ABSTRACT
This paper describes the mobile application, Imbue, that uses a GPS/compass-enabled AR algorithm to display to the user relevant events, building information, and other pertinent details within the user’s current surroundings. The mobile application has two AR views, a Camera AR view and a Map AR view. The Camera AR view is fully augmented, while the Map view simply orients the map to the current heading and location of the user. The app supersedes the accuracy recent attempts to offer AR in conjunction with a worldwide point of interest database, and it is one of the first consumer AR mobile apps to be released. This application was developed over the course of the semester and is available to the public at no cost via the Google Play Store and the Apple App Store.

General Terms
Human Factors, Performance, Algorithms

1. INTRODUCTION
We present Imbue, a novel mobile application for Android and iOS that uses Augmented Reality to display social media and located-related information to users. The application displays information about nearby buildings, Facebook events, and the location of nearby friends to users of the app. In addition to allowing users to view information about buildings and events around them, it also lets users identify their current location to Facebook friends.

The goal of Imbue was largely to create an app that brings AR functionality to the general public, offering data of interest to the general public like social media and location information. While other apps exist for displaying information about nearby points of interest using AR, there are few publicly available applications intended for the consumer market. Moreover, there are virtually no applications that draw from worldwide databases of point of interest to display AR related information in worldwide environments, and those that do are available for highly specialized audiences like blind users [?]. This particular app suffers from significant accuracy issues as well. Popular apps have recently added Augmented Reality technology. Yelp has an AR feature called "monocle," but the algorithm suffers from significant accuracy and display issues.

Additionally, applications that display points of interest throughout the world, such as Google Maps and Google Earth, do not presently include AR technology, but integrating AR into such apps is not extremely difficult.

In designing and implementing Imbue, we demonstrate that providing AR for points of interest worldwide, Facebook events, and other social media data like user location can be done to a high degree of accuracy with mobile phones. Imbue is also, to our knowledge, the first consumer-level social media app using AR released for Android and iOS.

The app can be used in a variety of contexts. We have currently added points of interest for all campus buildings at the University of Puget Sound to the app. This could allow the app to easily facilitate campus tours and navigation, and a similar AR-based approach could be used on other campuses. We have also acquired points of interest from Google Places and events and other information from Facebook to demonstrate the app’s ability to display content using AR throughout the world.

Imbue is the first mobile application of its kind and attempts to fully integrate the user’s social media experience into their physical interpretation of reality.

2. BACKGROUND AND RELATED WORK
2.1 Augmented Reality
Augmented reality refers to software that presents an environment to the user along with additional data and information not contained in that environment. Typically, an AR
application will overlay informative text and graphics on a video of an environment. AR has been used in a wide variety of settings, including in surgery, in navigation and tour applications, and to enhance popular location-aware apps like Yelp.

In order to provide AR features, an application must first identify objects and locations in the user’s environment and then augment the environment with additional information. Typically, this is done by first identifying features of the user’s environment and then using a visual display to overlay text, images, and other data on a video of the environment drawn from a camera.

Presently, there are three common approaches to AR: 1) sensor-based approaches, 2) computer vision approaches, 3) and hybrid approaches [?]. Sensor-based approaches use GPS, accelerometer, compass, camera, and other sensors to identify locations. Computer vision uses image-detection algorithms and cameras to identify nearby objects. Hybrid methods combine both sensor and computer vision techniques.

Sensor-based approaches are faster than computer vision-based approaches, and they are also simpler to implement. The primary disadvantage of sensor-based approaches is that they can be susceptible to electromagnetic interference, making them difficult to use in indoor environments. Computer vision-based approaches are comparatively slower in addition to being more difficult to implement. Their primary advantage is that they perform well indoors. Hybrid approaches aim to overcome the disadvantages of both approaches by combining sensors and computer vision [?].

In developing Imbue, we chose to use a sensor-based approach. We found that we can achieve acceptable performance indoors with sensors alone, particularly because we are not trying to identify interior objects with a high degree of sensitivity. The majority of objects we identify are buildings outside, and the only interior identification we must perform is identifying the building in which a user is located. Moreover, computer vision is relatively slow compared to sensor-based approaches, and we wanted to maximize identification speeds whenever possible in order to immediately identify buildings and to optimize CPU usage for mobile devices. Finally, given the benefits and performance of sensor-based approaches, we saw no need to implement more complicated computer vision approaches for this app.

While off-the-shelf packages exist for computer-vision based AR (such as OpenCV), there are few packages for sensor-based AR. As a result, we implemented our own sensor-based AR algorithm. In addition to the fact that no pre-existing libraries were available to implement the algorithm we wanted to use, implementing our own version of sensor-based AR gives us significant control over the algorithm, and has allowed us the flexibility to optimize and adjust the algorithm as needed.

Mobile phones are useful for AR presentation because they offer both video cameras and displays, and their motility makes them more flexible than computers for AR in outdoor environments.

2.2 Related Work

Recently, several mobile applications have been developed to help users navigate large environments. Both sensor and computer-vision based AR systems have been proposed and implemented for mobile AR applications.

Mata and Claramunt [?] developed an AR-based mobile app for navigating cities that uses a combination of GPS and compass bearing to identify nearby buildings. The app draws building information from an SQLite database. The authors note that the compass algorithm is in need of further refinement.

Guan et al. [?] provides an alternate approach to AR-based navigation of large cities. They developed a mobile app that uses computer vision to identify nearby buildings. The app produces a list of nearby buildings using GPS, then uses a learning algorithm (k-nearest neighbor) to identify images using the phone’s camera. The app compares images from the camera against web-based image search results. A combination of real-time web searches and computer vision techniques makes the app’s performance relatively slow, operating at peak speeds of 40-345 milliseconds. These peak speeds were achieved after optimizing the algorithm by matching only portions of images, and they still likely produce a human-perceptible lag.

Reitmayr and Drummond [?] have developed a hybrid augmented reality mobile application for navigating urban environments. The app combines GPS and compass location with computer vision based recognition of buildings. It compares images of buildings to a database of 3D models. The hybrid approach is intended to use computer vision to overcome the limitations of GPS in urban and building-heavy environments and the impact of sources of magnetic interference on compasses. The disadvantage of this approach is that the computer vision approach to building recognition introduces a 40 ms delay into the app, which, like Guan et al., likely produces a human perceptible lag.

Blum et al. [?] comes closest to our own implementation. They developed an iPhone app to help blind users navigate urban environments using AR. They rely on compass and GPS to determine location to points of interest. They use the Google Places API to generalize the app such that it can help users navigate in any location on earth. Published in 2013, the authors note that they are one of the first AR apps to draw from a dataset that can provide points of interest for any location in the world. They conclude, however, that the current state of GPS and compass accuracy of iPhones makes accurate identification of urban buildings intractable, and that AR technology will have to be delayed until hardware improves. We find that our app performs with a high degree of accuracy even in urban environments such as Seattle, and conclude that iPhone hardware is not presently too limited to perform accurate GPS/compass based AR identification.

Watts has developed TigerEye [?], a mobile application for providing AR-based campus tours of Clemson University. As with Mata and Claramunt, the app uses GPS and compass heading to provide AR information about nearby buildings. The app stores building corners in an SQLite database.
The algorithm for detecting buildings within the camera’s field of view involves calculating a vector between the user’s location and every building corner in the database. The app then eliminates vectors falling outside of the camera’s field of view. It then averages together headings that correspond with corners for the same building. From there, it determines a single heading and provides building information onscreen by converting between heading and pixels with a simple conversion factor. The app appears to include a number of redundant calculations. It calculates a bearing to every building corner on campus, even buildings far away from the camera, which could be eliminated with a simpler SQLite query. The app also calculates a bearing to each building corner only to average bearings for corners corresponding to the same building. Since the app is not using building corner information, it could simply use average bearings in the first place as long as identifications are accurate.

Other than mobile AR apps, there are a variety of apps available for viewing content about points of interest, the most well known being Google Maps and Google Earth. The Google Earth app displays icons for clickable information about points of interest. The clickable items range from photos taken at a location to wikipages about a point of interest. Google Earth displays clicked information similar to the Building Info and Event Info activities. It is difficult to develop an app with the same access to data as Google’s database, since Google places limits on how much content you can pull from their places API, and does not let you pull data from as many sources of information as they have access to (until recently, for example, it was difficult to integrate Google Streetview with Google maps). By adding other sources of information, however, we provide more information than most mapping apps currently display to users.

3. IMPLEMENTATION

3.1 Overview

The following section details the implementation of the Android and iOS versions of Imbue. We first discuss the Augmented Reality algorithm that we developed for Imbue. We then document the Android and iOS implementations in detail. We then outline the two views We then discuss the two AR views Imbue presents: a map view that rotates and moves with the user’s location and a camera view that displays overlays with information about buildings and events within the camera’s field of view. From there, we discuss the types of data that Imbue displays to the user with AR: Buildings, Events, and User Locations.

We maintain a database of points of interest and user locations on a server hosted by Parse. Parse is a database and app backend platform that allows for cloud hosting of MongoDB databases. It also provides a wide variety of methods through its API for interfacing with the database, and for social media platforms like Facebook, that significantly simplify the process of accessing the database from Android and iOS.

We use the Parse server in order to allow POIs to be updated and expanded at any time. The app communicates with the server instead of storing POIs locally. We also use the database to maintain user locations. At this time, we have added points of interest for all buildings at the University of Puget Sound. In the future, the database could store additional information about points of interest acquired from Google Places.

3.2 Design Patterns and Philosophy

Both the iOS and Android apps use the Model View Controller (MVC) pattern. MVC involves separating an app into three components, a model that stores data, a view that displays data, and a controller that interfaces between the data and the view. The data that we host on the Parse server resides outside of the iOS and Android app implementations and is the app’s model. Each app contains its own sets of views, which are separated from program logic. Program logic is contained in the controllers of the iOS and Android apps. The controllers are a set of classes that perform program operations, download data from parse, and display data in the app views.

The app also makes substantial use of the client-server design pattern. This allowed us to separate our data from the Android and iOS implementations, placing it in an easily updatable and implementation agnostic environment. Because much of the data, such as events, is subject to frequent change, and because the app is designed to be used in locations throughout the world rather than one fixed location, we found that offloading the app’s data to a server would also allow the app to be used with the greatest amount of flexibility and to be readily expanded. Since the app was originally intended for use at the University of Puget Sound with goals to expand beyond it, using a server architecture from early in the implementation process made expansion relatively seamless, and will allow the app to expand further in the future.

We used agile design techniques for both Android and iOS development. We made frequent use of pair programming, brainstorming and implementing ideas in group meetings. We often prototyped features on one platform before porting to another. Generally, we also aimed to rapidly implement features before going back and refining designs and layouts.

3.3 Augmented Reality Algorithms

3.3.1 AR Algorithm

We developed our own AR algorithm for both iOS and Android. It is used to display overlays on the camera view. The algorithm is detailed below:

1. Create a list of all points of interest within a 111 meter radius of the user’s current location.

2. Find all points of interest in this list that fall within plus or minus 1/2 the angle of view (50 degrees). This is accomplished by first determining the heading from the user to each point of interest, then determining if the difference between this value and the user’s compass bearing is less than or equal to half the angle of view.
There are two special cases in which the calculated headings must be adjusted. Generally speaking, these special cases occur when the user’s compass bearing is within the field of view of a point of interest, but the compass bearing and point of interest sit across the threshold from 365 degrees to 0 degrees. This would result in reporting a large difference in bearings when there is really a small one. In these cases, 365 degrees is added to one of the values to normalize the difference in bearing as follows:

(a) If the user’s current heading is >360-1/2 the angle of view and the heading to the point of interest is < 1/2 angle of view, then 360 degrees is added to the heading to the point of interest.

(b) If the user’s current bearing is < 1/2 angle of view and the heading to the point of interest is >360-1/2 the angle of view, then 360 degrees is added to the user’s heading.

3. Add 1/2 the angle of view to the heading value of the point of interest (providing a range of 0 to the angle of view) to get a raw value.
4. Divide the number of pixels the screen is wide by the angle of view to obtain a scaling factor.
5. Multiply the value by the scaling factor to find where on the screen the pixel should be drawn (see below).

### 3.3.2 Displaying Overlays Onscreen

Screen x-axis coordinates for displaying overlays for points of interest within the user’s field of view are calculated using the following formula. Let s be the device’s screen width in pixels, f be the field of view size in degrees, b be the bearing to a building coordinate in degrees, and c be the current compass bearing in degrees.

\[ x = \frac{s}{2f} \times (b - c) + f \]

The formula first calculates a conversion factor to convert between screen pixels and degrees in the field of view. This value is multiplied by the difference between the bearing to the point of interest and the current bearing, a maximum value of half of the field of view, to yield the difference in pixels between the camera’s line of sight and the point of interest. The size of the field of view is then added to the product in order to position the point of interest in relation to the start of the field of view.

Y-coordinates are generated arbitrarily. Since there may be multiple points of interest at the same GPS location, each new coordinate is placed a constant y value away from the previous coordinate in order to prevent overlap.

### 3.3.3 Augmented Map View Algorithm

Another simpler augmented reality algorithm is used for the augmented map view. This algorithm simply aligns the map’s camera heading in response to compass sensor changes to keep the map oriented with the user’s heading.

3.4 Android

Android is one of the mobile OS platforms that this app was developed for. The language to create Android applications is Java. There are many aspects that go into creating a mobile application. Aspects such as design guidelines and the Android framework. The Android Framework is what makes up a mobile application. It provides many different APIs to use the features Android has to offer. There are many small parts that all work to have a application run on a phone. We will touch on a few of the most important aspects of the framework.

### 3.4.1 Activities

An activity can be best thought of an interaction point between the user. The activity allows a place for UI to be placed. When the app first starts up a map is loaded. There is a pullout side menu also. This entire screen can be thought of as an activity. This application has many different activities. These activities each serve a particular purpose, such as displaying building or event information, the augmented view and map view. It is important to keep track of Activity lifecycles. As a user moves throughout the application, they may close different activities and open new ones. It is important to manage the lifecycle of these activities or else they add up and take up more of the phones resources. There is the ability to pause, resume, restart and stop activities and we stop all activities when they are left. When accessing a new activity a new instance of the activity will be loaded. The only activity that is not killed when the user navigates away is the map view activity. This activity does restart though when the user navigates back to provide an accurate location and query of nearby buildings.

### 3.4.2 Layouts

A layout is a way to define the user interface of an activity. Layouts are made of XML language. XML is very similar to the HTML syntax because of the use of the tags, attributes and elements. When an activity is created, a xml layout can be loaded along with other user interface aspects. Layout elements can be created at runtime also, such as the event list in the Building Info Activity. Every XML layout must have a particular layout structure as the root element. This could be a commonly used linear layout, grid layout or relative layouts. Custom layouts can also be created to further
Figure 2: The view when the application is started in Android

customize the user interface. Everything from margins, text size, font, colors and titles are determined in each element of a layout.

3.4.3 Map Activity
The map view of the app is what is loaded on the launch of the app. This activity does some important tasks on the creation of the activity listed here:

1. The app makes a connection to Parse
2. Then the sensor service is initialized
3. The main view of the activity is paired with the XML
4. Next a Google Map is created and placed on the screen. This map is enabled to have on touch listeners.
5. Last the side drawer menu is created to provide the user with an easy way to navigate throughout the application.

3.4.4 Camera Activity
Android handles the camera view different from the iOS version. In Android, the Camera Activity is where the majority of the math to calculate augmented reality takes place. Some work goes into setting up this user interface for the user to interact with.

3.5 iOS
The iOS app is structured similarly to the Android app, but there are a number of differences in implementation stemming from differences inherent to iOS and Android development.

The iOS implementation is comprised of three Objective-C classes, ViewController, ContentViewController, and Bearing.

While Google Maps provides an Android method for calculating bearings to points of interest, iOS does not. The Bearing class calculates a bearing from the user’s compass bearing to a point of interest using the following formulas:

\[ y = \sin(dLon) \times \cos(lat2) \]
\[ x = \cos(lat1) \times \sin(lat2) - \sin(lat1) \times \cos(lat2) \times \cos(dLon) \]

The ViewController class provides the majority of the app’s functionality. When the app is first launched, it establishes a connection with Parse and downloads points of interest, storing them in a mutable array. It also downloads Facebook events on launch along with any available user locations. When a GPS and compass reading is first acquired, it orients the map in relation to these readings.

When the app is rotated into landscape mode, the ViewController allocates and initializes a UIImagePickerController. The image picker controller is typically used to quickly take a picture and add it to a user’s library, but with the addition of overlays, it provides an easy way to create a customized camera view on iOS. There are few other approaches to displaying a customized camera view. To implement this custom view, text and image layers were added to the view as subviews to display AR overlays. Additionally, the camera view has been transformed to fill the entire screen, as the default camera view does not match the screen size.

The ContentViewController displays detail views for points of interest. It is passed the title and description of points of interest. It acquires images either from the ViewController for buildings that are loaded into the Parse database or by making a call to Facebook’s API for Facebook event images. This lazy initialization is done to avoid loading a large number of Facebook images upfront.

There are some other minor differences in implementation between iOS and Android. The iOS app does not display
events in the augmented camera view while the Android version does. This was a performance concession; after finding that events significantly impacted the performance of the Android app, we decided that they were not worth implementing for the iOS app. The iOS app also does not implement a search bar. By contrast, the Android app does not display the user’s location but displays the location of other users. The iOS app makes use of a lightweight menu rather than a swipe-out "drawer" menu because Apple has recently discouraged the use of this type of layout.

3.6 Map View
The map view snaps to the user’s current location. The map will rotate similar to a compass based on the user’s bearing. When the screen is tapped or when a marker is clicked, the map stops rotating with the user’s bearing to allow the user to zoom and otherwise adjust the map. In order for the user to quickly snap back to their location, the compass icon in the top right can be pressed.

The map view displays data via several different types of markers. There are red markers which represent buildings, yellow markers that represent interior points of interest, purple markers representing the location of nearby Facebook friends, and blue markers representing public and private Facebook events.

The map view activity also has a search function in the top action bar. The magnifying glass button opens the search bar. The search queries the titles of all currently visible buildings to check for a match.

3.7 Camera View
The user has the ability to get to the camera activity by (on Android only) selecting the camera menu button or by rotating the phone to a landscape position. This starts the augmented view. The augmented view shows all nearby buildings and, for Android, Facebook events. The camera activity has two parts. The camera interface and the camera overlay interface. The camera interface displays the image that the front camera is pointing at. The camera overlay interface is used to draw overlay content on top of the camera interface. The camera overlay allows for drawing objects and text on the screen. This method was found to be the most efficient for time and timeline constraints when working on the project, but it would be desirable to, as future work, implement this overlay in OpenGl, as this would allow for overlaying content in a 3D plane instead of the 2D plane the camera overlay currently uses.

3.8 Buildings
Imbue displays information about nearby buildings and points of interest to users in its AR views. Building information comes from two data sources: Google Places and our own Parse server database. We used Google Places to collect and display information about buildings throughout the world. We query Google Places directly for buildings near the user.

We also maintain our own server database with building names, descriptions, images, coordinates, and rectangular coordinate regions used for calculating whether the user is inside of the building. The server currently contains information about all buildings for the University of Puget Sound. This allows the app to be used for campus tours and by students of the University of Puget Sound, but, in the future, we could also store building information acquired from Google Places in order to minimize Google Places queries (which are limited to 1000/day within the free payment tier) and to augment Google Places data with additional sources of information.

3.8.1 Google Places Buildings
We include building information from Google Places to acquire AR overlays for buildings throughout the world. Google Places is a rich source of information; Blum et al. found it superior to other data sources like FourSquare in terms of coverage.
Google Places buildings were drawn from Google’s data repository using a javascript API query performed with Parse Cloud Code. We used Cloud Code because Google Places only provides a javascript API. When the phone launches the main screen, we call a method in Cloud code to query the Google Places API. The process of querying Google Places is as follows:

1. The current user location is placed in a dictionary that is passed to our Cloud Code function as a parameter.
2. The app calls the "placesRequest" Parse Cloud Code function using an asynchronous HTTP request method provided through the Parse API. This function submits the following API query to Google places in the form of a GET request:
   
   https://maps.googleapis.com/maps/api/place/nearbysearch/json?location=’ + request.params.location +’&radius=1000&sensor=false&key=[Our API Key]

   The query returns a JSON object containing the names and locations of nearby buildings within a 1000 m radius or, if unsuccessful, returns an error. The location of the query is determined from the dictionary passed to Cloud Code as a parameter.
3. When the function completes we parse the JSON object to obtain the names and locations of all buildings. We store these in a list of buildings.

The information provided is extremely limited (only building names and locations are returned with some limited user review information if available). In the future, it would be ideal to query for more building information from other sources. We could also crowdsource the acquisition of information and allow users to upload information about buildings to our server.

3.8.2 Parse Server Buildings
We collected information for all buildings at the University of Puget Sound and store them in a Parse database. The database contains fields for building tile, description, image, coordinate location, and coordinate boundary.

The coordinate boundary allows the app to determine whether the user is currently inside of a building. It also allows us to present information about what is happening in the building such as what Facebook events fall within those coordinate boundaries.

In the future we would like to host our own database of building information for points of interest throughout the world, relying on Google Places only when the database does not contain sufficient information about nearby places. This would reduce our reliance on Google Places, which offers a limited number of daily queries, allow us to synthesize information from additional sources, and also allow us to offer additional information not available from Google Places. As long as we pull information directly from Google Places without also maintaining our own database of points of interest, we will be limited by the information available from Google Places.

3.9 Events
To show users Facebook events we used FQL (Facebook querying language). We wrote a query that would pull any events that a user could see based on their current location. Originally we limited the events to being in a few mile range of user. We decided to generalize the query to return events throughout the world, as the query still returns a relatively small amount of data. This allows us to show a user all the events that are public or that are available to their friends or themselves. This means that there will be a unique map for each user based on their social media connections. The FQL query for obtaining events is as follows:

SELECT name, pic_big, venue, description, start_time, end_time, eid FROM event WHERE eid IN (SELECT eid FROM
The query obtains pictures, coordinate locations (contained in the "venue" term), descriptions, and start and end times for events. It obtains events for the user's friends ("FROM friend") or for the user ("uid = me()"). It performs a check to determine whether the event is currently taking place, and limits the number of events obtained to 2500 since the map view begins to lag when it must display more than several thousand markers. Event information returned is parsed and stored in a list of events.

### 3.10 User Location

Imbue allows users to log their current locations in order to share their location with their Facebook friends. Users can choose to log their current locations and a message about their current location. The message is recorded along with the user's name and current GPS coordinate. This information is stored on the Parse server for each user. In addition, the Parse server maintains a boolean field `shareLocation` that specifies whether or not the user's location is currently being shared. Users can manually choose to stop sharing their locations, in which case this field will be set to false. The field is also set to false if users move a certain radius away from their current location or if a certain amount of time elapses since the user's location was last logged.

Locations of the current user's Facebook friends are plotted on the map view. To do this, the app's controller filters the list of app users for only those who are Facebook friends with the current user and then plots the locations of any users choosing to share their locations. The app controller does this by first querying Facebook for a set of unique id numbers for users who are Facebook friends with the current user. It then queries the Parse server for a dictionary of app users ordered by Facebook unique id. For each Facebook friend whose unique id is in the dictionary, it checks if the `shareLocation` field is set to true, and, if it is, it obtains GPS coordinates and the message for the user's location and plots these on the map.

The FQL query for identifying Facebook friends of the current user is:

```
SELECT uid FROM user WHERE is_app_user AND uid IN (SELECT uid2 FROM friend WHERE uid1 = me())
```

### 3.11 Updating User Data

Because the app is highly dependent on the user's current location, we frequently query our various data sources as the user's location changes. We query for Facebook events, the locations of nearby Facebook friends, Google Places, and our own database of points of interest when the app is first launched, when the app is relaunched from the background, and whenever the user moves outside a significant radius.

### 3.12 Libraries

#### 3.12.1 Facebook

Facebook is used for many of Imbue’s features. We use the Facebook API to display events to users. We also use Facebook in conjunction with our nearby friends feature. We query the Facebook API to obtain a list of Facebook friends in order to determine whether a nearby user is Facebook friends with the user and should be displayed on the map.

#### 3.12.2 Google Maps

The Google Maps API brings in features such as 3D rendering, vector tiles, indoor floor plans, gesture controls, location services and the ability to draw on and customize the map. Imbue uses all these components. 3D rendering is shown where available. Indoor floor plans are very useful in Imbue, because a user can share their location to their Facebook friends and friends will be able to see what part of the building they are in. In addition, indoor maps provide a variety of free points of interest displayed directly on the Google map.

#### 3.12.3 Google Places

The Google Places API provides a database of general info about points of interest throughout the world. Google Places allows 1,000 requests per day. With credit card verification, the amount of requests can be increased to 100,000 per day. The Google Places API allows for requests to be made for nearby buildings, and it can display details about the buildings and images of the buildings. Given Google Places’s severe restrictions on daily queries and our restricted budget, we will need to find a solution for obtaining or storing point of interest data without or in addition to Google Places in the near future.

#### 3.12.4 Parse

Parse was acquired by Facebook in 2013. Parse is an app backend development service. It provides database hosting for mongoDB databases, as well as APIs to interact with databases from Android and iPhone apps. Parse is very popular with hosting data for many mobile applications. Parse offers other services such as Parse Cloud Code, which hosts javascript files that can be called from mobile devices and executed on Parse’s cloud servers. Parse recently changed its pricing plan to allow up to 30 requests per second, which should be enough to accommodate Imbue until it acquires a significant user base, and even then Parse is designed to be scalable and relatively inexpensive.

### 4. EVALUATION

We measured the performance in milliseconds of the AR algorithm across three trials. The time is measured for the
Our AR algorithm performs 20 times faster on iOS than the most optimized version of Guan et al.’s computer vision AR algorithm and 272 times faster than the least optimized version of their algorithm. The Android performs 40 percent faster than the most optimized version of Guan et al. The least optimized version of Imbue on Android runs 270 percent faster than their application. Their algorithm achieves a maximum speed of 40-60 ms after several optimizations and using fast sensors, and a worst case performance of 490-540 ms with no optimizations and slowest sensors [?]. The app likewise outperforms the algorithm developed by Reitmayr and Drummond, which, like Guan et al., achieves a peak performance of 40 ms. We credit this to the superior performance of sensor-based approaches over computer vision techniques.

When we add Facebook events that are displayed to the screen on Android, we observe an average time of 274 ms. The events dramatically slow down the algorithm simply due to the size some type of sorting could dramatically improve both event and event free versions of the algorithm. We do not report event data for iOS because we elected not to display events in the camera view largely due to the significant performance drop we observed on the Android platform.

5. DISCUSSION
As noted throughout this paper, Imbue provides a variety of functionality that improves upon AR applications currently available.

Imbue runs faster than Guan et al.’s computer vision AR algorithm which was running on top quality hardware in 2010. It similarly outperforms Reitmayr and Drummond’s 2006 computer vision/sensor hybrid algorithm. In general use, we have found that Imbue also does not suffer from the same degradation in sensitivity characteristic to sensor based AR methods, although we have not yet evaluated its accuracy at the level of sensitivity used to evaluate computer vision based algorithms. However, we have found that it does consistently recognize points of interest accurately even in urban environments, which commonly trouble compass and GPS readings. Moreover, even as our datasets have grown to include buildings, events, and user locations, Imbue’s AR algorithm has not suffered from significant performance degradations.

In fact, Oculus Rift has recently noted that an algorithm running below 20 ms does not suffer from human-perceptible lag, although others have argued that application should aim for 7-15 ms. Anything over 40 ms results in a “disturbing lag.” [?] Imbue consistently performs at approximately 2 ms, while the majority of recent computer-based AR applications published perform at or above 40 ms.

Imbue has also expanded past work such as Google Maps and Yelp’s mobile application. Imbue creates a unique combination of social media, augmented reality and maps. We feel that Imbue, when used in combination with mapping services like Google Maps, offers useful functionality that can be easily intergrated with existing navigation technologies. Imbue supercedes other technologies that have attempted to integrate AR with existing social media, like Yelp’s AR feature, “Monacle.” We account for the rotation of the phone around the x-axis where the Yelp app does not. Yelp’s failure to do this means that Yelp’s labels appear in incorrect locations. Yelp’s camera view does currently display graphical overlays superior in quality to Imbue’s, but Imbue’s camera view allows for smooth drawing of the markers that are overlaid on the screen, while Yelp’s are choppy and stutter frequently.

6. FUTURE WORK
Because Imbue includes both AR and social media features, we could provide a variety of new functionality for either the augmented or social media side of our app. On the augmented side, storing additional points of interest in our own database is one of our first goals. This will allow markers to display significantly more information not contained in Google Places, such as picture galleries and other information tailored to the specific point of interest, like menus for restaurants. Expanding our database will also reduce our Google Places query load, which will become unreasonable high if the app gains even a small or average-sized user base. In order to expand our database, we will need to look into collecting data from other public domain information sources or from crowd-sourced, user-inputted data.

On the social media side, improvements can be made that provide additional ways of interactions with friends. Implementing group chat options for friends sharing location would be a useful feature for people to have. Being able to show more information about friends sharing their location, such as profiles pictures, statuses, tweets from Twitter and real time updates of what friends are sharing could also be useful additions to the app.

Implementing graphics in Open GL is left as future work. While we attempted to implement Open GL, we ran into a variety of issues that made displaying text and graphics unfeasible. The implementation was too inefficient and stuttered. However, Open GL is widely used on mobile devices and should, with additional time, be relatively simple to im-
plement. Aligning graphics in three dimensional space will make the relationship between buildings and their overlays much clearer.

Finally, it would be ideal to implement Imbue for additional devices, such as the iPad and Google Glass. Preparing the app for iPad could be done extremely quickly, while implementing it for Google Glass, while more time consuming, could allow users to integrate the app into everyday life.

7. CONCLUSIONS
In developing Imbue, we created several novel features not currently available in apps for mobile devices while achieving many of the goals we set out to accomplish. We believe that soon, many of the features we have been working on will be seen in many applications we use today. We developed an efficient augmented reality algorithm ourselves, and we made strides in displaying information about the campus and surrounding areas. We also developed an interactive map view which has since been adopted in Google Maps. Overall we created an engaging augmented reality application for two platforms that we believe offers a fun, interesting, and new way to view location-based information.

8. ACKNOWLEDGMENTS
We would like to thank Prof. Brad Richards for overseeing capstone and providing project guidance, Mike Rottersman, Associate Director of Admissions, for providing links to Puget Sound campus and building information for our database, and the University of Puget Sound computer science department for a quality computer science education.