Web-GIS Final Project Report

Group member: Jianfang Li, Wei Wu, Jong Young Lee

1. Introduction of NYC life web system

Our project is called NYC life which will provide location and direction information for some useful and popular sites, for example library, hospital, museum, tennis courts, football courts, swimming pools etc. Figure 1 is an overview of our web system.

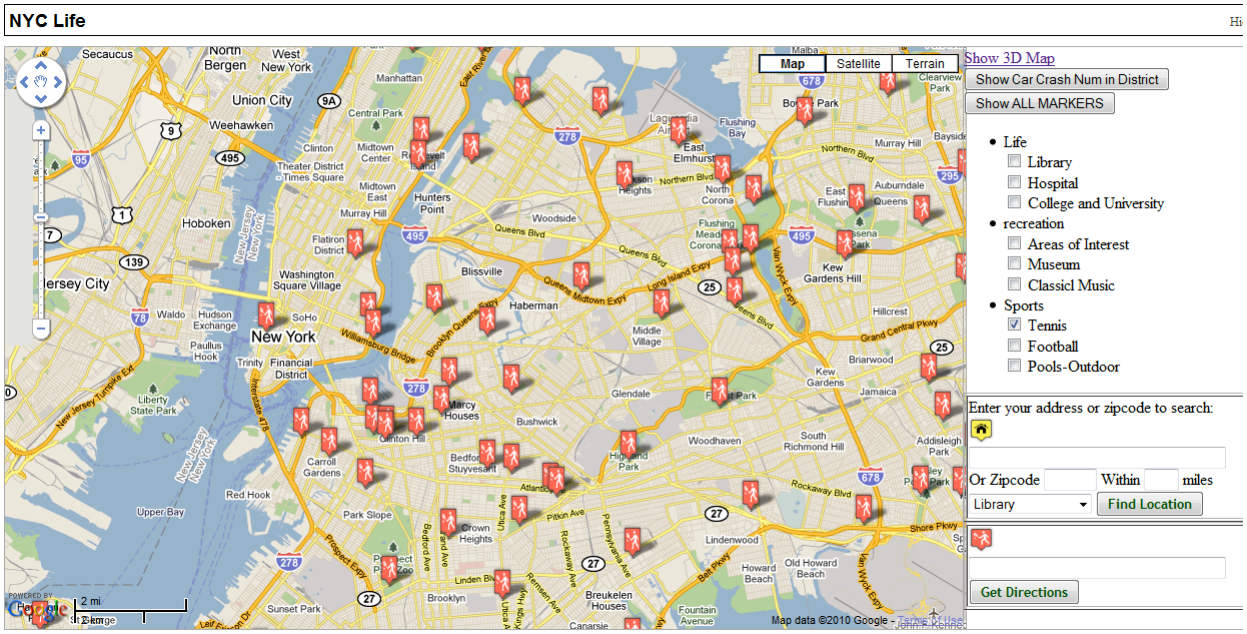


Figure 1: an overview of our web system

Our web system is built based on Google Map API V2 /Google earth API and PostgreSQL database and includes the following major functions:

- Customer Location markers and large number of marker management
- Multiple Google markers management: check box and markers cluster
- KML overlay for NYC Police Precinct and car crash information
- Address decoding and reverse addressing decoding
- Location searching based on address or zip code and distance
- Location direction with a new Google window and local window.
- Location Streetview
- Basic google earth 3D view of tennis courts

For Google earth 3D part, we only implement basic function because time is limited.

2. Data Processing

Most of the data we used is downloaded from NYC government website: http://www.nyc.gov/html/datamine/html/data/geographic.shtml. Date includes: tennis, football, pools, library, hospital, college and university, areas of interests, museum, and classical music. NYC Police Precinct data is from http://www.nyc.gov/html/dcp/download/bytes/nypp_10bav.zip, and car crash number data is from http://geoteci.engr.ccny.cuny.edu/WebGIS/nyccrash.txt.

2.1 Import data into database

All the geometry information data is shapefile, we have to (shp2pgsql and psql) convert shapefile into .sql file then import into database.

Command:

```
shp2pgsql  -s 2263 -i –I –D dpr_tennis.shp dpr_tennis > dpr_tennis.sql
psql -U wu -d d118 –f dpr_tennis.sql
```

2.2 Convert geometry polygon to geometry point

The geometry type of tennis, football and pools are polygon, but we want to show the points in the Google Map. So we have to convert polygon geometry into point geometry. Use tennis data as an example.

Since every tennis court in the map is a square, there are more than seven hundreds of squares. First, we merge the tennis courts in the same park as one tennis marker location using "st_convexhull(st_collect(the_geom))", and calculate the number of courts in every location, then the center point of each new geometry using "ST_centroid(the_geom)", and using "Geography(ST_Transform(tennis1.centergeom,4326))" and "ST_Y(geometry)/ ST_X(geometry)" to calculate the longitude and latitude of each points. At last we get a new table named "tennis", which includes "gid", "centergeom", "longitude", "latitude", "name" and "type" columns.

```
CREATE TABLE  tennis AS
        select count(t.gid) as courtnumber, st_convexhull(st_collect(t.centergeom)) as the_geom
        from dpr_tennis t, dpr_parks p
        where st_contains(p.the_geom, t.centergeom)
        group by p.gid;

ALTER TABLE tennis1 ADD COLUMN gid serial;
ALTER TABLE tennis1 ADD COLUMN centergeom geometry;
ALTER TABLE tennis1 ADD COLUMN xlat numeric;
ALTER TABLE tennis1 ADD COLUMN ylong numeric;

update tennis
set centergeom = a.geom
```

```
from (select gid as gid, ST_centroid(the_geom) as geom from tennis1 ) a
where tennis1.gid = a.gid;

update tennis
set xlat = b.long
from (select a.gid as gid, ST_Y(a.geog::geometry) as long from (select gid as gid,
Geography(ST_Transform(tennis1.centergeom,4326)) as geog from tennis1) a) b
where tennis1.gid = b.gid;

update tennis
set ylong = b.long
from (select a.gid as gid, ST_X(a.geog::geometry) as long from (select gid as gid,
Geography(ST_Transform(tennis1.centergeom,4326)) as geog from tennis1) a) b
where tennis1.gid = b.gid;
```

2.3 Merge all the data in different table into one table

We use “union” to do create the table nyclife.

```
CREATE TABLE nyclife AS
        select name, courtnumber, centergeom, xlat, ylong, category from tennis
        union
        select name, courtnumber, centergeom, xlat, ylong, category from football
        union
        select name, courtnumber, centergeom, xlat, ylong, category from pools
        union
        select name, courtnumber, the_geom as centergeom, xlat, ylong, category from aoi
        union
        select name, courtnumber, the_geom as centergeom, xlat, ylong, category from hospital
        union
        select name, courtnumber, the_geom as centergeom, xlat, ylong, category from university
        union
        select name, courtnumber, the_geom as centergeom, xlat, ylong, category from library
        union
        select name, courtnumber, the_geom as centergeom, xlat, ylong, category from music
        union
        select name, courtnumber, the_geom as centergeom, xlat, ylong, category from museum;

ALTER TABLE nyclife ADD COLUMN gid serial;
```

2.4 Generate KML file

Our website will display NYPP precinct information in the map. NYPP precinct data comes from shapefile. However, Google Map API can’t use shapefile directly. In order to display NYPP precinct, we use Google Earth professional to convert shapefile to KML file. There are one difference of KML file display in between Google Earth and Google Map. In Google Earth, every polygon will show the information in <ExtendedData> … </ExtendedData> when we click; but in

Google Map, it just can show information in <name>…</name> and < description>…</ description> part after < Placemark>. So we manual add different color layer and car crash information into KML file.

KML file is very large when there is lots of information in the file; so it is loaded very slowly by Google Map. KMZ stands for KML-Zipped, a compressed version of KML file. It is just one third sizes as the original KML file after we converted.

3. Structure

This Google Map API application includes five files: life.html, nyclife.php, style.css, filter.php, zipcode.php ,nypp.kmz and some open source javascript library files .

nyclife.php is used to get data from database system. We use "$connection=pg_connect()" function to connect to the database system, and use "$result = pg_query($connection," SELECT gid,name,courtnumber,xlat,ylong,category FROM nyclife");" to fetch data.

style.css is designed primarily to describe webpage presentation, including layout, colors, and fonts.

life.html is designed webpage using JavaScript.

Filter.php and zipcode.php are used for dynamical location searching based on address or zip code and distance submitted by clients. In filter.php and zipcode, we will use ST_Distance() function to calculate distance between source and locations in the PostGIS. Because data in the PostGIS use SRID 2263, however Google map use SRID "4326" instead, so we use ST_Transform() to convert SRID from "4326" to "2263" during query.

4. Marker Management

4.1 Custom marker icon

We set a baseIcon, then all the new icons inherent the base icon's format, and show different piture.

```
var baseIcon = new GIcon(G_DEFAULT_ICON);
baseIcon.iconAnchor = new GPoint(9,34);
baseIcon.iconSize = new GSize(20,34);
baseIcon.infoWindowAnchor = new GPoint(9,2);

gicons["tennis"] = new GIcon(baseIcon,"icons/tennis.png");
gicons["football"] = new GIcon(baseIcon,"icons/soccer.png");
gicons["pools"] = new GIcon(baseIcon,"icons/pool.png");
gicons["aoi"] = new GIcon(baseIcon,"icons/beautiful.png");
gicons["hospital"] = new GIcon(baseIcon,"icons/hospital.png");
gicons["library"] = new GIcon(baseIcon,"icons/library.png");
```

```
gicons["music"] = new GIcon(baseIcon,"icons/music-classical.png");
gicons["university"] = new GIcon(baseIcon,"icons/university.png");
gicons["museum"] = new GIcon(baseIcon,"icons/museum.png");
```

4.2 Check box to arrange markers

We want to show nine different category data in the map, includes tennis, football, pools, library, hospital, college and university, areas of interests, museum, and classical music. All these data are in one table, and we can read these data at a time using "GDownloadUrl("nyclife.php", function(data))". Create a marker according to its category, for example, if the category is "tennis", this location will show the tennis icon. Push all the markers into an array "gmarkers=[]". When check box is checked it will active show () function to show all the markers which have the same category, and when uncheck box is unchecked it will call hide () function. Then check box will show different data layer.

The following Figure 4.1 shows two layers: tennis and museum.

Figure 4.1: display Tennis and Ares of Interest layers

4.3 Show marker information window

By clicking a marker, there will be an info window, which includes: specified info, address, get direction and street view. (Figure 4.2)

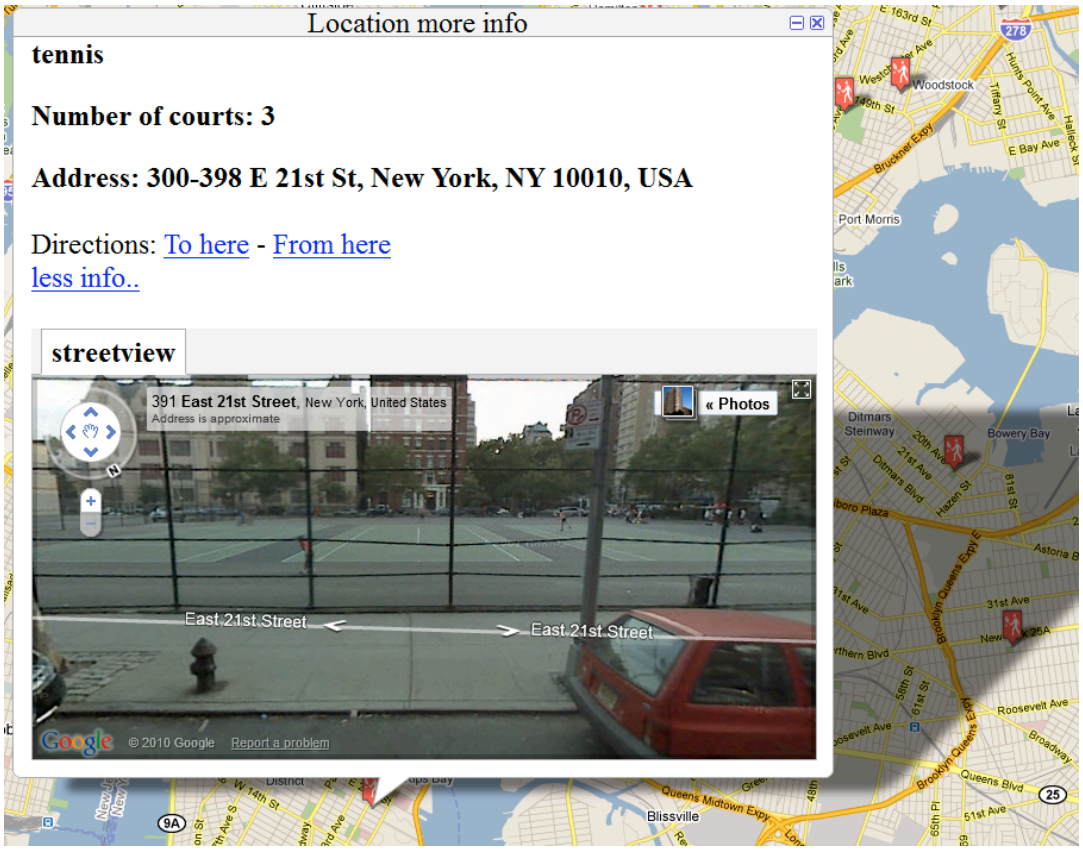


Figure 4.2 Marker info window

(1) Show specified icon information

Using a variable named “html” to show the information of icons; this is specified for every icon, and get from database.

```
if(category == "tennis" || category =="football" || category == "pools"){
        html = "<b>number of courts: " + courtnumber + "<\/b><p>" + category;
}else{
        html = "<b>name: " + name + "<\/b><p>" + category;
}
```

(2) Get direction

Using two arrays (to_htmls[] and from_htmls[]) to set the information of get direction: “from here” and “to here”.

```
to_htmls[i] = html + '<br>Directions: <b>To here<\/b> - <a href="javascript:fromhere(' + i + ')">From here<\/a>' +
    '<br>Start address:<form action="http://maps.google.com/maps" method="get" target="_blank">' +
    '<input type="text" SIZE=40 MAXLENGTH=40 name="saddr" id="saddr" value="" /><br>' +
    '<INPUT value="Get Directions" TYPE="SUBMIT">' +
```

```
'<input type="hidden" name="daddr" value="' + point.lat() + ',' + point.lng() +
    // "(" + name + ")" +
'"/>';
```

(3) Get the marker location

Using geocoder.getLocations() function to get the current address of marker from google sever.

(4) Set street view

There is an option for clients to check streetview provided by Google server in our web system. This function is implemented by GStreetviewPanorama object in Google MAP API version. In order to display streetview in the infowindow of the marker, we use javascript open library “tabbedmaxcontent.js” to open a big window for streetview purpose.

```
map.openMaxContentTabsHtml(latlng, html, htmlsmall, tabs,
        {maxTitle: "Location more info",
        selectedTab: 'streetview'        });
```

4.4 Marker cluster to arrange markers

There are more than eight hundreds markers when we show all markers. It’s horrible when all the markers show at the same time when the zoom size is small, see Figure 4.3. We use marker cluster to manage markers. When zoom size is small, the markers will merge by themselves and show a large circle including the number of markers merged. Because there are fewer numbers of markers showed in map, the speed will faster than former method. And the markers are arranged well and the map is beautiful (Figure 4.4).

After importing javaScript library: markerclusterer.js, we use MarkerClusterer function to achieve marker cluster and variable mcOptions to design the zoom parameter.

```
mcOptions = { gridSize: 70, maxZoom: 15};
markerCluster = new MarkerClusterer(map, gmarkers, mcOptions);
```

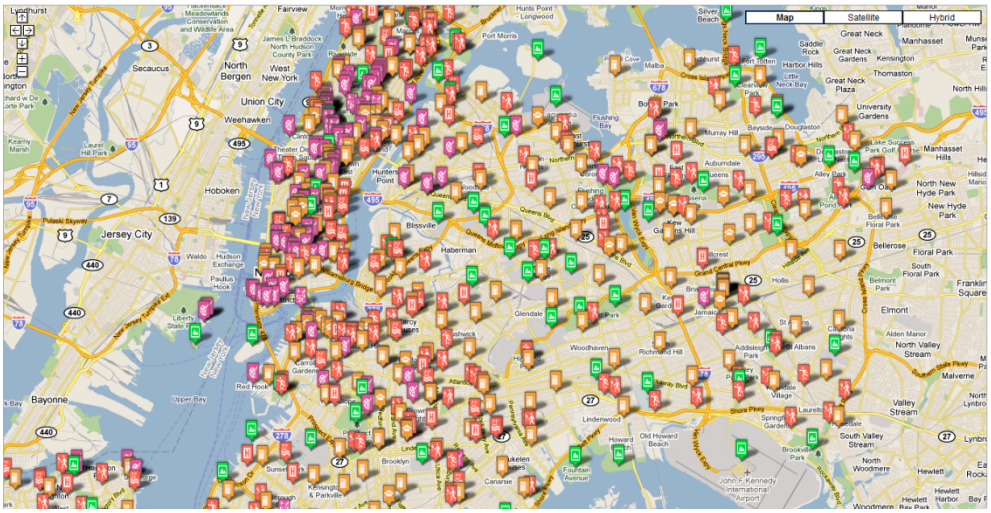


Figure 4.3: show all layers

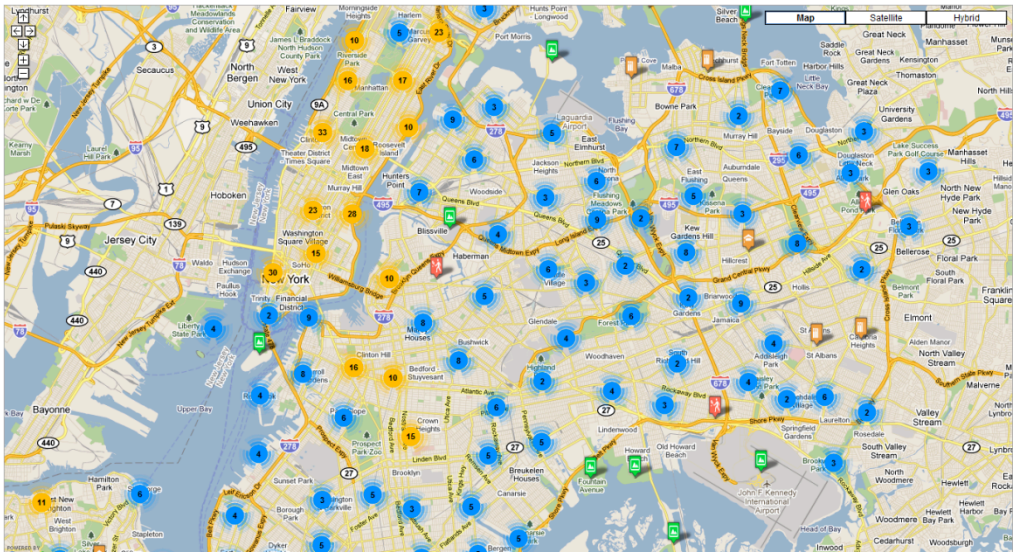


Figure 4.4: show all layers using marker cluster

5. KML overlay

GGeoXml object in Google map API v2 is generated by calling the following function:

```
geoXml = new GGeoXml("http://134.74.146.40/~wu/proj3/nypp.kmz")
```

When the button of “show nyc district” is clicked, nypp precinct overlay will be added to the map implemented by “map.addOverlay(geoXml)” in toggleMyKml() function, and use “map.removeOverlay(geoXml);” to remove KML overlay. The following Figure 5.1 shows the nypp precinct, the dark color means more car crash happen in this district.

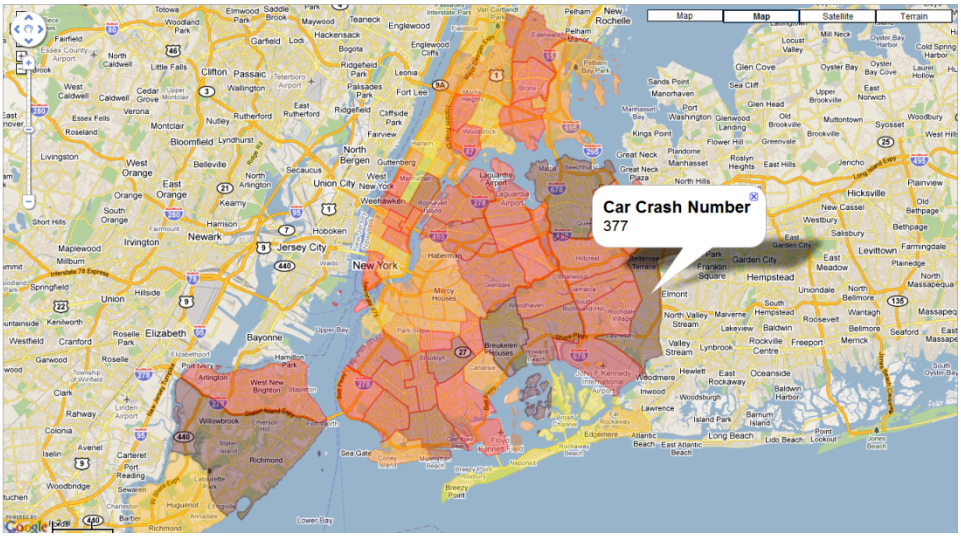


Figure 5.1: NYPP precinct and car crash

6. Location search by address and Zip code

Our website will provide clients with location search by location category and address or zip code function as the following figure 6.1.

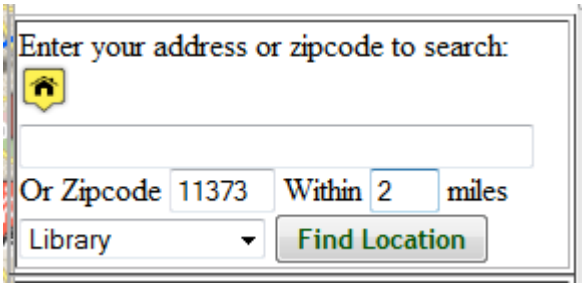


Figure 6.1 Location searching by address and zipcode

In the input text area, clients can submit address or zip code with distance together and select the category of location.

Because the geospatial data information of all locations are stored in the postgresql database, in order to implement search function, we use geospatial data in the database with column of “centergeom” , “xlat”, “ylong” and category to query.

We use the following SQL statement to query locations with the latitude and longitude of Google map.

```
SELECT courtnumber,xlat,ylong,category FROM nyclife where category= '{$category}' and
st_distance(sports.centergeom, ST_Transform(ST_GeometryFromText('POINT({$ylong}  {$xlat} )', 4326),2263) )
< {$distance}
```

Google map API will projection “4326” to store latitude and longitude, however our database use SRID “2263” to store geospatial data, in order to make data consistent between Google map API and PostGIS, we use **ST_Transform** in PostGIS to calculate distance between submitted address and locations in the PostGIS database.

In the next, we will explain the detailed methods to be used in our project for location search.

6.1 Location searching by address function

In Google map API v2, latitude and longitude information of map can be retrieved through GClientGeocoder object. In GClientGeocoder object, getLatLng(address:String, callback:function) will Sends a request to Google servers to geocode the specified address. If the address was successfully located, the user-specified callback function is invoked with a GLatLng point.

After retrieving GLatLng point from Google server, we will pass latitude, longitude and category parameters to php file “filter.php”. PHP file will query PostGIS database and return back results with XML format. Then, we use GXml object in Google map API to parse XML.

At last, the Google map API v2 will generate Gmarker ojects for each location returned by PostGIS server and add markers as overlay on the map.

Figure 6.2 is the example of location searching with address of “160 Convent Avenue New York, NY” and with category of football.

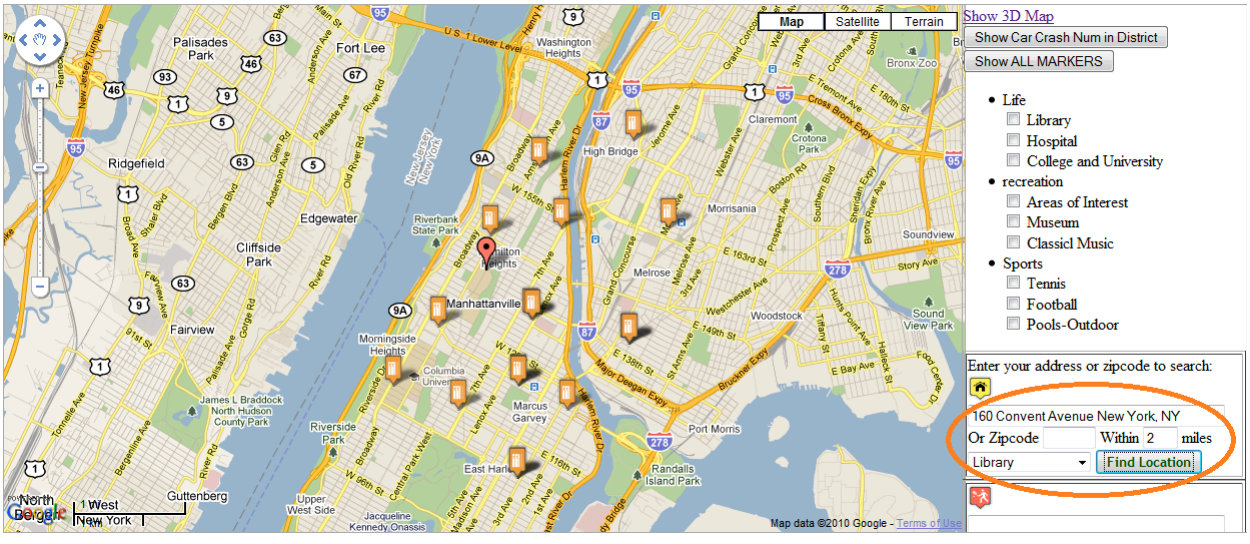


Figure 6.2 Location searching with address of “160 Convent Avenue New York, NY” for category of football

6.2 Location searching by zip code

Different from the method used in address searching as previous section described, our project will use Google Map Geocoding API web service to get latitude and longitude from specified zip code.

The Google MAP API web service use in our project as the following example:

http://maps.Google.com/maps/geo?output=csv&key={$mapsApiKey}&q=11373

In this example, Geocoding API requests a CSV response for a query with zip code “11373”.

The Google web service will return response with an array including latitude and longitude as the following:

```
200,5,40.7290878,-73.8766212
```

Then, we will pass latitude, longitude and category parameters to php file “zipcode.php”. PHP file which are used to query PostGIS database and return back results with XML format. Then, we use GXml object in Google map API to parse XML.

At last, the Google map API v2 will generate Gmarker ojects for each location returned by PostGIS server and add markers as overlay on the map.

Figure 6.3 is the example of location searching with zip code (11373) and distance (2 miles) and with category of tennis.

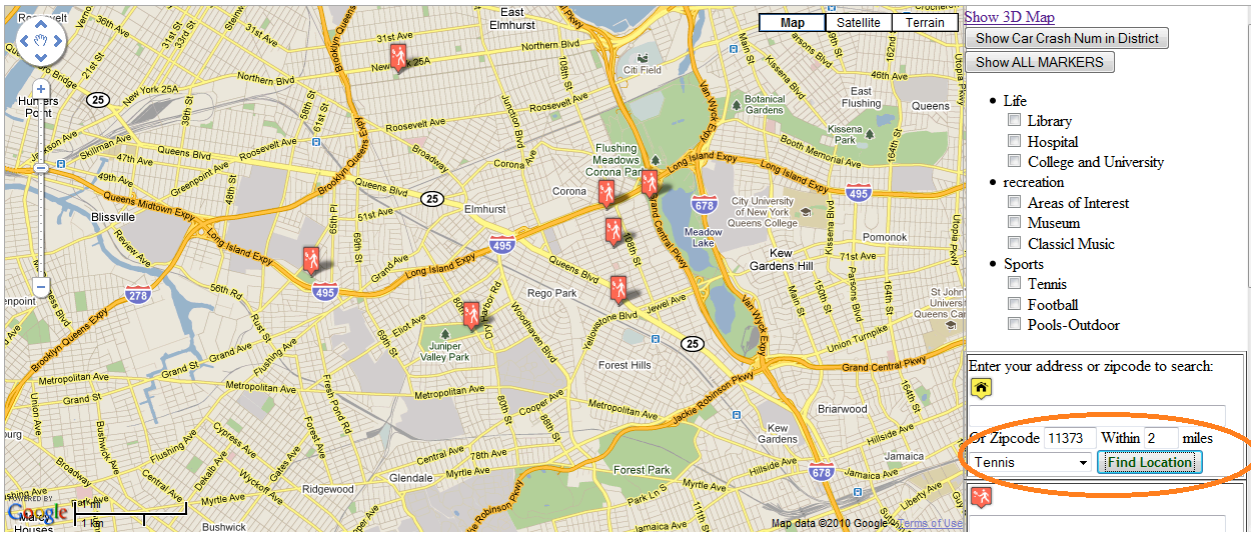


Figure 6.3 Location searching by zipcode 11373 and distance 2 miles for tennis category.

7 Get Direction by Google Map Window and Local Window

Our project provides two solutions for location direction. The first solution is to open direction in a new Google map window. The second solution is to display direction in the local window. Next, I will introduce two solutions separately.

7.1 Get direction by Google map window

When clients clicked a marker of location, an InfoWindow will be displayed around the marker as the figure 7.1.

In this solution, our website uses the lat() and lng() method of Gmarker object to get latitude and longitude separately and pass latitude and longitude to Google server.

Because PostGIS does not contain address information for each marker, we use geocoder.getLocations() function to get address of the latitude and longitude from Google server and add the address into the infowindow of marker.

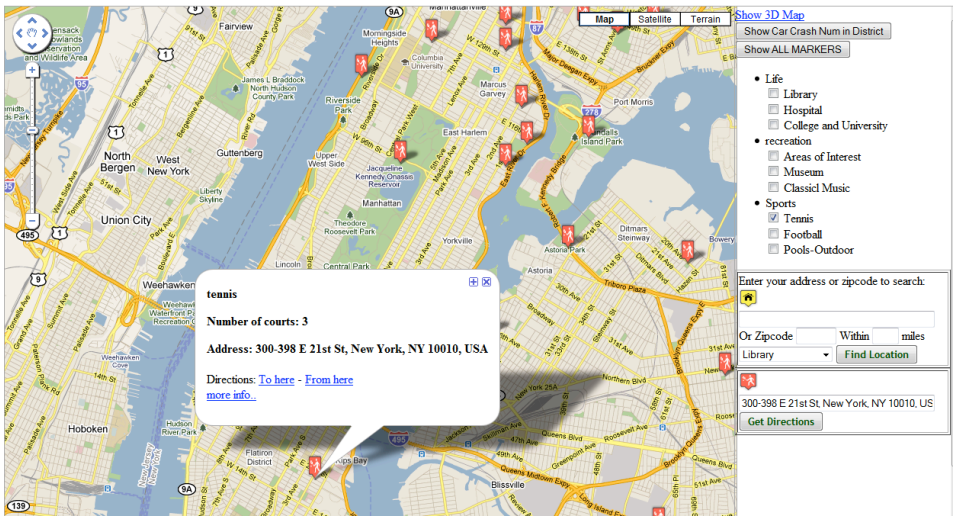


Figure 7.1 Infowindow of location marker

In the infowindow, two options of “From Here” and “To here” are listed and used to users to submit source or destination address. A submitted address is displayed as figure 6.2.

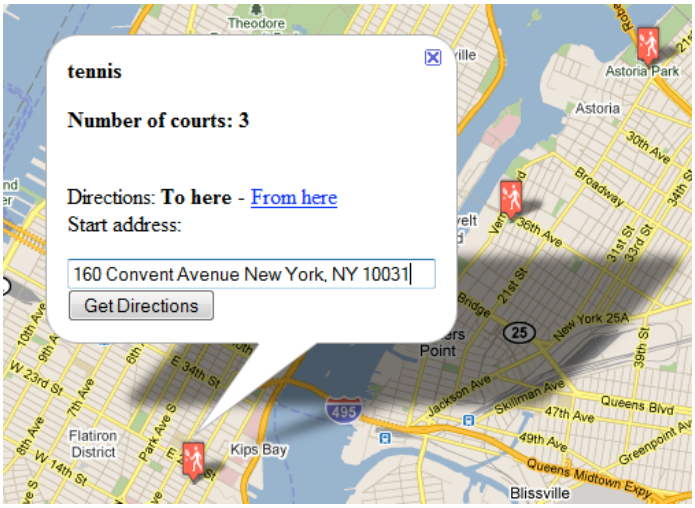


Figure 7.2 Get direction by a new Google map window

After clicking “get directions” button, our website will redirect to a new Google map window and display the direction in the new window.

In this solution, Google map web service of getting map will be used to open a new Google map window.(Figure 7.3)

The code used for this purpose as the following:

Start address:<form action="http://maps.Google.com/maps" method="get" target="_blank">

End address:<form action="http://maps.Google.com/maps" method="get"" target="_blank">

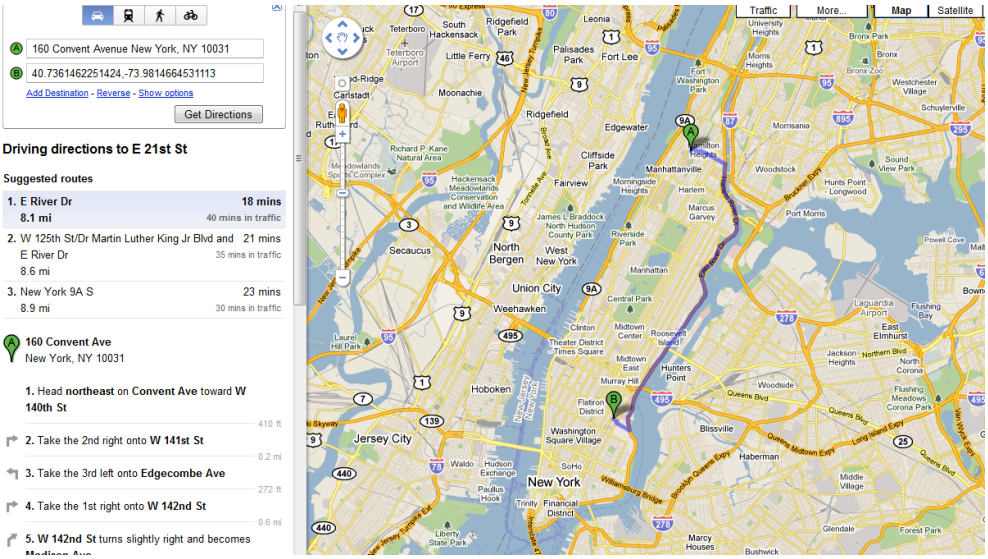


Figure 7.3 Get direction result

7.2 Get direction by local window

Sometimes, new Google map window is inconvenient during location searching. Clients have to go back to our website and check location again in case clients need to search other locations. In this case, we provide another solution to display direction.

By clicking the marker of location, the address of location will be copied into the lower address input area automatically. After clicking “get direction” button, a direction information will be display in the local window.

In this solution, GDirection object in Google map API v2 will be used to get direction from Google server.

The getDirection() function in “life.html” is used for this purpose.

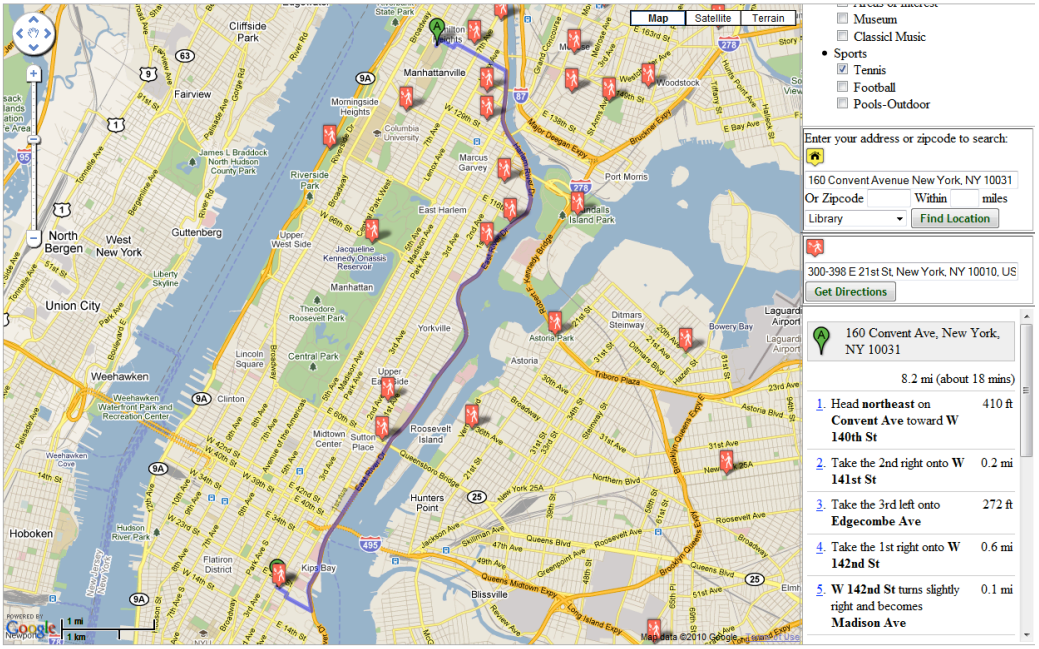
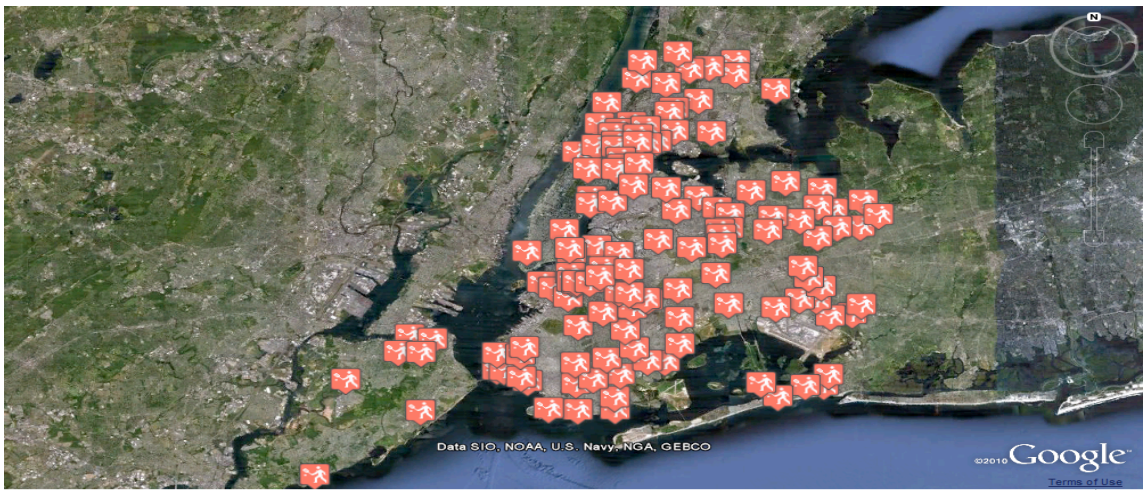


Figure 7.4 location direction displayed in the local window

8 Implement 3D map using Google Earth

8.1 Google earth view of tennis courts

Our website will display tennis courts by Google Earth 3D Map. First, you should install Google Earth plug-in and then you can view Map information by 3D.



The first step, Loading the Google Earth API

We have to replace API Key with your own key like this on java script head.

```
<script src="https://www.google.com/jsapi?key=ABCDEF"></script>
```

The tag's SRC points to a JavaScript file with a single method, google.laod, used to load individual Google APIs. Inside a new <script> tag, call: google.load("earth", "1");

This tells Google to load the earth module into the google.earth namespace, and specifies version 1. To specify the latest test version of the API, specify “1.x”.

The second step, creating a container for the plugin

The Google Earth Plugin is loaded into a DIV element with a unique id. Add the DIV to your page's <body> section:

```
<div id="map3d" style="height: 400px; width: 600px;"></div>
```

The third step, creating initialization functions

You attempt to create a new instance of the plugin. The first function looks like this:

```
function init() {
          google.earth.createInstance('map3d', initCB, failureCB);
         }
```

google.earth.createInstance here shows three options: the DIV element into which the instance should be added, the function to call when success is returned, and the function to call if a failure is returned. The success callback function will contain all of the code required to set up your 'first run' experience - all of the objects and views that will first appear when your plugin instance is loaded in the browser. This function must contain the GEWindow.setVisibility method, setting the window visibility to true, so that the plugin is visible inside its DIV:

```
function initCB(instance) {
                ge = instance;
                ge.getWindow().setVisibility(true);
               }
```

The failure callback can contain any code to deal with a failure to create the plugin instance. The error code is passed to the callback function, and can be repeated on the page or as an alert, if desired. Throughout this guide, the error callback function is left empty:

```
function failureCB(errorCode) { }
```

The final step, calling the initialization function when the page is loaded

The google namespace includes the setOnLoadCallback() function, which calls the specified function once the HTML page and requested APIs have been loaded. Using this function ensures that the plugin is not loaded until the page's DOM is completely built out.

```
google.setOnLoadCallback(init);
```

8.2 Additional Information about Google Earth 3D Map

Google Earth displays satellite images of varying resolution of the Earth's surface, allowing users to see things like cities and houses looking perpendicularly down or at an oblique. The degree of resolution available is based somewhat on the points of interest and popularity, but most land is covered in at least 15 meters of resolution. Google Earth allows users to search for addresses for some countries, enter coordinates, or simply use the mouse to browse to a location.

For large parts of the surface of the Earth only 2D images are available, from almost vertical photography. Viewing this from an oblique angle, there is perspective in the sense that objects which are horizontally far away are seen smaller, but of course it is like viewing a large photograph, not quite like a 3D view. Because the three-dimensional map functionality is built in to the same map control as the standard two-dimensional maps, you do not need to change how you build your map-enabled Web site. You can program against the new 3D APIs in the same way that you use the 2D APIs. On the programming side, including the 3D maps in your existing sites requires very little additional effort.

For other parts of the surface of the Earth 3D images of terrain and buildings are available. Google Earth uses digital elevation model (DEM) data collected by NASA's Shuttle Radar Topography Mission (SRTM). This means one can view the whole earth in three dimensions. Since November 2006, the 3D views of many mountains, including Mount Everest, have been improved by the use of supplementary DEM data to fill the gaps in SRTM coverage.

Many people use the applications to add their own data, making them available through various sources, such as the Bulletin Board Systems (BBS) or blogs mentioned in the link section below. Google Earth is able to show all kinds of images overlaid on the surface of the earth and is also a Web Map Service client. Google Earth supports managing three-dimensional Geospatial data through KML. Google Earth is simply based on 3D maps, it has the capability to show 3D buildings and structures such like bridge, which consist of users' submissions using Sketch Up, a 3D modeling program software. In prior versions of Google Earth (before Version 4), 3D buildings were limited to a few cities, and had poorer rendering with no textures.

Recently, Google added a feature that allows users to monitor traffic speeds at loops located every 200 yards in real-time. In version 4.3 released on April 15, 2008, Google Street View was fully integrated into the program allowing the program to provide an on the street level view in many locations.

9 Conclusion

In this project, we use lots of Google Map API tools: markers with info windows, address decoding by latitude and longitude and reverse address decoding by address, getting directions, location searching sending KML files to Google Maps, adding custom icons, Marker Categories, Marker Cluster, Google streetview , Google earth view etc.

Map loads data from PostgreSQL database system through .php file. It shows different layers of NYC life information, and users can get information when click markers and get address and direction by query.

10 Reference

1) http://code.Google.com/apis/maps/documentation/javascript/v2/reference.html
2) http://www.nyc.gov/html/datamine/html/data/geographic.shtml
3) Google Map API Tutorial: http://econym.org.uk/gmap/
4) http://itouchmap.com/?r=googleearth
5) http://code.google.com/apis/earth/
6) http://code.google.com/apis/earth/documentation/#using_the_google_earth_api