White Paper Report

Report ID: 99039

Application Number: HD5099510

Project Director: Joan Decker (joan.decker@phila.gov)

Institution: City of Philadelphia, Department of Records

Reporting Period: 3/1/2010-2/28/2011

Report Due: 5/31/2011

Date Submitted: 5/31/2011

azavea

PHILADELPHIA DEPARTMENT OF RECORDS

Implementing Mobile Augmented Reality Technology for Viewing Historic Images

An Azavea and City of Philadelphia Department of Records White Paper

Azavea • 340 North 12th Street
Philadelphia, Pennsylvania • 19107
(215) 925-2600
www.azavea.com

City of Philadelphia Department of Records • City Hall, Rm 156
Philadelphia, Pennsylvania • 19107
(215) 686-2260
www.phila.org/records

Copyright © 2011 City of Philadelphia Department of Records and Azavea
All rights reserved.
Printed in the United States of America.

The information contained in this document is the exclusive property of the City of Philadelphia Department of Records and Azavea. This work is protected under United States copyright law and other international copyright treaties and conventions. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, or by any information storage or retrieval system, except as expressly permitted in writing by the City of Philadelphia, Department of Records. All requests should be sent to Attention: Commissioner, Department of Records, City Hall, Room 156, Philadelphia, PA 19107, USA.

The information contained in this document is subject to change without notice.

U.S. GOVERNMENT RESTRICTED/LIMITED RIGHTS
Any software, documentation, and/or data delivered hereunder is subject to the terms of the License Agreement. In no event shall the U.S. Government acquire greater than RESTRICTED/LIMITED RIGHTS. At a minimum, use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in FAR §52.227-14 Alternates I, II, and III (JUN 1987); FAR §52.227-19 (JUN 1987) and/or FAR §12.211/12.212 (Commercial Technical Data/Computer Software); and DFARS §252.227-7015 (NOV 1995) (Technical Data) and/or DFARS §227.7202 (Computer Software), as applicable.

Contractor/Manufacturer is Azavea, 340 N 12th St, Suite 402, Philadelphia, PA 19107, USA.

Azavea, the Azavea logo, www.azavea.com, and @azavea.com are trademarks, registered trademarks, or service marks of Azavea in the United States, and certain other jurisdictions. Layar, Apple, Apple's iPhone, Apple's iPad, iOS, Android, and other companies and products mentioned herein are trademarks or registered trademarks of their respective trademark owners.

Notes:
We would like to acknowledge Commissioner Joan Decker at the City of Philadelphia Department of Records for her ongoing and enthusiastic support of *Philly*History.org.

We would also like to acknowledge the National Endowment for the Humanities Office of Digital Humanities for their support of this project in the form of a Digital Humanities Start-Up Grant. Any views, findings, conclusions, or recommendations expressed in this paper and project do not necessarily reflect those of the National Endowment for the Humanities.

Portions of this white paper also appeared in *Museums and the Web 2011: Proceedings* under the following citation.

Boyer, D. and J. Marcus. Implementing Mobile Augmented Reality Applications for Cultural Institutions. In J. Trant and D. Bearman (eds). *Museums and the Web 2011: Proceedings*. Toronto: Archives & Museum Informatics. Published March 31, 2011. http://conference.archimuse.com/mw2011/papers/implementing_mobile_augmented_reality_applicat

Introduction

Synopsis

In February 2010, the City of Philadelphia Department of Records (DOR) was awarded a Digital Humanities Start-Up Grant to investigate the use of mobile augmented reality technology in displaying historic photographs. Administered by the National Endowment for the Humanities Office of Digital Humanities, the Digital Humanities Start-Up Grant program is designed to promote innovative digital projects in the humanities and share information on those projects via a series of publicly accessible project white papers. The DOR sought to use the Digital Humanities Start-Up Grant to investigate augmented reality (AR) as a new and more immersive way for users to access the over 93,000 images and maps—as of May 2011—available in the *Philly*History.org database (www.phillyhistory.org).

Entitled "Historic Overlays on Smart Phones," the original grant application proposed utilizing mobile augmented reality technology to develop a mobile application that would enable users to view historic photographs of Philadelphia, via their smart phones, as overlays on a camera view of the current urban landscape. A selection of 500 images would be geographically "pinned" in 3D space and the coordinates used to more accurately align past and present views of a specific location. An advisory group of local historians would provide contextual information to accompany a select twenty images with additional resources also available.

While the initial grant called for research into creating a prototype application, we have chosen to release that prototype as an application available to the general public. The Augmented Reality by *Philly*History.org application is available for both the iPhone and Android phone platforms via the respective app stores.

Mobile augmented reality includes many technologies and is used in a variety of ways ranging from art installations to non-profit activities to commercial ventures. We recognize that AR applications vary widely and what is possible in one application may not be available in another. This white paper specifically looks at our research into mobile augmented reality and its use for displaying historic photographs as 3D overlays. We outline our investigations into various augmented reality technologies and provide a description of how we built the final application, including both a general outline of our process as well as more technically detailed explanations of certain features. We have also included notes on how we would like to continue our research as well as the future role that augmented reality may play in cultural institutions.

Background

The City of Philadelphia Department of Records manages and provides public access to the historic resources available at the Philadelphia City Archives including a photograph collection estimated at nearly 2 million images. Originally taken by City photographers for risk management purposes, the photograph collection dates back to the latter half of the nineteenth century and has become a rich resource of historic visual material. From images of the building of the mass transportation system in the early 1900s to a quiet residential street in the 1950s, the photos document Philadelphia's past and provide insight into the people, places, and events that encompass the city's history. Rather uniquely, the collection is very geographic in nature; many of the images are of a specific location identified by the original photographer.

Figure 1: The *Philly*History.org map-based search page emphasizes the geographic information available for many of the images.

While the photographs were available at the City Archives, public access of the images was limited to those who could visit the Archives during open hours and much of the photo collection was not fully catalogued. To provide better access to the images, the DOR, in 2005, launched *Philly*History.org, a web-based digital asset management system that provides both public access to historic images of Philadelphia as well as administrative management of the related archival metadata. Built by Azavea, a local software company specializing in Geographic Information Systems (GIS), *Philly*History.org includes a publicly visible search page that enables users to search through the images by geographic location (address, intersection, place name, neighborhood), keyword, date, topic, and other criteria (Figure 1). Each image is visible as a thumbnail and in a larger detail view that includes metadata fields, a small map of the location of the photo, and the ability to view that location in Google Street View or Google Earth. The site also includes interactive features that encourage public engagement with the images. On the administrative side, authorized users can add, update, geocode, and delete records in a simple web-based system. Administrators can also respond to user-submitted error reports, scan requests, and comments as well as leave internal management notes, create featured photos and searches, answer licensing requests, and view site statistics.

Initially, *Philly*History.org contained a mere ninety images from the City Archives. Over the past six years, it has grown to contain over 93,000 historic photographs and maps from five Philadelphia organizations – the Philadelphia City Archives, the Philadelphia Water Department, the Free Library of Philadelphia, the Library Company of Philadelphia, and the Philadelphia Office of the City Representative. The site is updated regularly based on changing technology, administrative requirements, and public feedback.

In summer 2007, the DOR launched *Philly*History.org Mobile, making the entire collection accessible via cell phone and other Internet-capable mobile devices and enabling users to view the images while standing at the location where they were originally taken. Mobile access improved in 2009 with the launch of a web application specifically optimized for the larger touch screens of the new generation of smart phones

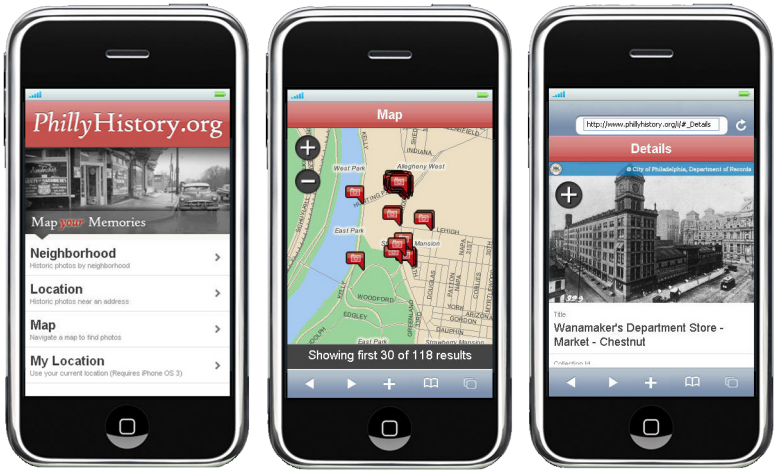

Figure 2: A version of *Philly*History.org optimized for smart phones provides users with geographic search access to the entire collection of images.

such as the Apple iPhone and Google Android devices. With this application, users can search for images based on location, view the search results as flags on a full-screen map, and select a flag to view a much larger image with complete metadata (Figure 2). The application also makes use of location technology. If a phone has an available internal GPS system, users can simply visit the *Philly*History.org search page and the mobile version will load historic photos near their current location.

Mobile access to the images on *Philly*History.org created an opportunity to introduce these historic images to new audiences who might not have known of their existence. It also led the DOR and Azavea to question how else mobile technology could be used to interact with students, historians, and the general public and assist them in experiencing and investigating the history of Philadelphia. Some of the most striking opportunities appeared to be in the field of augmented reality, loosely defined by the *Philly*History team as the overlaying of digital data or computer generated imagery on a live view of the physical world. In February 2010, the Department of Records received a Digital Humanities Start-Up Grant from the National Endowment for the Humanities Office of Digital Humanities to research and develop a prototype mobile phone application that would enable users, via their smart phones, to view historic photographs of Philadelphia as overlays on the current landscape – a form of augmented reality. By using the geographic coordinates tracked as part of the *Philly*History.org digitization process, the DOR hoped the prototype application could more accurately place the images in 3D space and create a stronger user experience. The final goal was a prototype application that would enable a more immersive experience with the historic images.

Augmented Reality: The Original Plan

In the original NEH grant application, the DOR proposed researching and developing a prototype that would provide mobile access to approximately 500 images as overlays on the current urban landscape. Since each image is geocoded when it is added to *Philly*History.org, the goal was that users would be able to point their phone at a location connected with a photo and view the historic images as an overlay on the view shown through the phone's camera display. Each image would be accompanied by descriptive text including title, date, collection name, and location. The DOR would also work with an Advisory Committee, which includes the co-editors of The Encyclopedia of Greater Philadelphia project, to create additional interpretive text for at least fifteen of the images. The images would be selected from the collections of the five organizations contributing to *Philly*History.org, although the majority of the images would come from the City Archives. While the prototype application would focus on the neighborhoods in the

downtown area, the project team also would include images from neighborhoods throughout the city in order evaluate accuracy issues related to tree cover, building height, and other multi-path errors that could affect the display.

With this overall goal in mind, the DOR proposed investigating three different options for creating the augmented reality prototype. The approaches reflect the myriad and swiftly changing options that currently exist for augmented reality development.

1. Use an existing proprietary framework from one of the leading companies working in augmented reality such as Layar, Metaio, or Wikitude.
2. Create a custom application that will run on the Android platform.
3. Create a custom application that will run on the Apple iOS platform, which includes iPhone, iPod Touch, and iPad devices.

While the project seemed technically feasible, especially based on the success other cultural institutions had found with augmented reality applications, the DOR recognized several significant questions requiring investigation.

- Will the level of GPS accuracy inherent in smart phones allow true point and view functionality in the crowded urban built environment?
- Will the available technology allow for the images to be placed in the 3D space where they were taken?
- What is the most effective user interface for displaying historical images and additional text?
- Will processing times be fast enough to provide the augmented reality data in real-time fashion?

We hoped to answer these questions and more by experimenting with the available augmented reality technology and constructing a prototype application.

The Devices

Before further explanation of our project, however, some background information on the creation of AR technology is useful. The concept of augmented reality has existed for many years. In the 1990s, one form of developed augmented reality consisted of special goggles that could overlay a video image from a small computer onto the screen of the goggles. Although an intriguing idea, interacting with the data was awkward, and the concept did not become popular. Increased development of AR technology returned in summer 2009, however, with the release of a new version of the Android operating system. What made this possible? The key to the answer lies in the phones and the operating systems that run them. A contemporary smart phone is not just a phone that can run applications; it is a device packed with sensors that application developers can access and utilize. The iPhone and Android devices (and later the iPad, Windows Mobile 7, and Blackberry devices) have sensors that include:

- Microphone
- Global Positioning System (GPS)
- WiFi
- Cell tower radio receiver
- Camera
- Compass
- Accelerometer

The most recent devices have continued to add new sensors, such as a gyroscope. Except for the microphone, all of these sensors are important to enabling the creation of the many new AR applications. A basic geographic location is provided by the GPS sensor, and the location is frequently augmented by WiFi and cell tower locations. The compass provides information on the direction the device is pointing and the accelerometer indicates how the device is moving through space. The summer 2009 release of a new Android operating system added the final piece – the ability for developers to take control of the camera view and add graphics and other media to the display. Combined with the other sensors now common on Android and iPhones, this innovation opened up important new possibilities, and, within weeks, some of the initial experiments with contemporary augmented reality appeared. The iPhone operating system (later renamed iOS) added a similar capability in a release shortly thereafter.

First Steps: Selecting Materials

We wanted to take advantage of new developments in mobile technology. Mobile augmented reality applications, however, are dependent upon having assets that are associated with a particular location. To create an effective mobile augmented reality app, an asset - whether it be an image, an object, or an actual physical point – must be assigned latitude and longitude coordinates. Since each image in *Philly*History.org is geocoded when it is added to the website, this information was readily available for the project. Truthfully, we had too many photos and too much information. Of the 93,000 images, we needed to select 500 images to be visible as 3D overlays in the application. Of those 500 images, 20 photographs would be selected for additional attention. A group of local scholars including the editors of The Encyclopedia of Greater Philadelphia would research the 20 select images and write explanatory text on the place, events, and people shown in the photograph.

The *Philly*History.org team with feedback from the Advisory Committee created the below list of criteria to follow while selecting images.

1. Time period depicted in photo
2. Historical interest
3. Educational value
4. Aesthetic interest
5. Taken from all collections available on *Philly*History.org
6. Mixture of locations throughout Philadelphia
7. Locations that have changed dramatically
8. Locations where elements of the historic and current photo are similar
9. Geocoded
10. Taken at street level or matched well to Google Street View

The last criterion was required based on our desire to place the images as overlays in 3D space. To do this, we needed additional coordinate data that would place the images in the correct angle, position, and scale in 3D space. Rather than always directly facing the user, the images would have an absolute positioning in space that allowed them to be more accurately aligned with the same location in the current landscape.

This level of positioning data is rarely tracked in archival photo management software, but we were able to gather this data through the Google Street View feature added to *Philly*History.org in early 2009. If 360-degree street level views are available for the location where an image was taken, users can select to view that location in Google Street View. This feature provided a simple method for users to compare a historic photo to the contemporary landscape where it was taken without requiring physical travel to that location. The management features of *Philly*History.org also included an option for administrators to coordinate the angle (pitch and yaw) and zoom of the Google Street View imagery to match the angle and zoom of the historic photograph (Figure 3). These additional coordinates of pitch, yaw, and zoom enabled us to calculate the location of the image as if it were a billboard sitting in 3D space on the side of a road. To use the pitch, yaw, and zoom coordinates, however, we first had to make sure that information was available for the 500 images. Two dedicated *Philly*History.org interns set the Google Street View imagery for each of the selected images. Now, not only did we know the location on the ground (the latitude and longitude), we also knew the angle the photo occupied in space, enough geographic information to possibly facilitate the display of the images as an augmented reality style overlay.

Unfortunately even with trying for as wide a selection as possible, we found many areas of the city still had gaps of several blocks between selected photos. After much discussion, we chose to include all geocoded images from the *Philly*History.org database in the augmented reality application for a total of nearly 90,000 photographs (of the 93,000 photos in the full collection). The 500 selected images would appear as overlays "pinned" in 3D space while the remaining images would be visible in the application (since latitude and longitude information was available) but not as accurately placed as they would be if pitch, yaw, and zoom data were available. Expanding the application to include a large collection of assets created better access to the images. Creating an augmented reality application with 90,000 data points, however, also had unique challenges as described later in this white paper.

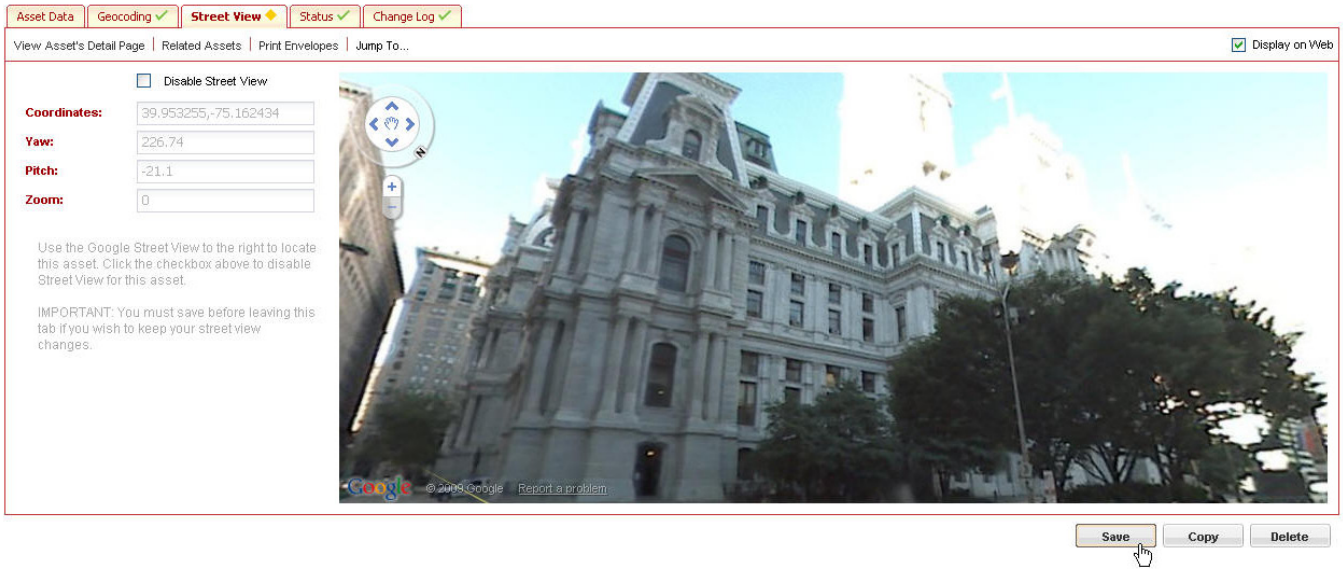

Figure 3: The administrative tools include an option for setting the Google Street View to match the angle and scale (zoom level) of the original photograph.

Initial AR Technology Research

While the archival team selected images and wrote text for the application, the development team began researching different methods for creating the application. Throughout the research and development process, we consistently encountered rapidly changing technology and the introduction of new options. The technology portions of this white paper reflect our experience at the time of the research and development of our application. The state of various AR technologies may have changed since our research, and we encourage readers to investigate the latest software releases and other AR options.

The application development team began the project with a broad review of augmented reality technologies to identify the most promising technology strategies for developing an AR application. Given how rapidly the relevant technologies are changing and advancing, we wished to investigate what are likely to be the most promising platforms and techniques in the next few years. We also performed a series of initial experiments to learn the real strengths and weaknesses of the platforms available, as we discovered there was often a significant gap between the advertised and actual performance of augmented reality technologies.

While we learned that many of the technologies are still quite immature, we found that the most promising and powerful technologies fell into what we categorized as "sensor" AR and "computer vision" AR. A simple way to frame the difference between these two approaches is with a question of their approaches to AR. Sensor AR asks the question "Where are we?" while computer vision AR asks the question "What can we see?" In the sensor AR approach, an application combines data from the sensors in a modern smart phone, like the GPS and the accelerometer, to make the best guess possible as to where the user is standing and where they are looking. The computer vision AR approach analyzes the images coming in from a digital camera (e.g. the camera on a smart phone) to determine what is visible in the world around the user and where those objects are in relationship to the user. Both approaches are discussed in more detail below.

Computer Vision AR

We performed a series of experiments with two of the most popular and influential open source computer vision libraries - OpenCV and ARToolkit. In general, we found that computer vision technologies have promising futures for AR work in the humanities. While they are extremely powerful for certain applications, this technology was unfortunately not the right match for the *Philly*History.org application.

The OpenCV project was launched in 1999 by Intel with the goal of creating open and optimized libraries that implemented

the leading edge ideas of computer vision algorithms. These libraries would create a foundation for future work by providing a common infrastructure for researchers and practitioners. OpenCV is widely used in a variety of computer science domains, including medicine, security, and robotics. For example, it was a key part of the Stanford vehicle that won the $2 million DARPA Grand Challenge by driving itself across the desert. We highly recommend the book *Learning OpenCV* for an excellent technical introduction to using OpenCV. It is important to realize, however, that OpenCV is a set of libraries that could be used by a software developer to build an augmented reality application. It is a set of tools, not a pre-packaged platform or application that can be used "out of the box."

ARToolkit, created by Hirokazu Kato of Nara Institute of Science and Technology in 1999, was a seminal development in augmented reality. It uses "markers" (distinctive, easy to process images that are printed and placed in the real world) which the computer can identify and track using a digital camera. The power of this technique is its simplicity. By placing an image in front of the camera that the computer expects and can recognize, the system can determine how the image must be oriented in space for it to appear in its current form and then create a model of where the camera must be placed relative to the "marker." At that point, the computer can add 3D objects to the incoming image, which it shows to the user. This technique works quite well, and we were able to print markers and project 3D images onto the markers (viewed on a computer monitor). As we moved the piece of paper with the "marker," the illusory 3D model would move along with it, creating a powerful illusion. We also assessed a long list of derivative tools that use this technique in the AR arena.

While this technique works in many cases, there are a few critical disadvantages that led us to not pursue computer vision AR technology for our project. The primary issue is that this technology requires a great deal of control over your environment. The most obvious constraint is the necessity of placing the markers in the physical environment (perhaps as a poster) and then creating a 3D model in which your system can know where the markers have been placed. Additionally, slight variations in lighting and surrounding objects can confuse these systems. We had the best success configuring our experiments for a particular goal – a particular marker in a particular place with particular lighting. The algorithms were also not as general purpose as we might have hoped. While the algorithms are extremely powerful, they are also extremely limited. Many impressive results are possible, but the technology is still very far from the advanced uses shown in movies. Despite these limitations, computer vision AR could be particularly exciting for indoor augmented reality experiences where markers could be placed in a building and provide additional visual content based on where the user is standing within the building. Given the fact that phone sensors (like GPS) do not work well indoors, a marker based approach might be one of the only currently feasible approaches for such an installation.

Sensor AR

As discussed in the Devices section, an impressive and powerful array of sensors has been added to everyday mobile phones, making it possible for an application on a mobile device to learn an impressive amount about where the device is and how it is oriented in the world. In sensor AR, the phone uses its sensors to determine where it is and where the camera is looking. The application then queries its source of data about the augmented reality assets to be shown and renders anything that should currently be in the camera's view. Many augmented reality projects use this approach, which is both flexible and powerful. One advantage is that it is straightforward to use a remote source of data since it can be queried over a network. The application queries the data service asking, "What assets are near me, and where should I show them?" Once the application gets a response, it renders a 3D object in the camera's view, based on its understanding of where the camera is looking.

Sensor AR is the most common technique in the available augmented reality platforms, but its power is bounded by the accuracy of the phone's sensors. Each of the commonly available sensors has its own limitations regarding what it can know, knowledge that can be subject to errors. The Global Positioning System (GPS) that has become common in car navigation allows mobile devices to determine roughly where the device is located. The GPS receiver receives timing information from satellites above the earth. Small timing errors cause the GPS

receiver to continually make different assessments of where it is located, causing a user's supposed location to bounce around implausibly. These errors can be particularly bad in the city, where signals can bounce off buildings and do not reliably penetrate the indoors. Compass sensors tell you where you are currently headed, but their output oscillates quickly over a range, like a magnetic needle in a compass quivering back and forth. The phone's accelerometer measures gravity's acceleration force on the phone, which it uses to determine how the phone is oriented. If you shake your phone, however, the accelerometer cannot tell the difference between that acceleration and the effects of gravity. See the later Fixing the Jitter section for more on overcoming these limitations in sensor AR technology.

Custom Application Development

As outlined in the original grant proposal, we wanted to compare the feasibility and advantages of a custom augmented reality development with the use of an existing AR client and platform. Our hope in building a custom application for Android and iOS platforms was that we could use or leverage existing open source AR libraries to substantially speed the development of a proof of concept application. Unfortunately, our review of existing libraries revealed that while there are a wide variety of powerful augmented reality libraries in the "computer vision" sphere -- where 3D objects are placed in the visual plane based on the detection of coded images in that visual plane -- there is not a robust, mature open source solution for GPS-based augmented reality solutions. We evaluated a number of open source projects that aim to implement a sensor-based AR client including mixare, Open Gamaray Project, iPhone ARKit, Locatory, locative, augmentia, and "augment-this!" We also evaluated a range of open source projects that were focused on computer vision applications but weren't appropriate for our needs. Unfortunately, extremely few of the open source projects were being actively developed and only one platform had clients for both iPhone and Android platforms. For our initial research into a custom application, we chose mixare, an open source augmented reality engine, as a base for our development for a few reasons. First, it has both an Android and an iPhone implementation. Second, it contains the basic marker functionality -- points of interest can be loaded from a variety of sources and shown in the camera view. Third, the code is relatively straightforward and well organized. However, while mixare has interesting possibilities, it is not yet an extremely robust or mature application and does not support 3D objects or the newer phone sensors that would improve the overall augmented reality experience by presenting markers in a less "jumpy" and more accurate fashion. Unfortunately, it is no longer being actively developed; the last code check in was in January 2011.

Selecting an AR Framework

After determining that we would use an existing AR framework for the *Philly*History.org AR app, we investigated a variety of proprietary AR technologies before selecting the Layar platform, including Junaio, Wikitude, PanicAR, 3DAR, and Qualcomm's AR platform. Critical criteria for us included support for both Android and iPhone platforms as well as strong support for 3D placement of 2D photos. Given these criteria, only Layar and Junaio were viable candidates. A mobile augmented reality platform developed in The Netherlands and launched in 2009, Layar has quickly become ubiquitous as a platform for augmented reality applications. When we began our research, we were skeptical that Layar was an appropriate platform. It was possible to load thousands of 3D locations into Layar, but users were only able to find the points by going through the Layar user interface and sifting through the thousands of other layer collections. We wanted to enable users to go directly to an app that only contained data from *Philly*History.org. However, in the months since we had originally conceptualized the project and our decision on a framework, the Layar team had made significant enhancements to their platform that would assist us in achieving our project goals. We selected Layar for five key reasons:

1. Support for the placement of 2D images in 3D space without the use of modeling tools
2. Ability to place the Layar layer within our own application
3. Customization and branding opportunities
4. Documentation of technology
5. Regular technology updates

As one of the key companies developing AR technology, Layar provided significant documentation for their platform as well as regularly technology updates. These useful features ensured we would have access to information about the technology and the most current features available. While the regular technology updates made development challenging at time, the recent enhancements, particularly in the support for 3D objects in the form of 2D "billboards" and the customization opportunities, ultimately convinced us to utilize Layar technology in the creation of the *Philly*History.org AR app. The placement of images in 3D space, placing a layer within a custom application, and the customization options are discussed in detail later in this paper. Several shortcomings remained with using Layar, but after examining several options, it was the best available toolkit for our purposes.

Creating Data Services

Once we selected Layar, we needed to decide how we would create data services, meaning how we would provide the data (both text and images) for the AR application. Since the project also included experimenting with multiple client applications (Layar is one type of AR client), we wanted to build an architecture that separated the data services from the client-side augmented reality viewers. Regardless of the client technology that runs on a phone to provide the AR experience, we wanted to create a single source of digital asset information. To implement an augmented reality layer in Layar, one publishes the "augmentations" (the points of interest that are visible in the AR app) by creating a web service that client applications can query for information about what's around them. A web service is simply a term for a standard method of allowing two computer programs to request and communicate data in a structured way. For example, a request to this "augmentation" database might request all of the points of interest within 200 meters of a given latitude and longitude. While the Layar web service format has some limitations, it is relatively simple to implement both server-and client-side support for it. It is not strictly what is often called a "RESTful" web service, which is a lightweight style of web service that in many ways is similar to loading a web page, but the service can be implemented with a simple web application that can read in POST variables. As there is no independent standard for requesting and publishing AR points of interest, the Layar service is as close as we could find. It had the additional advantage that we could directly test the result in the Layar clients available for both the Android and iPhone platforms.

While there are many advantages to this architecture, it is important to remember that it means all imagery is being transmitted to the mobile device while the user is using it. This creates a number of issues. If the user has no or poor connectivity, the application will not be able to load photos. Even under good circumstances, there will be a noticeable delay, and there are restrictions as to how many photos can be sent in a short amount of time over a network. An alternate approach would be to package the asset images with the application and install those images along with the application. While this approach might work well for a custom built application with 100 photos, the storage requirements for a collection of 90,000 photos would make this impractical.

There are challenges involved in any data translation project, but some challenges we faced are specific to creating augmented reality services. As discussed earlier, we used Google Street View as a tool to identify and select the desired angle in 3D space where we wished to place each photograph. However, Google Street View and Layar specify this angle differently. Both Layar and Google Street View represent a viewer facing north with a value of 0 degrees (in Layar this is the "angle" parameter and in Google Street View it is called "yaw"). In Google Street View, however, rotations go clockwise (so 90 degrees is East and -90 degrees is West) whereas rotations in Layar go counter-clockwise (so 90 degrees is West and -90 degrees is East).

We chose to use a spatial database (PostgreSQL database with the PostGIS spatial extensions) to store our assets. A spatial database is designed to store and reason about objects in space. For example, it is possible to ask a spatial database to find the assets that are within a specific distance from the viewer. It is also possible to add a stored procedure to a non-spatial database to make the same query (the Layar wiki documentation provides such a function), but we found that with large numbers of points the optimizations found in a spatial database were necessary for reasonable performance. The creation of a “spatial index” allows the database to limit its

searches very quickly to likely candidates found within the database instead of needing to search through all of the assets in the database. In practice, however, the overall performance of an AR application is limited by the network transmission time far more significantly than the backend server performance.

Some image processing also needs to occur before images can be displayed on the small screen of a mobile device. This is extremely important because we found that clients (such as Layar's client) will silently drop images that did not fit their specification. Because the client will not include the photo for a number of distinct reasons but there is no feedback explaining why the photo was not included, the process of diagnosing missing photos can be tricky. For Layar, the file size of all images must be smaller than 75 KB, and there are specific resolution limitations (e.g. full images in Layar must be less than 640x480). Given that mobile device screens have significantly smaller resolution than 640x480, that resolution is probably much higher than necessary. Additionally, some clients (like Layar) do not support making images transparent. We wanted to provide an option to view the images as either transparent or opaque. It was therefore necessary to set the alpha channel of the photos in a pre-processing step and store transparent as well as opaque versions. For example, using the open source ImageMagick package, the following command line invocation could perform the necessary scaling and transparency conversion on an incoming image stream: "convert - PNG32:- | convert -scale 240x180 -channel Alpha -evaluate Multiply 0.8 output.png".

There are already a number of open source platforms for publishing Layar content, most notably PorPOISe (PHP), djangolayar (Python) and LayarDotNet (C#), with PorPOISe being the most fully featured of the platforms we reviewed. However, PorPOISE lacked some crucial 3D features at the time of our review. The beta release of an online service called Hoppala Augmentation does support 3D layers (http://augmentation.hoppala.eu/), but we were unable to successfully run the 3D service and found the documentation and usability to be underdeveloped. It is certainly necessary to have a full understanding of the Layar protocol to use the Hoppala service (at this point) as the API allows developers to set a range of settings without explanation or checks on invalid or conflicting settings. Given these limitations and our desire to implement our own interactive capabilities and user settings, we developed our own data web services in Python.

Layar and 3D Points of Interest

Along with creating data services, we also needed to ensure that our images could be displayed in 3D space. In the past, Layar offered a 3D-like visualization of nearby points of interest, but it did not in fact support the visualization of 3D objects placed in 3D space. Given that a key goal of our project was to explore the feasibility of placing historic photos in 3D space with an orientation that matched the original perspective of the photographer, we needed the ability to place the 2D photos as a 3D dimensional "billboard" with a specific fixed placement and angle.

Often, augmented reality applications will place an image in the visual plane of the user, but the "billboard" (if you imagine the image or icon as a billboard being viewed by the user) always directly faces the user in a relative rotation rather than maintaining any form of angled orientation. Sometimes a 3D effect is created by changing the apparent placement of these "billboards" at the edges, but we wanted to have a fixed placement of images as if they were real, physical billboards with a specific location and orientation. We were excited to discover that Layar implemented support for 3D models and 2D images as 3D billboards. As a recently developed feature, several known bugs regarding 3D objects are listed in the Layar Player software development kit (SDK), though some were fixed during our development period. Despite the issues, we were able to create 3D "billboards" with our photos that we placed at a specific location, altitude, and angle (an absolute rotation) (Figure 4).

As compared to other similar platforms, the Layar support for 3D objects enables the placement of 2D images in 3D space without the use of 3D modeling tools. In some systems, it is necessary to build a 3D object using a 3D modeling tool, with the leading open source tool being Blender. Once a model has been created, it must be transformed into the appropriate format for one's service. Placing 3D models in Layar, therefore, usually requires both additional tools for 3D model editing and

Layar's own tool to convert standard formats into their own 3D model format. The "billboard" configuration, however, allows for a 2D image to be given an angle in 3D space without conversion to a new format and the requisite use of 3D modeling tools. We chose to use this method when creating the application.

There are some key limitations that should be kept in mind. First of all, we found scale -- the real world size of the image -- to be implemented very inconsistently across Layar clients. Some images were quite small and others were so large they filled the entire screen. In the end, we had to add support in our data services to detect whether requests were coming in from an Android or iOS client and change the "size" parameter based on which client was requesting the images. Secondly, while our Google Street View approach gave us the angle of the photo, we did not track the precise size of the image in 3D space -- so we needed to use a rough estimate. Thirdly, scale is tied to pixel width and height of the photo, which means that scale must be calculated for each photo based on the height and width. Our photos differ in aspect and resolution (pixel height and width) which would require that we calculate scale for each photo to simulate the correct placement of the photo in real world space.

Photos: Josh Marcus

Figure 4: The image on the left shows relative position where the photo always directly faces the user. The image on the right shows absolute positioning where the photo is angled based on additional coordinates.

Challenges in Displaying the Images

After setting up a basic data service that provided for 3D display of select images, we encountered several problems.

1. It was difficult to predict how many points would come back for a given query.

2. As the number of points increased, load times became long enough that it was difficult to be sure the application was working.

3. Some points failed to ever draw, an issue that was not resolved even after fixing the file size and resolution.

4. Points located very near each other caused interference and shearing, a consistent issue as we had many images geocoded to the same location.

The number of images available in our AR app created many of these problems. Few AR applications contain 90,000 images or "augments" in a single layer. With many images geocoded to the same location, we needed to find a way to speed up the application and accurately display the photographs.

After much discussion, we chose to speed up the layer and set the number of points coming back for a given query by only displaying photos for the closest four points. The remaining nearby images were represented by icons, which are faster to load and cached by the Layar client. For points displayed as icons, the photo and description are visible when that point is selected, and the image can be loaded in the AR viewer with the "Load Photo" action we created. This seemed to be the best way to keep layers loading quickly while also letting the user know what images were available.

Discovering why images were not showing up was more difficult. Eventually we learned that the Layar client will not draw 2D images whose viewing angle is incompatible with the viewer (when rendering with absolute position). To visualize this, imagine that the images are a billboard. When the user is behind the billboard (and therefore can't see the ad on the

Figure 5: By calculating the viewing angle, we adjusted the images as necessary to make them more easily visible.

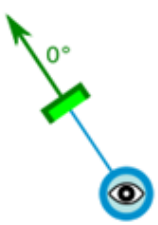

Fig. 1 Viewing at the Intended Angle

Fig. 2 Viewing from the Side at 60° Angle

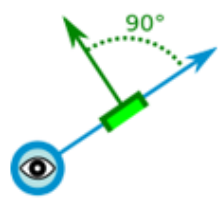

Fig. 3 Viewing the Edge at a 90° Angle

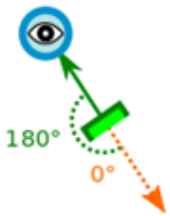

Fig. 4 Viewing from Directly Behind

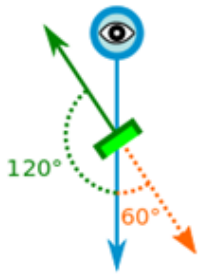

Fig. 5 Viewing from Behind and to the Side

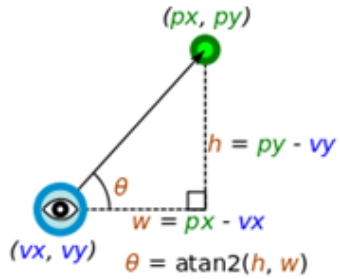

Fig. 6 Calculating the Viewing Angle

front of the billboard), Layar does not display the image or any other hint that there is a point available. A similar problem occurred when the viewer was at a perpendicular angle to the image as the image would appear as a barely visible vertical sliver. In these cases, it is not clear what else Layar could do that would be more correct, and there was no mention of this issue in the documentation nor any options in the layer related to fixing it.

We made all images display by slightly altering the positioning. Since we know the user's position via the GPS coordinates of the phone and the image's position, we can find the user's viewing angle on the point and thus determine if the image will be invisible. In such cases, we decided to reverse the image (as if seen in a mirror) and flip the angle 180 degrees (as if it were pointing toward the viewer instead of away from the viewer). This would mean that landmarks like trees, spires, etc. would be on the side the viewer sees them and the image would be visible. We also decided to give some images a "nudge" to make their angles less sharp. We found that angles sharper than 70 degrees were hard to see, and angles approaching 90 degrees started looking like graphical errors. Figure 5 details some calculations used in adjusting the angles. A more extensive discussion of the code used to adjust the images based on the viewing angle of the user is available in the next section of this paper.

After processing the images and adjusting their angles, all the images appeared as planned. Unfortunately, we now had a different problem as the closest images would often pile up or block each other. When the points were evenly dispersed, we were able to fit ten to twenty photos into the AR viewer. The points were often clustered around a few places, however, such as City Hall or a school where the photographer had taken multiple images of a construction project or other event. Not only did the images block each other, but when they were too close they would cause visual interference and tearing as they became spliced together.

Initially, we tried to solve this problem by making sure all points were more than a certain distance (e.g. 1 meter) apart. If two points were too close together, we'd take the nearest one and drop the one further away. This helped, especially when images were close to the viewer, but we still had the problem that close images obscured other points behind them. Finally, we created a bounding circle around each point (the 1 meter distance) but also calculated a 20-degree viewing "slice" which the point claimed. Any additional points located in that 20-degree slice would be dropped and not available for viewing (Figure 6). This enabled us to return up to 16 points which are near the user but are guaranteed not to block each other or cause display issues.

Although we had solved the issue of the overlapping photos, we also had created more questions. If only a certain number of images were shown for a location, how did we prioritize what was shown, especially with nearly 90,000 possible images? Fortunately, the images fell cleanly into three categories that made prioritization fairly straightforward. The application had a small number (20) of photos considered especially important which were accompanied by additional text, a larger number (500) of photos which had been more accurately positioned in 3D space, and a very large number of photos (around 87,000) which made up the remainder of the *Philly*History.org photo collection for which location information was available. In prioritization discussions, we referred to these as gold, silver, and bronze photos.

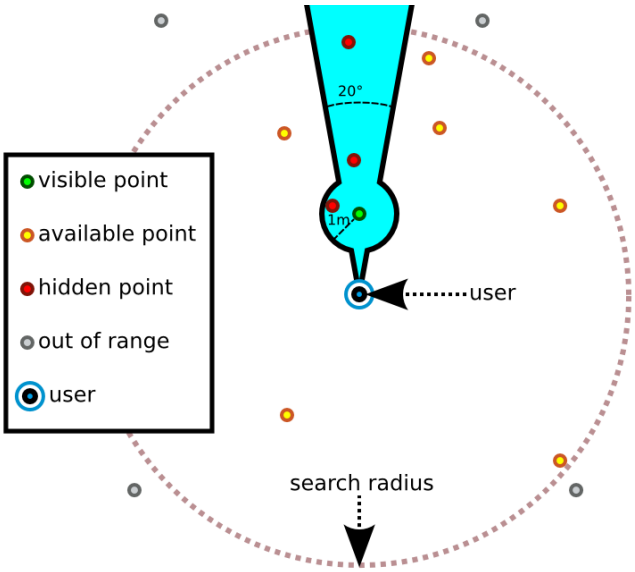

Figure 6: Only a select number of images can be displayed within a search radius to prevent display issues related to images appearing at the same location.

Early tests with only gold and silver photos showed there were many parts of Philadelphia where we did not have complete enough coverage, which prompted us to test including the entire collection. However, many of the bronze photos may lack extensive descriptive text or conventionally interesting subject matter. For a given search radius then, we'd like to show gold and silver photos if possible. When necessary, we can "pad out" the results with bronze photos, and for some searches, the results will be entirely bronze photos.

We decided to sort the points first by priority (where gold had the highest priority and bronze had the lowest) and then by distance. If any gold points would be returned in a given search radius, we made sure to try to place them. We used the same rules as described earlier (using proximity and viewing slices) but with the catch that a more distant gold point might block a nearby silver or bronze photo from displaying. Since these were the photos we had spent time selecting and describing, it seemed like a good trade off. With additional resources, we could have created more priority levels and distributed the large bronze group into many smaller groups (e.g. ranking pictures of houses higher than pictures of sidewalks).

Technical Detail: Walkthrough of Viewing Angle Calculations

In this section, we describe in more technical detail some of the code that handles viewing angles and the adjustments described in the previous section that provide for a better viewing experience. We have included python implementation code to serve as a guide to other software developers.

First, we must compute the viewing angle to a point of interest. We can accomplish this by dusting off a little trigonometry. Imagine that viewer (vx, vy) and the point of interest (px, py) form a right triangle. In this case, we want to compute the angle at the viewer point, which we can do with atan2 given the lengths of the opposite and adjacent sides (which end up being py - vy and px - vy, respectively). The resulting angle will be 0 when facing East (when px and py are the same) and proceed counter-clockwise (so pi/2 is North, pi/-pi is South and -pi/2 is West). We convert this into the form that Layar uses (where 0 is North, -90 is East, 180/-180 is South, and 90 is West).

```
def get_angle(vx, vy, px, py):

    # find the angle in pi radians
    # (-pi to pi)
    theta = math.atan2(py - vy, px - vx)

    # convert from pi radians to degrees
    # (-180 to 180)
    degrees = (theta * 180.0) / math.pi

    # return the viewing angle relative
    # to the positive-Y axis
    return degrees - 90
```

We will also use the standard Euclidian distance function to calculate how close points are to each other.

```
def get_distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    return math.sqrt(dx ** 2 + dy ** 2)
```

When photographs are visible at a particular angle, we need to determine how much difference there is between the viewing angle the user has on a point and the angle at which the photograph should be viewed. In this case, a difference of 0 would mean the user is viewing the photograph at the exact angle at which it was taken, 180 would mean that the user is behind the photograph, and 90 would mean that the user is at a right angle to the photograph. Since there is a point (180/-180) where angles wrap around, it's important to handle this correctly. For instance, -160 and 179 are only 21 degrees apart. We can use modular arithmetic to normalize angles to 0-360. Here is an implementation:

```
def angle_diff(angle1, angle2):

    # calculate the difference between
    # angle1 and angle2 in degrees
    # this value will range from 0-360

    diff = abs(angle1 % 360 - angle2 % 360)
```

```
    # if the difference between angle1
    # and angle2 is more than 180
    # degrees we should instead return
    # the different between angle2 and
    # angle1 which will be less than 180
    # degrees.

    if diff > 180:
        return 360 - diff
    else:
        return diff
```

When an angle is too close to a 90 degree angle, the image won't be visible; in these cases we can soften the angle so the user can see the image better. This function will nudge the start angle closer to the goal angle by a given number of degrees (amount):

```
def nudge_angle(start, goal, amount):

    # calculate the difference between
    # start and goal in degrees.
    # this value will range from 0-360
    diff = abs(start % 360 - goal % 360)

    # if the diff is less than the nudge
    # amount, use that instead.
    if diff < amount:
        amount = diff

    # here we figure out whether we need
    # to subtract or add diff.
    # if start is greater than end then
    # we will be subtracting diff,
    # and otherwise we will be adding
    # diff.
    subtract = start % 360 > goal % 360

    # however, if diff is greater than
    # 180 then we need to reverse our
    # previous decision (because going
    # the other direction means the
    # difference is less than 180).
    opposite = diff > 180
    if opposite:
        subtract = not subtract

    # add or subtract the correct amount
    # and return the new angle.
    if subtract:
        return start - amount
    else:
        return start + amount
```

We can put this all together to implement our strategy for dealing with oblique angles and image flipping. The code could be more terse, but it's easy to get the math wrong so we try to do things in a well-commented procedural way.

```
# points are represented as (x, y)
# tuples in web mercator.
   # angles are given in degrees
   # counter-clockwise from North, as
   # earlier.

def calc_angle(self, viewer_pt, point_pt,
img_angle):
       vx, vy = viewer_pt
       px, py = point_pt

   # calculate the angle the viewer is
   # pointed when seeing the point.
   angle = get_angle(vx, vy, px, py)

   # compute the difference between the
   # angle the viewer is pointed
   # when looking at the point, and the
   # direction the point's photograph
   # should be faced from.
   diff = angle_diff(angle, img_angle)

   # possibly flip the image and angle
   # if the viewer is behind the photo.
   if abs(diff) <= 90.0:
       # use the angle as-is
       flipped = False
```

```
else:
    # flip the image and the angle
    flipped = True
    angle = angle + 180.0

    # recalculate the new viewer
    # angle
    diff = angle_diff(angle,
    img_angle)

# soften angles that are within
# wiggle degrees of img_angle
wiggle = 15
if diff2 >= 90 - wiggle:
    angle = nudge_angle(angle,
    img_angle, wiggle)

# return the angle the image is
# viewed at, and whether we flipped
# the photo or not
return angle, flipped
```

The result should enable users to have a better viewing experience in the AR app.

Branding Opportunities

In addition to the ability to display images as 3D overlays, we also wanted to create an application that we could customize and brand as much as possible. We harbored a number of concerns regarding the Layar technology in terms of how users accessed an individual layer and the options for branding and skinning a layer. A layer refers to the series of augmented reality points on a particular topic, location, or event that have been compiled by an organization. Developers can build an individual layer using the Layar framework which is then accessible using the free Layar reality browser. When we initially reviewed the Layar infrastructure, it required the user to access a particular institution's augmented reality layer by first searching for it in the Layar mobile app. The resulting layer contained the assets selected by the institution but did not include many opportunities for skinning or packaging the layer so that it was easily identifiable as connected to the institution. Since *Philly*History.org is a project that exists almost solely as an online presence, it was crucial that our users could quickly identify our augmented reality prototype as being connected to the *Philly*History.org website. A recent Layar update, however, both extended the technical capabilities of the platform and provided additional branding and packaging options that allayed some of our concerns.

Our first challenge concerned how users would find and view our AR points. Requiring users to first access materials through the Layar app made locating an institution's augmented reality layer a multiple step process that might confuse users. Even a willing user might not understand each of the intermediate steps or why the application was branded 'Layar' with no reference to the content the user thought they were trying to load. But two developments -- one publicly released during the period of our research project -- have removed several of these steps and made Layar much more customizable. On the Android platform, it is possible to create an application with one's own branding and content that launches the user into Layar. The app prompts the user to download Layar via the Android Market if necessary and provides a more guided process for accessing the application. The user experience includes the following steps:

1. User downloads the AR application from the Android Market. The application has its own branded icon and any desired custom content such as an informational splash screen.

2. The app then sends the user directly to the layer in Layar at a designated point such as when the user taps a button or after a certain number of seconds.

3. The user is prompted to download Layar if necessary. Depending on the phone the user is using, he or she will either be returned to the same place in the AR application or need to restart the application. If a user has Layar, the Layar logo fills the screen as the Layar application loads.

4. The user is then brought to the 'settings' screen of the Layar layer which can include up to five custom parameters. After changing or agreeing to the parameters, they can then view the AR layer created by the organization.

While this method of accessing an institution's AR app requires custom Android platform development, an application called the "Layar Launcher Creator" (documented on the Layar wiki at http://layar.pbworks.com/w/page/28695224/Layar-shortcut-tool) creates the basic skeleton of an Android application that launches a Layar layer. The core technology is the Android "intent" that would also allow the inclusion of a web page link directly to a Layar layer if the Layar application has been installed. While not perfect, this functionality enables limited branding that can frame the user experience and significantly simplify the process of bringing a user into the augmented reality experience – a key requirement for the *Philly*History.org augmented reality application.

On January 27, 2011, the "Layar Player" service for the iPhone platform launched. The service contains a software development kit (SDK) for including Layar as a component within a custom application, similar to the Layar launcher for Android. The "Layar Player" includes some features that were not previously available such as the ability to provide information from your external application in the requests from the Layar client to the web services that provide the point of interest information. This SDK also obviates the need for the user to download Layar as a separate application. Overall, the idea is very similar to the Android launcher -- one can 'sandwich' the Layar experience with one's own branding and content.

As part of placing the Layar experience within the framework of our own application, we created an informational splash screen that users would view upon downloading and launching our app, which we named "Augmented Reality by *Philly*History.org." The screen includes options for viewing help text, reading credit information, and launching the AR layer (Figure 7).

While this screen was a useful tool for introducing the app to users (many of whom might not have prior experience with AR), it required significant design work. Graphics requirements varied depending on the screen size and type of phone running the application, requiring the graphic designer to create multiple versions of each element of the splash screen and launch button. The help and credits pages also required styling for mobile viewing. Despite the time commitment, we felt the creation of our own application to surround the Layar layer was absolutely necessary to providing the easiest user experience.

Branding our content in the app was our second concern with using Layar. As much as possible, we wanted to set the Layar site to use the colors and other branding connected with *Philly*History.org. While several design elements including the location of thumbnail images, text fields, and radar are set, Layar provides customization options in terms of color, icons, and labeling. A full listing of the customizable design elements is available on the Layar site at http://layar.pbworks.com/w/page/30676821/Edit-a-layer#Lookandfeel. Adjusting colors and icons can create a strong visual connection to the brand of an organization and provided the connection we desired between the AR app and the *Philly*History.org website. Figure 8 includes an overview of the custom color skinning we added to our AR app.

Figure 7: The information splash screen guides users through the process of using the app.

Figure 8: A diagram of the color customizations available in Layar. The logo featured in the bottom banner and the icons indicating an AR point are also customizable.

Perhaps the most time intensive element of skinning the AR application was creating all the required icons and images in multiple versions. Layar refers to the icon indicating an augmented reality asset as a point of interest or POI. A custom icon for the POI, referred to as a Custom POI Indicator Widget (CIW), can replace the standard POI which is a colored circle. We created a CIW that included a PH (for *Philly*History) within a circle, similar to the icon we used for assets displaying in Google Earth. Each CIW needed to be created in four states. The **outer**, **middle**, and **inner** versions were displayed depending on how close the user was to the point. The **focus** version was displayed when the user selected the asset.

Due to the nearly 90,000 images visible in our AR app, we chose to create three different CIW – all similar in design but with color variations – to provide guidance to users in identifying the different types of assets available in the app. A white icon indicated the full collection of nearly 90,000 images, a gold icon indicated the 500 "pinned" photos, and a red icon indicated the 20 images which included accompanying text created by local scholars and the editors of The Encyclopedia of Greater Philadelphia (Figure 9). Each of the icons needed to be created in four states thus requiring a total of twelve icons.

The ability to brand the AR layer by framing it in our own application and customizing the icons and colors was an important reason we chose to use Layar for our AR framework. However, creating our own application around the layer and implementing significant customization was time consuming and required development and graphic design skills that may not be available at all cultural institutions. We feel that the time commitment was worth it, ultimately, as the result was an application that could be published in both iTunes and the Android Market. Users could thus more easily locate the Augmented Reality by *Philly*History.org application and identify it as a project of the Philadelphia City Archives and Department of Records.

Figure 9: Different icons indicate the various types of assets available in the application.

Layar Limitations

While we were pleased with our selection of Layar for use in the *Philly*History.org AR app, we also encountered several key limitations.

1. Although some configuration is possible, fully custom controls are neither available nor allowed even when using Layar as an integrated SDK. This prevents the creation of a custom experience or interactions beyond what the Layar clients already offer.

2. Layar is a platform in rapid development and bugs are not uncommon.

3. While it is possible to integrate Layar into your own application, the integration is not seamless and users will still see a full-screen Layar logo before reaching your content.

4. Debugging is complex. Issues frequently arose when we could not debug directly, such as when Layar had cached our content and didn't seem to be querying the database on our servers.

During our application development period, there were service outages, API changes, and very significant bugs. Layar is in rapid development, however, and new features and bug fixes were also rolled out during our development period. Their technical support was also relatively responsive and documentation is available on many topics. When developing on the Layar framework, however, it is important to remember that the platform is not bug free or and is rapidly changing and developing. This has long-term support implications as an application released on the Layar platform will not necessarily behave in the same manner from one release of the toolkit to the next.

Fixing the Jitter

An unfortunate fact of most augmented reality applications is that screenshots or mockups of an application give a much better impression of the functionality than actual use demonstrates. In the real world, augmented reality applications are often frustrating. Images do not inhabit a fixed place in your view of the world around you; they wobble, bounce, jitter, fly away, or disappear entirely. Sensors on mobile devices still have very significant limitations. In addition to the other research described above, we explored the feasibility of a next generation of augmented reality applications that could more successfully create the illusion of a virtual object in a fixed location in 3D space. Some newer phones, like the iPhone 4 and some Android-based phones like the Google Nexus S, include a gyroscope that can be leveraged to create a more stable understanding of the phone's orientation in 3D space. While the gyroscope has its own limitations and errors, there are techniques to integrate the gyroscope's data with the data from the accelerometer, compass, and GPS.

While developing our custom augmented reality application, we extensively explored modifying existing open source augmented reality frameworks to use the gyroscope to improve the overall user experience. We were able to identify a promising approach in which the compass and GPS are only occasionally sampled to prevent the inconsistency of GPS readings from suddenly shifting the user's location in a way that disrupts the illusion. The gyroscope and accelerometer sensors are also used in an integrated fashion to establish a less frustrating and more consistent user experience. While we found the documentation regarding the use of the Android and iOS gyroscopes to be incomplete, it is possible to model the phone's six degrees of freedom (Figure 10) by mathematically combining or comparing the results from the phone's gyroscope, accelerometer, and compass using methods ranging from the complicated (e.g. using a kalman filter) to the relatively simple (e.g. high & low pass filters, simple weighted averages over time). Unfortunately, this type of gyroscope support has not been integrated into the commercial or open source clients we investigated and would be a significant contribution to an open source AR client in the future.

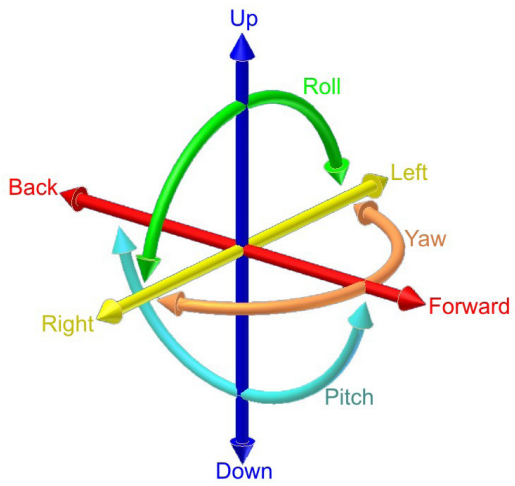

Figure 10: Visualization of the six degrees of freedom available with a gyroscope. Source: Public Domain (Wikimedia Commons at http://en.wikipedia.org/wiki/File:6DOF_en.jpg**).**

There are many technical challenges in using gyroscopes and other sensors to display AR materials. The best sensor correlation techniques are complicated, use of the newer sensors is often not well or accurately documented, and none of the existing frameworks we identified provided a working example on which to base our own work. There are a variety of small challenges as well: the sensors provide data at different rates, the variety of hardware on different phones necessitates separate testing, and the APIs are often restrictive and poorly documented. There are also necessary decisions to be made based on the goals of the application including the trade-off between smoothness and speed of response.

While we were able to demonstrate the feasibility of a custom application approach in our initial proof of concept development for the Android and iOS platforms, we decided to build our full application development on top of an existing AR client. In the absence of a fully developed open source augmented reality resource, creating a custom application was ultimately not possible due to the limited timeframe for the project, the number of developers available to work on the application, and fiscal constraints. We do feel, however, that further work in the area of sensor research would make an extremely valuable contribution to an open source augmented reality application and possibly make future custom AR application development both feasible and more successful.

Research Findings

While creating the Augmented Reality by *Philly*History.org application, we learned much about what is and is not possible. We succeeded in building many features and were frustrated by an inability to build others. Creating an augmented reality application can be fairly straightforward (such as creating a layer in Layar) or much more complex (displaying a large number of images in 3D space). A few of our summarized research findings are below.

Choices are Numerous

Beyond simply choosing an AR framework such as Layar lie myriad decisions that must be made in creating an AR app. How many points should be included? Should points be represented with floating information bubbles, billboards, markers, or some other graphic? In what geographic area can users expect to get results? What information is necessary for points to display properly? How many points is the user expecting to see? What methods should be available for the user to filter or search the possible results? How much text should be displayed?

Questions of what to include are common in any exhibition or digital humanities project, but it is important to recognize that the same issues apply to an AR project for viewing images. What may seem as simple as putting a few points into a Layar layer can be more complex than it initially appears. As with all projects, gathering user feedback and focusing on how to create the best user experience in terms of design, interaction, and display of information is crucial.

The Type and Number of Assets Influences Development

Mobile AR applications require assets that are geographic in nature. Each item (photo, point, object, etc.) must have at least a latitude and longitude and be connected to a location. Having more than one asset connected to a location can make display of both assets difficult as we described previously. Questions of display and user interface design become even more complex when dealing with a large collection of assets. The amount and quality of data associated with any given point may vary widely, complicating the representation of those

points. In working with a collection of nearly 90,000 images, we found it necessary to create groups of points with a priority ranking and have the interfaces vary depending on the group to which the point belonged. We wish we could more easily display all the assets at a particular location but the available technology does not allow for that option. While there are solutions to creating an app with a large number of assets or assets associated with the same place, it may require advanced development knowledge.

Geography is Key

The location and density of assets in an AR app has a great influence on the user experience. Applications with a small number of points can assure a fairly constant density of points throughout the coverage area. If the coverage area expands, however, the number of points may increase in a way that causes density to vary wildly. In areas with sparse asset coverage, users may be disappointed if they can't find results, and in areas with dense coverage, users will have trouble navigating through clusters of points to find interesting content. In these cases, prioritizing points or limiting the number of available points is important, although creating useful priorities and auditing the collection can be daunting tasks. While we chose to place points within categories, adding interfaces for searching and filtering points represents another way to deal with point density.

Data Services

With larger collections, there is also the question of how to get the data to the user. A small collection of assets could easily be embedded in an AR application, whereas this is impossible with a large collection. Querying a web service to download information about points and image data, the approach Layar uses, can scale to large asset collections but also introduces significant latency and fails without a good data connection. This strategy also introduces more failure points (web service delays or crashes will affect the application). In general, loading a large number of assets or images can cause displays that will negatively affect the user experience.

Inaccurate Locations and Jittery Images

Augmented reality blends images of the real world with other kinds of image data (in our case historical photographs). The goal is to produce a seamless experience where the collection's objects seem to take on their own reality and behave as if they are physically rooted to a place. Unfortunately, this illusion is difficult to create and maintain, and the hardware and software we worked with could be improved. One of the biggest problems AR developers contend with is inaccurate GPS readings. In many locations, a phone's GPS will return results with low accuracy, particularly in cities where large buildings can create shadows and bounce signals around inaccurately. Furthermore, the GPS sensor is constantly updating its idea of the user's location, creating more jitter which needs to be smoothed. The accelerometer and compass jitter meant that our images were often floating and moving around the view screen, even when the user was holding the phone still. Some of this could be corrected by using better filters in software, but without using a gyroscope, it is impossible to accurately detect and respond to fine movements and rotation. Some newer smart phones (iPhone 4, Android Nexus S) contain gyroscopes but most existing smart phones do not have these sensors. In our experience, it is often better to make a "best guess" about where the user is and then stick with that location.

Phone Screens are Small

While contemporary smart phones have significantly larger screens than previous earlier phones, they remain a small space on which to overlay a lot of information on top of the camera view. Although the small screen size is an important limitation of all AR applications aimed at smart phones, the newest crop of tablet devices (iPad, Android 2.3 tablets, Blackberry Playbook, etc.) have all of the same sensors as the smart phone devices while adding a much larger screen. These devices will likely provide a more suitable target for future AR applications.

Users are Enthusiastic about History in AR

Historical collections, whether they are images, objects, stories, or something else, lend themselves well to an augmented reality project. If AR is about combining something physical with something digital, then the combination of a view of the physical world with a digital representation of the past makes history well-suited to a new life in augmented reality. We received much positive feedback about our project, both when it was in development and after the app was available. Despite any limitations, users are still wowed by augmented reality applications and excited to access them and learn more about the images or other collections (Figure 11).

Figure 11: The Augmented Reality by *Philly*History.org application enables users to learn more about the history of City Hall in Philadelphia.

Creating an Open Source AR Client for the Humanities

Over the course of this project, it became clear to us that there is a significant need for shared, freely available tools for the creation of augmented reality tools and applications for the humanities. While Layar, the leading commercial provider of AR platforms and clients, has a strong product and a quickly growing set of features, not all humanities applications will fit into a "one size fits all" model for augmented reality. Just as not all websites need the same functionality and layout, the specific goals and content of a particular augmented reality application suggest customizations and content that go beyond what a single proprietary client can offer. While working with the Layar client, we often wished we could integrate the innovations we designed for our custom application experiments including help content and transparency sliders. Integrated sensor calculations by using the gyroscope in addition to the sensors currently used would also provide significant improvements in the user experience.

While greater control of the platform's behavior would be ideal, the development of a core application framework for AR is a very significant amount of work and beyond the available resources for the implementation of any one specific application. A common, open source platform (with code that would be freely shared and maintained by a community of developers and users) would provide a strong foundation for the development of custom applications of individual institutions, while allowing new innovations and customizations to be shared in an ongoing manner. Also critical to an effort such as this would be the formulation of a documented standard for the publishing of augmented reality assets on the web and the protocol used by backend data services. This would allow for multiple, alternative clients to be created that could leverage the same services. While an open source client could support proprietary protocols, it might not be sustainable to rely on protocols that could be changed at any time or burdened with licensing requirements.

An open source framework could provide options for interesting tradeoffs based on the needs of the humanities project. Is it more important to create an illusion of a fixed object in space or to have the greatest degree of accuracy? Is it more useful to view assets in the AR app first or on a map or list first? An interesting observation from our field testing was that the List and Map functionality in Layar (where the assets were shown in a simple list or on a map) was often more useful than the standard camera view. The List view provides a list of nearby photos with accompanying thumbnails as well as dynamically updating a distance to each of the nearest photos. While a very different user experience from the standard camera view, it evokes a feeling not unlike using a metal detector with the distance counting down as one moves nearer to a photo location. Is additional text necessary for a point or additional images of that point? How can users contribute materials to the application? The kind of customizations and user interactions that are

possible with proprietary platforms are extremely limited. An open platform that could be directly modified would make a much larger range of experiences possible and meet the needs of future, varied humanities augmented reality projects.

Conclusion

In April 2011, the Augmented Reality by *Philly*History.org application was released for public use via a free download from iTunes or Android Market. We are excited to hear the public reaction to the application and hope that it serves as a useful and enjoyable tool to assist users in experiencing the connection between the past and present.

Creating the app provided us with the chance to investigate many of the technical issues connected to displaying historic photos as overlays on a view of the current landscape. Perhaps more importantly than learning what can be done with augmented reality, we discovered where future research needs to be conducted and additional features developed. The technology is changing rapidly and new features and better options become available regularly. With the increasing technological capabilities, augmented reality applications have many potential uses in cultural institutions. As a technology that still makes users gape in amazement, augmented reality projects can introduce an organization and its collections and mission to new audiences and create a variety of additional educational and interactive opportunities. It is our hope that other cultural institutions will build upon our research and the projects done by other groups to create more innovative and useful augmented reality experiences.

Resources

1 Boyer, D. (2011, January 17). Augmented Reality Coming Soon! In *Azavea Atlas*. http://www.azavea.com/blogs/atlas/2011/01/augmented-reality-coming-soon/

2 Boyer, D. and J. Marcus. Implementing Mobile Augmented Reality Applications for Cultural Institutions . In J. Trant and D. Bearman (eds). *Museums and the Web 2011: Proceedings*. Toronto: Archives & Museum Informatics. Published March 31, 2011. http://conference.archimuse.com/mw2011/papers/implementing_mobile_augmented_reality_applicat

3 Bradski, G. and A. Kaehler. (2008). Learning OpenCV. Sebastopol, CA: O'Reilly Media, Inc.

4 "Historic Overlays on Smart Phone." National Endowment for the Humanities Office of Digital Humanities Library of Funded Projects. http://www.neh.gov/ODH/Default.aspx?tabid=111&id=152.

5 Johnson, L., Witchey, H., Smith, R., Levine, A., and Haywood, K. (2010). *The 2010 Horizon Report: Museum Edition*. Austin, Texas: The New Media Consortium. http://www.nmc.org/pdf/2010-Horizon-Report-Museum.pdf

6 Marcus, J. (2011, January 24). *Philly*History Augmented Reality: Developer Journal 1. In Azavea Labs. http://www.azavea.com/blogs/labs/2011/01/phillyhistory-augmented-reality-developer-journal-1/

7 Marcus, J. (2011, April 5). *Philly*History Augmented Reality Journal 2: Building Data Services. In Azavea Labs. http://www.azavea.com/blogs/labs/2011/04/phillyhistory-augmented-reality-journal-2-building-data-services/

8 Osheim, E. (2011, April 5). *Philly*History Augmented Reality Journal 3: 2D Billboards in Layar. In Azavea Labs. http://www.azavea.com/blogs/labs/2011/04/using-2d-billboards-in-layar-2/

9 Wang, X. (2010). "Edit a layer." On Layar.pbworks.com. Last edited April 8, 2011. http://layar.pbworks.com/w/page/30676821/Edit-a-layer

10 Wang, X. (2010). "First Layar Tutorial – Create a simple layer." On Layar.pbworks.com. Last edited January 19, 2011. http://layar.pbworks.com/w/page/30832324/First%20Layar%20Tutorial%20-%20Create%20a%20simple%20layer