Technical Design Points

Wits:

* Wits will be supplied as a simple array for the purposes of this prototype (the real data model will be defined in Parse later on, but we don’t need to worry about it right now)
* Each wit will be specified as a latitude, longitude, altitude, type (which will be always “image” for the purposes of this prototype), owner (string representing a user), status (claimed or unclaimed) and content (the actual image); we may need to add other fields later, so you should create a class with appropriate init/get/set methods to manage a wit and its attributes programmatically
* For now wits will be specified relative to a location so that we can test them at our geolocation and you can test them at yours. For example:
  + we are at geolocation:

lat = 37.4280726; long = -122.1700931; alt = 72 ft

* + you are at geolocation:

lat = 49.84735; long = 24.02700; alt = 910 ft

* + wit w1 has a relative geolocation of:

lat = -0.0004686; long = 0.003144; alt = 297 ft

* + for us this w1 appears at absolute geolocation of:

lat = 37.427604; long = -122.1669495; alt = 396 ft

* + for you this w1 appears at absolute geolocation of

lat = 49.8468814; long = 24.030144; alt = 1207 ft

* The main challenge is to translate wit coordinates into the app’s coordinate system described below. Based on these wit coordinates the app will determine:
  + is this wit here and visible in viewFinder; if yes - we need to determine where to put this on the screen in x, y, z coordinates
  + is this wit here but NOT visible; if yes - it is a green witMarker and we need to determine where to put the marker around the edges of the screen pointing to its location off screen
  + is this wit near (and by definition it is a witMarker only and NOT visible); if yes - it is a yellow witMarker and we need to determine where to put the marker around the edges of the screen pointing to its location off screen

Coordinate System:

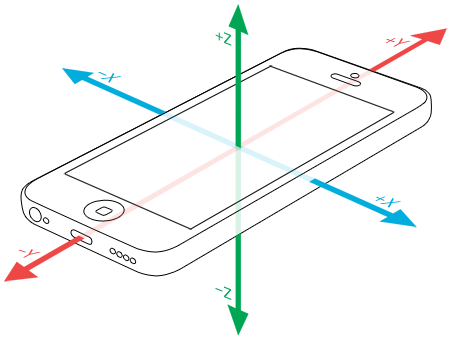
* The following coordinates need to be obtained from iPhone’s sensors:
  + user’s geolocation (latitude, longitude, altitude)
  + heading (north, south, east, west)
  + gyroscope readings (pitch, roll, yaw; i.e. which way is the iphone pointing)
  + accelerometer readings
  + iPhone focal length is between 29mm to 35mm depending on model. We need to translate that into z coordinate, i.e. how far iphone sees

All of these can be obtained using CoreLocation (<https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/LocationAwarenessPG/CoreLocation/CoreLocation.html> )

and CoreMotion (<https://developer.apple.com/library/ios/documentation/CoreMotion/Reference/CoreMotion_Reference/>) libraries;

This is a great explanation about using iPhone’s sensors to construct a coordinate system - <http://nshipster.com/cmdevicemotion/>

* Based on all of these we need to calculate “iphone’s field of vision” in 3D space, i.e below



x, y, and z axes extending from -1 to 1; with user’s position being center screen and having coordinates of 0,0,0

Accuracy:

* We are interested in maximum accuracy, which may be 1-2 meters or better but could be as poor as 10-20 meters; We may choose to use assisted GPS and/or iBeacons to deal with lack of accuracy in certain locations, but we will not worry about implementing this functionality right now; We will also not worry about locating wits indoors for now.
* User’s geolocation in the coordinate system described above will need to be updated continuously but while being aware that such updates are computationally expensive and reduce battery life, so we may be looking for optimization points here.

Displaying Wits:

* We will not be using markers or marker-based approaches, such as OpenCV, Vuforia, Wikitude, etc.
* We think we will use SceneKit (possibly in combination with SpriteKit since the two are integrated) to represent wit content when it is an image. We like those libraries for their relative simplicity and flexibility to have a 2D or 3D wit. To implement this with SceneKit or SpriteKit you can use a ViewController that has the VideoPreview as subview and above this view a SCNView or SKView with a transparent background.
* One challenging aspect of implementing a 3D model is that natural light is missing. To compensate for that we need to place some lights at the same positions they are in reality e.g. one for the sun. We can figure out where the sun is based on time of day and pre-compute some displays for morning, mid-day, afternoon, evening.
* As you are implementing this you will want to display and refresh wits and witMarkers with a rate of 60 FPS so that the augmented reality content moves smoothly and in a natural way. This gives you only 166ms for all your calculations and update methods, so you should start using Instruments Time Profiler tool early in your development.
* Other gaming libraries that we have explored and may come back to depending on our needs
  + Metal
  + OpenGL ES
  + Cocos2D/3D
  + Unity
  + Others ???

We are also open to your suggestions and expertise in this area.