Cal Poly Xpress Project

Exploring Interactive Storytelling through Digital Multimedia Platforms

Eugene Bistolas
June 2012

Advisor: Dr. Michael Haungs, Dr. Aaron Keen

Cal Poly XPress

Project Overview

The primary goal of the Cal Poly XPress project is to deliver a framework for immersive, interactive storytelling and educational material in a mobile environment. The Cal Poly XPress (CPXpress) project acts as a new creative tool for educators and storytellers, unique from traditional mediums utilized in the past. Leveraging today's powerful new interaction technologies, CPXpress allows creative individuals to deliver new engaging and simulating experiences to their audiences that were previously not achievable.

The ever-increasing power of mobile devices has seen the rise of multimedia applications that serve similar purposes as CPXpress; however, they are all bounded to the confines of the mobile device they run on. Additionally, these applications tend to be single serving, catering to the direct needs of one author while not being openly available for other's use. CPXpress differentiates itself from these applications in two ways. Firstly, the user interactions within the CPXpress framework are directly tied to the user's environment, utilizing geolocation and augmented reality technologies to drive the creative process. Secondly, the CPXpress framework is free and open to all, providing an easy to use, open schema for defining new stories or material.

CPXpress leverages a wide variety of multimedia and sensor technologies to achieve its goals. Modern mobile devices have a variety of input mechanisms and sensors, such as cameras, touch screens, GPS modules, digital compasses, and gyroscopes, all of which are exposed by the framework to provide creative tools for content authors. The tools provided to content authors are general enough to support an enormous variety of potential material, such as interactive stories, tour guides, augmented reality games, and more, only bounded by the imagination of the author.

The primary plot drive behind the majority of CPXpress applications hinges on geolocation-based events, allowing the user to organically explore his or her environment, gaining new information or knowledge based on the locations the user encounters. This puts importance on environmental interaction first, augmenting the user's exploration process instead of requiring direct and constant interaction with the mobile device. These events can have additional interactions attached to them such as modifying how the event is triggered, such as time-based filtering, or an action to take after the user encounters it.

CPXpress provides a wide variety of actions that can be performed when the user encounters one of the aforementioned geolocation-based events, catering to the creative needs of the content creator. The most common type of event trigger is multimedia playback, leveraging the powerful multimedia capabilities of the mobile device to present information or drive the story forwards. These triggers include images, text, video or audio playback, or any combination of these three. In addition, the device's camera can be utilized for event triggers as well. The first trigger type allows the user to take a static image of something in their environment, often prompted by some clue or text created by the author. The image taken can then have a variety of filters applied to it, as well as text overlays and other effects, which provide additional information to the end user. More interestingly, the camera, when used in combination with the device's internal gyroscope, accelerometer, and compass, can provide an augmented reality view of the user's environment, injecting in real time an image, text, or 3D model over a live view of the user's surroundings.

Content creators also have a few other non-multimedia, but still essential tools at their disposal. The first is the concept of a "Lockbox". The Lockbox provides an end goal for the user for mystery solving or story based content, bridging together information learned from several geolocation-based events and their triggers to solve an overall question or riddle. Content created for this framework can contain multiple Lockboxes in sequence, unlocking each of these Lockboxes provides a clue to the next, or presents more events to be explored therefore driving the plot forwards. The proper use of these Lockboxes can help engage the user and provide

structure to the information presented in the story. In addition, the content author has access to a journal, whose entries are comprised of events the user encounters during their exploration of the environment. These journal entries can be used as a record-keeping device, allowing the user to go back and revisit events already encountered, or as a place for the user to gain different, supplemental information about the event, perhaps a clue or other piece of information needed to unlock a Lockbox. These journals are unique for every story stored on the device, and persistent across all devices the user may run CPXpress on.

The CPXpress framework provides a wide variety of tools for content authors; however, the content creation capabilities are not present on the mobile device itself. Instead, the mobile device is utilized as a playback mechanism for pre-made content that is downloaded or otherwise transferred to the device. Content creation is done via a web-based authoring tool, providing authors the interface necessary to create new stories to be downloaded to devices.

Cal Poly Xpress Walk-Through

The CalPoly Xpress mobile application contains a myriad of features available to content authors, as well as a user interface designed with a quality user experience in mind. Below is a brief walkthrough of the different functions of the CalPoly Xpress mobile application. See the Software Design section of this document for the technical aspects behind each of these functions.

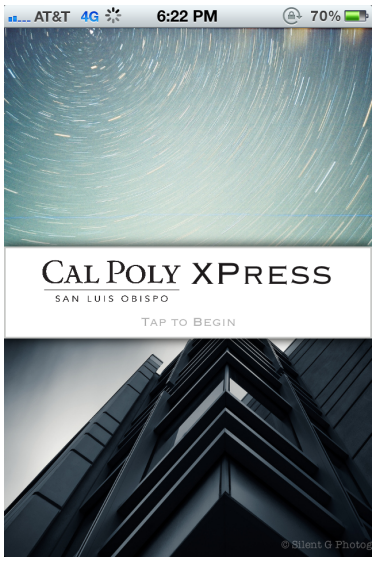


Figure 1: Initial Login View

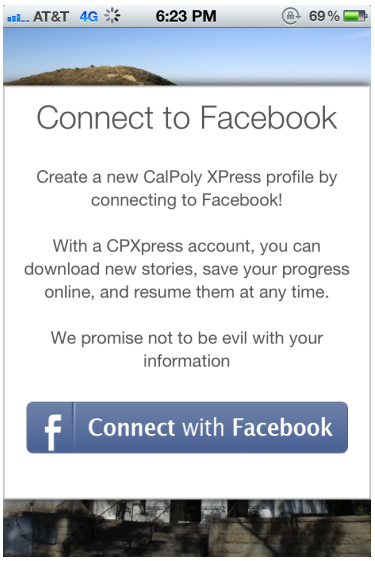


Figure 2: Connecting to Facebook

Figure 3: Facebook Authentication

Login View: This is the first view presented to the user upon application start, as seen in Figure 1. The images seen in the screenshot above cycle in a two second interval, downloaded from the CPXPress backend. If this is the first time the user has launched the app, the user will be prompted to log in via Facebook, as seen in Figure 2. This redirects the user to the Facebook CPXpress mobile application authentication page, as seen in Figure 3. After the user's credentials are verified, the application transitions to the myth selection view.

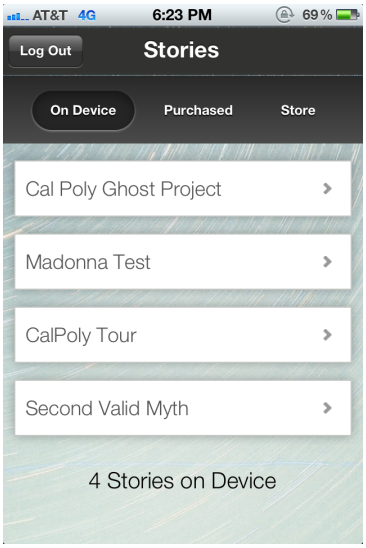


Figure 4: Story Selection View

Myth Selection View: After the user logs into the application, the myth selection screen is presented as shown in Figure 4. It acts as a launching point for entering the stories already on device. The tab bar at the top of the view allows the user to switch between Myths already loaded on the device, Myths purchased but currently stored in the cloud, and a Myth store to purchase and download new stories. The latter two options are not yet implemented

Figure 5: Chapter Selection View

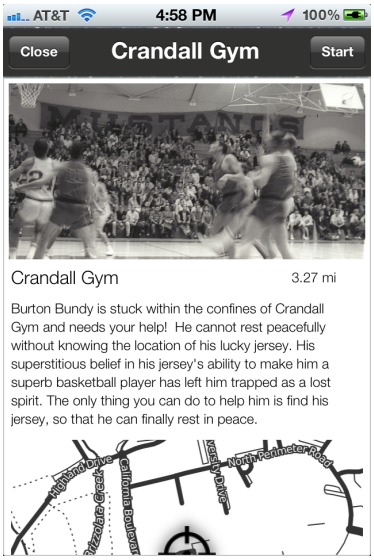


Figure 6: Chapter Preview

Shade Selection View: The shade selection view lists all the shades present in the myth, as well as pertinent details about each shade, visible in Figure 5. To the user, shades act like chapters in the story they're about to pursue, breaking down a potentially large amount of content into smaller, more playable bites. The shade selection view also contains a button to the story's Journal, allowing the user to go back and investigate tasks in this story that have already been completed. Upon pressing one of the shades in the list, the user is presented with a shade detail screen, which presents descriptive and geographic information about the shade as well as a start button to begin playing. This is shown in Figure 6.

Figure 7: Shade Exploration View

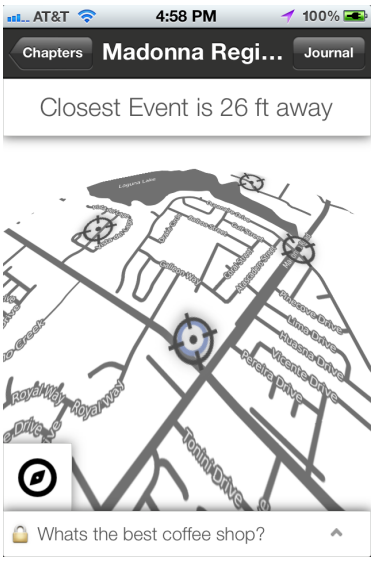


Figure 8: Perspective Shade Exploration View

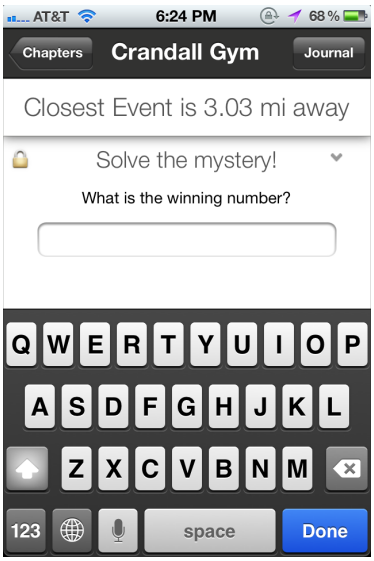


Figure 9: Lockbox View

Shade Exploration View: The shade exploration view provides the main gameplay element for the CPXpress mobile application, and is the view users interact with the most. The dominating central element in this view is the map, which provides the user with information on events yet to be discovered as well as previously triggered events as well as seen on Figure 7. Undiscovered events fade in and out as a function of the user's distance to the target event, to add a more organic explorative aspect (compared to displaying all undiscovered events at one time). The black arrows visible on the first image point to events that have not yet been discovered and are not currently in the map's field of view. Since the primary map view is locked to a high zoom factor, the "perspective" button on the bottom left hand side of the screen provides a 3D "perspective" like view, zooming out the map and plotting additional triggerable events on the map for the user to explore, as seen in Figure 8. Additionally, the user is supplied with the distance to the nearest undiscovered event, giving another subtle navigation clue for organic discovery.

The bottom of the view contains the current lockbox question for the shade. Pressing the question pops up the lockbox view, allowing the user to enter their answer as derived from events they've triggered as seen in Figure 9. If they get the question incorrect, a hint (as defined by the content author) appears, if its answered correctly, the lockbox moves on to the next one in the shade until all lockboxes are answered.

Event Views: The CPXpress mobile application contains a myriad of event views, which are presented when the user discovers a new element in a shade. These event views are designed to provide a wide range of multimedia options for the user, and are broken down into several different types to serve this purpose.

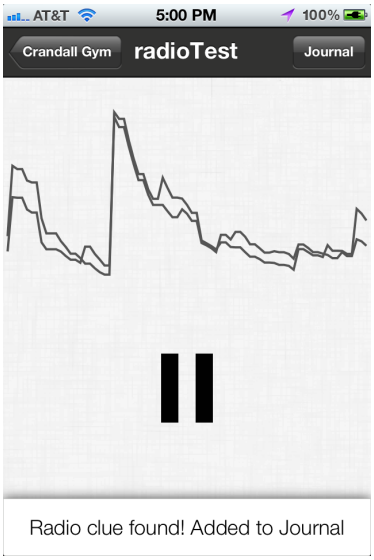


Figure 10: Radio View

Radio Event View: The Radio event view is responsible for the visualization and playback of RadioEvents.

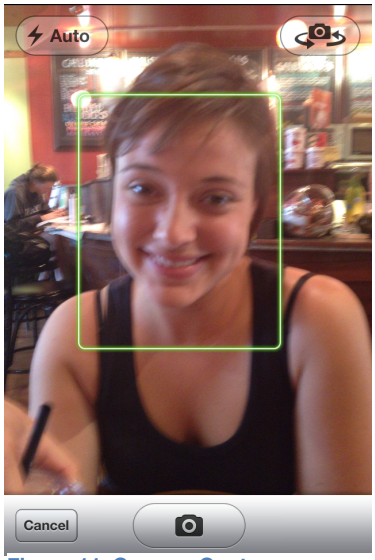


Figure 11: Camera Capture

Figure 12: Image with filter applied

Camera Event View: The camera event view is used to display Camera events. When the user encounters this event, a popup is shown giving a clue of what the user should take a picture of. Upon dismissal of the hint, the iOS image capture modal popover is presented, as in Figure 11. At that point, the captured image has a filter applied to it as well as any pre-defined image or text overlays, and presented to the user as seen in Figure 12.

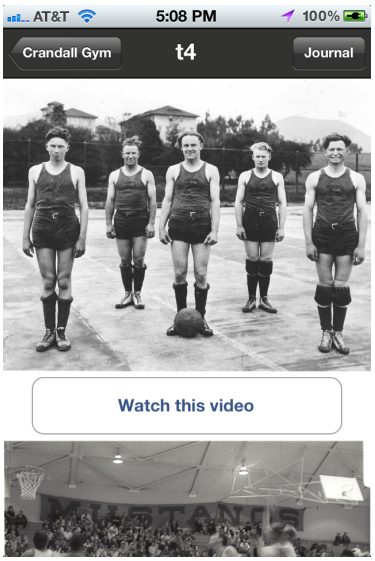


Figure 13: Scrapbook Media

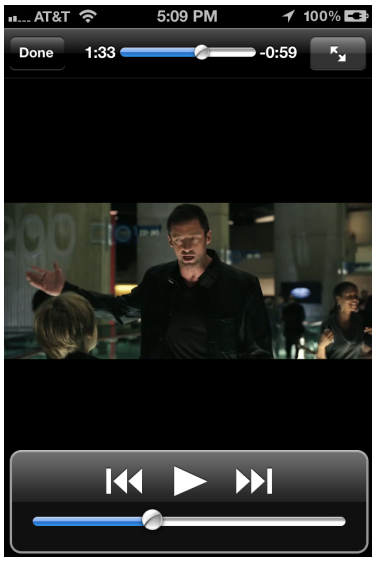


Figure 14: Scrapbook Video

Scrapbook Event View: The scrapbook event view controller is the most versatile of view types. It displays a wide variety of media, including images, text, and video, expandable to fit the needs of the content author. Figure 13 demonstrates how text and image content is rendered in the view, while Figure 14 shows video playback functionality.

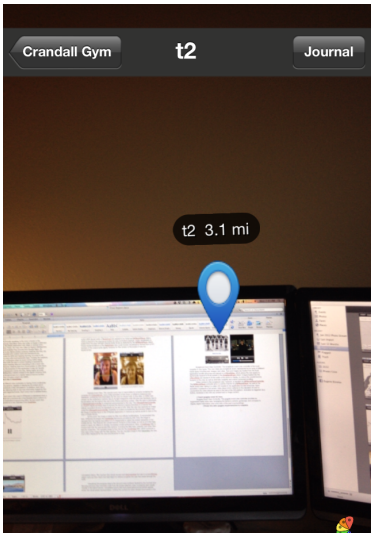


Figure 15: AR Camera View

Goggles Event View: The goggles event view provides a real-time augmented reality "goggles" view, as demonstrated in Figure 15. Its capable of displaying a variety of image content, mapped to a particular geopoint. The goggles view then overlays this media over a live camera view, allowing for organic content discovery.

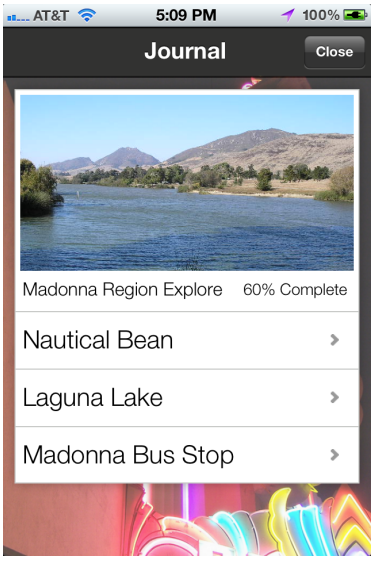


Figure 16: Journal View

Journal View: The journal view is persistent across the myth, shade exploration, and event views throughout the CPXpress mobile application. It's designed to provide a way for the user to go back and revisit their progress through a particular story, or perhaps discover new information about events that have already been triggered. While the content author defines the media in the journal, the discovery process through each story on the part of the user is unique resulting in a unique journal state for every user.

Design and Implementation

Design Requirements

The CPXpress mobile application, the component of the CPXpress framework under my charge, was designed to target Apple's iPhone platform running iOS 5 and above. This design choice was made for a variety of reasons. Firstly, the pervasiveness of the iPhone among CPXpress' potential target users is high, providing a large install base on a platform users already understand. In addition, the iPhone device, specifically the iPhone 4 and 4S targeted by this project, have a wide variety of sensors and input methods required to make this project a success, including cameras, Wi-Fi, GPS, high speed 3G, gyroscopes, accelerometers, digital compass, high resolution touch screen, and more. All of these integrated technologies are required to engage CPXpress' users in a meaningful way. The device is also small and unobtrusive, lending to the goal of allowing the user to interact with their environment first, and the device second.

The iOS software platform, which runs exclusively on the iPhone, is also ideal for CPXpress. Unlike other mobile operating systems, it provides a consistent way to interact with the device's sensors and multimedia playback capabilities, virtually guaranteeing a compatible and consistent user experience across our entire install base. In addition, iOS allows us to leverage Apple's extensive application distribution system, giving CPXpress a wider potential audience as well as a predefined means for acquiring more content for the player.

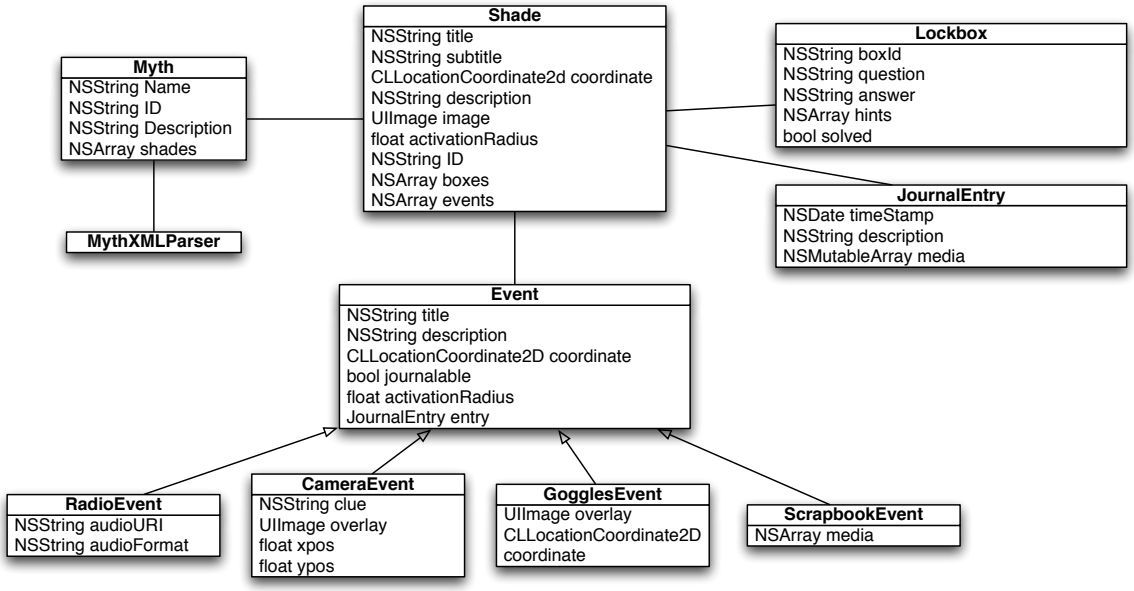
The CPXpress player is written exclusively in Objective-C for a variety of reasons. Firstly, Objective-C in combination with Apple's iOS development environment, XCode 4, compiles natively to iOS compatible binaries and provides an extensive debugging and analysis suite for iOS applications. In addition, Apple provides an enormous library of device-specific APIs, allowing easy access to core device functionality allowing the development process to focus more on creating a quality user experience over dealing with core device functionality access. The one downside of this development target is incompatibility with other mobile operating

systems, such as Google's Android or Microsoft's Windows Phone 7. While other languages do exist that compile to iOS, Android, and WP7, they do not offer the same kind of deep device integration and media playback capabilities required to make an immersive application like CPXpress.

Software Design

Core to the CPXpress experience are the stories and content created by authors, therefore, a robust digital representation of each story needed to be developed for the iOS implementation of the application. Each story is broken down into three distinct elements: a myth, shades, and events. The relationship between each of these elements can be seen in the chart 1 below:

Graph 1: UML



At the top level, the Myth type represents an entire story object, encapsulating any and all information for story playback. This includes the story's name, internal ID, and description. It also contains Shades, which can be considered to be the story's chapters. Each shade holds a subset of information itself, such as its own name, description, a descriptive image used in the UI, and the Lockboxes associated with that portion of the story.

Additionally, Shade objects have a reference an array of Event objects, which represent the information needed to encounter and trigger an event. There are many different types of Event objects, such as a Camera, Radio, or Scrapbook event, however they all inherit from a common Event class, which provides information about its location, activation radius, description, and more. In addition, Events, Shades, and Myths all adhere to the MKAnnotation protocol defined by Apple, which allows any of these objects to be plotted in an MKMapView in the user interface.

Specialized event objects contain additional information to present to the user upon activation. For example, a RadioEvent object contains URI's that point to local audio resources within the main application bundle. A ScrapbookEvent could contain an array of media, such as videos, images, or text. While a shade stores all event objects as generic Events, the UI controller classes introspect these Event objects to determine how they should be displayed upon activation, for example, a CameraEvent would trigger the device to display the camera viewfinder, and pass the captured image back into CPXpress.

Myth objects, and their associated Shades and Events are generated by a static MythXMLParser class, which Myth XML data generated by the content authoring tool. To

accomplish this task, the MythXMLParser class utilizes the TBXML library version 1.4 by Tom Bradley <2>, one of the fastest and most efficient DOM XML parsers for Objective-C. TBXML is given the resource URI of the requested myth XML, which it then parses it and loads it into memory as a TBXMLElement. TBXMLElements have knowledge of their own data and attributes, as well as information on its children and siblings. Using this knowledge, we can iterate through the parsed information, and load it into a Myth structure. Dr. Aaron Keen, Cal Poly Computer Science Department, handled all schema development, and will eventually be responsible for creating the CPXpress authoring tool. This XSD schema document provided a basis for the parsing algorithm that was used in conjunction with TBXML to load myth data into memory.

The last major object in the CPXpress project is the JournalEntry class. This class represents the media stored in an event's associated journal entry, if such entry exists as defined by the content author. Journal entries usually contain a simple set of information, such as a title and description, as well as a specialized Event type as defined earlier for use in the Shade class. The ambiguity between Events used in Shades and the content in a journal entry allows us to reuse view controllers for both the journal and event views, simplifying the overall codebase.

The CPXpress application adheres strictly to the model-view-controller (MVC) design paradigm as outlined by Apple's application development requirements. While the Myth, Shade, and Events objects represent the data source for the application, classes were also written to drive the primary user interface. Below is a brief outline of the CPXpress mobile application user interface and the technical design considerations that went into making them a reality.

Login View Controller: As the first view presented to the user upon application start, the Login view controller has the responsibility of verifying a user's credentials, and if none exist, prompting the user for the creation of new credentials. If this is the first launch of CPXpress, or there is no preexisting user credentials cached to disk, the login view controller requests the user log in via Facebook. Facebook authentication was chosen because of its easy, one click login, and its tight integration with the Parse backend. User credential data is then persisted to Parse, along with all of their progress through the Myths on their device which can be restored to any device or synced to a different iPhone if needed.

Myth Selection View Controller: This view controller is initialized and presented after the user successfully logs into CPXpress. At this point, the MythXMLParser has already parsed all myths loaded on device into memory, which will persist for the duration of this application session. Since a normal list view would lack visual appeal to the user, a randomly chosen background is presented semi-transparently behind the myth options, which moves with a parallax effect as the user scrolls the list. This is accomplished via the UIScrollView class handling user scroll interaction while computing a translation transform that is applied to the background image view.

Shade Selection View Controller: The shade selection view is a visual representation of the shades present in a particular myth, responsible for giving the user information on the location, description, and content of all the shades. This is done in a similar manner to the myth selection screen, with the notable exception of the list element. Each shade in the list contains a visual representing the shade, a title, and the distance to the shade. This distance is calculated via GPS location updates provided by a CLLocationManager, part of Apple's CoreLocation framework. GPS data is updated at one-second intervals, which in turn updates the distance labels on the shade list items.

Upon pressing one of the shade list items, the item expands to fill the full view, providing a detailed description of the shade as well as a map pinpointing its exact location. This is done via a custom UIView subclass called a ShadePreviewView. This shade preview can scroll dynamically with the help of a UIScrollView to display descriptions of any length, as defined by the content author. Animation between the list state and details state is handled by the UIView

CoreAnimation framework, interpolating animations between to pre-defined view frame sets as well as varying visible content within each preview by modifying the preview view's subview's alpha properties.

Shade Exploration View Controller: The shade exploration screen, technically speaking, is the most complex in the CPXpress mobile application. While previously described views rely solely on Apple's UIKit and Foundation frameworks for UI rendering, the shade exploration view relies on several other frameworks as well to perform its functions. Firstly, the map view relies on not on Apple's MapKit framework, but the Route-Me open source map view library, with map tile data sourced via CloudMade. This alternative map source was chosen for aesthetic reasons, since Apple's MapKit does not have theming options available. A custom CloudMade theme was created for CPXpress that matches the overall UI theme for the app.

Additionally, a 3D "perspective" view was created to offer an additional view to assist shade exploration. Unlike the normal flat map view, which is zoomed in to only show the user's immediate surroundings, this perspective view is zoomed out and provides an overhead view of a greater surrounding area. Events within this area are plotted on the map, only visible in this mode and not the other. This transformation leverages CALayer's CoreAnimation properties, allowing for the generation of a transformation in 3D space. The map view's transformation matrix is directly accessed to provide a 45 degree X-axis rotation as well as a perspective skew at the top of the view. In addition, the view is dropped a pixel offset in the negative Z-axis direction to provide the "overhead view" illusion. This transformation is then stored, and applied to the map view whenever the user presses the perspective button on the lower left hand corner of the screen.

The CoreLocation framework is used for GPS and heading tracking, accurate to 5ft under optimal conditions. CoreLocation is used with event activation logic, tracking the positions of each event, displaying an "event found" notification when an event comes within range. Additionally, the map rotation in conjunction with the user's current heading is powered by CoreLocation as well, taking digital compass heading data and translating it into a local view transform applied to the MKMapView. However, heading data from CoreLocation is often times jumpy and inaccurate, while functional for the purposes of this application it was not visually appealing. To solve this, the CoreMotion framework comes into play, taking highly accurate gyroscope data and applying it to the MKMapView rotation transformation instead of the digital compass data. The compass was then used to calibrate the gyroscope on a regular basis (2 second update interval), whenever the device detected that compass output was stable.

Arrow overlays are also visible when the map is not in an overhead perspective view. These are simply UIImageViews with a modified anchorPoint property to ensure rotation around the center of the map view. The technical complexity comes into play in calculating the rotation transform necessary to ensure the arrows point in the appropriate directions, in this case, towards any event not within the map's current field of view. The Haversine algorithm was used to accomplish this task, as it is capable of calculating the bearing between two points on a curved surface. For the purposes of the CPXpress mobile application, the Haversine function was implemented as such:

```
        double deltaLong = targetLong - currLong;
        double y = sin(deltaLong) * cos(targetLat);
        double x = cos(currLat) * sin(targetLat) - sin(currLat) * cos(targetLat) *
cos(deltaLong);
        double radiansBearing = atan2(y, x);
```

These four lines of code generate a bearing in radians towards a target event. Unfortunately, this information on its own is not enough to generate a rotation transform to apply to the arrow UI element. At this point, a delta between this bearing and the device's current compass heading is calculated, and subsequently negated. This radian value is then used to create a CoreAnimation rotation transformation that is individually applied to each of the arrow views.

Event Views: CPXpress leverages several different view controllers to display Event multimedia content. Each of these views are built to be generic, capable of displaying the widest set of multimedia possible within the design constraints of the iOS platform. In addition, each has a button pointing back to the user's Journal, allowing them to see new information logged by this event as well as the ability to reference past triggered Events. Each event view is detailed below, as well as any pertinent technical details.

Much like the Event data types, each event view used in CPXpress is subclassed from a single EventView view controller. This allows for a common set of functionality and a uniform UI experience across all event views without duplicated programming effort. As new event view types are added, they can be easily integrated as a new subclass of an EventView.

Radio Event View Controller: As mentioned previously, the Radio event view is responsible for the visualization and playback of RadioEvents. Audio playback is handled via the AVFoundation and MPMediaPlayer frameworks provided by iOS. These frameworks allow for the fast and efficient playback of media located within the application bundle resources, without much development effort. The audio URI's stored within a RadioEvent are passed to an instanced MPMediaPlayer object, which is bound to the play/pause button for user-controlled playback. Audio visualization is pulled from the peak left/right channel audio levels as detected by the MPMediaPlayer object. This data is then plotted against time via the F3PlotStrip class by Brad Benson <1>.

Camera Event View: The camera event view is responsible for properly rendering the contents of a CameraEvent. Upon initial activation of this event, a UIAlert is instanced and displayed, containing clue information for the user to interact with. In this case, a UIAlert was chosen as the notification method, due to its tight integration with iOS and user familiarity due to its pervasive use throughout the rest of the iPhone operating system. After the alert is dismissed, a UIImagePickerController is instanced and presented to the user. The UIImagePickerController is a built-in Apple provided class to access the device's camera functions in a manner that utilizes consistent UI and UX standards as the rest of iOS. After the user has captured an image using the UIImagePickerController, hopefully of content prompted by the previous UIAlert clue, the image is passed back to the CPXpress application in memory as well as saved to disk on the user's mobile device.

At this point, the image is ready for filter and effects modifications. To accomplish this task, CPXpress leverages the CoreImage framework provided by iOS. First, the image is rescaled from its original size (either 5 or 8 megapixels) to match the screen resolution of the mobile device. This reduces the overall memory footprint of the image, as well as providing fewer pixels to apply the effects and thereby increasing overall performance. Next, a CoreImage filter is constructed with the appropriate filter options, including the filter type, intensity, and other filter-specific properties. The rescaled image is then fed into the CoreImage filter, which produces a new copy of the image with the filter applied. The old, rescaled image is then discarded; the new one is then displayed for the user and persisted to disk.

Scrapbook Event View Controller: The scrapbook view controller takes any scrapbook event, represented as an array of different media types including text, images, and video. Text and images are loaded from the local application bundle resources and placed in a UIScrollView, which allows the view space to dynamically expand to fit all content in the ScrapbookEvent object. Images loaded into the UIScrollView are resized to fit the width of the device's display, to reduce the overall memory footprint of the view as well as eliminate unsightly and uneven horizontal scrolling.

Video content in the scrapbook view, however, is handled via MPMoviePlayerController. The MPMoviePlayerController acts in a similar manner to a MPMediaPlayer, as used in the Radio event view, taking in a video URI located in the application bundle resources. Upon activation, it presents a full-screen video player which adheres to the standard UX and UI conventions for

video players as used in the rest of the iOS platform. Activation is triggered via a button, rendered in-line with any present text or image content.

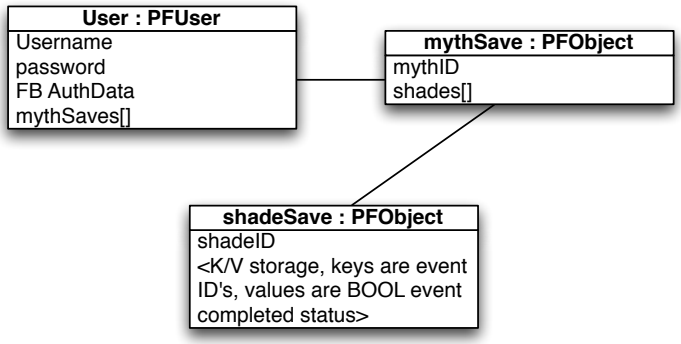
Goggles Event View Controller: The goggles event view controller provides an augmented-reality style view, leveraging the device's camera, gyroscope, and compass to display digital information over a live view of the user's surroundings. The SM3DAR library by Spot Matrix is leveraged to accomplish this task. SM3DAR acts in a very similar manner to Apple's MapKit framework, in fact, much of its capabilities are subclassed directly from MapKit. However, it also has the ability to tap into the device's camera, gyro, and accelerometer to display an augmented reality view. Points of interest are added to SM3DAR in the same manner as MapKit, as long as a class adheres to the MKAnnotation protocol, it can be used as a SM3DAR point.

Journal View: The journal view controller is responsible for tackling two major problems: backend synchronization and UI.

Background synchronization is paramount to the operation of the Journal view. To accomplish this task, CPXpress relies on the Parse service for user account creation and management, as well as object management for individual users of CPXpress. Essentially, Parse acts as a cloud based, schema-less database and file storage system for mobile applications, set up in such as way that very little setup is required on the part of the application developer. Parse provides a robust iOS framework for interacting with their cloud services, allowing object retrieval and persistence, complex queries, and local device caching without significant additional development effort. Since the data persisted to Parse by CPXpress is simple in nature, it suited the needs of this project well. Additionally, it allows users to synchronize their progress in CPXpress across multiple devices, and potentially multiple platforms in the future.

To accomplish user progress persistence, a very loose structure was defined within the CPXpress mobile application, which was then associated with the user's Parse object. Every user that logs into CPXpress has an associated PFUser object, as generated by the Parse login systems. This PFUser object acts as a dictionary on device, capable of having additional values and keys associated with it. After modifications are made, it can be asynchronously persisted back to Parse. This takes network load off the main thread, resulting in a responsive UI across the application regardless of network activity. The structure defined for user progress persistence can be seen below:

Graph 2: Parse UML



As demonstrated by the above diagram, each user has an associated array of mythSave objects. MythSaves are PFObjects, the most basic unit of object storage in Parse, and acts like a dictionary in the same manner as a PFUser. Each mythSave represents one of the myths present on the user's device, including its ID and an array of shadeSave objects that represent all the shades contained within that myth. ShadeSaves, PFObjects as well, contain a set of keys that correspond to the events in the shade, its values being a Boolean YES or NO to indicate

completion status. The Journal view simply access and re-synchronizes the user's current PFUser object, and can then reach into that object to retrieve progress the user has made through the myth.

Visualizing this progress data is the second major problem tackled by the Journal view. Reusing previous view controller code from the shade selection view, it presents each shade visually in the same manner. Completed events that have associated journal entries appear below the visual shade representation, shifting the content for other shades downwards in the view. Pressing one of the completed events in this view presents a new Scrapbook event view controller, rendering media from the Scrapbook event stored in that event's journal entry property.

Cal Poly Xpress Technical Analysis

While the Cal Poly Xpress mobile application's primary focus is on overall user experience, it is important to note the significance of the technical performance of the application and how it affects this goal. By analyzing several key data points about the application, it may be possible to improve the overall performance and stability of the application leading to a superior user experience. All analysis is done via Apple's Xcode 4.3 Instruments test suite. The Instruments suite inspects running processes on a tethered iOS device, and can gather information on memory usage (including individual object allocation and deallocation within the program stack), CPU and memory usage, estimated power draw, network connectivity, disk use, GPU usage, and more. Given the capabilities of the CPXpress mobile application, several performance metrics will be analyzed: CPU and memory usage, estimated power draw, and network activity. The data gathered from Instruments will be used to make recommendations for optimizing the performance and reliability of the Cal Poly Xpress mobile application.

CPU Usage Analysis

CPU utilization plays a direct role in the overall performance of the Cal Poly Xpress mobile application. A breakdown of CPU utilization by type was obtained using the Instruments test suite, as a user performed a series of tasks mimicking an average use case for the application. CPU utilization was divided into four parts: overall system CPU utilization, foreground application utilization relative to overall CPXpress CPU utilization, as well as audio and graphics CPU utilization calculated in the same manner as foreground application utilization. A plot of these utilization characteristics over time can be seen below:

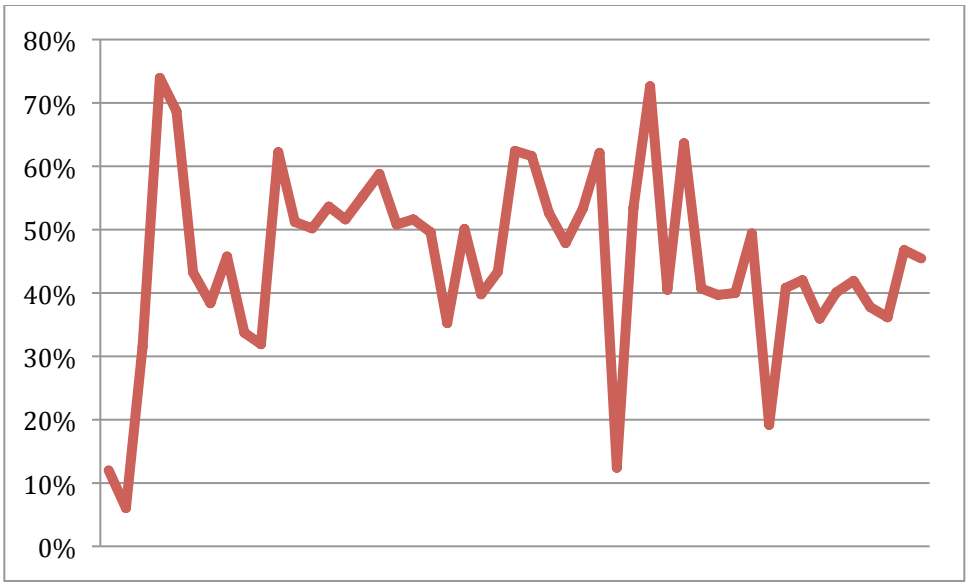


Figure 16: Overall system CPU utilization

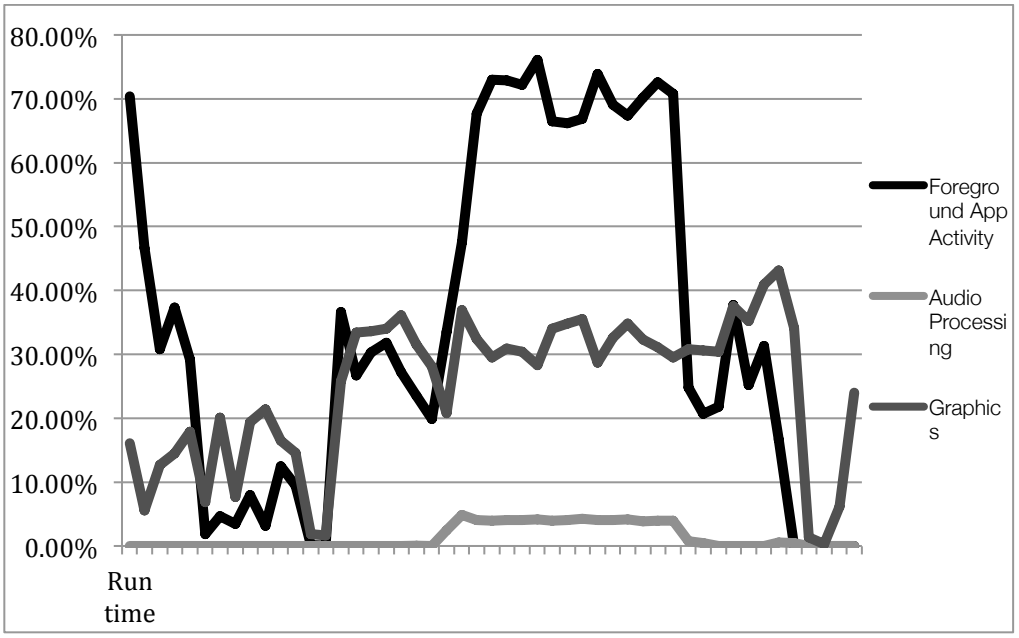


Figure 17: CPU Utilization By Type

As seen by Figure 16, the CPXpress mobile application constitutes a relatively constant load on the device's CPU, hovering around 60% total utilization for the duration of the application lifecycle. However, several spikes in overall utilization can be seen, especially the drastic spike to 75-80% upon application launch. This spike is directly related to the MythXMLParser class, reading through myth data located in the local application bundle and loading it into memory. This parsing process is extremely CPU intensive, and actively adds 1-2 seconds to the launch time of the CPXpress project. This leads to the conclusion that this content layout is not appropriate for a mobile environment; instead, some format that does not require parsing would be optimal. For example, placing content in a CoreData database would greatly increase local access speeds. Optimally, future versions of CPXpress can download

content directly from the Internet on an as-needed basis, entirely negating the need for local storage in the first place.

Figure 17 represents a breakdown of CPU activity by type over the same application lifecycle as Figure 16 by foreground application activity, as well as graphics and audio tasks. At application start, a large spike in foreground app activity is recorded, corresponding to the observations noted in Figure 16. However, this plot brings to light other CPU loads previously not observed that impact overall performance. The first one is a spike in graphics tasks on the CPU, around when the Shade Exploration screen is instanced and becomes visible. The Shade Exploration screen relies primarily on CoreAnimation for all visual effects, which provides a certain amount of GPU acceleration to increase performance. The GPU is used to ensure a 60fps target for all animations created within the view, however, it is apparent that the CPU still plays a significant part in making this happen, making up about 30% of the application's CPU utilization at that time. To alleviate this CPU load, the Shade Exploration view could be rewritten to be an OpenGLES view, which relies 100% on the GPU for rendering.

The largest foreground application CPU utilization spike seen in Figure 2 is due to the RadioEvent view's plot-based visualizer. Unfortunately, this visualizer has no GPU acceleration at all, as it does not leverage CoreAnimation for plot generation. Switching this visualizer to one that utilizes OpenGLES or CoreAnimation should ease CPU load in this view.

Memory Utilization

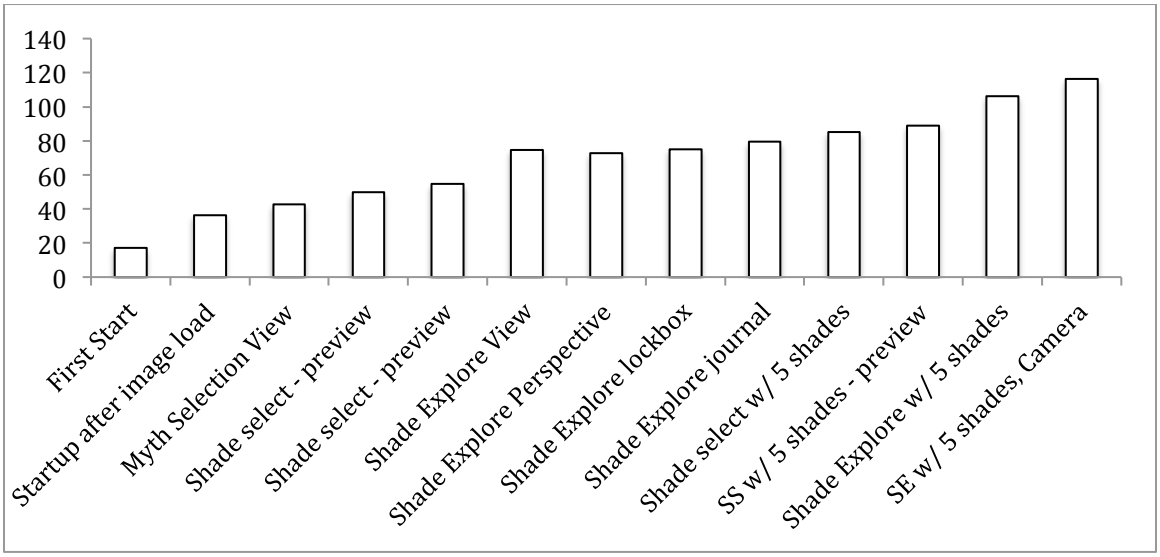


Figure 18: Memory Consumption per task

Cal Poly Xpress relies on iOS 5.0's ARC, or Automatic Reference Counting for memory management, leaving very little developer interaction in the memory allocation or deallocation process. However, a general analysis was performed of real system memory usage over a variety of tasks to determine the effectiveness of ARC. As seen from the Figure 18 below, ARC has a very liberal memory usage model, allowing the application to use 100-120MB of system memory for some tasks. At this point, it was observed that ARC forces the deallocation of application content to bring memory usage down to more manageable levels, however, this process comes with a temporary decrease in performance and therefore the overall user experience of the app. To alleviate this issue, design choices can be made in the CPXpress application programming to reduce memory usage. This includes

- Shade selection view

    - The shade selection view has the potential to play host to a large number of views, depending on the number of shades in the particular selected myth. More shades results in an exponentially higher number of views that need to be instanced for the preview, such as the map, image, and description views. However, these views are not always visible to the user. If fewer views could be instanced at once, and then reused for different content later for other previews, the overall memory consumption could be reduced.
- Myth parsing
    - At application start, all myths on device are parsed and loaded into memory. This is an unnecessary process, and consumes needed memory resources. Persisting myth data to disk instead of memory would be a more efficient option.

Estimated Power Consumption

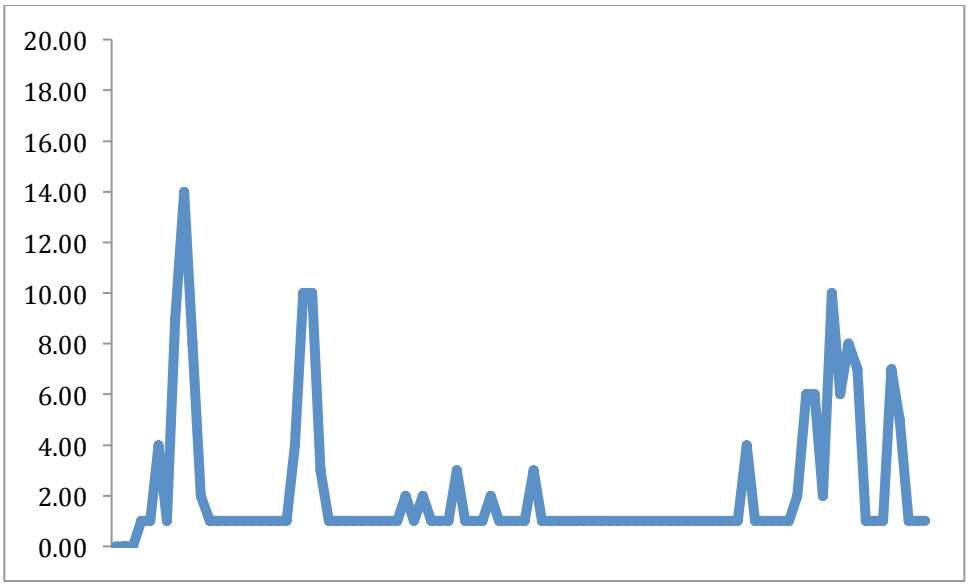


Figure 19: Estimated Power Consumption

Power consumption is determined via XCode's Instruments Energy analysis template. This analysis process differs slightly from CPU and memory consumption metrics, as it does not run tethered. Instead, energy logging is enabled on a target test device while untethered from power or a computer. At this point, the CPXpress mobile application is used in the same manner as the other two tests. Upon application termination, the target test device is plugged back into Instruments, which proceeds to offload power log data, parsing it into human-readable information.

Power consumption is measured on an estimated 0 to 20 scale, 20 indicating maximum power consumption, 0 indicating relatively average power consumption. Unfortunately, these are only estimates; there are many variables that could potentially impact device battery life during normal user operation including other applications open on the device and the state of the device's internal sensors. The power consumption analysis is helpful, however, for generalizing the potential power draw of different operations within the CPXpress mobile application, exposing any glaring power drains that may be present.

A few interesting conclusions were drawn from the analysis of the power consumption data. Firstly, GPS did not appear to have a significant impact on overall power usage, even though CPXpress leverages it extensively for the majority of its functions. This could be due to the efficiency of the CoreLocation framework provided in iOS. Spikes in power consumption were seen during high periods of CPU utilization, perhaps indicating that high CPU load should

be avoided. GPU utilization did not appear to have a major effect on power consumption as well, moving the exploration view over to a GPU-accelerated OpenGLES view should alleviate power drain further.

Related Works

Upon extensive research, it has been determined that there are no mobile applications to date that match the exact goals of the CPXpress project. However, there are several products and applications that do fulfill a subset of the CPXpress project's capabilities, or are similar in nature.

National Geographic Parks for iPhone

The National Geographic Parks application for iOS provides an informative guide to several US national parks. Similarities to the CalPoly Xpress project include geocode events, as well as the ability to present relevant information based on current user coordinates. However, this application is single-purpose, and does not have the ability to allow the user to organically discover their environment via the device's location services.

Source: http://itunes.apple.com/us/app/id518426085?mt=8&src=af&ign-mpt=uo%3D6

Layar Reality Browser

The Layar application provides an augmented reality view of a wide variety of information culled from a variety of online sources, such as Yelp and Wikipedia. Similar to the CPXpress mobile application, its primary purpose is to overlay digital information over the user's physical environment. Additionally, its capable of being used for a variety of purposes, depending on the information source selected. However, it does not lend itself to the story telling paradigm established by the CPXpress project, instead acting as a reference tool, a way to get information out of the internet and into the real world.

Source: www.layar.com

Geocaching

The Geocaching application for iOS facilitates the new, popular sport of geocaching, or finding physical objects in a user's environment via GPS location services. With the exception of the physical object aspect of geocaching, the goals here are very similar to that of the CPXpress project. However, much like Layar, it does not support the storytelling features that CPXpress touts.

Source: http://itunes.apple.com/app/geocaching/id292242503?mt=8

Conclusion

As demonstrated, the CalPoly XPress mobile application has proved the concept of location based storytelling and educational material to be a feasible option on today's mobile devices. CPXpress leverages the full technology stack available in iOS 5.0+ and the iPhone 4 and 4S to create an engaging augmented-reality experience for the end user. Additionally, it provides a new, innovative creative tool for content authors to create their stories with. The potential applications for the CPXpress project hare huge, its up to the creativity of our users to bring it to its full potential.

Future Work

The CPXpress mobile application in its current form is capable of performing a wide variety of tasks, however, there are many areas in which future development would benefit it greatly. A few of these key development tasks are outlined below

- Parse persistence for Myth data
    - Currently, myths are required to be located in the local application resource bundle for the CPXpress project. This is inconvenient, as it requires the content authors to recompile and reload the application upon every content modification. Additionally, requiring all content to be local on device requires a significant amount of disk space, a rare commodity on mobile devices. The Parse object storage service has proven to be extremely effective at storing user sync and persistence data, and already has tight integration with the CPXpress mobile application. It would not be difficult to migrate myth content over to Parse as well, providing one, central, cloud-based solution for myth and user data storage. Additionally, this would alleviate the distribution problems in the current rendition of the project, as it would not require a content author to recompile the application upon myth modification. Instead, modifications could simply be pushed from the authoring tool to Parse, where it would immediately show up on the end user's device.
- Desktop Authoring Tool
    - Currently, CPXpress content is generated via the creation of an XML file that is compliant with the CPXpress schema developed by Dr. Keen. This, along with associated multimedia content for that story has to be manually loaded on the mobile device in order for users to interact with it. While the XML development process was effective for the purposes of this project from a development standpoint, it is not user friendly for non-technical content authors. Therefore, the creation of a graphical web or desktop-based authoring tool would be an important asset for the CPXpress project. Such a tool would make content creation for the project extremely accessible especially to non-technical persons, leading to an increase in overall content for the CPXpress mobile application.
- Indoor Navigation
    - The CPXpress mobile application, at this time, is the most effective in outdoor environments. This is due to the sensors available on the iPhone 4 and 4S leveraged by the project, and their ineffectiveness indoors. In an indoor environment, GPS and digital compass readings become unreliable to the point of being useless, rendering the CPXpress mobile application ineffective. Therefore, a new method of indoor navigation needs to be investigated. Possible options include QR code based navigation, utilizing the device's camera. However, this would remove some of the organic exploration features that are central to the project's goals. Additionally, new emerging technologies may provide for accurate indoor navigation in the near future, the use of devices that support this feature would greatly enhance indoor navigation.
- Cross-Platform Compatibility
    - While the CPXpress mobile application targets iOS 5.0+ and the iPhone line of devices, it would be remiss not to recognize the large install base of other mobile platforms today. Additionally, these other mobile platforms are host to devices that are very similar to the iPhones used for CPXpress, and would be prime candidates for the project. If development resources allow it, it would be a good idea to port the project to other platforms such as Google Android or Microsoft Windows Phone 7.
- Social Integration

- The current integration of the CPXpress mobile application is restricted in terms of user experience to one user at a time. Social media integration could be explored to facilitate collaborative organic discovery, or perhaps the sharing of information found in CPXpress event encounters with peers.

References and Resources

1. Benson, Brad. *F3PlotStrip*. N.p., n.d. Web. <https://github.com/ChiefPilot/F3PlotStrip>.

2. Bradley, Tom. *TBXML*. Computer software. Vers. 1.4. N.p., n.d. Web. <http://www.tbxml.co.uk/>.

3. Bukovinski, Matej. *MBProgressHUD*. Vers. 0.4. N.p., n.d. Web. <https://github.com/jdg/MBProgressHUD>.

4. Morrissey, Jason. *JMTabView*. Computer software. N.p., n.d. Web. <https://github.com/jasonmorrissey/JMTabView>.

5. OpenStreetMap. *CloudMade*. N.p., n.d. Web. <http://cloudmade.com/>.

6. TestFlight Inc. *TestFlight*. Computer software. N.p., n.d. Web. <https://testflightapp.com>.