iOS7 Tutorial Series: Map Kit

Map Kit has some nice additions for iOS7 that build on the strong foundation present in the iOS 6 SDK. This post includes a sample application that demonstrates each new feature: you will find how to circle the world, enclose your pins in one view, find new ways to draw overlays, get directions, create a camera and take a photo with it.

Spanning the 180th Meridian.

You may have already noticed on your device that you can now pan around the world in an East-West direction so that you can view Hawaii and Japan at the same time. On iOS 7 you can programmatically show regions that span the 180th meridian as well. Here is an example of setting a MKMapView with a one thousand meter radius on the 180th meridian at the equator:

```
map.region = MKCoordinateRegionMakeWithDistance(
     CLLocationCoordinate2DMake(0, 180),
     1000.0 * 1000.0,
     1000.0 * 1000.0
);
```

Ensuring Visibility of Annotations

In a recent project we spent time calculating how large a region needed to be to display a group of pins on the map. With iOS7 there is a new method of MKMapView **showAnnotations:animated:** on MKMapView that positions the map to show the given annotations to the fullest extent possible. In the sample project you will see where pins were added to the map using this method:

```
[_mapView showAnnotations:@[pointRavens,pointSteelers,pointBengals,
pointBrowns] animated:NO];
```

I have found that this doesn't always work perfectly with small regions. Although the point might be on the screen, the pin associated with it might not. One way a developer can increase the size of the region slightly is by using the camera property of the map view to zoom out:

```
_mapView.camera.altitude *=1.4;
```

Additionally, annotation views in iOS 7 now track the map as it is manipulated by the user, which means they will stay upright, face the screen, and stay the same size. Apps that were doing lots of calculations attempting to keep the annotation in the right place as the map changes won't work anymore. With 3D maps the information is non-linear and you app will have difficulty interpolating the correct position on the screen, so let the Map Kit do the heavy lifting for you.

MKOverlayRenderer replaces MKOverlayView

MKOverlayView is now deprecated and replaced by the more efficient MKOverlayRender, which doesn't need to inherit from UIView. Accordingly, all subclasses and other classes that were associated with MKOverlayView have been replaced with renderer equivalents. For example, an MKPolygonView (deprecated) is now an MKPolygonRenderer. Here is a sample of using MKOverlayRenderer:

```
-(MKOverlayRenderer *)mapView:(MKMapView *)mapView rendererForOverlay:
(id<MKOverlay>)overlay{
     MKPolygonRenderer *polygonRenderer = [[MKPolygonRenderer alloc]
      initWithOverlay:overlay];
     polygonRenderer.strokeColor =[UIColor blackColor];
     polygonRenderer.fillColor =[UIColor redColor];
     return polygonRenderer;
}
```

<u>Overlays and Z-Indexing</u>
The view hierarchy of an MKMapView consists of seven layers of the map that are stacked on top of each other. At the bottom of the stack is the map grid, followed by the Base Map, MKOverlayLevelAboveRoads, Labels, MKOverlayLevelsAboveLabels, 3D Buildings and Annotation Views. You probably noticed that the two MKOverlayLevel values are the two oddly named layers, and they are related to a new iOS 7 feature of Map Kit. When creating your overlay you can now choose one of those two MKOverlayLevel options, and Map Kit will insert your overlay at that layer:

```
[_mapView addOverlay:parkPolygon level:MKOverlayLevelAboveRoads];
```

This new feature is very useful if you have a secondary overlay that is less important than the labels on the map, and would otherwise impair the user's use of the map if simply placed on top as in the iOS 6 SDK. A good example would be an app that overlays a background on all state parks. If the user happens to be at a park and wants to browse points of interest or navigate a hiking trail, the background overlay could obscure those map features if it is drawn at the highest layer.

<u>MKGeodesicPolyline</u>
MKPolyline will draw a straight line from one location to another on a 2D map; however, in the iOS7 SDK you can now draw a line that follows the earth's spherical dimensions with MKGeodesicPolyline. As a developer, swiching between the two behaviors is as easy as changing the class from MKPolyline to MKGeodesicPolyline. There is not much difference between the two when showing short distances, but with long distances MKGeodesicPolyline looks like a line showing an arcing flight path.

```
geoPolyLine = [MKGeodesicPolyline polylineWithCoordinates:points
count:2];
```

<u>MKTileOverlay</u>
New in iOS 7 is the ability to replace areas of the map using MKTileOverlay. A new MKMapView property **canReplaceMapContent** allows you to choose if you want Apple Map data to be loaded and drawn. If this value is YES, you will need to provide your own tiles in a template string used to build URLs so the map view can locate

the map tiles it needs. Unless you are restricting your tiles to a small area, you will most likely want to store these on a webserver instead of including them in the app bundle. The MKTileOverlay coordinate system uses square grids with 0,0 at the upper left, although you could set **geometryFlipped** to YES and have the origin in the lower left instead. Zoom levels grow by powers of two, so at zoomLevel zero there is one square and for example at zoomLevel three there are 2^3 or 8 tiles in each direction. Since this grows exponentially you can see how much data you might need even for a small area. Below is an example of setting the tile overlay with a URL template:

```
NSString *template = @"http://.../tile?z={z}&x={x}&y={y}";
overlay = [[MKTileOverlay alloc] initWithURLTemplate:template];
overlay.canReplaceMapContent = YES;
```

<u>MKDirectionRequest and MKDirectionResponse</u>
Many mobile users rely on directions from their device and in iOS 7 developers can now provide them within the context of your app instead of switching to the Maps application. You start your request with a source and destination. Other options include alternate routes, transport type and time of departure/arrival.

```
MKDirectionsRequest *request = [[MKDirectionsRequest alloc] init];
    request.source = source;
    request.destination = destination;
    request.requestsAlternateRoutes = YES;
    MKDirections *directions = [[MKDirections alloc]
initWithRequest:request];
    [directions calculateDirectionsWithCompletionHandler:
     ^(MKDirectionsResponse *response, NSError *error) {
         if (error) {
             NSLog(@"Error is %@",error);
         } else {
             [self showDirections:response];

         }
     }];
```

The MKDirectionsResponse provides a plethora of information that includes an array of MKRoutes, each of which contain an array of MKRouteStep so that you can provide each step of directions to get your user to their location. By putting the MKRoute and MKRoute steps in an array, it makes it easy to display the information in a table or display the information on a map as in:

```
   for (MKRoute *route in _response.routes) {
  [_mapView addOverlay:route.polyline level: MKOverlayLevelAboveRoads];
   }
```

Results may be updated frequently as a user moves with their device so don't cache the results for long.

With a similar feature in the Google Maps API, Google imposes a limit to the number of requests your app can have in a day. Apple has chosen to go a different direction and rate limit each device instead., which means there are no per-app or per-developer usage limits. A device may be throttled if it reaches high usage, which is intended to restrain apps that would be doing something like recursively making requests.

3D Maps and MKMapCamera

You likely have seen the 3D perspective in the Maps app, that uses a two finger rotation to rotate the map and a two finger vertical pan to change the pitch in the map. In the simulator you can also rotate with option+drag in a circle. For pitch use option+shift+drag vertically, although I've had more success flicking vertically instead.

To take full advantage of 3D with iOS7 you want to use the MKMapCamera, which allows you to programmatically adjust the same values available to the user through gestures. If you have never worked with cameras in a 3D environment I might recommend downloading Alice, educational software in a 3D environment, for free from Carnegie Mellon University. It's a fun way to learn about 3D viewing in programming. An MKMapCamera has four properties:

1. A **centerCoordinate**, which is a location on the ground.
2. The **altitude** the camera is from the ground.
3. The **heading** that is cardinal values from 0 to 360 with 0 pointing north.
4. The pitch that is angle the camera is tilted, with 0 meaning pointing straight down.

A straightforward way of setting up your initial camera is to use **cameraLookingAtCenterCoordinate:fromEyeCoordinate:eyeAltitude:** for example:

```
CLLocationCoordinate2D coordsGarage =
     CLLocationCoordinate2DMake(39.287546, -76.619355);
CLLocationCoordinate2D blimpCoord =
    CLLocationCoordinate2DMake(39.253095, -76.6657);
MKMapCamera *camera =[MKMapCamera
      cameraLookingAtCenterCoordinate:coordsGarage
      fromEyeCoordinate:blimpCoord eyeAltitude:100];
```

iOS 7 strongly encourages state restoration, and users will expect to return to the same map perspective upon returning to your app, so it recommended that you archive the camera whenever your application becomes inactive. For convenience the MKMapCamera implements NSSecureCoding.

Static Map Snapshots with MKMapSnapshotter

True to its name, MKMapSnapshotter allows developers to take a static image snapshot of a map view. First you need to set up the options with MKMapSnapshotOptions to set items such as the size, scale, camera, and mapType. Alloc and initWithOptions the MKMapSnapshotter. Call startWithCompletionHandler on the MKMapSnapShotter, which returns a MKMapSnapshot that contains an UIImage property, image. There is a sample of taking a snapshot and saving to the user photos in the attached project. The MKMapSnapshot will not include annotations or overlays, the developer must draw them afterword. The **pointForCoordinate** method can be used to translate the position of the annotation or overlay coordinate value to its respective location inside the image's coordinate space.

Closing

This post presents new features of MKMapView and related classes that help extend functionality to your iOS7 apps. Now you can take on the world with confidence, or at least display it in your app. If you have questions you may contact me at strohtennis @ gmail.