So I incrementally add new zoom

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

levels as I complete the art. Every time I add a new one all I have to do is go into this script and change the min or max value to make it available.

7.7.2. See Also

- "Add Your Own Custom Map" **[Hack #66]**
- "Automatically Cut and Name Custom Map Tiles" **[Hack #68]**

—Will James

Hack 68. Automatically Cut and Name Custom Map Tiles

Make time to really enjoy a cup of coffee.

Google Maps are made of dozens to thousands of tile images, depending on the zoom level. At the distant zoom levels you only need a few images to cover a large area. For example, the NYC Subway Map uses 49 images to cover the greater metropolitan area at zoom level 5. But at the closer zoom levels, thousands of images are required. It takes more than 3500 images to cover the NYC metropolitan area at zoom level 2. "How Big Is the World?" **[Hack #16]** discusses the amount of disk space needed to tile the entire planet. Figure 7-21 shows a custom tile at zoom level 5, while Figure 7-22 shows part of the same area at zoom level 2.

Creating and then uniquely naming each of these images for even a small part of our world would be a daunting task if you had to do it by hand. That's why we use a script to do this. The following hack will show you how to configure a batch processing script to use with Photoshop 7 or CS that will carve all the tiles you need from one big image into hundreds of smaller GIFs and name them exactly as you need them named.

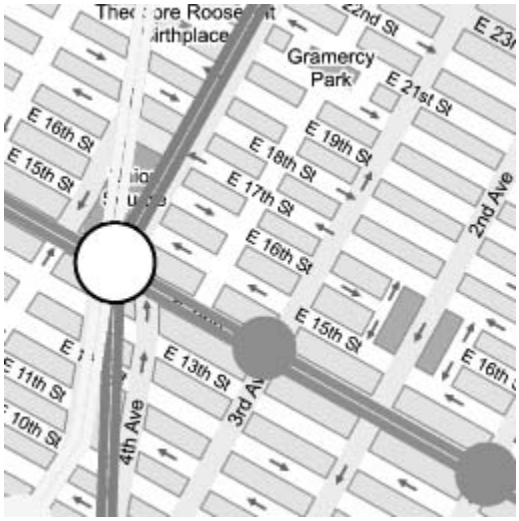
7.8.1. The Google Map Tile Structure

Before we set up and use our script we need to know a few things about Google's tiles. As we learned in "Serve Custom Map Imagery" **[Hack #67]**, each 256 X 256–pixel tile has a longitude, a latitude, and a zoom value. These are represented in the URL by the `x`, `y`, and `zoom` values.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 7-21. A custom tile at zoom level 5

Figure 7-22. A custom tile at zoom level 2



Here is what a tile path from Google looks like:

```
http://mt.google.com/mt?v=w2.4&x=1206&y=1539&zoom=5
```

7.8.2. Figuring Out the Values for Your Tiles

So how do you get the `x`, `y`, and zoom values? Fortunately you don't have to get *all* the `x`, `y`, and `z` values for all your images. We only need to get a set of values for one tile and from there we can derive all the other values we need.

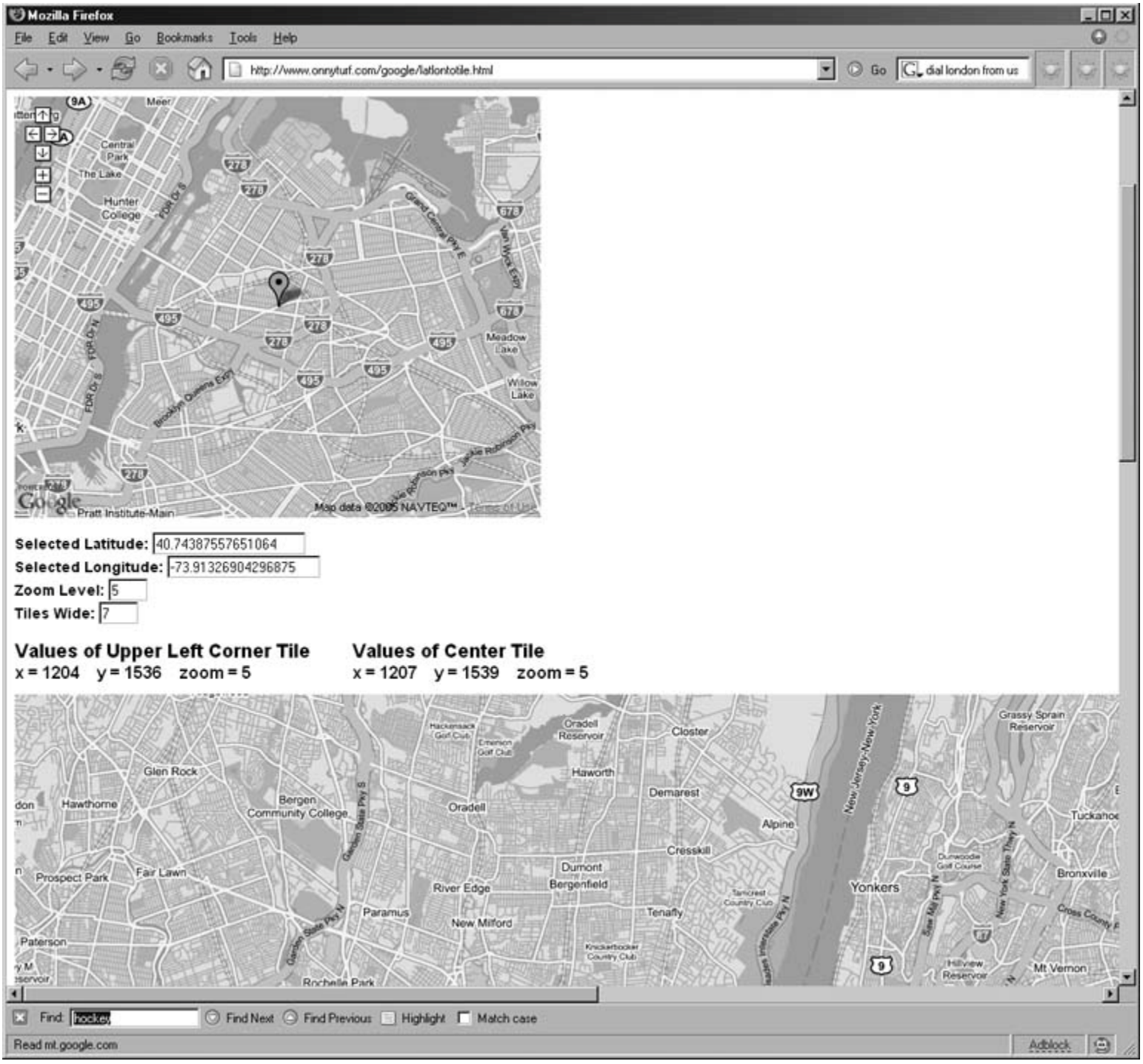
Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

We start with a limited area for our map because we assume you don't want to remap the whole world, but just a small corner of it. All we need then is to get the `x`, `y`, and `z` value of the upper-left corner tile of our map area and know how many tiles wide and high our map area is. For simplicity, our example map is square so that the area width and height are the same, but you could tweak this script to work with rectangular areas. Once we know the upper-left corner tile values, our batch processing script will calculate the rest for us.

Unfortunately getting just these corner tile values is not easy on our own. Fortunately again there is a web site that will help do this for us. The site at http://www.onnyturf.com/google/latlontotile.html has a set of tools that will help us figure out this information for the upper-left corner tile values. Figure 7-23 illustrates the process of converting lat/long to tile row and column.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 7-23. Converting latitude and longitude to tile row and column



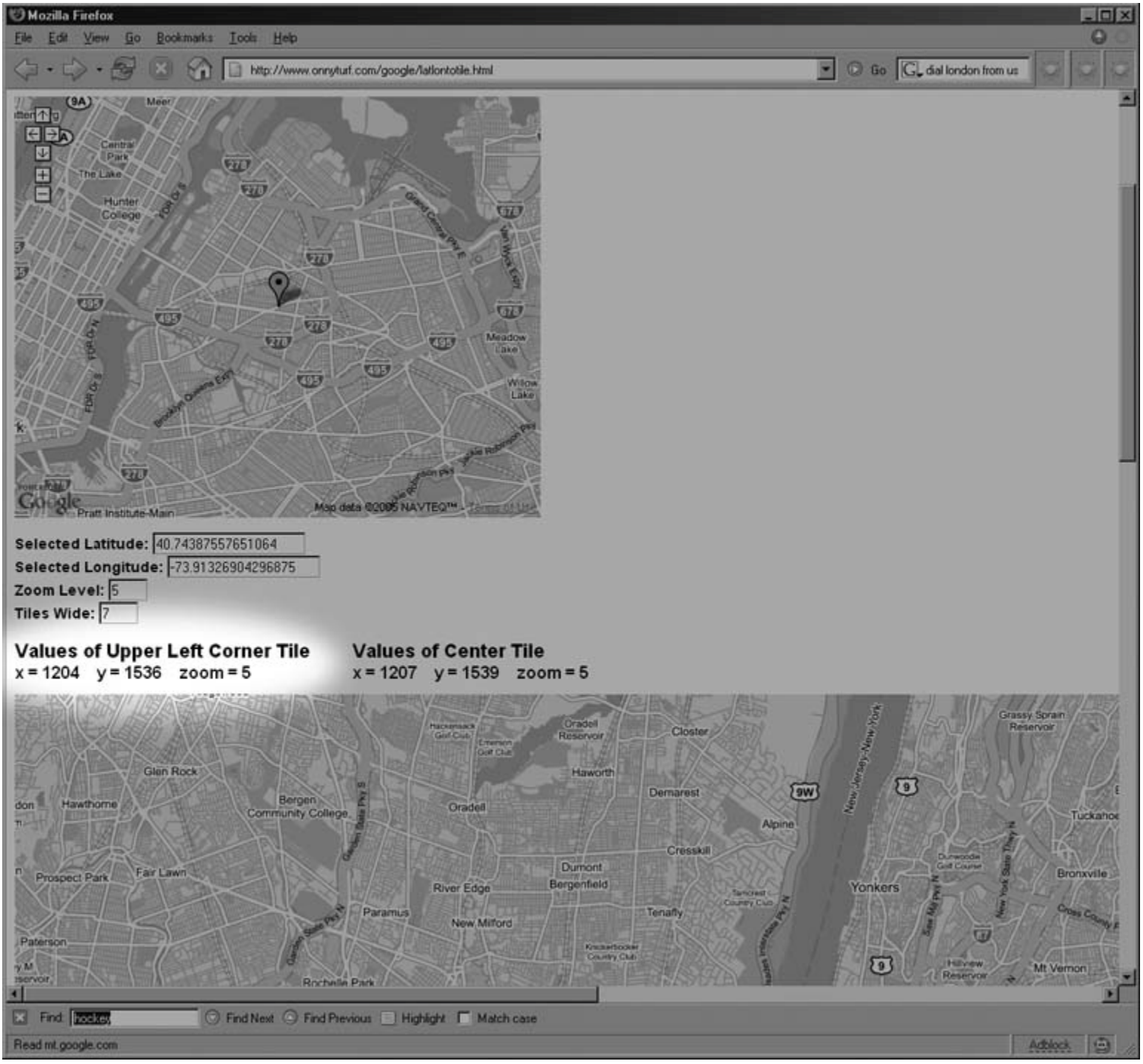
Using the map on the right side of this page you can find the center of your map area. When you click there the web page will get the latitude and longitude for the center of your map. You can also set the zoom level and tile width here. When you press submit, the web site will return Google's tiles that make up your requested map area. For our batch processing script, you will want to do this for your furthest zoom level.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

You'll probably have to experiment a little until you get the right values that match your map area. For my NYC Subway map, the furthest zoom level is 5 and my tile width is 7, as shown in Figure 7-24. Your maximum zoom level will probably be somewhere between 5 and 8.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 7-24. Finding the upper-left corner for a given latitude/longitude



To get the values for the upper-left corner tile, we now just right-click on that tile and select View Image. This will load the image into the browser window, and now we can see our *x*, *y*, and `zoom` values in the image URL:

```
http://mt.google.com/mt?v=w2.4&x=1206&y=1539&zoom=5
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Whew. So there it is. Now we know our start values for this example are:

```
x = 1206
y = 1539
zoom = 5
tile width = 7
```

7.8.3. Configuring the Script

The batch processing script provided is too long to go through in detail here. This hack focuses on showing you how to configure it for your map area so you can make your tiles.

To use the script provided you must enter some custom values, including the ones we just got for our upper-lefthand tile. The values you must customize come at the beginning of the script.

7.8.3.1. ZoomLevel.

Last things first. The first variable you encounter sets the Zoom Level you want to generate tiles for. To create your custom map you will probably want tiles at various levels. For each level you want to create tiles for, you must change this value and rerun the batch processing script on your master art. Here it is set to 3, but you will change this every time you want to run this script for another level.

```
var ZoomLevel = 3;
```

The rest of the variables you must customize you will only customize *once* for your map area. Let's go through them one by one.

7.8.3.2. FolderPath.

This next one sets the path to where your tiles will be saved. This path is for use on a PC. The path on the Mac will be slightly different. This sample path uses a separate folder for each zoom level, and those folders are in a folder *GoogleMap* that sits on the user desktop. You will have to precreate your zoom folders, but the path to those folders is dynamically generated in this example.

```
var FolderPath = "~/Desktop/GoogleMap/Zoom"+ZoomLevel+"Maps/";
```

7.8.3.3. Furthest zoom upper-left corner values.

Lastly, we put our upper-left corner values to work. These are the originating values, off of which the batch processing script will figure out the values for all the tiles you are creating.

```
var OrgX = 1204; //<-- the X value
var OrgY = 1536; //<-- the Y value
var OrgZoomLevel = 5; //<-- the zoom level
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
var OrgTileWidth = 7; //<-- the number of tiles wide your full map area is
var OrgTileHeight = 7; //<-- the number of tiles high your full map area is
```

That's all you have to configure! Here is how the whole thing looks with our customized values at the top:

```
/* Tile Carver for Photoshop 7 and CS
   Created by Will James
   http://onNYTurf.com
*/

//**** CUSTOMIZE THE FOLLOWING VARIABLES FOR YOUR MAP AREA ****
var ZoomLevel = 3;

var FolderPath = "~/Desktop/GoogleMap/Zoom"+ZoomLevel+"Maps/";
var OrgX = 1204;
var OrgY = 1536;
var OrgZoomLevel = 5;
var OrgTileWidth = 6;
var OrgTileHeight = 6;

//**** EVERYTHING BEYOND THIS POINT SHOULD NOT BE TOUCHED
//**** UNLESS YOU KNOW WHAT YOU ARE DOING!!! ****

var StartX = OrgX * (OrgZoomLevel - MakeZoomLevel) * 2;
var StartY = OrgY * (OrgZoomLevel - MakeZoomLevel) * 2;

var xTiles = OrgTileWidth * (OrgZoomLevel - MakeZoomLevel) * 2;
var yTiles = OrgTileHeight * (OrgZoomLevel - MakeZoomLevel) * 2;

var PixelWidth = 256;
var PixelHeight = 256;

var TotalTiles = (xTiles)*(yTiles);

preferences.rulerUnits = Units.PIXELS;

var xm = 0;
var ym = 0;

var TileX = StartX;
var TileY = StartY;

for (n=1; n<TotalTiles+1; n++) {

    if (ym == yTiles){
        xm += 1;
        ym = 0;
        TileX += 1;
        TileY = StartY;
       };

    MyXO = xm*(PixelWidth);
    MyXL = xm*(PixelWidth)+(PixelWidth);
    MyYO = ym*(PixelHeight);
    MyYL = ym*(PixelHeight)+(PixelHeight);

    var srcDoc = activeDocument;

    srcDoc.selection.select(Array (Array(MyXO, MyYO), Array(MyXL, MyYO),
      Array(MyXL, MyYL), Array(MyXO, MyYL)), SelectionType.REPLACE, 0, false);
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
        srcDoc.selection.copy();

        var pasteDoc = documents.add(PixelWidth, PixelHeight, srcDoc.resolution,
    "Paste Target");
        pasteDoc.paste();
        pasteDoc = null;

        var saveMe = activeDocument;

        saveMe.flatten();

          saveFile = new File(FolderPath + TileX+ "_" + TileY+ "_" + ZoomLevel
             + ".gif");

        gifSaveOptions = new GIFSaveOptions();
        gifSaveOptions.colors = 64;
        gifSaveOptions.dither = Dither.NONE;
        gifSaveOptions.matte = MatteType.NONE;
        gifSaveOptions.preserveExactColors = 0;
        gifSaveOptions.transparency = 0;
        gifSaveOptions.interlaced = 0;

        saveMe.saveAs(saveFile, gifSaveOptions, true,Extension.LOWERCASE);
        saveMe.close(SaveOptions.DONOTSAVECHANGES);
        ym += 1;

        TileY += 1;

    }
```

At http://mapki.com/index.php?title=Automatic_Tile_Cutter, you can get copies of this script. Save your script as *TileCutterNamer.js*, as it is a JavaScript file. Put the script somewhere easy to find, because next you are going to open it from Photoshop.

7.8.4. Running Our Script in Photoshop

We are going to cut our tiles from one big image. For the NYC Subway map, I created my map art in Adobe Illustrator and then imported it into Photoshop. As you create your custom map, an important thing to keep in mind is to know how wide and high to make your map art. I started in Illustrator with the dimensions for the furthest zoom level; then, when I open the art in Photoshop, I tell Photoshop to create a bitmap to match whichever zoom level I am working on.

We can figure out how wide and high our art must be by multiplying the total number of tiles across and down by the individual tile width of 256 pixels. For my furthest zoom, the width and height of the subway map ends up being 1,792 pixels: 7 tiles wide by 256 pixels. But in our example batch processing script, we are making tiles for zoom level 3; 1,792 X 1,792 pixels is going to be too small an image to carve all our tiles from.

So how do we get the right width and height for other zoom levels? Every zoom level is twice the height and width of its previous zoom level. So, if our first zoom level was 1,792 pixels wide, we end up with a matrix like Table 7-1.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

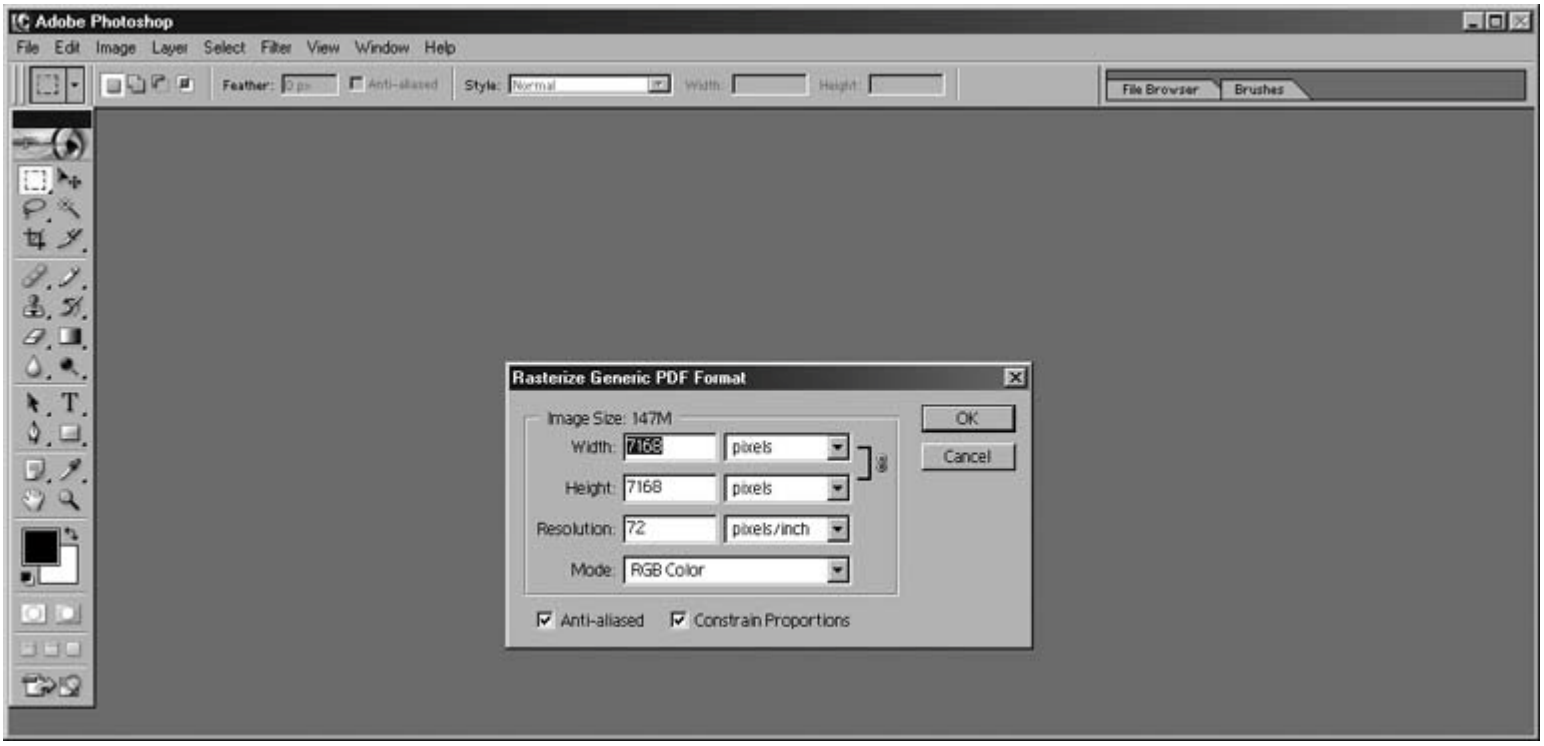
Table 7-1. Zoom level versus pre-tiled map width

Zoom level	Pixel width
5	1,792
4	3,584
3	7,168
2	14,336

To batch process tiles for zoom level 3 then, I start with a file 7,168 pixels wide. By the way, this is 28 tiles across. We know that because 7,168 pixels divided by 256 pixels is 28. For a square map area 28 pixels wide by 28 pixels high, we are then making 784 tiles! Thank goodness this script also names these tiles for us!

Getting back to our batch processor, I open my Illustrator art into Photoshop at 7,168 X 7,168 pixels, as shown in Figure 7-25.

Figure 7-25. Photoshop's File Import dialog

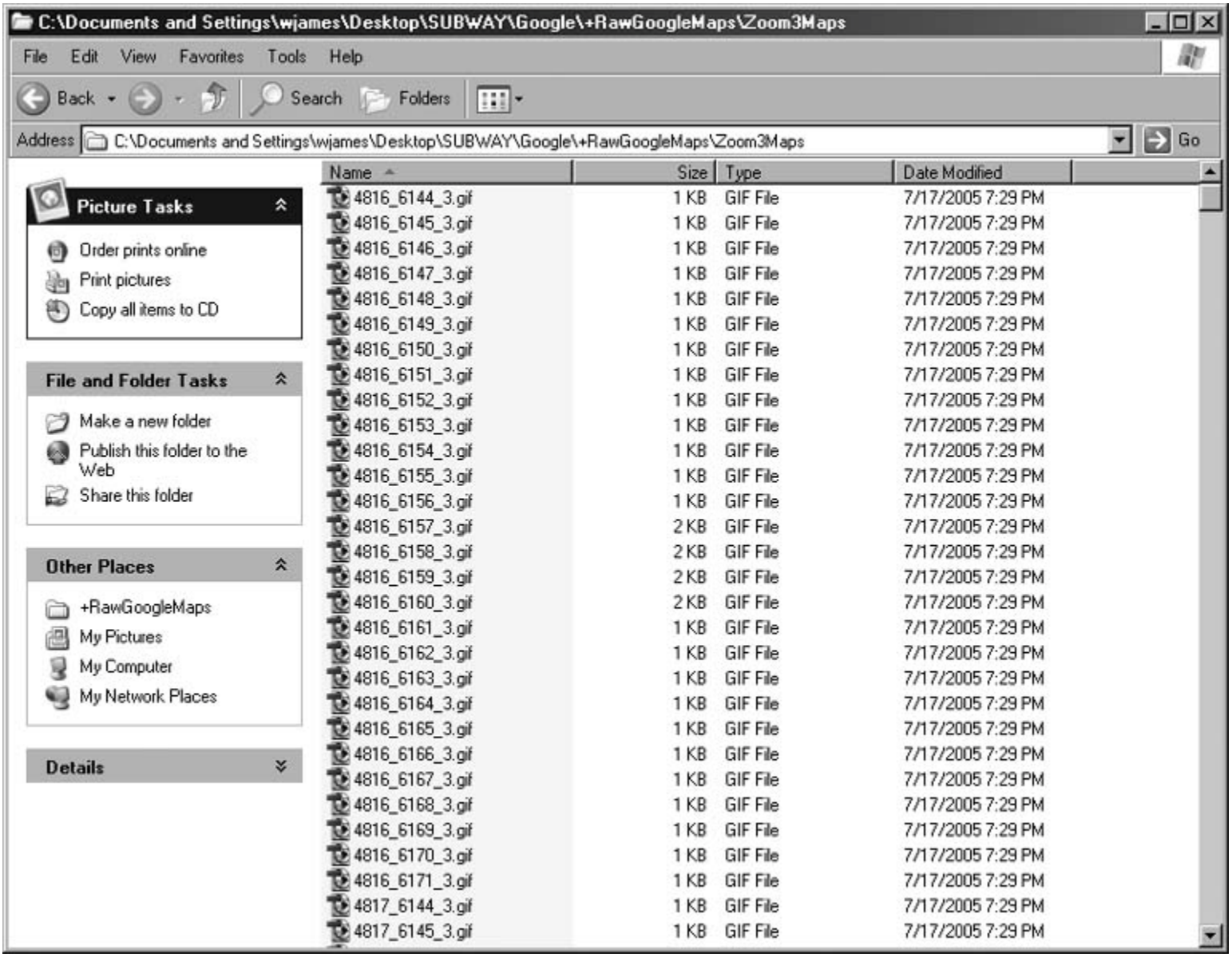


Once the file is rasterized, we run the JavaScript by going to File → Automate → Scripts. In Photoshop 7, this will pop up a small dialogue box. Choose Browse and then find the *TileCutterNamer.js* file and click Open. Off it goes! As the script runs, you can watch Photoshop make new tiles and save them. Fun, I know. If you are cutting a small number of tiles—less than 60, say—the process will take only about a minute. For something like 700 tiles, it will take 10 minutes or more. Level 3 of my subway map is made of more than 3,500 tiles. If you are cutting that many, you might want to make some other plans for the next hour. This might be the time to enjoy that cup of coffee.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

When the script has finished cutting the last tile, you will have a folder full of images, as shown in Figure 7-26.

Figure 7-26. The list of completed custom map tiles



The only thing left to do then is upload your new zoom level to your *maptiles/* folder on your server and then go and check out your new map in your web page!

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

7.8.5. Hacking the Hack

This batch processing script is written to produce GIFs. It could also be written to produce JPGs or PNGs. You will have to customize the settings in this part of the script:

```
//Set path to file and file name
saveFile = new File(FolderPath + TileX+ "_"
   + TileY+ "_" + ZoomLevel + ".gif");

//Set save options
gifSaveOptions = new GIFSaveOptions();

gifSaveOptions.colors = 64;
gifSaveOptions.dither = Dither.NONE;
gifSaveOptions.matte = MatteType.NONE;
gifSaveOptions.preserveExactColors = 0;
gifSaveOptions.transparency = 0;
gifSaveOptions.interlaced = 0;
```

7.8.6. See Also

- "Add Your Own Custom Map" **[Hack #66]**
- "Serve Custom Map Imagery" **[Hack #67]**

—Will James

Hack 69. Cluster Markers at High Zoom Levels

When necessary, avoid cramming a map too full of markers by clustering them together.

Two major problems can occur when you try to put too many markers on a Google Map at once. First, the large quantity of markers can take a long time to render, consume a lot of memory in the browser, and pretty much slow things to a crawl. We address this particular problem to some degree in "Show Lots of Stuff—Quickly" **[Hack #59]**. The other, equally serious problem is a more general matter of good digital cartography, in that too many markers placed too close together can make the markers difficult to click and the map difficult to interpret. Both problems can be fairly effectively solved by first applying a clustering algorithm to a large data set of points, and then by tailoring the display of the point clusters, such that they become readily identifiable and easy to interact with.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

7.9.1. The Hack

To experiment with this idea, we took over 600 waypoints from Rich Gibson's GPS receiver, representing the highlights of several years' worth of travels. Since Rich lives in northern California, most of the points are centered around the western United States, while the remainder are scattered from Mexico to Massachusetts to Venice, Italy. We figured that about 600 points would qualify as too many to show on a map at once, but not so many as to make experimentation difficult.

The solution that we eventually hit upon for visualizing all of Rich's travels at once can be seen at http://mappinghacks.com/projects/gmaps/cluster.html, as shown in Figure 7-27. The standard Google Maps marker is used to represent a cluster of points, roughly bounded by the red box around the marker. Clicking on a marker pans and zooms the map to the area covered by that cluster. As you get closer in, more localized clusters appear, eventually resolving into the small Google Ridefinder markers, which represent Rich's individual waypoints. Clicking on one of these smaller markers opens a small info window, showing the name and coordinates of the waypoint, as shown in Figure 7-28 .

Figure 7-27. Clusters of points show the general outlines of Rich's travels

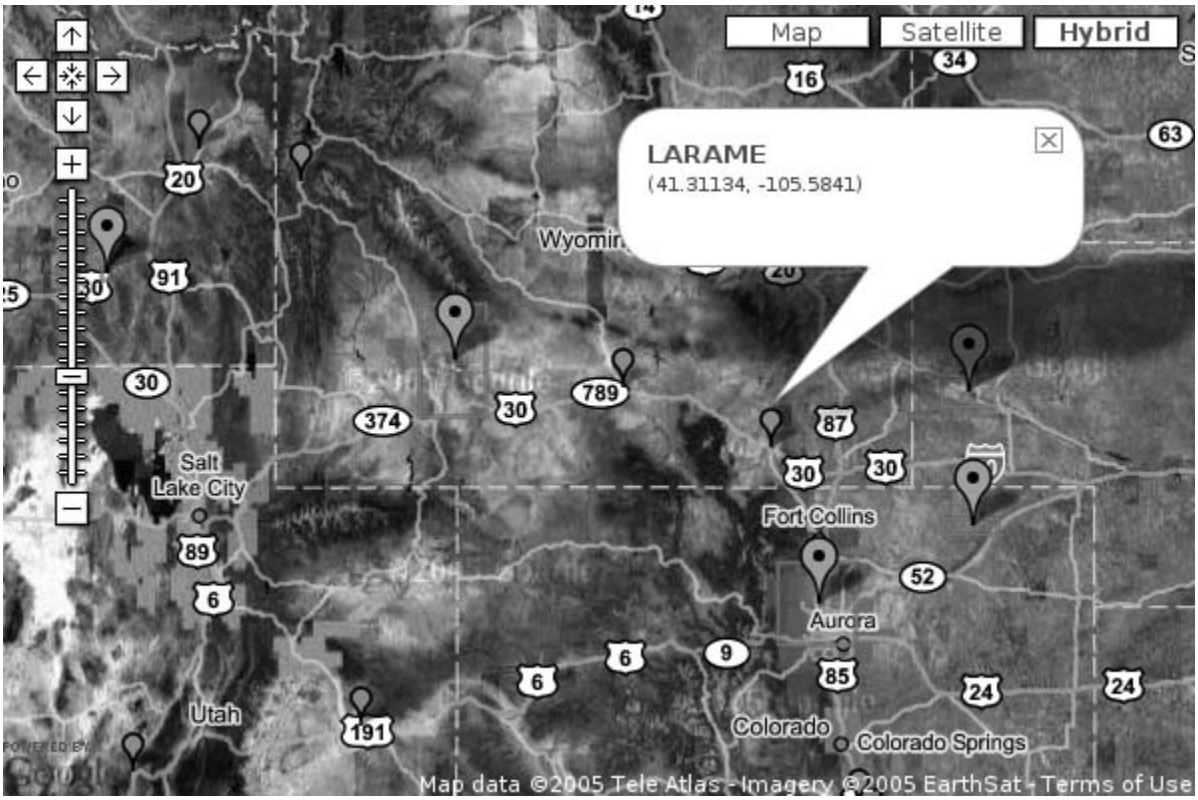
Although this demo has some obvious deficiencies, we think that, to some degree, it does succeed at summarizing the data at high zoom levels, while still allowing exploration of the individual data points. Coming up with a lightweight and reasonably fast solution to this problem took a lot of trial and error! Before we take a look at the code, let's peer down the blind alleys we followed before finally settling on our current implementation.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

7.9.2. Choosing the Right Clustering Algorithm

Stepping away from the map for a moment, let's restate the problem: we have a large number of points in a two-dimensional coordinate system, which we want to assign to a much smaller number of clusters, based in some way on their relative proximity to each other. This is a fairly common problem in computational statistics, and several well-documented algorithms already exist.

Figure 7-28. Zooming in causes the clusters to resolve as individual waypoints



7.9.2.1. k-means clustering.

The first one we tried was the *k-means clustering* algorithm, which is one of the most popular clustering algorithms in the literature, on account of its relative simplicity:

1. Select a center point for each of *k* clusters, where *k* is a small integer.
2. Assign each data point to the cluster whose center point is closest.
3. After all the data points are assigned, move each cluster's center point to the arithmetic mean of the coordinates of all the points in that cluster, treating each dimension separately.
4. Repeat from Step 2, until the center points stop moving.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

The other advantage to *k*-means clustering, aside from its ease of implementation, is that it is relatively quick, usually completing in less than $O(n^2)$ time. The most obvious disadvantage of *k*-means is that you have to somehow select the value of *k* ahead of time, but we could try to make an educated guess about that. The primary disadvantage of *k*-means derives from the ambiguity of Step 1: how, exactly, does one select the initial center points? We found that selecting random points in space led to some clusters never acquiring points at all, while randomly selecting points from the data set as initial centers led to the clusters themselves clumping up, which is something we were trying to avoid. Any attempt to choose *k* or the initial centers based on the data itself only complicated matters, in a sort of recursive fashion. Also, to top it all off, *k*-means apparently offers not only no guarantee that a globally optimum clustering will result, but not even any guarantee that a "locally" optimum clustering will result. It looked like we were back to the drawing board.

7.9.2.2. Hierarchical clustering.

The next approach we tried was a series of variations on *hierarchical clustering* algorithms. Instead of iteratively building clusters from the top down, as *k*-means does, hierarchical algorithms attempt to cluster the data from the bottom up, instead. A typical hierarchical clustering algorithm works as follows:

1. Assign each point to its own cluster.
2. Calculate the distance from each cluster to every other cluster, either from their respective mean centers, or from the two nearest points from each cluster.
3. Take the two closest clusters and combine them into one cluster.
4. Repeat from Step 2, until you have the right number of clusters, or the clusters are some minimum distance apart from each other, or until you have one big cluster.

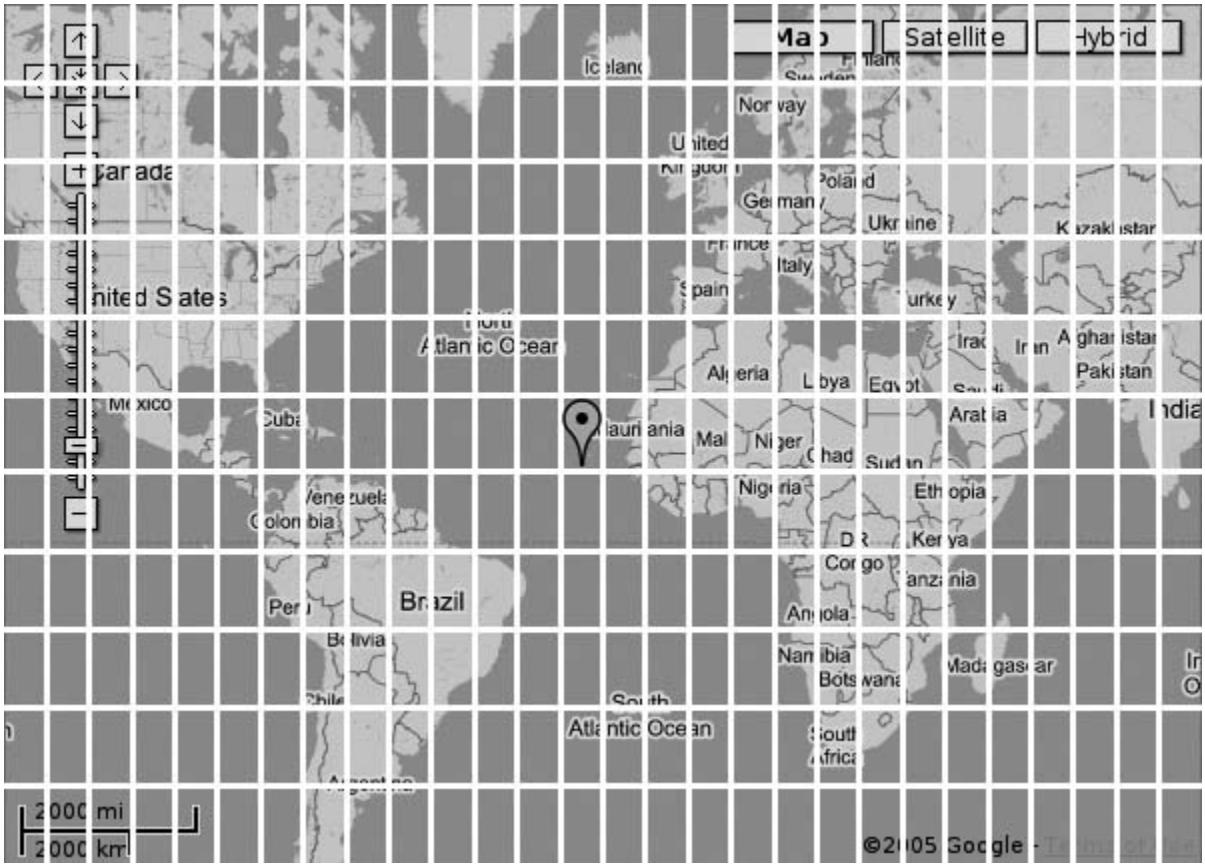
By now, you can probably guess that hierarchical clustering presents at least one of the same problems as *k*-means—namely, that you have to know somehow when you have enough clusters. Also, though hierarchical clustering can be shown to produce globally optimal results, optimality comes with a cost: hierarchical clustering algorithms are frequently known to run to $O(n^3)$ time in the worst case. Although this can be mitigated somewhat by clever use of data structures, such as using a *heap* to store the list of cluster distances, we found that our best attempt at implementing hierarchical algorithms provided reliable results, but it was, quite frankly, too slow. Polynomial-time algorithms are absolutely fine when *n* is small, but we wanted an approach that could potentially scale to thousands of points and be able to run dynamically in a reasonable time for an HTTP request. The hierarchical approach apparently just wasn't it for us.

7.9.2.3. Naïve grid-based clustering.

We realized, of course, that we were looking at the problem the wrong way: we didn't need optimal clustering, or mathematically precise clustering, or anything like that. What we were after was some way of consolidating points on a map such that: (a) the markers we display don't wind up overlapping, and (b) we don't show too many markers at once. We can solve requirement (a) by drawing a grid on the map, as shown in Figure 7-29, such that each "cell" in the grid is guaranteed to be at least as large as our marker image when the map is displayed. Next, we take all of the points within any one grid cell and call them a single cluster. For added measure, we can also check each cluster formed this way and see if there are any clusters in the immediately adjacent cells. If there are, we combine them them into a single larger cluster.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 7-29. Assignment of points to clusters based on grid cell membership



By definition, this approach is guaranteed to solve requirement (a), but how likely is it to solve requirement (b)? Initially, we note that the number of clusters we wind up with isn't sensitive to the number of points being clustered, just to their distribution. Let us assume the worst case, where there is at least one point in every potential cell. Let us suppose that a Google Map of the entire world could be as large as 1,024 pixels wide and 768 pixels high, thereby taking up all of an ordinary desktop screen. If the default Google Maps icon is 20 X 34 pixels, then we get a worst case of (1,024 ÷ 20) X (768 ÷ 34) = about 1,156 cells or, at most, 1,196 cells if we round up both sides of the product.

However, since every nine cells will be combined into a supercluster, we wind up with an absolute worst-case maximum of under 150 clusters, which, although not great, isn't horrifying, either. Let's just say that if you have a data set that covers the entire surface of the planet, you might think about using something other than Google Maps to illustrate it. Also, you can see that the algorithm is more sensitive to the size of the grid cells than anything else. At the very least, even if our naïve grid-based algorithm isn't perfect, it can be shown to run in O(*n*) time, which is an improvement over our earlier experiments.

7.9.3. The Code

The code we developed to test the grid-based clustering algorithm has two parts: first, a CGI script written in Perl runs on the server side, which filters a file of waypoints for those points within a given bounding box, clusters those points as described above, and then spits out an XML file describing the clusters; and second, an HTML page on our server contains client-side JavaScript that uses the Google Maps API to fetch the cluster description for the current map view and plot the clusters on the map.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

7.9.3.1. The server-side CGI script.

The CGI script itself is only 120 lines of Perl, but we'll just show the highlights here. If you're interested in looking at the full source, you can find it at http://mappinghacks.com/projects/gmaps/cluster-grid.txt.

Much like the Google Maps interface, our script expects a lat/long center point for the map, and a map span, measured in degrees latitude and longitude. Additionally, because the map geometry matters to our calculations, we also expect a width and height for the intended map in pixels. The following Perl code takes care of that:

```
my $q = CGI->new;
my %var = $q->Vars;

if ($var{ll} and $var{spn}) {
    my ($lon, $lat) = split( " ", $var{ll} );
    my ($width, $height) = split( " ", $var{spn} );
    $bbox = [ $lon - $width/2, $lat - $height/2,
              $lon + $width/2, $lat + $height/2 ];
}

if ($var{size}) {
    my ($width, $height) = split( " ", $var{size} );
    $size = [ $width, $height ];
}
```

The result is that the `$bbox` variable contains a reference to an array listing the minimum longitude and latitude, and the maximum longitude and latitude for our map, in that order. The `$size` variable similarly stores the width and height of the map in pixels. We then pass `$bbox` to a function that loads the waypoints from a comma-delimited file, keeping only the ones that fit within our bounding box:

```
    while (<FH>) {
        chomp;
        my ($lat, $lon, $name) = split /,\s+/, $_;
        push @data, [$lon, $lat, $name]
            if $lon >= $bbox->[0] and $lon <= $bbox->[2]
            and $lat >= $bbox->[1] and $lat <= $bbox->[3];
}
```

This hack would be more elegant if we parsed the waypoints from a GPX file instead, but we were aiming to simplify the problem as much as possible, from a technical standpoint. The points themselves are returned as a reference to an array of arrays. We then estimate the grid size in degrees, starting with a hard-coded icon size in pixels, as follows:

```
my $icon_size = [ 20, 34 ];
my $grid_x = $icon_size->[0] / $map_size->[0] * ($bbox->[2] - $bbox->[0]);
my $grid_y = $icon_size->[1] / $map_size->[1] * ($bbox->[3] - $bbox->[1]);
```

Basically, we divide the width of the marker icon (in pixels) by the width of the entire map (also in pixels), which gives us a proportion of the map that a single marker would cover, in the horizontal dimension. We multiply this fraction by the total span of the map in degrees longitude, to yield the number of degrees longitude that a marker would cover on a map of this size and span. We then perform the same calculation the marker height, map height, and map span in degrees latitude to get the marker height in degrees latitude. By making our grid cells at least this many degrees wide and high, we can guarantee that none of the markers we place on the map will visually overlap each other by much, if at all.

We then call the `cluster()` function, which takes the bounding box, the grid cell size, and the data points as arguments:

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
sub cluster {
    my ($bbox, $grid, $data) = @_;
    my %cluster;

    for my $point (@$data) {
        my $col = int( ($point->[0] - $bbox->[0]) / $grid->[0] );
        my $row = int( ($point->[1] - $bbox->[1]) / $grid->[1] );
        push @{$cluster{"$col,$row"}}, $point;
    }
```

These few lines of code actually do all the work of assigning each point to a cluster. We divide the difference between the point's longitude and the western edge of the map by the grid cell size, which, discarding the remainder, gives us the grid column that the point lies within. We follow a parallel calculation to determine the row of the point, counting from south to north. We then push the point on to a hash of arrays that lists the points within each cell. Since we plan to take each cell that contains one or more points as an initial cluster, this hash is simply called `%cluster`. We're already halfway finished.

The next bit of code is a bit tricker, as it concerns the neighbor analysis that we use to combine adjacent clusters into superclusters. As a heuristic, we assume that the points are already naturally clustered in some way and conclude that the densest of our initial clusters are most likely to be the "centers" of the combined superclusters. We implement this heuristic by sorting the list of clusters we've found, according to the number of points within each one, in descending order:

```
my @groups = sort { @{$cluster{$b}} <=> @{$cluster{$a}} } keys %cluster;
```

We then iterate over the list of clusters, looking at the eight cells immediately adjacent to each one. If any of the adjacent cells also contains a cluster, we add the points from that cluster to our new supercluster and remove the subsumed cluster from our list. Finally, when all is said and done, we return an array of arrays, containing our original points grouped into clusters.

```
    for my $group (@groups) {
        next unless $cluster{$group};
        my ($col, $row) = split ",", $group;
        for my $dx (-1 .. 1) {
            for my $dy (-1 .. 1) {
                next unless $dx or $dy;
             $dx += $col; $dy += $row;
                if (exists $cluster{"$dx,$dy"}) {
                    my $other = delete $cluster{"$dx,$dy"};
                    push @{$cluster{$group}}, @$other;
                }
            }
        }
    }

    return [values %cluster];
}
```

We won't bore you with the tedious details of generating XML from this data structure; once again, you can refer to the source code online, if you're particularly interested. However, in order to understand how the JavaScript frontend works, it's worth looking at an example output from this script:

```
<points>
     <cluster lat="40.76706" lon="-123.334565"
            points="187" width="1.85699" height="6.57074" />
     <cluster lat="44.40734" lon="10.832265"
            points="70" width="3.33653" height="5.03806" />
     <cluster lat="41.507245" lon="-72.426195"
            points="10" width="2.74580999" height="1.73063" />
     <cluster lat="39.27062" lon="-115.56396"
```


Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
            points="239" width="13.66804" height="11.94808" />
     <point name="CAIRO" lat="30.05671" lon="31.23585" />
</points>
```

As you can see, this document contains enough information to draw each cluster, as well as to allow the UI to zoom directly in on the extents of any one cluster. Finally, any cluster with only one point isn't really a cluster: it's just a point and is treated as such. The JavaScript user interface therefore only needs to be able to pass the map center, span, and size into this script, and then be able to parse the cluster and point descriptions from the XML output.

7.9.3.2. The client-side JavaScript.

Naturally, the JavaScript frontend to this clustering experiment needs to do all the usual things to import the Google Maps API code and set up the map itself. These tasks are already well-covered elsewhere in this book, and, of course, you can always View Source on the demo page, if you're interested in the details. We'll go straight to the `loadClusters()` function, which calls our CGI script when the page is loaded:

```
function loadClusters () {
 var request = GXmlHttp.create();
 var center = map.getCenterLatLng();
 var span = map.getSpanLatLng();
 var url = "http://mappinghacks.com/projects/gmaps/cluster-grid.cgi?"
    url += "ll=" + center.x + "+" + center.y
          + "&spn=" + span.width + "+" + span.height
       + "&size=" + map.mapSize.width + "+" + map.mapSize.height;

 request.open("GET", url, true);
 request.onreadystatechange = function() {
     if (request.readyState == 4)
          showOverlays(request.responseXML);
  };
}
```

This code initiates a `GXmlHttp` request in the usual way, passing in the map center, span, and size. The `map.mapSize` property used here is undocumented in the Google Maps API, but you could almost certainly use the DOM API to find out the width and height styles of the map `div` element instead. When the HTTP request returns, the parsed XML document is passed to the `showOverlays()` function, which is shown below:

```
function showOverlays (xml) {
    map.clearOverlays();
    var cluster = xml.documentElement.getElementsByTagName("cluster");
    for (var i = 0; i < cluster.length; i++) {
        showCluster( cluster[i] );
    }
    var point = xml.documentElement.getElementsByTagName("point");
    for (var i = 0; i < point.length; i++) {
        showPoint( point[i] );
    }
}
```

The `showOverlays()` function clears the map of its existing overlays and then uses the `getElementsByTagName()` method from the Document Object Model API to extract each `cluster` and `point` element from the XML document, passing them one by one to the appropriate rendering function. The `showPoint()` function doesn't do anything more exciting than put a marker on the map in the appropriate place and bind an info window popup to the click event. Let's look at the `showCluster()` function, instead, because that's where most of the excitement really is:

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

```
    function showCluster (elem) {
        var point = new GPoint(parseFloat(elem.getAttribute("lon")),
                               parseFloat(elem.getAttribute("lat")));
        var width = parseFloat(elem.getAttribute("width")),
            height = parseFloat(elem.getAttribute("height")),
            count = parseInt(elem.getAttribute("count"));

        var marker = new GMarker(point);
```

The `showCluster()` function parses the various attributes out of the `cluster` element and uses the latitude and longitude to create a new marker. So far, no surprises. The action that we bind to that marker's click event is much more interesting:

```
        GEvent.addListener(marker, "click", function() {
            var span = new GSize(width * 1.1, height * 1.1);
            // undocumented: http://xrl.us/getLowestZoomLevel
            var zoom = map.spec.getLowestZoomLevel(point, span, map.
viewSize);

            loadClusters();
            map.centerAndZoom(point, zoom);
        });
```

Instead of binding an info window to the click event on the marker, we bind an anonymous function that calculates the center point and span of the cluster, augmenting the span by 10% in each dimension, so that all of our points are well within the map view and not hiding out at the edges of the map. Next, we use the method described in "Find the Right Zoom Level" **[Hack #58]** to calculate the best zoom level for viewing the entire cluster. The anonymous function then calls `loadClusters()` to kick off a request for the new set of clusters and points for that map view, and then pans and zooms the map to that view.

The finale of the `showCluster()` function gives the cluster some visual clarity, by drawing a rectangular polyline around the extents of the cluster, so that the user can see which part of the map that the cluster covers:

```
        var bbox = new GPolyline(
           [new GPoint(point.x - width/2, point.y - height/2),
            new GPoint(point.x - width/2, point.y + height/2),
            new GPoint(point.x + width/2, point.y + height/2),
            new GPoint(point.x + width/2, point.y - height/2),
            new GPoint(point.x - width/2, point.y - height/2)],
           'ff0000', 2, .80);

        map.addOverlay(marker);
        map.addOverlay(bbox);
}
```

The user can zoom into a cluster by clicking on it, but what if she wants to zoom back out? A single event handler takes care of refreshing the map display when the user zooms in or out **[Hack #60]**.

```
GEvent.addListener(map, "zoom", loadClusters);
```

7.9.4. Conclusion

We've seen here that there can be viable solutions to the vexing problem of what to do when you have too many points to show on a map. We've also seen that some traditional clustering algorithms may be inadequate for certain tasks of practical digital cartography and that a more

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

customized approach, using grid-based clustering, can yield pretty good results. There's no doubt that this method isn't perfect, though, and we'd really like to see it improved upon. If you do use this technique as an inspiration for your own projects, please let us know!

7.9.5. See Also

- Wikipedia on clustering algorithms: http://en.wikipedia.org/wiki/Data_clustering
- "Find the Right Zoom Level" **[Hack #58]**
- "Show Lots of Stuff—Quickly" **[Hack #59]**
- "Make Things Happen When the Map Moves" **[Hack #60]**

Hack 70. Will the Kids Barf? (and Other Cool Ways to Use Google Maps)

Google Maps Hacks are turning up everywhere; here are some things we just didn't have time to cover fully.

My kids get sick on curvy roads. It is just a fact of life. One measure of the curviness of a road is the *detour index*. This is the ratio of the shortest road distance and the straight-line distance. Some mathematicians also call this the "fractal dimension of the polyline."

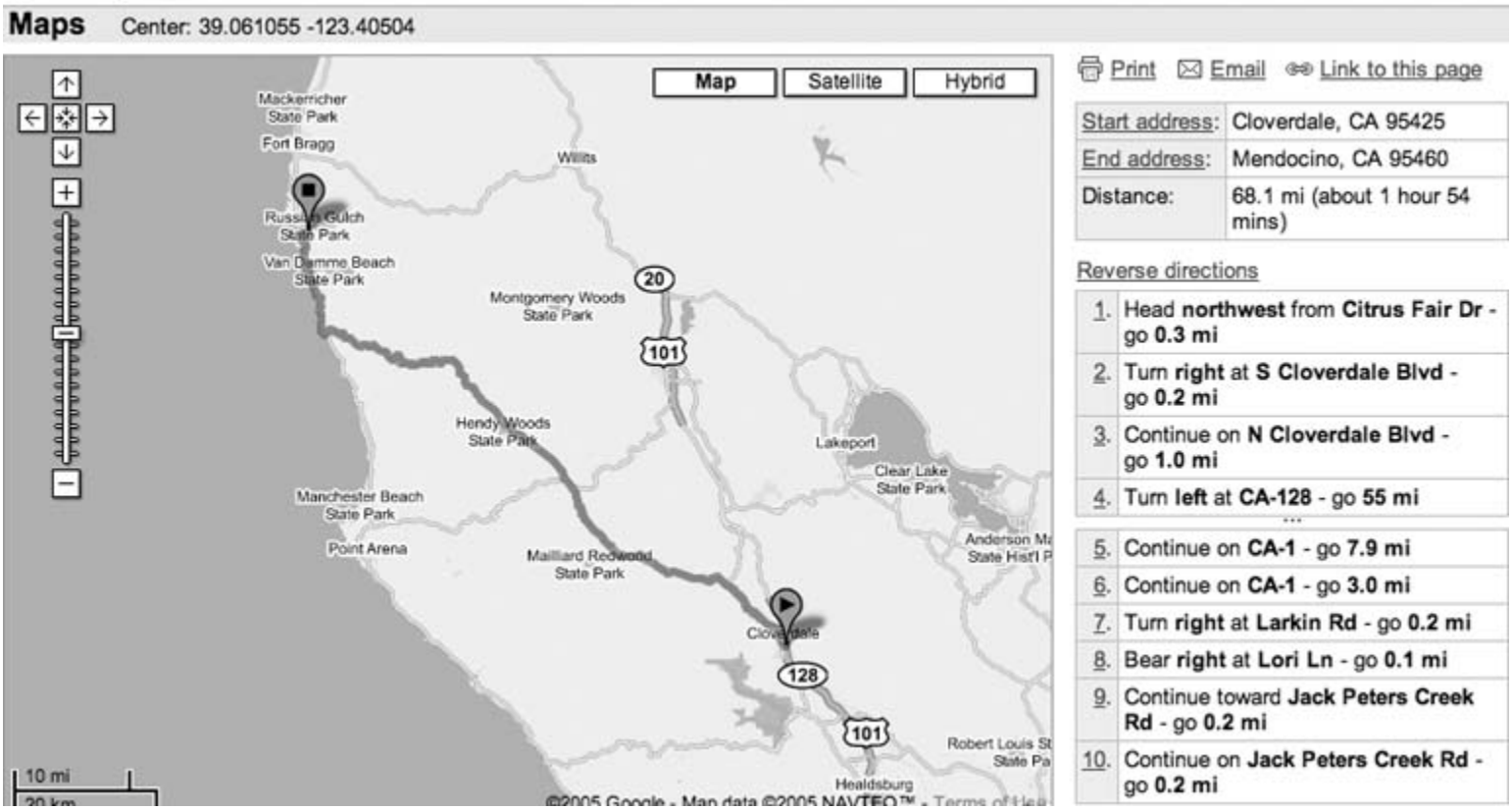
```
Detour Index = road distance ÷ straight line distance X 100
```

We can get the road distance by calculating the driving directions and grabbing that distance. Figure 7-30 shows it is 68.1 miles from Cloverdale to Mendocino.

See "How Far Is That? Go Beyond Driving Directions" **[Hack #12]** for different ways to calculate the straight-line distance. I used our sample tool at http://mappinghacks.com/projects/gmaps/lines.html. Be careful to measure the same distance! I noted the start and stop points of the Google route and tried to match those in calculating the straight-line distance as shown in Figure 7-31.

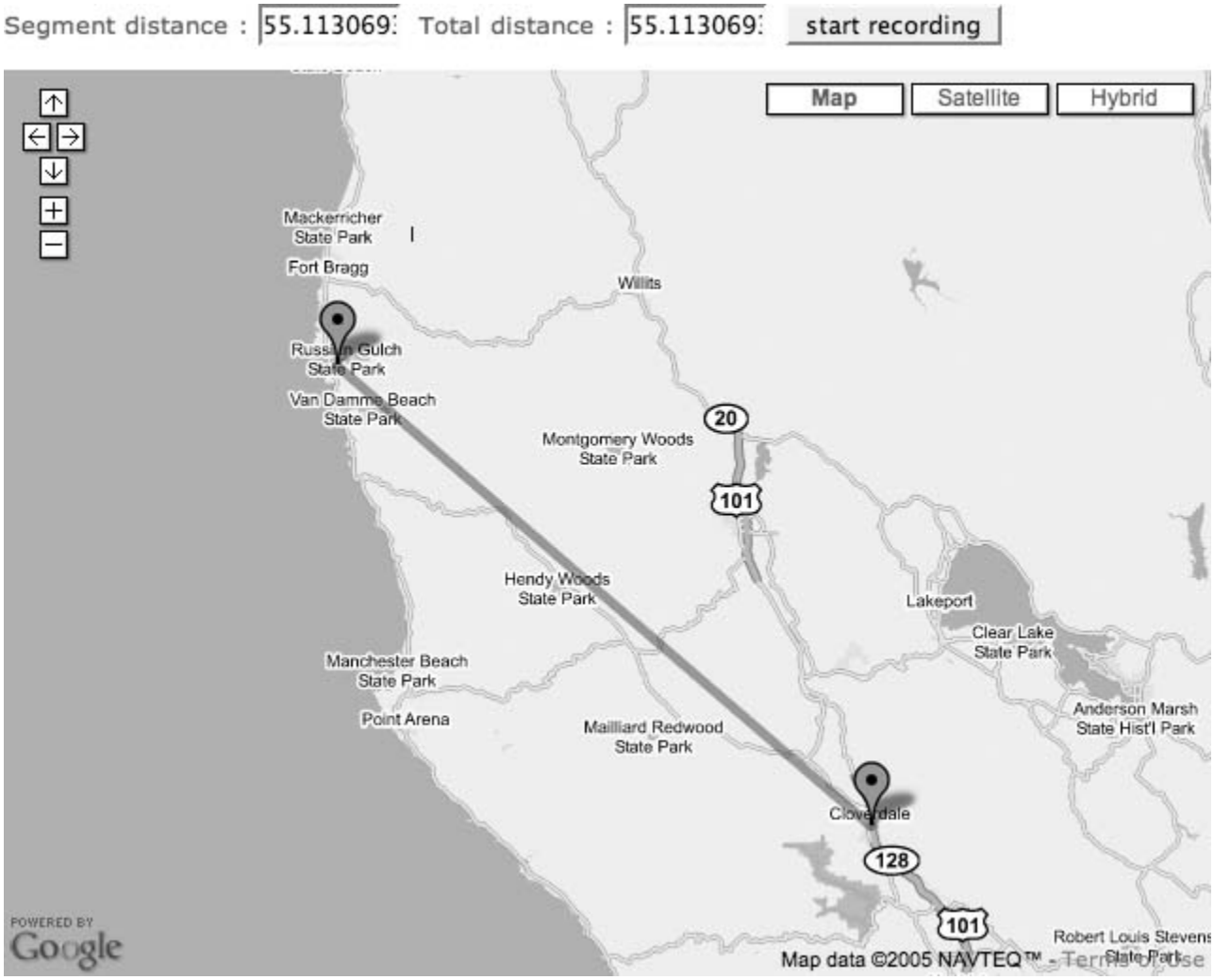
Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 7-30. From Cloverdale to Mendocino is 68.1 miles on the road



Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 7-31. Straight-line distance from Cloverdale to Mendocino is 55.1 miles



The straight-line distance is 55.1 miles, so the detour index is `68.1÷55.1 X 100=123.6`.

Through careful analysis of four or five trips, as partially described in *Mapping Hacks*, I've determined that a detour index over about 120 is a recipe for carsick fun. If you are a motorcyclist looking for curves, you can use the number for your own ends. But remember, as the heartless half of this book's writing team says, "For some people, motion sickness is no laughing matter. For the rest of us, it is."

We really wanted to turn this into a Google Maps demo that you could experiment with yourself, but we just didn't have the time!

7.10.1. Other Cool Google Maps Hacks

One of the painful challenges in writing this book during the heady days of the release of the official API was that we just could not keep up with all of the cool work that was being done, much less contribute to it ourselves. We'd watch our email, and the Google Maps API mailing list, and Del.icio.us, and more cool "must include" hacks appeared than we could possibly ever write about. Here are a few of the coolest.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

7.10.1.1. Google sightseeing.

What could be better than standing on a sunlit peak overlooking the Grand Canyon at sunset with your loving family by your side? How about looking over it without the 10-hour car trip in an overheating station wagon with sticky seats, grumpy children, and a fed-up wife?

We have Google, so why bother seeing the world for real? The Google Sightseeing site at http://www.googlesightseeing.com/ shows neat places on Google Maps. Consider it a map geeks blog: Alex, James, and Olly scour (imagery of) the Earth to bring you landscapes to amuse and amaze.

7.10.1.2. ZIP Code maps.

Postal code boundaries are often a bit arbitrary. You can go to http://maps.huge.info/, enter a ZIP Code, and see the area it covers. Figure 7-32 shows 95472 on a map. ("Examine Patterns of Criminal Activity" **[Hack #18]** shows another example of rendering arbitrary GIS vector data on Google Maps.)

The site is careful to note its own limitations:

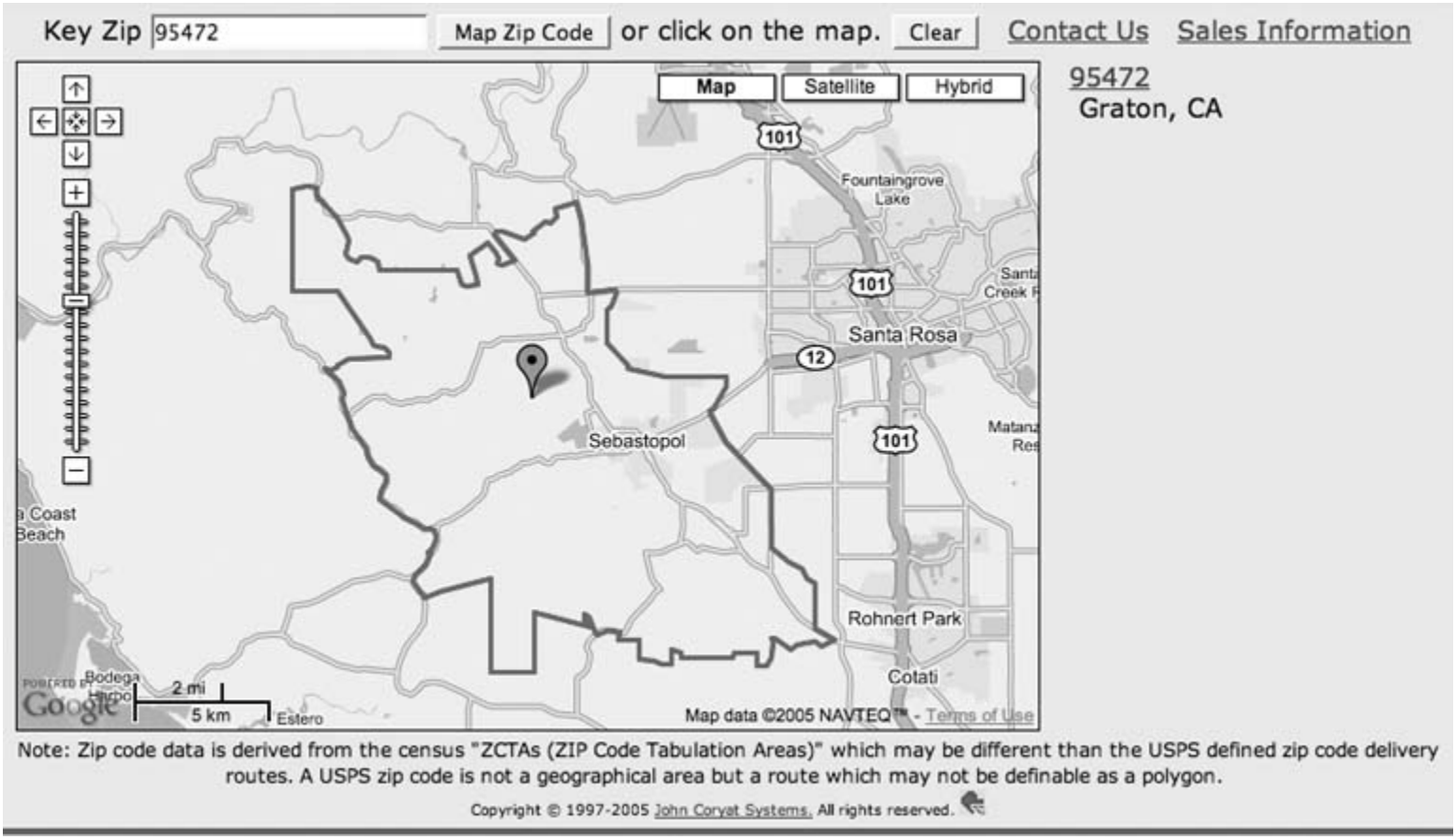
> ZIP Code data is derived from the Census 'ZCTAs (ZIP Code Tabulation Areas),' which may be different than the USPS defined ZIP Code delivery routes. A USPS ZIP Code is not a geographical area but a route which may not be definable as a polygon.

7.10.1.3. Google Planimeter.

You can use Google Maps to measure areas at http://www.acme.com/planimeter/. Navigate to your selected area and then click on at least three points, and it will calculate the area. Figure 7-33 shows that the Point Reyes National Seashore is roughly 137 square miles.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 7-32. ZIP Code 95472 displayed on a map



7.10.1.4. Play games on Google Maps.

In Tripods, you "battle invading Google Maps Tripod markers that are invading Manhattan." This is a multiplayer game, so you can gang up with other players to fight the menace, as shown in Figure 7-34. You can play Tripods at http://www.thomasscott.net/tripods/.

Find the Landmark, at http://landmark.mapsgame.com/, gives you a landmark and times your attempts to find it. Figure 7-35 shows my "I give up" time at finding the Space Needle in Seattle.

Other games are discussed on the Games On Google Maps wiki page at http://moloko.itc.it/trustmetricswiki/moin.cgi/GamesOnGoogleMaps.

7.10.1.5. Map versus satellite.

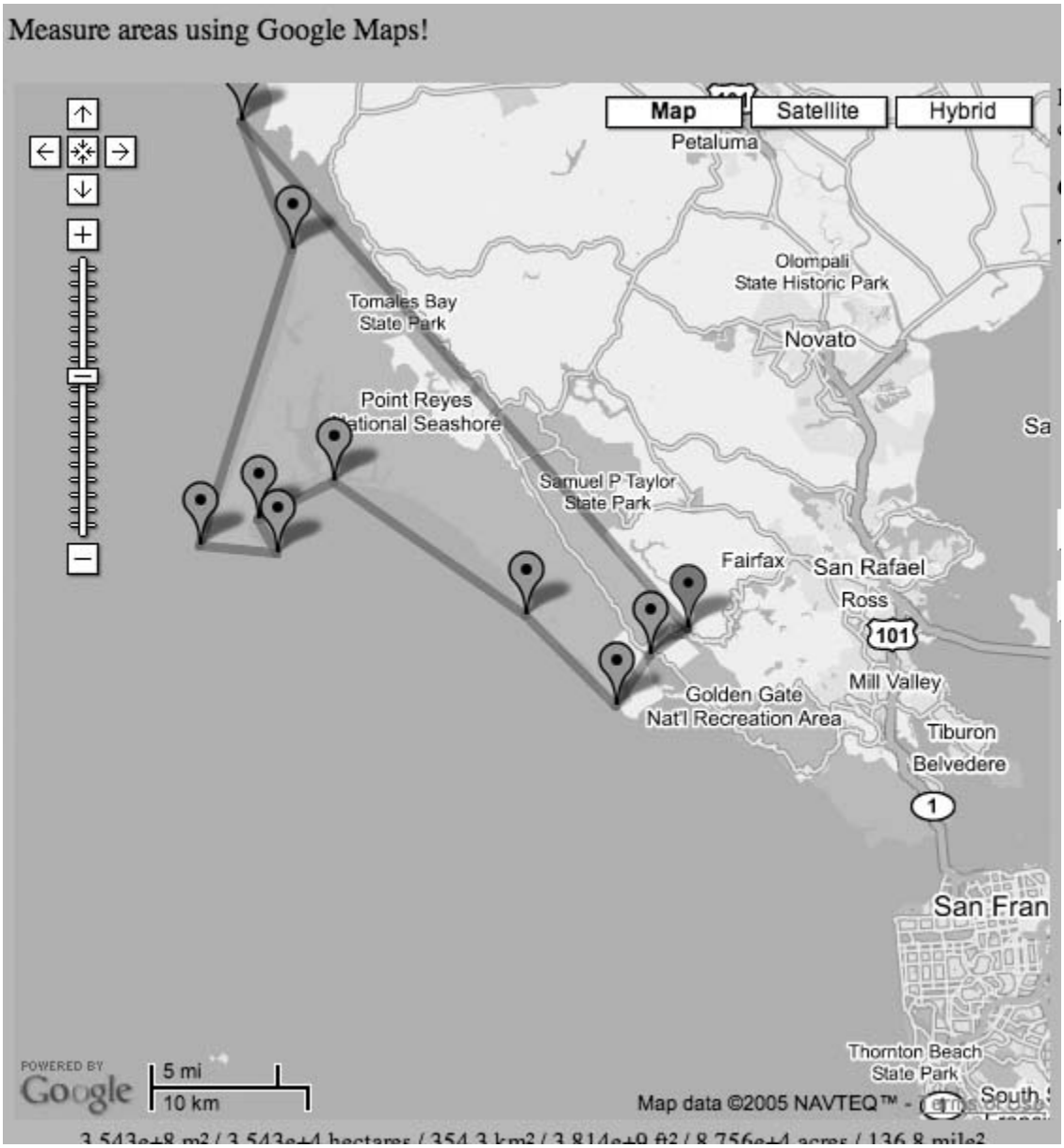
Overlay a small section of a map view over the satellite view, or vice versa, at http://www.kokogiak.com/gmaps-transparencies.html. This helps you to see discrepancies between the map and the imagery, as in Figure 7-36.

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

7.10.1.6. Edible plants in the public domain.

The Garden of the Commons at http://commonsgarden.org/ maps plants growing in public spaces that have food or medicinal value and offers users the chance to contribute their own finds.

Figure 7-33. Measure the area of Point Reyes National Seashore



Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

7.10.1.7. Animate a route.

There is a bookmarklet at http://www.ergul.com/maps/ that will animate your driving directions, even cross-country. Enter your end points and watch as the computer follows your route for you.

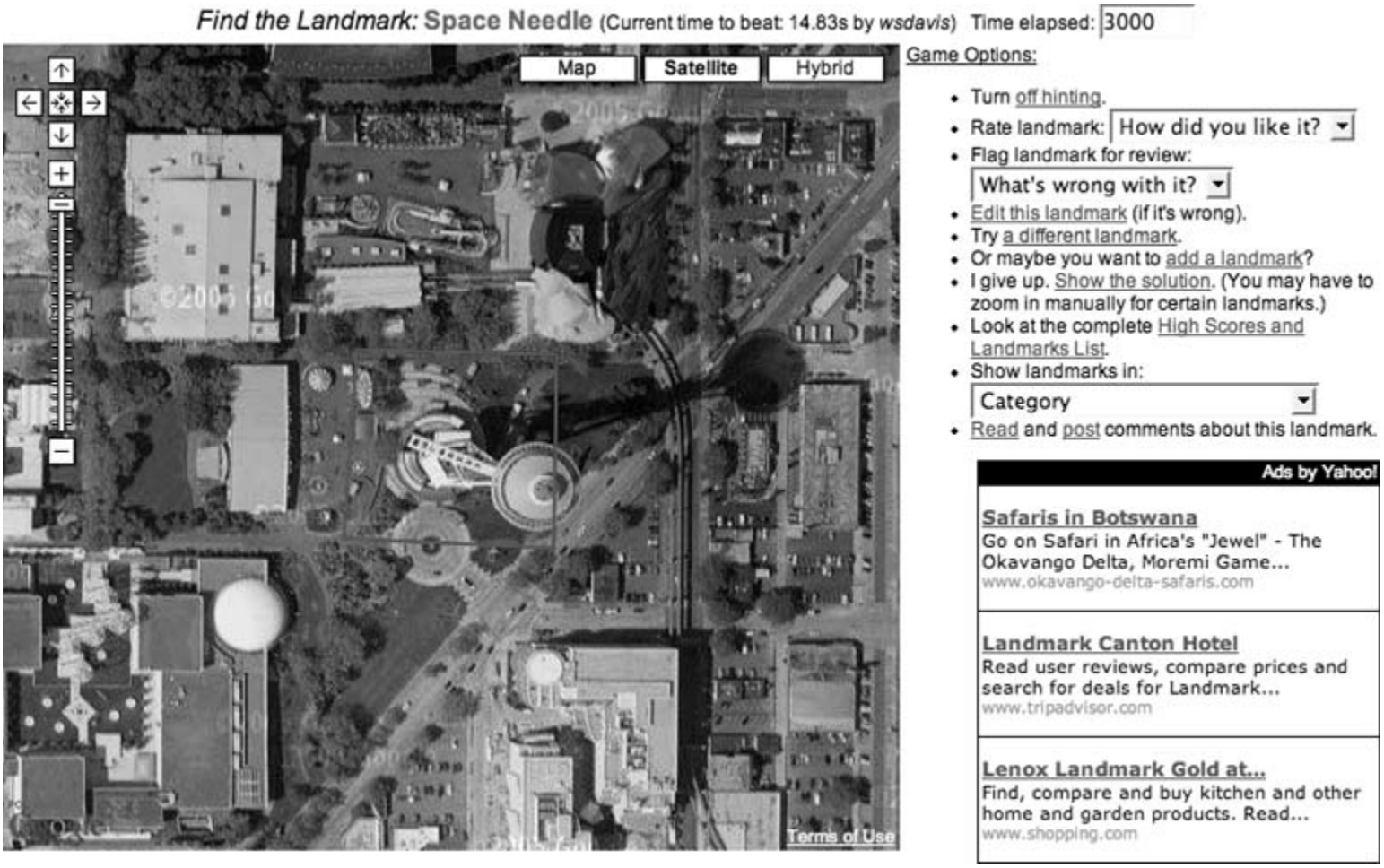
7.10.1.8. Track your credit card spending.

Greg McCarroll has done work on mapping his credit card statement at http://www.mccarroll.org.uk/~gem/creditcardtracker/. Being able to see where you spend money might just help you manage your spending habits!

Figure 7-34. Fight the Tripods!

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 7-35. Find the Landmark—The Space Needle



Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006

Figure 7-36. Compare the map with the satellite view

7.10.2. Where to from Here?

We can recommend a couple of web sites that track the Google Maps phenomenon in all its glory and bring you up-to-date on the latest developments. The first is Google Maps Mania, at http://googlemapsmania.blogspot.com/, a veritable cornucopia of Google Maps news and reviews. The other, of course, is our very own Mapping Hacks blog, at http://mappinghacks.com/ where we hope to present a more vaired and in-depth view of the rapidly evolving world of digital cartography, including, of course, Google Maps.

So where to now? Google Maps has provided a brush or two, an easel, and a few pots of paint. Get out there and see what you can make!

Google Maps Hacks By Schuyler Erle, Rich Gibson
ISBN: 0596101619 Publisher: O'Reilly Print Publication Date: 1/1/2006