Altus - Vector Maps on Android

2/26/2014

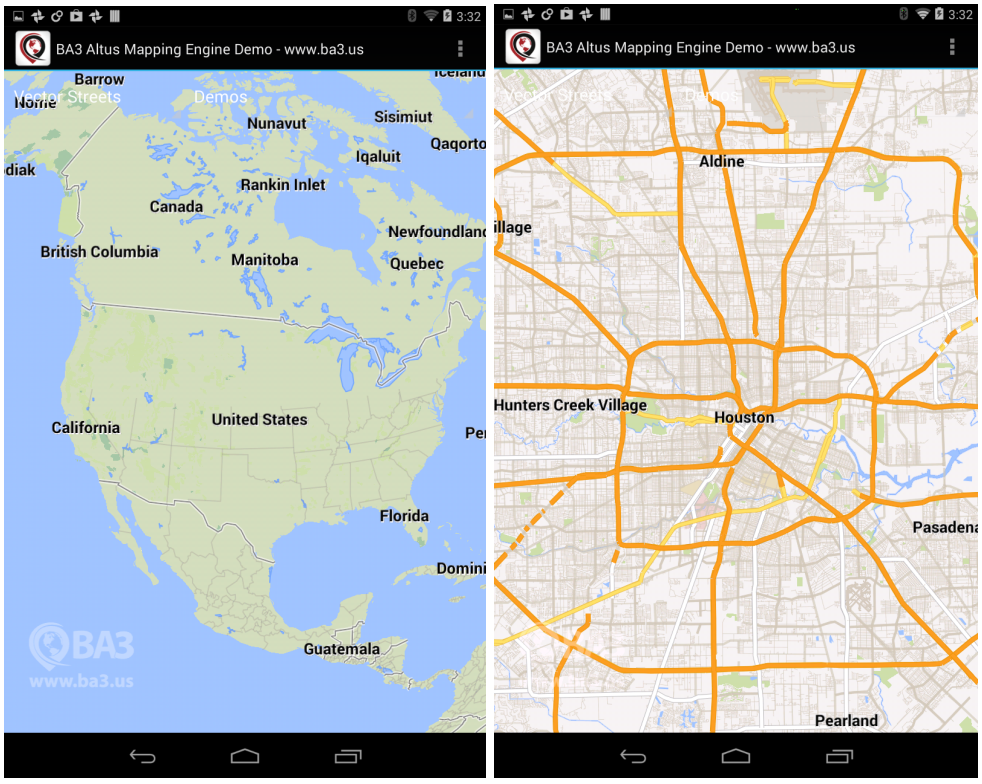
The Altus Mapping Engine provides several ways for you to put your vector data on the map. You can use:

- Dynamic Vectors - High-frequency updates / user interaction.
- Pre-Packaged Vector Maps - Styles may update, but data is fixed. Supports offline scenarios.
- Vector Tile Providers - Stream and convert data on the fly from a variety of sources, or read it from custom sources stored on the device.

Here we'll cover the the last two on Android.

Pre-Packaged Vector Maps - Streets and the World

This is a highly-optimized way to display a lot of vector data in Altus. It loads quickly and renders smoothly and is very compact.



NOTE: For reference see the file VectorWorldMapTest.java in the AltusDemo Android sample application.

In this example, we have used Altus MapShop to created a 13 MB highly optimized vector map from OpenStreetMap and NaturalEarth data sources.
(The map file is here http://mapserver.ba3.us/Drop24Maps/WorldVector/World_Style2.zip)

In addition, we are displaying a clustered marker map (generated with metool / AltusMarker) over the vector data which gives us some localized labelling that works in any screen orientation.

Note: Traditional street-labeling is a feature that we are still working on. It is at a prototypical stage and we are happy to discuss where we are headed with that feature with interested customers.

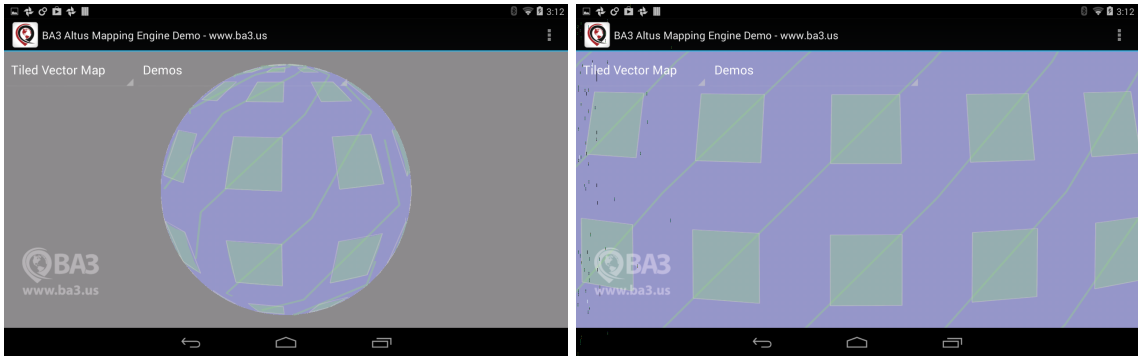
If you want to generate a map like this you can, we have several tutorials on how to set up MapShop to create a map like this:

- http://www.ba3.us/?page=pages/tutorials-mapshop&id=0
- http://www.ba3.us/blog/?p=529

But it is a rather difficult process and our tools were originally meant for internal use only. We put them out there due a few requests and we know it isn't easy to get the pipeline set up. With that said, we are working to simplify this system and provide something that will work more 'out-of-the box' for customers who don't want to spend the time to set up and maintain a PostGIS server.

Vector Tile Providers - Example Pattern

In this example, an algorithm supplied vector data information.



NOTE: For reference see the file TiledVectorMapTest.java in the AltusDemo Android sample application.

Here's the process:

- A virtual vector map is created and added as a layer
- Several line and polygon styles are added to the layer
- Altus then begins requesting vector data for the layer
- The tile provider provides a 'computed' set of geometry that represents a pattern.

The vector geometry is created by an algorithm when the mapping engine calls requestTile. In this case, the tile provider logic is relatively quick and executes and returns immediately. If this was a long-running function, this work would be done on another thread, and when completed, the thread would notify the mapping engine. This is demonstrated in the next sample.

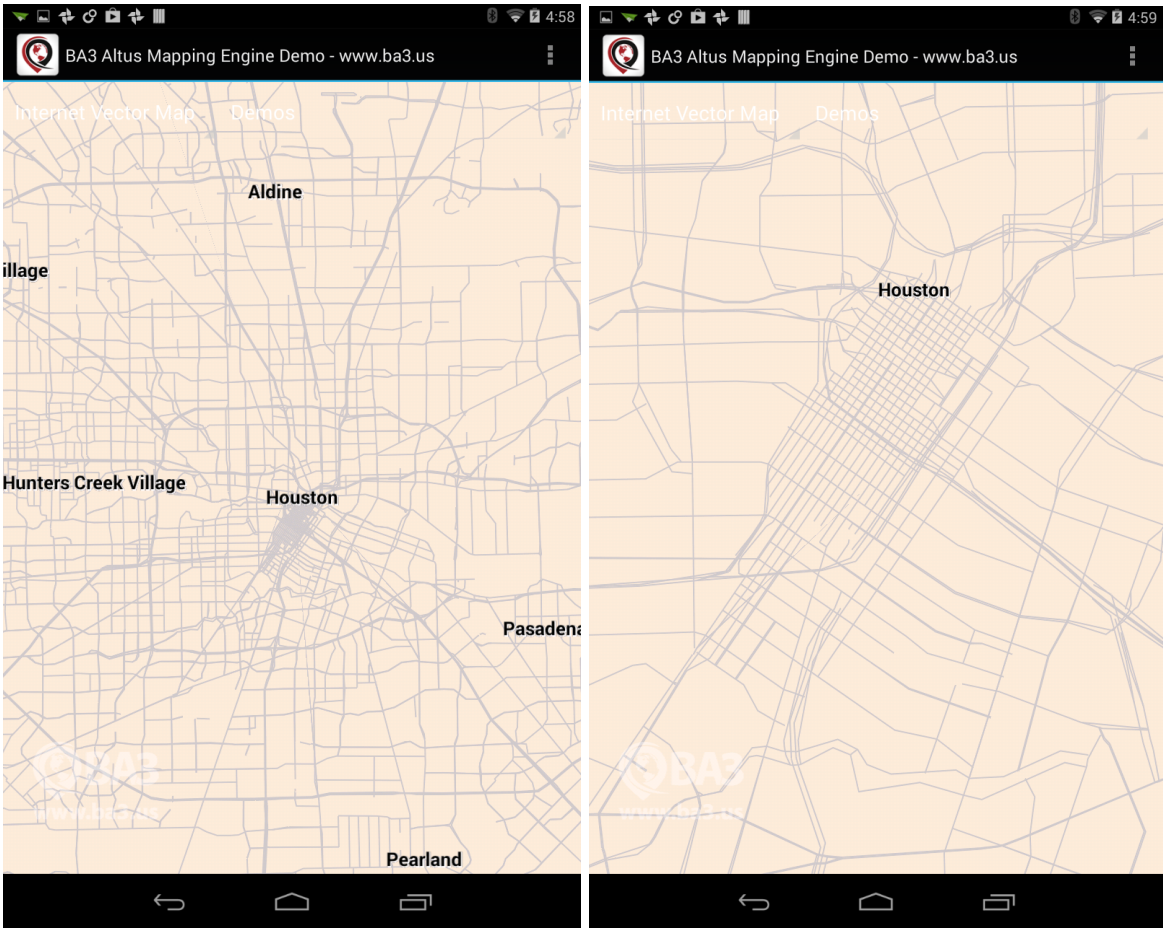
This is the function that serves up the looks like this:

```
@Override
          public void requestTile(TileProviderRequest request) {

               Location min = request.bounds.min;
               Location max = request.bounds.max;

               GeometryGroup geometry = new GeometryGroup();

               // add a line
               geometry.lines = new Line[1];
               geometry.lines[0] = new Line();
               geometry.lines[0].shapeId = 1;
               geometry.lines[0].coordinates = new Location[] {
                         min,
                         max
               };

               double width = max.longitude - min.longitude;
               double height = max.latitude - min.latitude;
               Location size = new Location(height, width);

               geometry.polygons = new Polygon[2];

               // add a polygon that covers the tile
               geometry.polygons[0] = new Polygon();
               geometry.polygons[0].shapeId = 2;
               geometry.polygons[0].shell = new Location[] {
                         min,
                         new Location(min.latitude + size.latitude,
min.longitude),
                         max,
                         new Location(min.latitude, min.longitude +
size.longitude),
                         min
               };

               // add a polygon that is in the center of the tile
```

```
                Location start = new Location(min.latitude +
height/4.0, min.longitude + width/4.0);
                geometry.polygons[1] = new Polygon();
                geometry.polygons[1].shapeId = 5;
                geometry.polygons[1].shell = new Location[] {
                          start,
                          new Location(start.latitude +
size.latitude/2.0, start.longitude),
                          new Location(start.latitude +
size.latitude/2.0, start.longitude + size.longitude/2.0),
                          new Location(start.latitude,
start.longitude + size.longitude/2.0),
                          start
                };

                // call this on the main thread to finish a tile
request
                mapView.vectorTileLoadComplete(request, geometry);
          }
```

Vector Tile Providers - Streaming JSON Data

This demonstrates a simple and flexible (though not efficient) way of serving up vector data over th internet.



In this example, as the mapping engine requests vector data from http://tile.openstreetmap.us/vectiles-highroad. It is served up as JSON text files from there and these are parsed on the device. The tile provider downloads, caches, and and parses JSON from the vectile-highroad example hosted by OpenStreetMap. We only have two styles here: a thin light line for roads, and a subtle color for the background. This example is only parsing lines from the source JSON data.

On most devices, this will run extremely slowly for 2 reasons:
1. This particular JSON data source has a lot of data in it, so it takes a while to download. We are only rendering a portion of the source data.
2. The JSON parsing takes a long time and consumes a lot of JVM resources.

While this inefficient way to transmit data of this nature, this sample demonstrates the ability to have a background worker thread download vector data from another server, convert it into native Altus geometry types and provide it to the mapping engine.

If you were going to do something like this in a real world scenario you would want to have a

more efficient transport, or you would want to greatly trim the data being streamed down to what is relevant.

If you have access to a PostGIS server, you could probably set up an end-point to issue queries that return data to your tile provider in a more optimal way. We believe a binary means is probably more efficient. Internally, we have some sample code that consumes TileMill2 protocol buffer-based vector tiles. If you are interested in how that works, let us know and we can fill you in. But even these in their current state these seem heavy-weight and best suited for generating raster tiles.

Notes on this approach:

- Altus supports the notion of being asked “Is this data still needed?” for streaming raster tiles. This API is demonstrated more-so on our iOS platform. It provides a way for the tile provider to have some work scheduled that it may cancel if the user pans away while other work is ongoing. This is not used in these samples.
- Out implementation for streaming vector data like this on Android presently only presently supports lines and polygons with an outer shell. Polygon holes are supported by the engine, but the JNI logic for converting the GeometryGroup is not fully implemented to support polygon holes at this time. If you need that support, please let us know and we can add it.

Projection Notes

Altus was originally built to cover the entire planet and we made some early technical decisions to have a different projection than spherical mercator above 75N and below 75S. That means, if you are storing your data in spherical mercator (slippy) tiles using an X,Y,Z scheme, then your data will only work to +-75 in Altus at this time without some form of conversion. In the case of raster imagery, Altus can resample spherical mercator tiles into its native projection. But with vector data, that functionality is not in place at this time.
This is not an issue if you have access to the raw vector data (i.e. in PostGIS) as you can query the subsets Altus requests based on the regions it asks for.