On-the-fly Map Generator for Openstreetmap Data Using WebGL

Advisor: Dr. Chris Pollett Committee Members: Dr. Chun and Dr. Khuri

> Presented By Sreenidhi Pundi Muralidharan

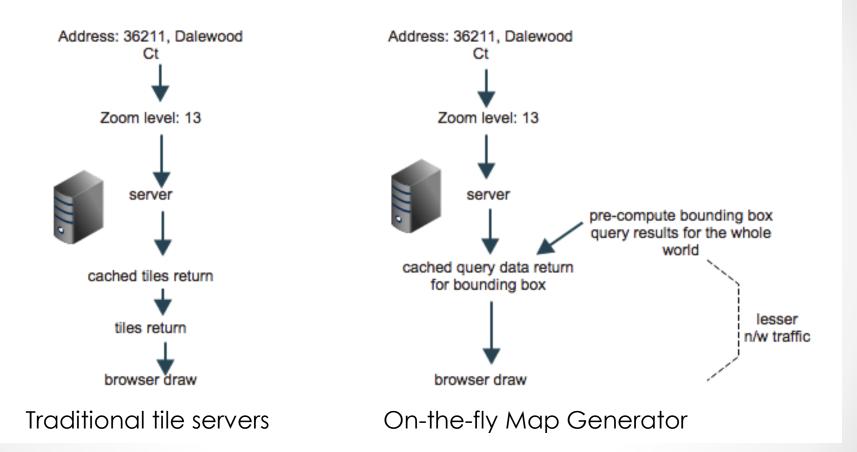
Agenda

- Project Idea
- History of online maps
- Background about Online Maps
- Technologies Used
- Preliminary Work
- Design of On-the-fly Map Generator
- Implementation
- Experiments and Results
- Questions to Ask!
- Conclusion
- Future Work
- Demo
- Q&A

Project Idea

- Render navigational browser maps using WebGL for Openstreetmap data
- Comparison of HTML5 graphics technologies to render maps in browser vs. traditional approach of rendering map tiles
- More on this later...

Project Idea (contd...)



Over the next few slides, we will explain our idea in detail as well as traditional approaches.

History of online maps

- Evolution of online maps
 - First online map was in 1989 Xerox PARC Map viewer, produced static maps using CGI server, written in Perl
 - Mapquest 1996, was used for online address matching and routing - navigation
 - Openstreetmap 2004, crowd-sourced
 - Google Maps 2005, used raster tiles, data loading was done using XHR requests
 - Other maps are being used now Yahoo, Bing, Apple maps

Background about Online Maps

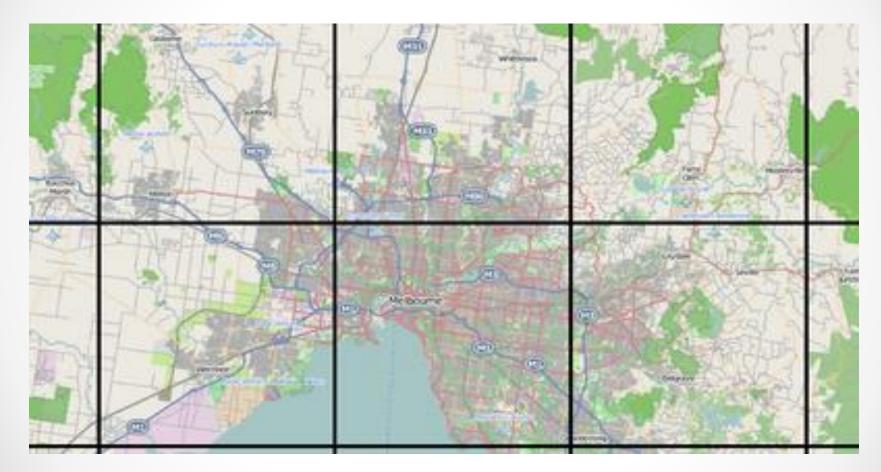
- Static Maps
 - Were used initially, with no interaction
 - Example: Initial topological survey maps, weather maps

Dynamic Maps

- User interactivity
- Traditionally used tile servers that store tiles 256 X 256 PNG images
- Image tiles for each zoom level, for all over the world
- Created beforehand, but rendered on-demand, fetched from a DB/tile server

- Why tiling system?
 - Initial mapping techniques used WMS an oldest standard
 - Each request single large image
 - Continuous map image at street level would be millions of pixels wide
 - Rendering pre-generated tiles on the browser is computationally faster than having to compute, generate and render map images
 - So maps tiles are fetched from a map server
 - Tiles cache efficiently, load progressively and are simple to use

- Raster and Vector data maps- the differences
 - Raster Maps-image tiles are used, mostly represented as z/x/y coordinate system
 - Storing tiles for all zoom levels, all over the world is cumbersome.
 - Rebuild the whole image tile if a certain topological feature changes
 - Rendering tiles now faster with high speed systems and caching
 - <u>http://www.opencyclemap.org</u>



Raster Maps - Tiled

- Vector Data data in the form of points, lines, etc.
 - Data represented in its original form
 - Still, tiling is used in almost all of today's vector-based maps
 - The need to re-render or rebuild the topology when data changes – still faster than having to regenerate a whole raster image tile.
 - Elevations can't be effectively represented with vector geometry



Mapzen's vector data tiling

- Initially, rendering these pre-generated tiles was time-consuming.
- Latest maps cache map tiles called seeding
 - Reduces communication between server and client
 - Disadvantages purging cached tiles is cumbersome when map data often changes

- Present day maps use a combination of both vector data on top of raster maps
- Nowadays, can draw on browser canvas
- GPU acceleration on browser
- Hope do more on browsers than traditional way



Vector Data on top of Raster Maps

Technologies Used

- OSM open source, crowd-sourced
- OSM Data Format
 - Primitives Tags, Nodes, Ways, Relations
 - Data storage format:
 - Several dumps available, with mirrors
 - Common ones are XML and PBF. Others include SHP, SHX, DBF, etc.
- Postgresql Database
 - o Open-source-relational
 - PostGIS extension that helps store geospatial data
 - pgAdmin 3 GUI tool for Postgres

WebGL - Introduction

- Web Graphics Library, designed and maintained by the Khronos Group
- Rendering of two and three dimensional objects on the web browser
- No external plugin required
- Rendering of effects and animations of objects on top of a HTML5 canvas
- Can mix WebGL with normal JavaScript code
- Several libraries available now for extended functionality support facilitates easy lighting, fixing camera/viewing angle
 - E.g., include three.js, CopperLicht, etc.
- Used in gaming industry for rendering 3D game designs, aerospace industry, particle analysis, rendering human face models

WebGL – shaders

- Everything is in terms of vertices of a geometry
- No fixed pipeline to manipulate vertices out of the box
- WebGL offers programmable pipeline we are responsible for manipulating vertex positions, colors, etc.
 - This is done by GLSL (OpenGL shading Language)
 - Has C-language bindings
- Shader source code is either defined in HTML <script> tags or as variables in JavaScript

- Two Main types of shaders
 - Vertex shader
 - Operate on vertices specified by buffers
 - Passed to a variable called gl_Position specifies the position of each vertex
 - Fragment shader
 - Defines the color of each pixel, hence each vertex
 - R,G,B,A values defined
 - Can interpolate different colors between vertices

- Compiling and linking C-like shaders
 - Mostly repeating boilerplate for almost all WebGL programs
 - Shader functions are called in WebGL and compiled using the WebGL context
 - The compiled program is linked and the program is made available
- WebGL Buffers
 - Send vertex information to the GPU
- Draw Objects
 - Use draw() calls to draw
 - Draw either as lines, points or other geometry modes

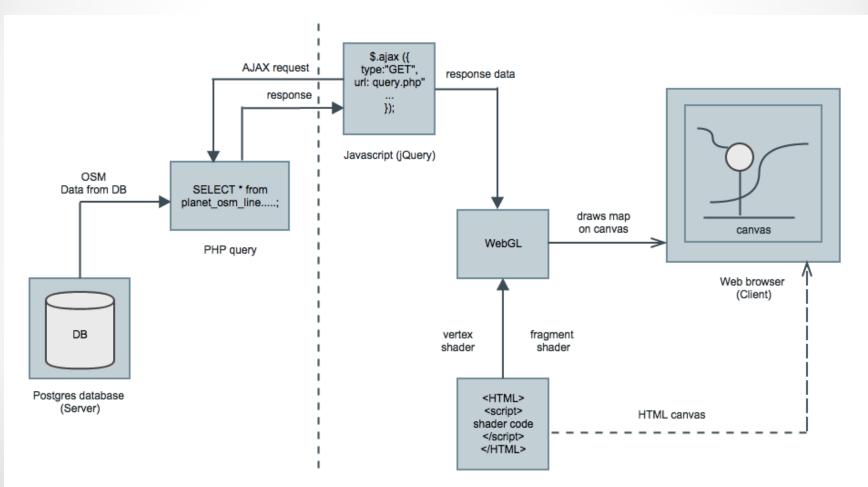
Preliminary Work

- Database Setup
 - Installed Postgres.app (PostgreSQL DB server)
 - Added extension "PostGIS", for storing geospatial data
 - PostGIS adds a table spatial_ref_sys by default
 - Added Google's spherical mercator projection, 900913 to spatial_ref_sys
 - Downloaded a .pbf file for North America, imported to Postgres
 DB
 - Installed pgAdmin 3 a GUI developmental platform for Postgres

Preliminary Work (contd...)

- Local Server Setup
 - Written and tested in Mac OSX, which has Apache and PHP bundle
- A machine with high-end Graphics Processing Unit
- A browser that supports WebGL

Design of On-the-fly Map Generator



Implementation

- Idea get current positional coordinates
- Create bounding box
- All geometry within this bounding box is drawn

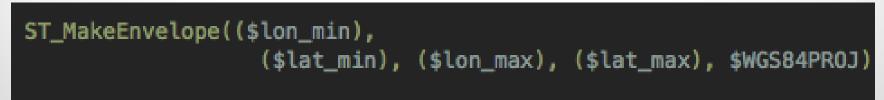
HTML5 canvas

- Defined using the <canvas> tag
- Canvas width and height is defined

Viewport

- Rectangular viewport
- WebGL defines the placement of vertices and colors
- Database Query
 - Two separate queries, one for drawing polygons(buildings) and the other for drawing lines(roads)
 - Their results are stored in separate arrays

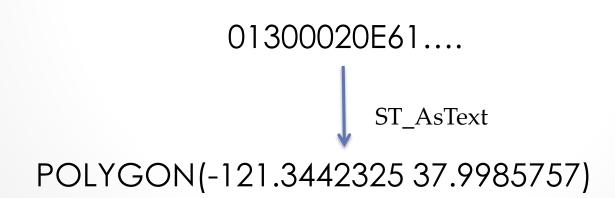
- Use of spatial geometry constructors PostGIS extension provides them for spatial geometry
 - For creating a bounding box, used ST_MakeEnvelope()
 - Creates an invisible rectangular bounding box specified by the minimums and maximums of latitudes and longitudes
 - The spatial referential ID (SRID) is used along with this to tell what type of projection is used.



- ST_Transform()- transforms the coordinates into a system of points that can be understood
 - The coordinate points are in the form of long, unreadable strings
 - SRID 4326 is used which transforms coordinate data into latlong coordinate points
 - Example: ST_Transform(way, 4326)
- ST_AsText() converts vector data into Well-Known Text (WKT) representation

Bank of America's coordinate geometry 0130002031BF....

ST_Transform, SRID = 4326



- The resulting data is sent to WebGL as a set of object arrays using json_encode()
 - Each object contains a set of latitude-longitude points for the geometry to be drawn
- Convert objects into a format that WebGL understands- pixels
 - Use mathematical formulae, which is written in JavaScript

$$x = \frac{\lambda + 180}{360}$$
$$y = \frac{1}{2} \ln(1 + \sin \varphi * \frac{1}{1 - \sin \varphi})$$

Lat-long points Lon: -121.3442325 Lat: 37.9985757

Converted to pixel (x,y)

X:41.254921 Y: 99.1307931

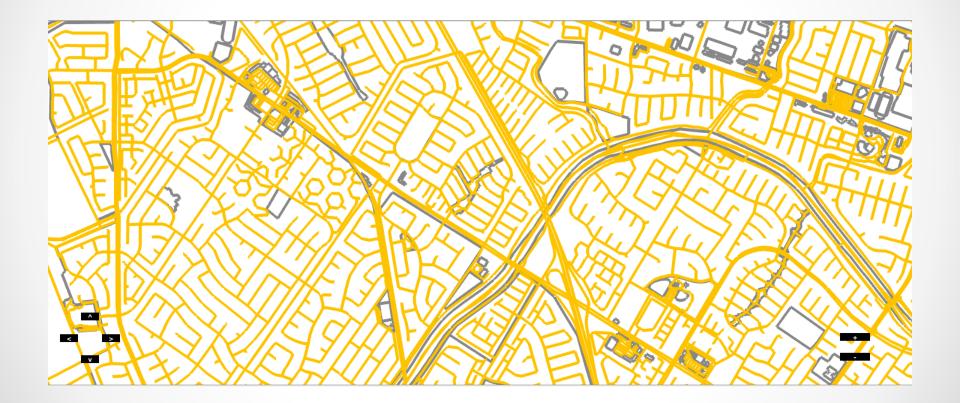
The Map

- Scale pixel coordinates according to zoom levels
- Vertex buffer now has the vertices to be drawn
- Used "index buffers" that were helpful in rendering "nothing" between the end point of the first polygon and start point of the next polygon
 - Index buffers are mainly used to hold indices for each polygon drawn
- Used different colors that specify polygons and lines
 - These colors were defined in the color buffer

- Now draw!
 - Use drawElements() call of the WebGL

gl.drawElements(gl.LINES, indexArray.length, gl.UNSIGNED_SHORT, 0);

- gl.LINES does not produce continuous lines, but with index buffers defined, it connects the vertices of a geometry
- Thus, two dimensional geometry is drawn



Final Map drawn for OSM data using WebGL

Zoom levels

- HTML <input> tags with type = button
- Zoom-in shows more finer details
- Zoom-out shows an overview of all data drawn



Map zoomed out at zoom level 12

- Panning around
 - Pans around- top, bottom, left, right, created with HTML <input> tag
 - Increments/decrements current positional coordinates by a "step" value
- Resizing the map
 - Map canvas resizes itself when the browser window is resized



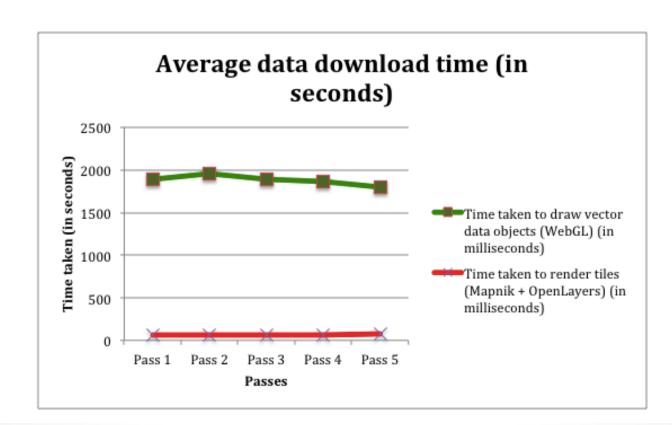
Experiments and Results

- Set up my own tile server on my local machine
 - Generates tiles according to zoom levels and bounding box coordinates specified
 - Used "mapnik" a Python-based open source toolkit for rendering maps
 - Installed mapnik tools generate_xml.py, generate_image.py, generate_tiles.py
 - Tested with: Zoom level 13, bounding box, closer to Newark, CA
 - This script generates tiles is a special hierarchy of folders, identified by the zoom level
 - Used OpenLayers to render these tiles generated
 - Finally, tiles are reassembled and rendered in client's browser

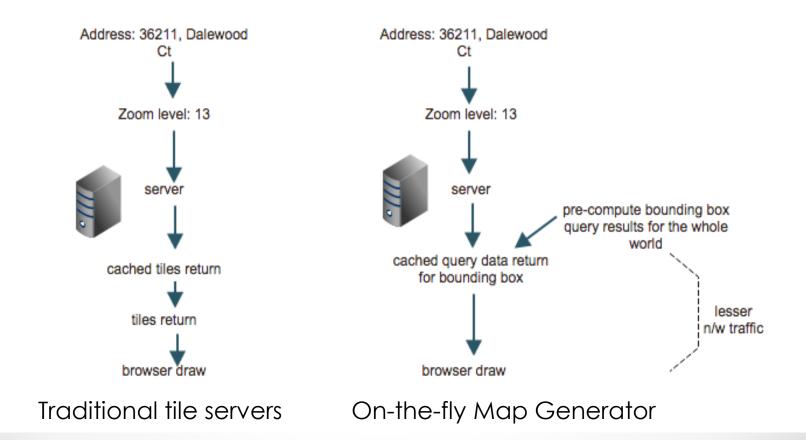
Experiments and Results (contd...)

- Timing tests
 - Test bench used
 - Mac OS X Yosemite
 - 1.8 GHz dual core CPU
 - 4GB memory
 - a high speed Intel HD Graphics 3000 GPU (Boost) MHz with 12 unified pipelines
 - Google Chrome version 47.0.2526.73 (64-bit), with WebGL 1.0 enabled

Type of data	Pass 1	Pass 2	Pass 3	Pass 4	Pass 5
download					
Time taken	1.88 s	1.95 s	1.89 s	1.86 s	1.79 s
for the query	1.000	1	1.07 0	1.000	
to execute					
and get back					
data (query					
time)					
Time taken	3.86 s	3.88 s	3.91 s	3.85 s	3.89 s
to generate					
tiles					
(Mapnik +					
OpenLayers)					



Real-world setting results are different



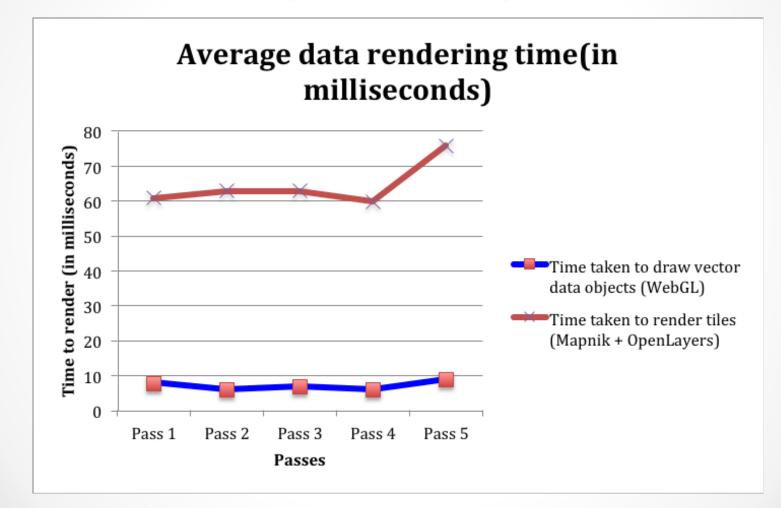
- The data returned initially was heavy
 - Converted the data to floats (because of the math involved!)
 - Truncated after the first 8 digits
 - Compressed data further using gzhandler gzip
- Resulting data was reduced from a whopping 5 MB (Newark, CA) to 684 KB.
- Did a comparison the tiles size and the queried data size

Table 6-2 Comparison of tile sizes versus queried data size, for bounding boxes

Places observed,	Size of tiles	Size of query data	
according to bounding			
boxes			
Fremont	762 KB	684 KB	
San Jose	2.4 MB	1.7 MB	

Table 6-3 Average Rendering times (in milliseconds) for both methods

Type of	Pass 1	Pass 2	Pass 3	Pass 4	Pass 5
rendering					
Time taken	8 ms	6 ms	7 ms	6 ms	9 ms
to draw					
vector data					
objects					
(WebGL)					
Time taken	61 ms	63 ms	63 ms	60 ms	76 ms
to render					
tiles					



Questions to Ask!

- Wait a minute! Isn't Google's MapsGL something similar?
 - Yes, but Google uses its own database for vector data
 - They have copyright issues, lets only API usage by developers
 - Can developers experiment?
 - It is still in "beta" stage
- Can WebGL be supported by all browsers?
- Does all hardware support WebGL?

Conclusion

- Pre-computing the bounding boxes and their query results will yield competitive results.
- Implementing this on a large scale, for the whole world might considerably reduce network traffic

Lesser data is sent to the browser

 In short, getting tiles (in terms of size) would be more than getting data from the query for bounding boxes.

Conclusion

- On-the-fly geometry rendering
- A novel way to generate online maps
- Bottleneck browsers without WebGL might not support this
- With evolving technology, this might not be a big problem
- Initial stages of project had Google maps underneath the geometry drawn

Future Work

- Separate colors for water bodies
- Labeling places and roads
- Differentiate between normal streets and freeways (different colors)
- Filtering feature search box for the user
- Extend to pre-computing bounding boxes for the whole world
- All this requires lots of experimentations, trial and error methods

Demo Time!



Thank You!