



# WPI

## Designing and Developing a Travel-Based Android Application

A Major Qualifying Project Report:

submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

---

Kevin Hufnagle

Date: May 6, 2014

Approved:

---

Professor Emmanuel Agu, Project Co-Advisor

---

Professor Jennifer deWinter, Project Co-Advisor

*This report represents the work of one of more WPI undergraduate students submitted to the faculty as evidence of a completion of a degree requirement. WPI routinely publishes these reports on its website without editorial or peer review. For more information about the projects program at WPI, see <http://www.wpi.edu/academics/ugradstudies/project-learning.html>*

## Abstract

Many people in North America enjoy capturing the visual, historical, and experiential contexts that landmarks offer as they travel. These individuals, however, use disjoint resources to plan trips, reducing the quality of the narrative experience they receive when visiting a place. To facilitate the narrativization of a place and create a shared experience of that place, I created an Android smartphone application, which allows users to explore basic facts, photographs, historical details, and travelers' experiences regarding New England's 117 coastal lighthouses. I developed three image-processing algorithms to serve as photograph filters and conducted two surveys and five usability studies to inform my iterative, audience-involved application design.

## Acknowledgments

- First and foremost, I thank my two project advisors, Jennifer deWinter and Emmanuel Agu, for their continuous support and dedication towards this project.
- Jay Hyland of the Lighthouse Preservation Society provided key background information about New England lighthouses in general and an avenue for distributing surveys to members of the society.
- Ryan Avery from Cape Neddick (“Nubble”) Lighthouse deserves a special mention for helping me collect pre-design survey responses.
- I am also grateful for Ryan Avery, Bob Gallagher from Old Scituate Light, Francene Webster from Highland (“Cape Cod”) Lighthouse, and Sarah Mumford and Keith Sisterson from Nauset Light for allowing me to use the grounds by their lighthouses to conduct my usability studies.
- Jeremy D’Entremont kindly permitted me to include information from his book and website within my app.
- Finally, I’d like to thank my family for their personal support throughout this project experience.

## Table of Contents

Acknowledgments.....	iii
List of Tables .....	vi
List of Figures.....	vii
1. Introducing an Application that Offers Mobile Traveling Experience.....	1
1.1. Why Humans Enjoy Travel-Based Experiences .....	1
1.2. Android Smartphone Use Continues to Grow Quickly.....	3
1.3. Computers Do Not (Yet) Satisfy Expectations Regarding Digital Image Processing .....	6
1.4. Challenges Associated with Current Options Available .....	8
1.5. Addressing These Challenges: Creating the Lighthouse Navigator Application.....	9
1.6. Overview of Report's Contents.....	11
2. Presenting Current Applications, Theoretical Foundation for Work.....	12
2.1. Current Websites Each Offer Narrow Scope of Travel-Based Information .....	12
2.2. Existing Smartphone Applications Combine to Form Narrative Space.....	26
2.3. Key Research Involving Images and Image-Processing Algorithms.....	37
2.4. Space and Place: Embedded Comfort Within the Unknown .....	49
3. Designing a Compelling Lighthouse Portal Using Audience Feedback.....	52
3.1. Android Design Guidelines Stress Simplicity, Usability.....	52
3.2. Design Paradigms.....	56
3.3. Guiding Users through the Application's Different Screens .....	59
3.4. Improving Design using Iterative, Audience-Based Process.....	72
3.5. Designing Universally Effective Application Involves Consideration of Market .....	79
3.6. Presenting the Modular Structure of the Android and Image-Processing Projects.....	81
4. Structure and Strategies for Implementing Application .....	85
4.1. Preparing Development Environment: Eclipse and ADT .....	85
4.2. Presenting an Overview of Lighthouse Navigator Application .....	86
4.3. Using Existing Data Structures and Packages to Enhance Application Efficiency .....	93
4.4. Creating New Data Structures for Organizing Run-Time Application Data .....	96
4.5. Using Flickr's API and Android Multithreading to Download Photographs .....	108
4.6. Creating a Flexible Back End: XML File and Photograph Cache .....	113

4.7.	Constructing Image Processing Algorithms with ImageJ .....	115
4.8.	Preparing Image-Processing Algorithms for Testing.....	125
5.	Reflecting on Algorithm Performance and Application Usability .....	127
5.1.	Findings from Image-Processing Algorithm Testing.....	127
5.2.	Presenting User-Friendly Elements in Mobile Application Increases its Appeal .....	137
6.	Reflecting on the Project’s Takeaways and Future Trajectories .....	142
6.1.	Application Attained Usability Goal, but Cross-Cultural Narratives Tough to Create	142
6.2.	Image-Processing Algorithms Achieved High Accuracy, but Tough to Generalize ...	143
6.3.	Completing App Functionality, Adding Filters, Investigating Simulacra in Future ....	144
7.	References.....	146
Appendix A:	Instructions for Installing Project Files (README).....	A-1
Appendix B:	Version History .....	A-6
Appendix C:	Interview with Jay Hyland .....	A-8
Appendix D:	Survey for Visitors of Cape Neddick Lighthouse .....	A-12
Appendix E:	Survey for Lighthouse Preservation Society Members.....	A-18
Appendix F:	Usability Study Materials.....	A-22

## List of Tables

Table 1-1. Distribution of Existing Android Devices by Operating System Version. (Accurate as of May 1, 2014).....	6
Table 1-2. Distribution of Existing Android Devices by Screen Density.....	7
Table 2-1. Primary and Secondary Categories of Information on Travel- and Experience-Based Websites.....	12
Table 2-2. Existing smartphone applications featuring specific points of interest. ....	27
Table 4-1. Naming Convention of Lighthouse Photographs Appearing within “Drawable” Resource Folder. ....	89
Table 4-2. Structure of URLs for Travel Experience Websites Featuring Lighthouses. ....	101
Table 4-3. Descriptions of Enumerated Lists within Application. ....	102
Table 4-4. Process of Transforming Test Image used in Shape-Based Feature-Matching Algorithm.....	124

## List of Figures

Figure 2-1. Flickr's user-level advanced search lacks location-based filtering. ....	14
Figure 2-2. "Search the map" button unnecessarily hidden within Flickr world map.....	16
Figure 2-3. Sample Search Results Page on Panoramio.....	17
Figure 2-4. Viewing location information, photographs, and user experiences for a point of interest on Yelp. ....	19
Figure 2-5. Viewing location information, photographs, and user experiences for a point of interest on TripAdvisor. ....	21
Figure 2-7. Filtering Options within Cowbird. ....	23
Figure 2-7. Story about a Lighthouse on Cowbird. ....	24
Figure 2-9. Log book entries and image gallery for the "Fundy Tides" geocache on Geocaching. ....	26
Figure 2-9. Sample location listing in Boston City Guide application. ....	29
Figure 2-10. Sample itinerary in Boston City Guide application. ....	29
Figure 2-13. Sample location listing in New York City – Everything NYC.....	31
Figure 2-12. First screen of main dashboard within New York City – Everything NYC. ....	31
Figure 2-14. Sample grayscale image and its corresponding histogram. ....	38
Figure 2-15. Sample grayscale image with its corresponding cumulative histogram. ....	39
Figure 2-16. The general histogram matching algorithm: Mapping cumulative histogram intensity values.....	42
Figure 2-17. Discretization of the general histogram matching algorithm: "stacking" intensity value "blocks" from the original image onto the reference image. ....	43
Figure 2-18. Approximation of Distances from a Pixel to Its 8-Connected Neighbors (3×3).....	46
Figure 2-19. Approximation of Distance from a Pixel to Its Nearest 24 Neighbors (5×5). ....	47
Figure 3-1. Multiple Elements within "Information" and "Photographs" Screens within Application.....	54
Figure 3-2. Areas of Screens Affording Real-Time User Interaction.....	55
Figure 3-3. Relationships among Artifacts and Ideas while Using Audience-Involved Communication.....	58
Figure 3-4. Relationship among Screens within Application. ....	59
Figure 3-5. Final Design of "Welcome" Screen within Application. ....	60
Figure 3-6. Final Design of "Search Results" Screen within Application.....	61
Figure 3-7. Final Design of "Information" Screen within Application ....	63
Figure 3-8. Final Design of "Photographs" Screen within Application. ....	66
Figure 3-9. Final Design of "History" Screen within Application. ....	69
Figure 3-10. Final Design of "Reviews" Screen within Application.....	71
Figure 3-11. Application Design Process. ....	73
Figure 3-12. Initial Design of "Welcome" Screen within Application.....	75
Figure 3-13. Initial Design of "Photographs" Screen within Application. ....	76
Figure 3-14. Class Diagram for Lighthouse Navigator User Interface Screens. ....	82

Figure 3-15. Class Diagram for Lighthouse Navigator Data Structures.....	82
Figure 3-16. Class Diagram for Lighthouse Navigator Utility Classes. ....	83
Figure 3-17. Class Diagram of Lighthouse Navigator Interface Adapters. ....	84
Figure 3-18. Class Diagram of Image-Processing Algorithm Logic Classes. ....	84
Figure 4-1. Structure of query to Flickr’s API for retrieving geotagged, .....	110
Figure 4-2. Excerpt of sample geotagged photograph response from Flickr’s API. ....	111
Figure 4-3. Sample Distribution of Pixels across Several Sub-Images. ....	119
Figure 5-1. Effects of Image Decomposition, Percentage Inclusion on Algorithm Accuracy. ..	129
Figure 5-2. Effects of Sub-Image Complexity on Image Classification Execution Time. ....	129
Figure 5-3. Changing Grayscale Intensity Value Affects Color-Based Feature-Matching Algorithm Accuracy.....	131
Figure 5-4. Moderate Color Difference Tolerance Best for Color-Based Feature-Matching Algorithm Accuracy.....	132
Figure 5-5. If Small Amount of Image Contains Color, Best to Consider it a Match .....	132
Figure 5-6. Execution Time of Shape-Based Feature-Matching Algorithm Constant when Using Single Reference Template. ....	134
Figure 5-7. Very Small Chamfer Match Score Allowances Lead to Accurate Shape-Based Feature-Matching Algorithm. ....	135
Figure 5-8. Even Smaller Chamfer Match Score Allowances Yield Accurate Shape-Based Feature-Matching Algorithm Results Given Multiple Reference Templates. ....	136
Figure 5-9. Shape-Based Feature-Matching Algorithm Using Multiple Reference Templates Takes Less Time to Execute as Chamfer Match Score Allowance Increases.....	137



## 1. Introducing an Application that Offers Mobile Traveling Experience

As people travel, they visit landmarks and capture photographs to mark the key moments along their journeys. Recently, people have begun capturing these memories more frequently on their smartphones and digital cameras, which offer convenient methods for storing entire scenes but cannot easily analyze the details within these scenes. Furthermore, these individuals need to develop an impossibly complex mental model while investigating different aspects of a location they plan to visit, preventing them from forming effective traveling communities. The application that I developed for this project presents informational, visual, historical, and experiential contexts of New England lighthouses. It offers image-processing algorithms for detecting global classifications as well as local features within images, and it scaffolds the narrativization of place as a shared experience of place, allowing the application to facilitate the formation of travel- and preservation-conscious communities.

### 1.1. Why Humans Enjoy Travel-Based Experiences

Humans crave interesting and memorable experiences, and by traveling, they can satisfy this need creatively. From visiting family and friends to embracing new cultures, people form lasting memories during their traveling adventures. These trips often involve visiting specific landmarks and capturing photographs, both of which help mark checkpoints along a journey within travelers' minds. This section discusses individuals' motivation for traveling in general and for consuming landmarks and photographs in particular.

Traveling in general offers plentiful opportunities for sociocultural exploration and personal introspection. Ramvie Santiago, Annabel Candy, and Adam Groffman each explore these ideas further in their respective online articles and blog posts. As people travel, they encounter opportunities to socialize with familiar faces and to immerse themselves in a different culture with new acquaintances. Santiago's article explaining why people travel explains that, in most cases, travelers wish to spend time with family and friends who have moved a significant distance away [17]. Adam Groffman's blog post discussing the traveling lifestyle agrees with Santiago that people travel to see family and friends, but he also believes that other individuals visit places to meet new people who offer interesting stories to share [11]. Annabel Candy expresses views similar to those of Groffman in her reflections on humans' motivation to travel, explaining that humans visit places to enjoy experiences that do not exist within their respective home areas. A particularly rich experience that Candy cites involves viewing new life

perspectives by exploring foreign languages and the ideas they express [8]. Santiago adds to this idea in her article, discussing how exotic cuisines symbolize new cultures abroad [17]. Humans express excitement with escapes to familiar memories and adventures involving new people and cultures, explaining their constant desire to travel.

In addition to experiencing sociocultural richness during their travels, people embark on journeys to learn more about themselves and to capture aesthetically pleasing spaces and places. Groffman indicates in his blog that traveling gives people a chance to discover their personalities in more depth and learn more about human life in general [11]. Candy expands upon this idea in her article, claiming that some people travel to remote places to experience different, potentially adverse, living conditions to gain a new perspective on life [8]. Santiago's article describes personal journeys in a more concrete manner, citing the scenic beauty that travelers capture as souvenirs and keepsakes once they return home [17]. Candy also remarks that people visit places in patterns, seeing variations of their favorite types of landmarks, such as waterfalls [8]. These types of traveling experiences offer people a welcome change from their daily routines and present opportunities to relive fond memories and develop new ones.

As people embrace these new experiences while traveling, they visit various landmarks to serve as symbols representing key milestones within their journeys. Hilleke and Hongkiat Lim expand upon Santiago's and Candy's ideas about landmarks, offering additional perspectives on their significance with regards to traveling. Hilleke explains in a blog post that landmarks, while used originally as tools for navigation, now serve as iconic places that tourists can visit and that others can recognize easily [13]. Lim adds to Hilleke's discussion within his article, describing how these places can represent pieces of history, culture, or architecture. Some landmarks, he claims, even represent important nationalist events and serve as sources of pride for the country in which they reside [15]. The symbolic weight that these places carry pique travelers' interests, motivating them to visit the landmarks.

While landmarks serve as temporary physical memories of a journey, photographs represent more persistent, symbolic memories of a traveling experience. Gary Arndt and Kate Berardo both explore why people capture pieces of their journeys with photographs so frequently, citing both memory recollection and cultural sharing as key reasons. Arndt explains in his blog post that, as people travel to new locations, their photographs symbolize the memories they acquire from a particular journey, allowing them to recall the fun experiences

they enjoyed from that trip more easily [5]. He and Berardo reference three key types of photography that occur during typical travel adventures [5, 6]:

- *Event-driven* – Some photographs serve simply to remind their owners of key events that occurred as they traveled, such as visiting monuments and eating ethnic food;
- *Art-driven* – Other photographs focus on interesting and unusual aspects of different places' scenery as their owners travel, serving as useful guides to others who wish to experience – and capture – these same vistas in the future; and
- *Culture-driven* – This rare category of photographs captures a collage of people, places, items related to the culture within a specific place. This approach to capturing a travel experience, unlike most others, casts the photographer's self and culture aside and focuses on the new experiences that the place offers.

No matter the medium or motivation that people use to capture their travel experiences, their journey artifacts encapsulate and symbolize stories of a traveling experiences that others can appreciate [6]. Furthermore, photographs can transcend linguistic and cultural barriers more easily, allowing a larger set of viewers to appreciate the snapshots representing a travel-based experience.

## 1.2. Android Smartphone Use Continues to Grow Quickly

Over the past several years, United States citizens have continued integrating smartphones into their daily lives. While senior citizens cite the need for assistance to adopt this new technology, they continue to use smartphones more frequently, as well. The Android platform enjoys a slight lead over its Apple counterpart in market-, subscriber-, *and* volume-based market shares, allowing Android application developers to enjoy a larger user base. These developers need to consider the variety in operating systems and screen densities that the Android platform contains, however, in order to offer an optimal experience for users. In this section, I describe increased smartphone adoption rates, Android's market share advantages over its Apple counterparts, and the Android platform's fragmented market share in order to illustrate the growing set of financial opportunities and logistical challenges that Android developers face.

### 1.2.1. Smartphone Penetration Rates in the United States Smallest among Senior Citizens

Smartphones have continued to grow in popularity within the United States over the past few years, particularly with older citizens, creating a need for universal usability within the

applications that these devices contain. According to the Pew Research Center's latest report on Internet use and technological advancement within the United States, 58% of all American adults own a smartphone, up from 56% in 2013 and 51% in 2012 [18, 20]. Given the country's population of almost 318 million (as of May 2014), this finding indicates that about 180 million individuals use these devices throughout the United States [4]. While smartphone users tend to achieve more popularity among younger people (83% of United States citizens own a smartphone), older individuals have also begun adopting the new technology, with 49% of those aged 50 to 64 using a smartphone [9]. Citizens within this age range have interacted with computers and smartphones for a smaller portion of their lives relative to their children's generation, so smartphone applications need to present user-friendly, easy-to-learn interfaces that allow them to understand and appreciate the information that these applications offer.

While smartphones have risen in popularity among working-age citizens of the United States, senior citizens have not adopted the technology as quickly. The Pew Research Center's 2014 report on senior citizen technology adoption indicates that only 18% of citizens aged 65 or older own a smartphone. Part of the reason why this demographic has continued using more traditional technologies, the center finds, lies in the perceived steep learning curve that smartphones possess. According to the report, only 18% of senior citizens within the United States would feel confident in learning how to use a smartphone without assistance while 77% of them would appreciate help from another person [19]. Smartphone application developers can use these findings as an opportunity to make their software as user friendly as possible so that, once these senior citizens have become familiar with smartphones in general, they can learn how to locate and use information within the application easily.

### 1.2.2. Market Share by Smartphone Platform

Android continues to perform well relative to other smartphone platforms, even considering that market share reporters use multiple metrics to convey a somewhat misleading sense of competition. During 2013, Android devices surpassed their iOS<sup>1</sup> counterparts in sales share within the United States. According to Matt Hamblen's report from ComputerWorld, Android sales increased from 46.2% to 50.6% of all smartphone sales between the final quarter of 2012 and the final quarter of 2013. Conversely, iOS device sales shares decreased from about

---

<sup>1</sup> Apple's iOS operating system forms the foundation for the company's mobile devices, such as the iPhone and iPad.

50% to 43.9% over the same time span [12]. When viewing smartphone platform shares based on the percentage of users subscribed to a cellular network while owning a particular type of device, Android performs similarly well relative to iOS devices. According to comScore's latest report, Android devices comprise 52.2% of the smartphone subscriber market share for users aged 13 or older (as of March 2014), up from 51.5% three months earlier. The corresponding iOS devices, on the other hand, witnessed a decrease in United States subscriber share, from 41.8% to 41.4%, during the same time interval [14].

These market- and subscriber-share victories for the Android platform cause more application downloads to occur worldwide on Android devices than their iOS counterparts, and Apple still dominates the application sales market share, this last remaining market-based advantage over Android continues to dwindle. In the first quarter of 2014, Android users downloaded 45% more applications on the Google Play Store than did their iPhone/iPad counterparts on the Apple App Store [16]. While the Android platform consists of more application *volume*, Apple's devices still enjoy larger application *sales*, primarily because many more applications available for iOS devices require payment than those for Android devices. According to a Flurry study conducted in April 2013, prices of applications developed for iOS devices continue to decrease steadily – 90% of App Store applications were free in 2013 compared to 80% in 2011. However, Android applications still cost far less on average; the study explains that Android applications cost \$0.06 on average, compared with still-steep prices on corresponding iPhone and iPad applications, whose costs average \$0.19 and \$0.50, respectively [10]. Within the last quarter or so, however, Android users in the United States and United Kingdom have expressed more willingness to pay for applications and make in-app purchases, yielding a 55% revenue increase in Google Play revenue during the first quarter of 2014 [16]. This emerging trend indicates that Android application developers could enjoy increased sales to accompany their existing increased volume over iOS versions of applications as the platforms continue to evolve.

### 1.2.3. Android Version Share

The Android platform releases new versions of its software on a relatively frequent basis, and vendors continue to create denser screen resolutions for these devices, so application developers need to understand the relatively fragmented set of software versions and screen

densities across the platform's user base. Table 1-1 below indicates the relative popularity of the different Android operating systems currently available on the market [2]:

Table 1-1. Distribution of Existing Android Devices by Operating System Version. (Accurate as of May 1, 2014)

Version	User-Friendly Operating System Name	API Level	Percentage of Devices Running This Version
2.2	Froyo	8	1.0%
2.3.3-2.3.7	Gingerbread	10	16.2%
3.2	Honeycomb	13	0.1%
4.0.3-4.0.4	Ice Cream Sandwich	15	13.4%
4.1.x	Jelly Bean	16	33.5%
4.2.x		17	18.8%
4.3		18	8.5%
4.4	KitKat	19	8.5%

As the above table shows, developers can ensure that nearly every Android user can install an application that works for Android API levels 8 and above. Even if developers chooses to use features exclusive to Ice Cream Sandwich (and later) versions of the operating system, they can still target over 80% of Android's user base. Developers not only need to select a minimum operating system version to decide which user interface elements and features will appear within the software, but they also need to design for multiple screen resolutions, as Table 1-2 on the following page shows [2]. This illustrates the lack of a "typical" screen density on Android devices; as a result, developers need to create user interface elements and images that can scale to multiple screen sizes, particularly medium, large, and extra-large ones (which collectively comprise roughly 77% of all Android devices on the market today).

### 1.3. Computers Do Not (Yet) Satisfy Expectations Regarding Digital Image Processing

Digital images now pervade many people's lives, regardless of whether they own the devices needed to capture them. Even those who are not professional photographers still interact with many types of digital images in their professional lives, including blueprints, copies of key documents, and screen captures [7]. Still, many individuals now own versatile digital cameras and/or smartphones and can transfer images from these devices onto their powerful personal

computers [7]. These devices contain sizeable collections, since many people use them while traveling and when significant events occur in their lives.

Table 1-2. Distribution of Existing Android Devices by Screen Density.  
(Accurate as of May 1, 2014)

Screen Density Category	Screen Density Range (in pixels per inch)	Percentage of Devices with This Screen Density
Low density (LDPI)	100-140	8.2%
Medium density (MDPI)	140-200	21.1%
High density (HDPI)	200-270	34.8%
Extra-high density (XHDPI)	270-350	20.8%
Extra-extra-high density (XXHDPI)	350-450	13.5%

While people appreciate digital cameras' and smartphones' ability to capture photographs, they often express surprise and disappointment at the lack of image analysis options available on these devices. Members of the general public often underestimate the difficulty involved with analyzing an image and identifying key elements – such as buildings, logos, and curves along a road – because the human visual system performs these actions easily and almost instantaneously [7]. Therefore, they expect the devices with which they interact to perform these image analysis tasks effectively. For example, many digital cameras and smartphones support facial recognition and red-eye detection, both of which improve the quality of digital family portraits.

Part of the reason why people express surprise at computers' lack of proficiency in analyzing images stems from the machines' relative mastery at textual analysis. For quite some time now, computers have helped navigate through textual information, identifying patterns and detecting the presence of certain terms that a document uses. People who view digital photographs, particularly photographers, would find similar features within image-filtering

technology particularly useful since they could then specify the elements of a scene they wish to capture and compare their ideas with those who have preserved similar types of scenes through existing photographs.

Even many though image-processing features still struggle to enter production systems, the recent advancements in technology allow researchers to perform and evaluate the related algorithms far more easily. A few decades ago, developers could complete only rudimentary image-processing operations given the poor operating system and hardware support for transferring and decoding images. Nowadays, however, they have access to several powerful high-level languages, including C, C++, and Java, each of which provides application programming interfaces (APIs) for completing basic but important image-processing tasks on images. [7]. These tools allow developers to implement more advanced and intricate image-processing algorithms, increasing the functionality of the digital imaging devices that use them.

#### 1.4. Challenges Associated with Current Options Available

Prospective travelers have access to the tools and information necessary for them to make informed decisions regarding the places they visit during a journey, but this information lacks the necessary organization for them to complete this task quickly or while traveling. In order to explore the different aspects of a particular place – such as general facts, photographs, historical events, and other visitors’ experiences – these travelers need to visit a separate website for each category of information. This disjoint, fragmented research process creates cognitive overload for these people, preventing them from forming an idea of a location’s “big picture” experience. Even when using mobile devices, these travelers experience similar problems, particularly since the applications they use for research contain information from only one source or present only one piece of information at a time. While a smartphone’s screen size limits the amount of information visible at any given time, many of these applications make meager attempts to tie different facts together with advanced navigation tools.

Even the mobile applications that offer photographs of a specific location tend to display these images as unordered collections, preventing users from exploring a particular aspect of the visual context surrounding that place. In particular, most of these applications lack image filters that show only photographs matching a certain configuration or containing a specific type of feature. For locations with several characteristic components, users would appreciate being able



to explore these parts of a place virtually so that they could embrace the culture and environment that the place offers once they arrive there physically.

### 1.5. Addressing These Challenges: Creating the Lighthouse Navigator Application

To address the drawbacks currently facing prospective travelers who wish to explore a location they wish to visit more easily, I created an Android smartphone application called the Lighthouse Navigator. As a proof-of-concept implementation, this application focuses on the 117 coastal lighthouses across New England. The software serves as a centralized repository of information about a given lighthouse, and it offers filtering options so that prospective visitors can explore specific components of the scenery surrounding the lighthouse.

#### 1.5.1. High-Level Overview of Application

The application contains a simple, search-based hierarchy that leads to a series of content-based information screens. Upon starting the application, users see a “Welcome” screen that invites them to find lighthouses near their current location or search for a specific lighthouse after selecting a specific state. After performing either search operation, users see a “Search Results” screen, displaying the lighthouses that best match their queries. A “View” button appears next to each lighthouse, allowing users to view more information about that landmark. This information appears within four separate screens:

- Information – Presents general facts about the given lighthouse, including hours, location, and visitor access allowances;
- Photographs – Features a gallery of photographs from Flickr’s servers that feature the given lighthouse, along with a series of filtering options for viewing subsets of this gallery;
- History – Includes several important historical events regarding the lighthouse; and
- Reviews – Provides users with other visitors’ evaluations of the area surrounding the lighthouse, courtesy of Yelp and TripAdvisor.

While these screens appear separately, users can change screens easily using a drop-down menu near the top of the application’s user interface. This connection among screens enhances the narrative that I convey about lighthouses within the application.

### 1.5.2. Project Goals

This proof-of-concept application allows users to experience the narrative space surrounding each New England lighthouse and create shared experiences of that space through photographs, knowledge of the monuments' historical events, and visiting experiences. The application also offers advanced filtering options for photographs featuring these seaside landmarks in order to simplify prospective visitors' planning processes for traveling to these places. The application also serves to raise awareness of the beauty and historical interest that each lighthouse possesses, potentially inspiring users to join a lighthouse visitor community that assists local and regional lighthouse societies in preserving these landmarks.

In order to create as feature-rich and user-friendly of an application as possible, I created several high-level goals for this project:

- Explore ideas related to concrete places and abstract spaces to present appropriate contexts for lighthouses within my application;
- Design the user interface using an iterative, audience-involved process to ensure optimal usability within the target audience;
- Create efficient and easily extensible data structures within the application to allow for simple updates to the information it presents; and
- Develop a set of image-processing algorithms that can classify photographs of lighthouses and identify the presence of color- and shape-based features within these images.

By fulfilling these goals, I created an application that lighthouse visitors express interest in downloading.

To further simplify the scope of this project, I worked closely with a regional lighthouse society – The Lighthouse Preservation Society – to learn more about efforts to maintain lighthouses in New England and to contact visitors of these coastal landmarks who have joined this society. Founded in 1983, the organization preserves lighthouses for future generations to enjoy, opens once-deactivated lighthouses for people to enjoy them now, and records the histories of buildings and keepers associated with these lighthouses [3]. These lighthouses represent important landmarks within the New England tourism industry, so maintaining these monuments benefits the regional economy as a whole. According to the “Accomplishments”

page on the organization's website, the society sponsors National Lighthouse Day and coordinated 70 coastal lighthouse celebrations during the first installment of this event in 1989. It has also funded the nomination of Maine's lighthouses to the National Register of Historic Places and has restored and relit several lighthouses [1]. Jay Hyland, the president of the society, continues to monitor the maintenance efforts of New England lighthouses today and also manages the "Dine at the Top of the Lighthouse" landmark in Newburyport, allowing visitors to enjoy gourmet meals atop the tower [1]. Each lighthouse in New England appears within a unique scenic, historic, and experiential space, which map very well to the "Photographs," "History," and "Reviews" screens that my application contains.

### 1.6. Overview of Report's Contents

Throughout the remainder of this report, I describe the process I used to create the application, from research to development. Each chapter focuses on a specific aspect of the process. Chapter 2 discusses the features and drawbacks of existing travel-based websites and applications and presents a high-level overview of ideas related to space and place. Chapter 3 presents the application's final design and the process involved in creating the different screens that users see. Chapter 4 describes the application's high-level implementation structure as well as the data structures, background threads, and image-processing algorithms that appear within the application logic. Chapter 5 reports on the key findings from testing the image-processing algorithms and conducting usability studies with prospective users. Finally, chapter 6 offers some reflections on the project as a whole and offers recommendations for moving the project forward.

## 2. Presenting Current Applications, Theoretical Foundation for Work

Several travel-based websites and applications exist within the market today, but none of them consolidate all aspects of a certain set of destinations, nor do they offer the immersive experience that users crave when investigating the spatial narrative of a place. This chapter discusses the features and drawbacks associated with six websites and eight Android applications, which collectively contain information, photographs, historical details, and recollections of experiences. It provides a theoretical foundation for my image-processing algorithm implementations, and offers some commentary on Henri Lefebvre and Yi-Fu Tuan's ideas of space and place.

### 2.1. Current Websites Each Offer Narrow Scope of Travel-Based Information

A number of dedicated websites deliver travel-based information to users planning on visiting specific landmarks, including lighthouses. Each of these websites, however, focuses on a specific aspect of traveling. Therefore, while website users planning on visiting a specific location can learn a great deal about, say, the scenery of a given location, they must visit a variety of websites in order to capture a clear idea of the scenery, key attractions, and visitor experiences that a particular place has to offer.

When investigating these travel-based websites, I found that they each discussed information belonging to one or two specific categories of travel information. Table 2-1 below maps each website to the category of information it displays. In particular, it illustrates how some of the websites featured in this section discuss more than one aspect of traveling (such as pictures *and* experiences). For the purposes of this review, I have designed a *primary category* of information that each website contains. Each of these websites focuses on travel-based landmarks in general, not any specific categories such as statues, bridges, or lighthouses.

Table 2-1. Primary and Secondary Categories of Information on Travel- and Experience-Based Websites

Website	Pictures	Locations	Experiences
<b>Flickr</b>	Primary	Secondary	---
<b>Panoramio</b>	Primary	Secondary	---
<b>Yelp</b>	Secondary	Primary	Secondary
<b>TripAdvisor</b>	---	Primary	Secondary
<b>Cowbird</b>	Secondary	---	Primary
<b>Geocaching</b>	Secondary	Secondary	Primary

Each website featured in the above table contains a great deal of information belonging to one category (the “Primary” category) and may contain some information characteristic of other categories (the “Secondary” categories). This category transcendence shows the versatility that some of these websites possess. Nevertheless, none of them encapsulate all of the information that appears within the application I have developed, preventing prospective travelers from appreciating the entire context of a place from one location in cyberspace. Readers can find a more detailed discussion of each website in the above table under the heading matching its primary category.

#### 2.1.1. Organized Digital Photograph Collections Rely on Search & Filter, Not User Popularity

Pictures showcase the interaction of color and light across a landmark, which viewers can understand regardless of their cultural and linguistic backgrounds. The following websites, Flickr and Panoramio, recreate a particular location’s visual element by presenting pictures of a landmark or monument from a variety of users and viewpoints. On each site, users can examine a picture’s properties – such as its resolution and location – and can even see how pictures relate to one another geographically on a virtual map. The layering of locations and photographs allows visitors of these websites to appreciate the different purposes for traveling to a particular location, such as viewing interesting scenery or exploring historic landmarks. The method in which these picture-based websites present photograph information and locations, however, affects the degree to which Android developers, such as myself, can incorporate this information into a smartphone application. In particular, Flickr’s API presents location information in a more organized and fair way than Panoramio’s interactive map and photograph popularity rankings.

#### *Flickr Presents Balanced Searching and Filtering but Favors Developers with Location Information*

Flickr allows end users and application developers alike to search for photographs of a landmark that satisfy a specific set of criteria. Both sets of users can then view and filter these results quickly and easily. By affording this functionality, Flickr presents photographs in a fair and focused manner, honoring all types of photographers and photograph types capturing a particular place.

### Search for

*Tip: Use these options to look for an exact phrase or to exclude words or tags from your search. For example, search for photos tagged with "apple" but not "pie".*

All of these words    
 Full text  Tags only   
 None of these words:

### Search by content type

*Tip: Check the boxes next to content you'd like to see come up in searches.*

- Photos / Videos   
 Screenshots / Screencasts   
 Illustration/Art / Animation/CGI

### Search by media type

*Tip: Filter to only display either photos or videos in your search results.*

- Photos & Videos   
 Only Photos   
 Only Videos   
 HD videos only

### Search by date

*Tip: Use one or both dates to search for photos taken or posted within a certain time.*

Photos taken  after  before    
 mm/dd/yyyy mm/dd/yyyy



*Tip: Find content with a Creative Commons license. [Learn more...](#)*

- Only search within Creative Commons-licensed content   
 Find content to use commercially   
 Find content to modify, adapt, or build upon

Figure 2-1. Flickr's user-level advanced search lacks location-based filtering.

This page, showing every “Advanced Search” option available to general users, lacks a location-based filter despite including date- and license-based filters.

Flickr offers general users a convenient two-step search and filter system to find specific types of photographs quickly, but it requires them to complete an unnecessarily long-winded scavenger hunt through its website to associate photographs with a particular location. When end users navigate to Flickr’s main page, they enter a set of words, requesting that all desired photographs contain at least one of these words in their titles, tags, or descriptions. After completing a search, end users see the first round of results and select a photograph to view more

information about it, including the photographer, licensing restrictions, and location (if the photograph is geotagged). Some users remain satisfied with this general search-and-view process, and Flickr's advanced features remain conveniently tucked away for them. Travelers who use Flickr as "power users," however, usually like to refine the results even more, and Figure 2-1<sup>2</sup> on the previous page shows that Flickr's "Advanced Search" page does not present any method of location-based filtering. Instead, users must navigate to the "World Map" page, located under the "Explore" menu that appears in the top-left corner of every page on Flickr.

As Figure 2-2<sup>3</sup> shows, the process for finding a lighthouse is far less intuitive and user-friendly than it could be. When the page first loads, Flickr populates the map with a seemingly random assortment of geotagged images from around the world. When users zoom in to a particular location, they would most likely expect the page to refresh automatically, but it does not. Instead, they need to select the "Search the map" button, which appears in an inconspicuous location on the page, then enter in a search term and location within a new navigation bar that appears, thankfully, in the same relative location on the screen.

This bar's transparency, however, forces users to forgo the text box hints because the suggestions remain too difficult to read. Once users finally figure out how to zoom in on a location and enter a search term, Flickr offers a good interactive interface, but most "power users" will likely accumulate too much frustration and confusion to ever reach this part of the website. I look to present a more satisfying user experience when users of my application search for photographs by location.

While Flickr does not present an intuitive interface for finding photographs based on location to end users, the website's application programming interface (API) makes this process far simpler while retaining the simple search and filter operations that general users can complete. Once developers have obtained an API key from the page entitled "The App Garden: Create an App," they can enter URLs with different query strings describing the types of photographs they wish to view [5]. In order to complete a search for photographs based on location and licensing restrictions, they can use the `flickr.photos.search` method, as described on a page within Flickr's API documentation [4]. A sample request-response pair

---

<sup>2</sup> Image from <https://www.flickr.com/search/advanced/?q=pemaquid%20point%20lighthouse>, accessed April 11, 2014.

<sup>3</sup> Image from <https://www.flickr.com/map>, accessed April 11, 2014.

using this method appears in section 4.5.1, “Structuring Calls to Flickr’s API to Obtain Geotagged Images,” of this report.

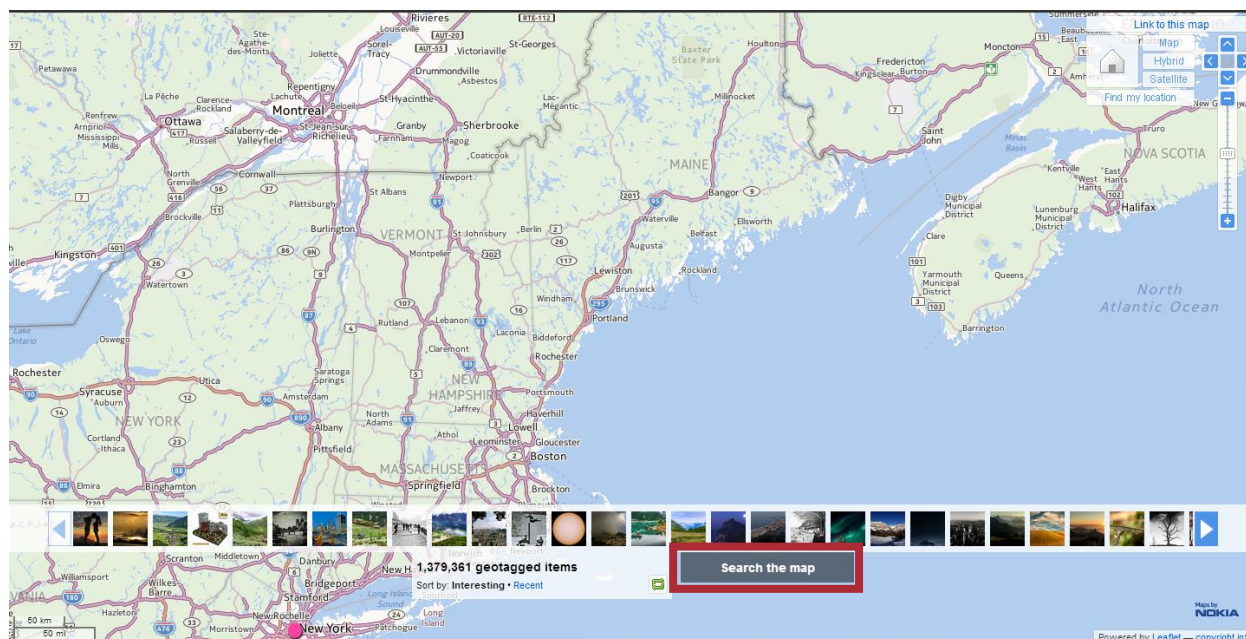


Figure 2-2. “Search the map” button unnecessarily hidden within Flickr world map.

When users zoom in on a region of interest, the pink dots indicating the relative locations of geotagged photographs do not refresh, and the “Search the map” button appears in the bottom-center area of the screen (shown with a red box around it above). This obscure location makes the entire page unnecessarily user-hostile.

### *Panoramio Panders to Popularity, Preventing Indie Photographers from Achieving Visibility*

While Flickr presents a method of finding each photograph satisfying a particular set of geographic and licensing requirements, Panoramio’s search mechanisms support – and favor – better-known locations. When users wish to explore photographs taken from a specific location, they can begin entering the name of that location in the search box in the top-center area on Panoramio’s home page. The website then uses its auto-complete feature to suggest several popular locations. If users wish to explore a location that Panoramio does not recognize, they must either (1) search for the area based on tags added to photographs around that location on Panoramio’s “Tags” page or (2) open a virtual world map and zoom in on the location of interest on the website’s “Photos of the World” page [28, 29]. The extra steps necessary to find less common locations cause users to favor more popular locations when searching for photographs, reducing the likelihood for a developing area to gain visual prominence on the website. After



users complete a search for photographs, they see the search results appear on a virtual map and as a series of thumbnails, as shown in Figure 2-3 below<sup>4</sup>:

By default, the website ranks these search results based on each photograph’s “popularity.” The website bases an image’s popularity on the following criteria:

- Number of views;
- Number of users who have marked the picture as one of their “favorites;”
- Number of users who have commented on the photograph; and
- Resolution of the photograph.

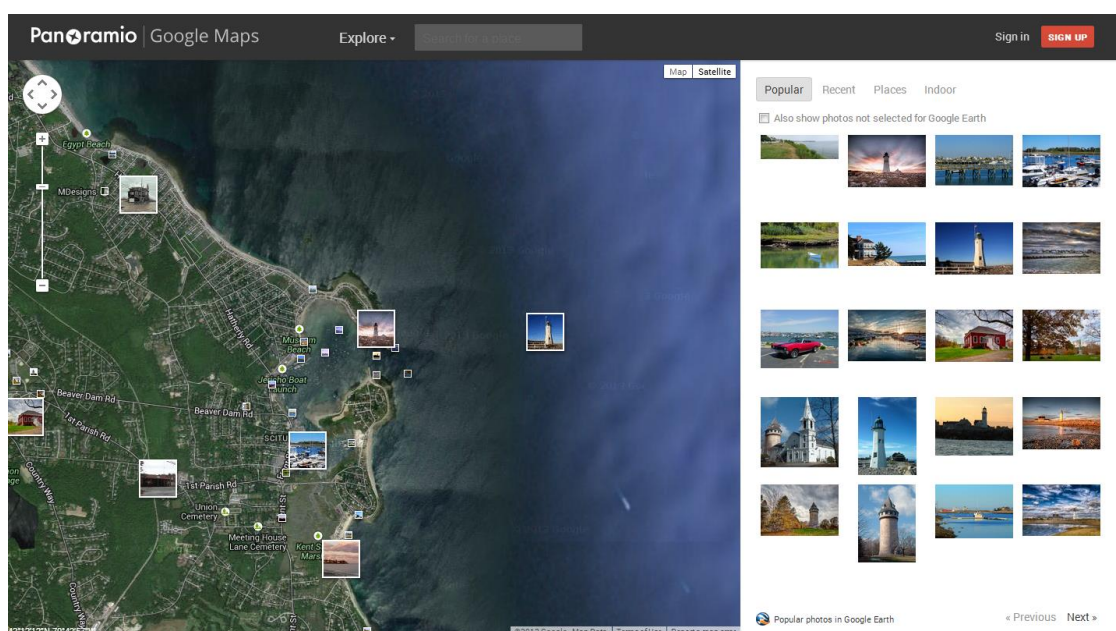


Figure 2-3. Sample Search Results Page on Panoramio

The most popular photographs appear first in search results, as Panoramio explains on its “Understanding popularity in Panoramio” page [30]. This “popularity algorithm” transforms Panoramio from a simple photo-searching website to a significantly more complex “photography social network,” where the places themselves accrue cultural capital. This ranking mechanism also reduces the chance that users will find collections from a photographer who is new to Panoramio, at least initially. New photographers might consider uploading their collections onto a different website as a result.

<sup>4</sup> Screen capture from <http://www.panoramio.com/map/#lt=42.203262&ln=-70.715817&z=3&k=2&a=1&tab=1&pl=all>, accessed September 10, 2013

Even though Panoramio offers a less than ideal method of searching for photographs and exploring specific locations, the image details screen for each photograph supports some interesting opportunities for exploring supplementary information about the image. The right-hand side of the page displays a virtual map indicating the photograph's location as well as a set of small thumbnails beneath this map that display nearby photographs. In addition, this right-hand sidebar displays some of the camera parameters associated with a photograph, such as the exposure time and focal length for the picture. Aspiring photographers find this information useful as they determine how best to capture a particular landmark. Panoramio presents this set of data within a single "details" screen, without forcing users to select multiple links to discover each image's location and camera data. Therefore, viewers spend less time gaining a better understanding and sense of appreciation for the location surrounding a particular photograph.

#### 2.1.2. Locations

The following websites provide information about the location associated with a particular landmark. In addition to offering reviews of a point of interest, these websites present related locations around a monument and suggest similar places for users. While Yelp and TripAdvisor support searching functionality similar to that in my application, Yelp includes "elite users" while TripAdvisor ranks nearby attractions, features that my application does not include.

#### *Yelp Introduces Popularity into Sorting Results, Uses Two-Step Search*

Yelp is an electronic listing website that connects top local businesses with their customers who use Yelp's services, according to the website's "About Us" page. The businesses themselves can upload paid advertisements onto the website, but Yelp clearly marks these promotional materials as sponsored, and the site prevents paying advertisers from modifying reviews that they have already uploaded. In addition, Yelp uses a filtering mechanism to block comments that seem fabricated automatically [37].

One of Yelp's most interesting features involves its set of elite users, each of whom serves as an ambassador to his or her local area and receives nominations from other users to attain "elite" status. These "superusers," with their visible badges on the Yelp website, provide some of the most influential reviews of places within their respective regions of expertise. They also receive new friends, invitations to parties at least once a month, and a variety of collectibles from the Yelp community. Yelp recommends on its "Yelp Elite Squad" page that, if users wish

to enjoy this prestigious status, they need to demonstrate an unbiased, supportive set of opinions about the locations they review and leave comments for landmarks that other users will find interesting or relevant [36]. By leaving such trustworthy content on the website, these users earn respect from their peers more easily.

Yelp offers a good balance of control and usability on the pages with which users interact in order to conduct and review a search for a particular landmark. To begin the process, users can enter the name and location of a landmark into the two text boxes that Yelp provides in the top navigation area on its homepage. Once the user selects one of the relevant search results, the website loads a summary page for that landmark. This summary page includes photographs and descriptions of experiences in addition to details about the location itself, as Figure 2-4 shows below<sup>5</sup>:

The screenshot displays the Yelp interface for the 'West Quoddy Head Lighthouse and Visitors Center'. At the top, the title is followed by a 'Write a Review' button and options to 'Add Photo', 'Share', and 'Bookmark'. Below the title, there are 4 reviews indicated by stars and a 'Details' link. A red box labeled '1' highlights a map showing the location at South Lubec Road, Lubec, ME 04652. Another red box labeled '2' highlights a gallery of photos, including a sunrise view from the lighthouse. A third red box labeled '3' highlights a recommended review by Jason P. from Medford, MA, dated 9/29/2011, which includes a note about RV access and a 'Useful' button.

Figure 2-4. Viewing location information, photographs, and user experiences for a point of interest on Yelp. Within the results page for a particular location, users can (1) view a map of the surrounding area as well as suggestions for nearby locations, (2) access photographs of the location, and (3) read about other users' experiences from visiting the point of interest.

<sup>5</sup> Image from <http://www.yelp.com/biz/west-quoddy-head-lighthouse-and-visitors-center-lubec>, accessed April 9, 2014.

After this page loads, several thumbnails of the landmark appear. Users can select one of these thumbnails to view a larger version of the thumbnail as well as dozens of similar photographs beneath it. The navigation area to the right of the vital information contains a small map of the area surrounding the landmark and offers suggestions for related places that the user may want to consider, as well.

Yelp lists user experiences and reviews, sorted according to “Yelp Sort” by default. This sorting algorithm takes into account how recently a user posted the review and the extent to which other users found this review useful. People who examine the user reviews can also choose to sort them by date and by rating. If a user has signed into Yelp before viewing search results, (s)he can also filter the results to show only those by “elite” users and Facebook friends. Each rating consists of a score (on a five-point scale), a date, and a description. The comments tend to focus on the logistics of visiting the location, the helpfulness of the staff at the landmark, and how the point of interest reflects the essence and culture of the nearby area.

Within my application, I provide a similar two-step process that allows users to search for a lighthouse. Since mobile applications favor interaction with widgets over keyboard typing, the application directs users to select a state from a list, then select a lighthouse within that state from a second list. The “Photographs” screen within my application provides the same geographic context as the map along the right-hand side of Yelp’s landmark information screen, as well.

#### *TripAdvisor Associates and Compares Places with Nearby Attractions*

While Yelp focuses on points of interest during a journey, TripAdvisor serves to guide users through their entire vacations. The latter site offers over 100 million pieces of candid advice about locations at which to stay and visit, such as hotels and points of interest, and even provides free travel guides, according to its website [31]. These travel guides include maps for different cities and present information about the restaurants, attractions, and lodging options in that city [16]. The guides and the website that provides them provide a more holistic sense of the location in which a particular landmark is situated.

TripAdvisor extends this general information approach to its searching process. Users begin looking for a location by entering a point of interest in the search bar near the top-right corner of the homepage. A results screen then appears, showing restaurants and hotels in addition to individual landmarks. Users can search for landmarks only by selecting the **Attractions** option

in the **Refine Search** box on the left-hand side of the page. In addition to displaying information about the point of interest, the main summary page displays users' experiences as well as links to other attractions, food, and lodging options, as Figure 2-5 shows below:<sup>6</sup>

Beneath the vital information near the top, several thumbnails of the point of interest appear. Users can select any of these thumbnails to open a pop-up window containing a larger version of the thumbnail as well as a scrollable list of thumbnails along the bottom of the window. The summary page also displays the ranking of the particular landmark among the other points of interest in the surrounding area as well as the average review score (out of 5) and the number of reviews. The individual reviews for the location appear further down on the page. By default, these ratings are sorted by date and show reviews written in English first. Each review contains a date, a score (out of 5), a description, and a number of "usefulness" votes from other users in the TripAdvisor community. These reviews tend to focus on sensory details – in particular the visual aspects of the location – as well as the overall experience and tips for other attractions to visit nearby. Finally, the right-hand side of the page contains links to nearby hotels, other attractions, and restaurants.

The screenshot shows the TripAdvisor page for West Quoddy Head Light. At the top left, there is a main image of the lighthouse and a summary box with the text: "Ranked #2 of 11 attractions in Lubec", "107 Reviews", "Certificate of Excellence 2013", and "Type: Lighthouses". Below this is a gallery of 52 visitor photos, with a "1" next to it. The review section shows a "107 reviews from our community" with a "2" next to it. The visitor rating is 4.5 stars, with a breakdown: Excellent (75), Very good (27), Average (4), Poor (1), and Terrible (0). A sample review by Donna T. from Eastport, Maine, is titled "Shore hikes" and reviewed on March 28, 2014. The review text describes the scenic views and hiking trails. To the right, there is a "Browse nearby" section with a map and a "3" next to it, listing nearby dining options like Breakfast, Eastland Motel, and Water Street Tavern & Inn.

Figure 2-5. Viewing location information, photographs, and user experiences for a point of interest on TripAdvisor. Within the results page for a particular location, users can (1) access photographs of the location, (2) read about other users' experiences from visiting the point of interest, and (3) view a map of the surrounding area as well as links to nearby locations.

<sup>6</sup> Image from [http://www.tripadvisor.com/Attraction\\_Review-g60947-d655485-Reviews-West\\_Quoddy\\_Head\\_Light-Lubec\\_Maine.html](http://www.tripadvisor.com/Attraction_Review-g60947-d655485-Reviews-West_Quoddy_Head_Light-Lubec_Maine.html), accessed April 9, 2014.

Unlike TripAdvisor, my application does not list attractions near a given lighthouse, since the application serves to place users within the lighthouse's visual and experiential space instead of those nearby. Also, since a sizeable distance exists between neighboring lighthouses along a coastline, I do not compare them directly in the manner that TripAdvisor does.

### 2.1.3. Experiences

While websites discussed earlier in this section focus primarily on the physical aspects of a location, the following websites emphasize the mental and emotional aspects of visiting a location and interacting with a point of interest on a personal level. The pages on these websites provide visitors with stories related to visiting a location and how these adventures illuminate the visitors' lives. In particular, Cowbird focuses on providing stories that last a long period of time while Geocaching uses a social treasure hunt to increase the popularity for specific landmarks.

#### *Cowbird Focuses on Timeless Aesthetics*

One website dedicated to experiences, Cowbird, turns the idea of learning about a location through short, fast-paced bursts of information on its head. According to this website's "About" page, it is dedicated instead towards creating and expanding a library of human experiences, creating a repository of collective wisdom that can grow over the course of generations. The stories that describe these experiences on Cowbird contain more depth and enjoy more long-term relevance than most other social media postings, in particular those on Facebook and Twitter. Cowbird even displays this thoughtful presentation paradigm on its "About" page, depicting testimonials from users of the website who share their appreciation of the slower-paced, more intense connections they can form with the more in-depth narratives that the website offers. Users agree that these stories allow them to categorize their thoughts in meaningful ways, enriching each of their life experiences [9]. Cowbird's "Culture" page reminds users to post original content in a humble manner that respects the diversity of cultures who post their life stories on the website [10]

In order to help modern-day Internet users navigate through the unusual cyber-landscape of thoughtful narratives, Cowbird has published a set of guidelines for finding and crafting stories on its "Guide" page. The primary posting unit on Cowbird is the "story," which contains one page of text, one picture, and (optionally) an audio track.<sup>7</sup>

---

<sup>7</sup> Cowbird recognizes users who sign up for a paid subscription as "citizens." These "citizens" can present more than one page of text and/or multiple pictures in a single story.

The textual element of each story must include:

- Characters (“who”);
- Description (“what”);
- Date (“when”); and
- Location (“where”).

Users can find stories satisfying a specific set of criteria – such as location, age range of storyteller, and category of story – by using Cowbird’s advanced filtering mechanism, shown in below:<sup>8</sup>

Once users have found a particular story to read, they can select the **Connections** link on a story to view related ones. They can also retell the story or create a derivative story by “sprouting” their own narrative from the one appearing in the search results. Some users work in tandem to create groups of stories, which Cowbird calls “Projects” [11].

While Cowbird does not yet have a saga (extensive project) dedicated to lighthouses, it does allow users to apply a “lighthouse” tag to individual stories related to these landmarks. These stories tend to focus on the beauty of the lighthouse and its surrounding landscape as well as on how lighthouse symbolize navigating through life’s obstacles and the passage of time. A sample story about lighthouses on Cowbird appears in Figure 2-7 on the following page.<sup>9</sup>

Figure 2-6. Filtering Options within Cowbird.

Cowbird offers a powerful filtering system to assist users in finding stories that match specific criteria. These filters include: age range, gender of storyteller, main character’s role, and location.

<sup>8</sup> Image from <http://cowbird.com/search/>, accessed April 9, 2014.

<sup>9</sup> Images from [http://cowbird.com/story/40159/Lighthouse On Slapton Sands/](http://cowbird.com/story/40159/Lighthouse%20On%20Slapton%20Sands/), accessed April 9, 2014.

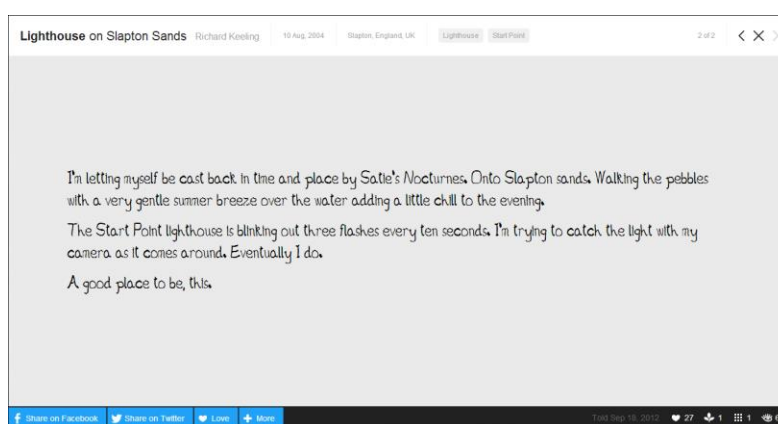
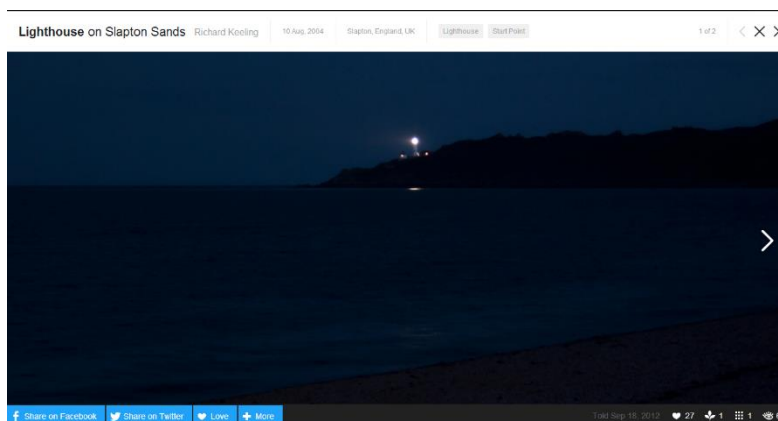


Figure 2-7. Story about a Lighthouse on Cowbird.

This story discusses the joy of capturing the fleeting flash of the light that the lighthouse lens produce during the nighttime hours. The photograph (top) complements the story about it (bottom) quite well. As of April 9, 2014, the story's statistics include: 27 "loves" (likes or +1's), 1 "sprout" (this story has inspired one other), 1 "collection" (compiled into one other user's compilation of stories (s)he enjoys reading), and 64 "read" (number of users who have viewed the story).

### *Geocaching: A Social Treasure Hunt that Increases Popularity of Specific Places*

While Cowbird focuses on the discovery of individuals' thoughts and feelings, Geocaching is a website dedicated to finding treasure boxes hidden around the world. According to the website's home page, the project involves 2 million locations and 5 million users. These users embark on a modern-day treasure-hunting experience, searching for a location using coordinates on a GPS or smartphone and sharing the experience of finding the location with fellow members of the community online [18]. The project's parent company, Groundspeak, aims to bring together online communities in a specific physical space, encouraging more outdoor exploration and socialization in the process [2].



The “Geocaching 101” page describes the simple process of exchanging ideas and items using these hidden geocaches. People who find these caches need to sign a log book online, sharing their experiences of finding the cache with others in the community. These hunters can take items from the geocache, but they must then leave something of equal or greater value in its place [17]. Some caches contain special items called “trackables,” which members of the Geocaching community can follow around the world. Some of these items merely symbolize users’ traveling experiences, but others contain promotions from Groundspeak that serve as incentives for users to find [19].

The website dedicates a page to each cache that players can find using their GPS-enabled devices. The challenge of finding a particular cache contains three aspects:

- Difficulty (out of 5);
- Terrain (out of 5); and
- Size (out of 4).

After listing this vital information, the page displays a brief description of the cache’s location that may include an encrypted hint if a player needs extra help finding a particular treasure box. The bottom of the cache listing includes a log book that contains a set of comments and a gallery of images, both of which appear in Figure 1<sup>10</sup> on the following page. The comments for a given cache tend to focus on the context of the players’ journeys, such as their trips to a set of caches, and the enjoyment of finding the cache. Two of the most common comments include: “fun find” and “TFTC,” which stands for “Thanks for the cache.” The images within a cache gallery usually show people around the cache as well as the landscape surrounding a picturesque treasure box.

---

<sup>10</sup> Images from [http://www.geocaching.com/geocache/GC15114\\_fundy-tides?guid=63b0df25-5359-4c59-8f1d-958d9a883e19](http://www.geocaching.com/geocache/GC15114_fundy-tides?guid=63b0df25-5359-4c59-8f1d-958d9a883e19) and <http://www.geocaching.com/seek/gallery.aspx?guid=63b0df25-5359-4c59-8f1d-958d9a883e19>, both accessed on April 9, 2014.

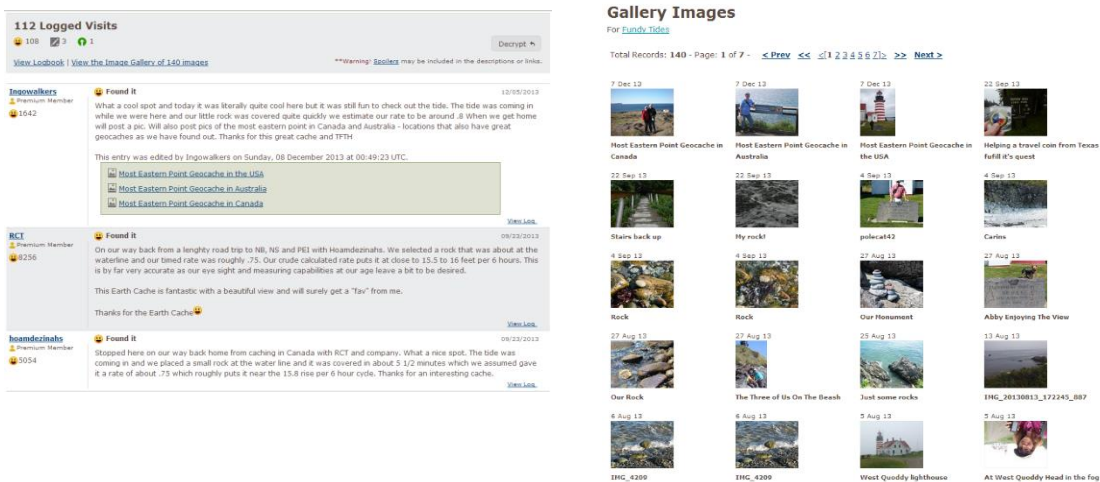


Figure 2-8. Log book entries and image gallery for the “Fundy Tides” geocache on Geocaching. These two screenshots illustrate the log book (top) and image gallery (bottom) features that Geocaching associates with each location that players can find. Note that the log book displays the searching success (or lack thereof), using a smiley face at the top of each entry, and that the image gallery depicts images of several landmarks surrounding the cache.

## 2.2. Existing Smartphone Applications Combine to Form Narrative Space

The Google Play market and Apple’s App Store already feature several applications that allow users to learn more about specific points of interest. Some of these applications serve as guides to a particular area while others focus more on the historical or experiential aspect of a landmark. Table 1 on the following page lists the applications that I discuss in this section, along with several high-level details about each program. I attempt to weave together a set of applications that describe location, history, experiences, and lighthouse information, which appear as content units within my application.

In general, these applications offer some compelling methods of interacting with the pieces of information they provide, but they often overestimate users’ abilities to understand the technological nuances of Android devices, use the application while connected to the Internet, and interpret ambiguous icons.

### 2.2.1. Location Guides Compartmentalize Categories of Information

These applications provide holistic overviews of a particular area for users who wish to visit that location. These overviews include places to visit and activities to complete while in a particular area. While the following products contain an abundance of information, users who are looking for a specific piece of information will likely become frustrated as they navigate through sections and screens that contain details irrelevant to them.

Table 2-2. Existing smartphone applications featuring specific points of interest.

The following applications serve a number of different purposes, but each focuses on virtual exploration of an area or place. All statistics shown here are accurate as of May 5, 2014.

\*All rating scores are out of 5.

Name	Publisher	Platform	Min. Version Required	Number of Installations	Average Rating Score*	Number of Users Rating App	Date Last Updated
<b>Boston City Guide</b>	TripAdvisor	Android	2.2	50,000 – 100,000	4.7	636	March 7, 2014
<b>NYC Way – Everything NYC</b>	MyCityWay	Android	1.6	100,000 – 500,000	4.1	686	December 10, 2013
<b>HISTORY Here</b>	A&E Television Networks Mobile	Android	2.3	100,000 – 500,000	3.7	624	February 5, 2014
<b>America’s National Parks</b>	LBS Solutions	Android	3.0	10,000 – 50,000	3.5	71	February 8, 2014
<b>My Disney Experience – WDW</b>	Disney	Android	2.3.3	1,000,000 – 5,000,000	4.0	13,069	April 4, 2014
<b>Foursquare</b>	Foursquare	Android	2.2	10,000,000 – 50,000,000	4.2	327,633	April 29, 2014
<b>US Lighthouses</b>	Lighthouse Friends.com	Android	4.0	100 – 500	4.8	12	January 16, 2014
<b>Lighthouse Locator</b>	MapMuse	Android	4.0	10 – 50	N/A	0	January 9, 2014

*Boston City Guide Divides Text into Headings, Allows for Continued App Use during Downloading*

Boston City Guide is one of several city-based products that TripAdvisor publishes to provide individuals with an electronic pocket guide to an area, including information about places to visit, eat, and stay. According to the application listing on Google Play, TripAdvisor members submit descriptions and reviews of Boston-area locations, and professionals then edit this information before the company publishes it to the application. The application also allows users to follow several pre-loaded paths and find their respective locations using their phones' GPS functionality. They can also create journals of their respective journeys, complete with entries containing photographs and textual descriptions. TripAdvisor even avoids the need for a constant Internet or data connection; all information about Boston is stored directly on users' phones after completing an initial update [7].

The application contains several levels of information that users can access. After it loads a "welcome" screen, it prompts users to download more detailed maps and reviews, assuring them that this data-gathering process occurs in the background and that they can still use the application in the meantime. My application presents a similar mechanism, allowing users to view details about a particular lighthouse while the application downloads photographs from Flickr. While waiting for the application to deliver more detailed information to their devices, users can read information about Boston by following the "Learn About the City" link. This link leads to a list of links to descriptions about specific aspects of the city, from neighborhoods to transportation options.

While the general overview article contains a discouragingly long block of text and the description of neighborhoods lacks a much-needed table of contents, the transportation page presents several levels of headings quite nicely, separating types of destinations from modes of transportation in a very reader-friendly manner. Also, some users have left reviews complaining that the application does not provide sufficient details about Dorchester or Davis [7]. However, the former area is probably too dangerous for most tourists to visit, and the latter is in Cambridge, not Boston, so TripAdvisor need not provide more than minimal information about either neighborhood.

Once the application has finished downloading more specific information, users can view these details from a variety of perspectives. Users can view listings of specific locations, such as the Neptune Oyster restaurant shown in Figure 2-9 on the following page. They can even select

the picture on a given results page to display a larger version of the image and a link to view additional photographs of the location associated with the listing. Users can also embark on guided tours that the application calls “itineraries.” An example of one such itinerary appears in Figure 2-10 below. The application breaks the guided path into a series of waypoints, each of which contains a textual description taking up one to two times the height of the screen, a photograph, and the distance from the user’s current location.

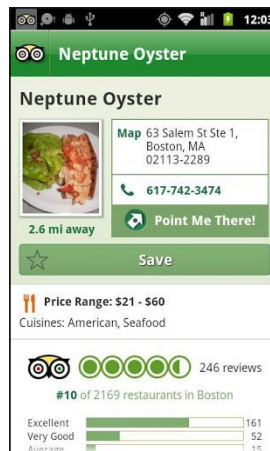


Figure 2-9. Sample location listing in Boston City Guide application.

This listing includes vital information and user reviews from TripAdvisor and allows users to select the thumbnail to view more images of the point of interest.



Figure 2-10. Sample itinerary in Boston City Guide application.

This itinerary shows a list of waypoints as well as the distance from the user’s current location to the beginning of the guided tour.

*New York City – Everything NYC Offers Dashboard for Main Menu, Discusses History of Buildings*

Another location guide publisher, MyCityWay, has created an application called “New York City – Everything NYC,” which presents a dashboard of “mini-apps” that allows users to view various details about the most populous city in the United States, including navigation to landmarks and cultural destinations. According to the listing on Google Play, this application supports check-in and reservation requests from users, as well. In 2010, FOX News touted the application as one of the five best travel applications available on the smartphone marketplace. One of its main drawbacks, however, is its inability to load the user’s current location without an Internet connection [26].

The application presents a structured but complex set of “mini-applications” for users to view after it loads. The application starts by displaying a startup splash screen containing a blurred image of New York City traffic before the main menu loads. This main dashboard screen, shown in Figure 2-12 on the following page, resembles the home screen on an iPhone. This screen presents a plethora of possibilities for users, and reviews of this application show that they appreciate the diversity and currency of information that this product offers [26]. However, the application only partially suggests a possible order for opening these “mini-applications” by placing them on three separate screens.

Users can view specific points of interest by searching the search icon on this main dashboard and entering a search term in the box that appears. The results screen allows the user to view the search results through a particular lens. For example, tourists can visit the Brooklyn Bridge as part of a standard journey to the city’s different attractions, or they can travel to a restaurant containing the phrase “Brooklyn Bridge” as part of a nightlife experience. Once users have selected a point of interest, a location listing appears in a format similar to that shown in Figure 2-12 on the following page. This page shows only basic vital information and relies on links to other applications for users to view more detailed information, such as the phone application to place a call or the browser application to view user experiences on Yelp, Cityscape, or Foursquare.

Finally, users view more detailed information about older locations within New York City. Users start this process by opening the “Tourism” mini-application and selecting the “Historic Tours” option from inside this screen. They then select the neighborhood or borough of interest and view a list of historical places. Each of these places contains a listing similar to the ones that discuss different attractions, but these historical-based descriptions also contain pictures of the location and a brief description of its journey through time. These stories tend to emphasize the building process and materials as well as the lore associated with a given location. My application includes a similar focus on structures and famous stories related to lighthouses within the “History” screen.



Figure 2-12. First screen of main dashboard within New York City – Everything NYC. This “home screen” provides a comprehensive – if overwhelming – perspective on the application’s capabilities.

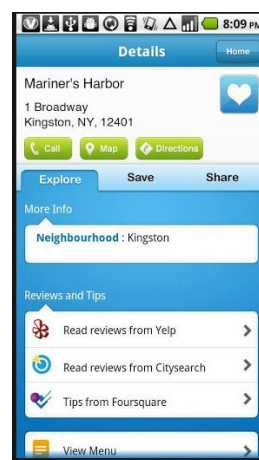


Figure 2-11. Sample location listing in New York City – Everything NYC. Listings of attractions in and around New York City rely on links, rather than direct textual descriptions, to provide users with information about them.

### 2.2.2. History Exploration Present Stories Well at Cost of Information Volume and Access Speed

The Android platform includes several applications for viewing historical information about local landmarks across the United States. Two of these applications – HISTORY Here and America’s National Parks – present historical information in a conveniently succinct, context-sensitive manner, but they each present few options for users to specify the type of landmark they wish to view.

#### *HISTORY Here Presents Good Stories but Lacks Content Volume*

A&E Television Networks Mobile provides an application called HISTORY Here that presents collections of photographs, videos, and interactive maps. It allows users to explore

thousands of historic locations across the United States [21]. The application achieves superb base functionality by presenting context-sensitive historic information effectively, but it offers a limited variety of content that users cannot easily filter.

Each landmark contains a focused, context-sensitive historical “story,” making the application very user-friendly to general audiences. The historical details about a particular landmark remain short – at most a screen and a half in length – and focus on the events that have taken place in a landmark in terms of the country’s general history. Furthermore, the application caters to each location being described, adding a welcome sense of variety for users. After seeing the beneficial effects of these location-driven information screens, I decided to follow a similar approach within my own application, including historical events that related to the lighthouse itself as well as the surrounding region.

While the details about landmarks featured in the HISTORY Here application present a welcome sense of relevance, the total scope of information remains somewhat lacking and difficult to filter. According to reviews on the Google Play Store website from Daniel Thomas and Stephanie Long, the application does not yet contain enough user-generated content. Furthermore, since each set of historical details discusses a landmark in a self-contained manner, the lack of places represented in the application prevents users from appreciating the spatial context of these landmarks. Also, users have the option to view the landmarks near a specified location in a “list view,” but they must view the places within this list in order of increasing distance. The application does not present any other filtering options, limiting users’ ability to investigate a specific type of landmark, such as a lighthouse. In order to give users a better sense of control when using my application, I present a variety of sorting options to users when they see the “search results” for a particular lighthouse query, including distance and name similarity.

*America’s National Parks Offer User Journey Feature after Obnoxiously Long Download Process*

Another history-based application, “America’s National Parks” (presented by LBS Solutions), includes information about hiking trails and recreational activities within each of the 58 national parks in the United States. While the application applies a fairly focused historical lens on each park and offers offline access to some of its features, the layout and functionality of its high-level navigation makes it difficult for users to access historical details and create their own journeys in the first place.



The application's initial screens do not present an ideal user experience and might even discourage people from using it entirely. As soon as the application loads, it displays a list of each park that the application supports, alphabetized by park name. Users cannot filter this list, nor can they sort it using any other method, such as by state. Also, the text within the list remains quite small, especially on my smartphone, where I have the "large text size" option enabled. Once users have selected a specific park to download, the application displays a message warning users that the ensuing download operation required to view information about the park will take a great deal of time. If users agree to continue with the downloading process, they see a progress dialog box for an unacceptably long period of time, and they cannot proceed within the application until it has downloaded an archive file containing the park's information and "imported" (unzipped) its contents within the application. The one saving grace with this process is that users can elect to import information from a local, cached archive file should they interrupt the unzipping process. I made sure that my application avoids such a long, modal operation by completing photograph downloading tasks as background threads.

Once users have survived the park selection and download steps, they can then appreciate the positive aspects of the application, including the focused historical details and well-designed "user journey" feature. When users view a specific park, they can then select the "Information" option from another list and view a set of historical events that have taken place at the park. These details fit within the area of the screen, allowing the user to capture an appreciation for the historical context of a specific national park without needing to scroll or access additional lists. I appreciated the ability to view historical details on a single screen; therefore, my own application presents a given piece of historical information on a single screen, as well. Users can also create their own "journeys" within a park, annotating a set of trails they wish to explore with photographs and comments. Users can even choose to save specific tours to their respective devices so that they can access them later without an Internet connection. This lack of reliance on a phone's network status gives the application a key advantage, especially since cell phone reception within national parks might be suboptimal. Since lighthouses also reside in areas with poor reception, I include as much offline functionality as possible within my application.

### 2.2.3. Collecting Experiences: Offering Personal Travel Logs with Technological Drawbacks

Android users can choose from several applications that enhance their traveling experiences to iconic landmarks – such as Walt Disney World. While these applications offer a

variety of user interaction options, they often contain technological shortcomings, such as unnecessarily stateless behavior and ambiguous icons.

### *My Disney Experience – WDW Offers Personalization with a Stateless Server*

As a method of marketing “the happiest place on Earth” to prospective and current visitors of its theme parks, Disney has created a set of applications that consolidates people’s travels through these places, aptly entitled “My Disney Experience” collectively. This paper focuses on the version of the application featuring Walt Disney World in Lake Buena Vista, Florida. While the application presents a variety of customization options that personalizes user experiences with the application – as well as their trips to Disney World in general – the application’s statelessness reduces the rate at which users can navigate through the different screens featuring elements of the theme park.

This application contains a great deal of personalization features that gives users the pleasing sense that Disney cares deeply about them as individual people. Upon selecting an event taking place at the theme park – such as Illuminations at EPCOT – the application gives users the option to add it to their “plan” for visiting the park, allowing them to accumulate events of interest within a personalized itinerary that they can access and share elsewhere within the application. My application does not support this “wish list” functionality as of yet, but a future version could allow users to form a lighthouse vacation plan more easily.

While Disney’s application affords user-based content, it presents this information behind a stateless server, forcing users to complete some unnecessary steps for accessing the itineraries that they have formed. Upon first opening, the application asks users if they wish to learn about its new features, which takes users to a self-contained screen detailing the latest updates to the application. While some users might enjoy learning about the application’s latest improvements, most would rather view their itineraries first and instead access this information when they wish to view it, not when the application wishes them to view it. Also, because the application does not maintain users’ identities, they must sign in and present their current location over Wi-Fi continually. This constant user information refreshing process limits the speed at which users can access their travel details and makes this process impossible in areas lacking a stable Internet or network connection. In particular, my application caches the photographs for the lighthouse that the user most recently viewed so that they can access them more quickly and in airplane mode as desired.

### *Foursquare Offers Great Social Experience, but Icons Not Easy to Comprehend*

This social travel experience application allows users to “check-in” to locations with their Facebook friends and contacts. Users can also browse recommendations for places to visit from these trusted companions. The application’s popularity and its prestigious “Editors’ Choice” designation indicate its influence within the travel experience sharing community.

Foursquare demonstrates an impressive variety of interactive options, but some of the visual representations of button options do not seem very clear. The “search results” page includes several useful filtering options, including sorting by “best match” or by “distance” and viewing only the places that offer specials or inexpensive services. These filtering options offer users a more personalized experience with the application, and the photograph filters within the Lighthouse Navigator allow users of my application to do the same when viewing photographs of a lighthouse. The information page for a particular landmark includes several sample photographs and a link to a gallery containing dozens or hundreds more. My application also benefits from showing small but coherent pieces of a photograph gallery upon loading information about a lighthouse. The bottom of the screen features some opportunities for users to share their own experiences of a location. In particular, the “Leave A Tip” and “Suggest an Edit” buttons include both an icon and text, giving users two modes of recognizing these options. However, while the presence of icons could help those with good visual memories, this particular pair features graphics that resemble each other too much for most users to retain their design for very long. My application sidesteps this potential issue by including only textual buttons within the different screens that offer users opportunities to share their lighthouse experiences.

#### 2.2.4. *Lighthouse Information Dependent on Internet but Rich in User Interface Aesthetics*

Several applications featuring lighthouses already exist within the Google Play Store. These applications tend to focus on the geographic or historical contexts of the seaside monuments. While these applications feature some aesthetically appealing user interface elements, they overvalue good design over extensive functionality in the process.

### *US Lights Relies Heavily on Internet Connectivity*

This application presents a mobile version of Kraig Anderson’s website, [lighthousefriends.com](http://lighthousefriends.com). As users view information about lighthouses within this application, they can listen to historical information read to them and keep a “personal journal”

of their journeys to different lighthouses, including dates traveled and notes for preserving memorable experiences from these trips.

Anderson's application presents several interesting representations of information but relies heavily on users' ability to connect to the Internet while viewing information about lighthouses. While viewing general information about a specific lighthouse, users can see a variety of icons near the top menu tabs. These icons present a variety of details, such as accessibility by car, appearances in pop culture, and types of technology used within the tower. Even better, users can select an icon to display a pop-up message explaining the meaning of each icon. This clear text-to-icon correspondence allows users to learn the meaning of the icons that they find interesting. The "search results" screen features a more subtle but equally effective visual technique. Each result contains the name of the lighthouse, its location, and a thumbnail of a photograph taken at that location, organized into a clean information collection. My application imitates this interface element, providing users with a geographic and visual context for lighthouses as early in the application experience as possible. The only drawback with the "US Lights" application involves its offline functionality. Without an Internet connection, users can view only one photograph and the history of the lighthouse. Since lighthouses do not always offer reliable phone service, I ensure that my application features substantial offline functionality, most notably the cache of photographs downloaded from Flickr.

#### *Lighthouse Locator Offers Tabbed Browsing, Crowdsourcing Improvements*

MapMuse's version of a virtual lighthouse navigation tool allows users to search for lighthouses on an interactive map or by using a list. Once information about a single lighthouse appears on the screen, users can view photographs and a brief description of each lighthouse.

The application features a simple but effective user interface and few key pieces of auxiliary information that further improve the user experience. The menu tabs ("Map," "List," and "Faves") near the top of the screen offer a simple method for navigating among the content-based screens of the application. My application does not include this tabbed structure only because it features four different content-based screens, and tabs become cumbersome when trying to display more than three options simultaneously. Also, the application presents an option inviting users to "Add a Missing Place" if they notice that a monument does not appear within the application. This crowdsourcing tool allows the application to contain correct, up-to-date information more reliably. While I do not present an interactive feedback screen within my

application, I still include my email address on the “Welcome” screen of my application, which allows users to offer suggestions for improvement as desired.

### 2.3. Key Research Involving Images and Image-Processing Algorithms

The general public tends to use the terms “image processing” and “image editing” interchangeably when working with digital images. However, *image editing* (or *image manipulation*) involves adjusting an image’s features using commercial software, such as Adobe Photoshop and the GNU Image Manipulation Program (GIMP). *Image processing* involves designing and developing the programs that people use to complete tasks related to image editing [41]. Numerous programs exist for *manipulating* images (such as Adobe Photoshop and GIMP) or *processing* images (Khoros/VisiQuest, IDL, MATLAB, and ImageMagick). However, few of these applications allow users to (easily) perform both manipulation and processing tasks together. ImageJ, on the other hand, combines image manipulation with image processing quite well. The software includes a number of built-in functions for manipulating images, and its plug-in architecture allows programmers to implement additional image-processing algorithms [41]. Even better, the ImageJ program contains only Java code, which combines well with the Android development environment. Most standalone image-processing applications, such as the ones listed above, use C or C++ instead, so I explain the benefits of using ImageJ within the Lighthouse Navigator application.

#### Image File Formats Offer Different Levels of Compression; JFIF Best for Web, Mobile

Given the variety of uses for photographs on digital machines, programmers have created several different types of image formats that image processing software can use; the trade-off usually involves storage space and image resolution. The *Portable Network Graphics* (PNG) format supports images containing a depth of 16 bits or less and includes an alpha channel (for transparency) with each pixel, which can also be up to 16 bits wide. It also supports indexed images that contain up to 256 colors. The format even supports lossless compression using a method similar to Phil Katz’s ZIP (PKZIP) [41]. In 1990, the Joint Photographic Experts Group (JPEG) established a standard for compressing images by a factor of 16. This compression ratio has since increased to around 25; as a result, the algorithm that the standard uses can now compress an 8-bit RGB image into one bit per pixel with minimal loss in perceived image quality [41]. An algorithm defined by the *JPEG File Interchange Format* (JFIF) transforms the RGB channels in a color image to the  $YCbCr$  space to separate the colors’ hues from their brightness. It then performs a significantly larger amount of compression on the hue portion of each pixel’s

color data because humans can tolerate drastic changes in hue more easily than sharp contrasts in brightness [41]. While PNG remains the most popular image format on the Web, the large file sizes that accompany the format make it less desirable in low-storage situations, such as internal storage space on smartphones. JFIF images, on the other hand, support compression necessary to save storage space. Therefore, I have chosen to store the images depicting lighthouses in the JFIF/EXIF format within my application.

### 2.3.2. Color Histograms Graph Relative Concentrations of Grayscale Intensities

The *histogram* corresponding to a given image displays the frequency distribution of the image's different levels of intensity. The horizontal axis of these graphs represent different intensity levels within an image, and the vertical axis represents the number of pixels within an image that contain a given intensity level [41]. An example histogram appears in Figure 2-13 below:

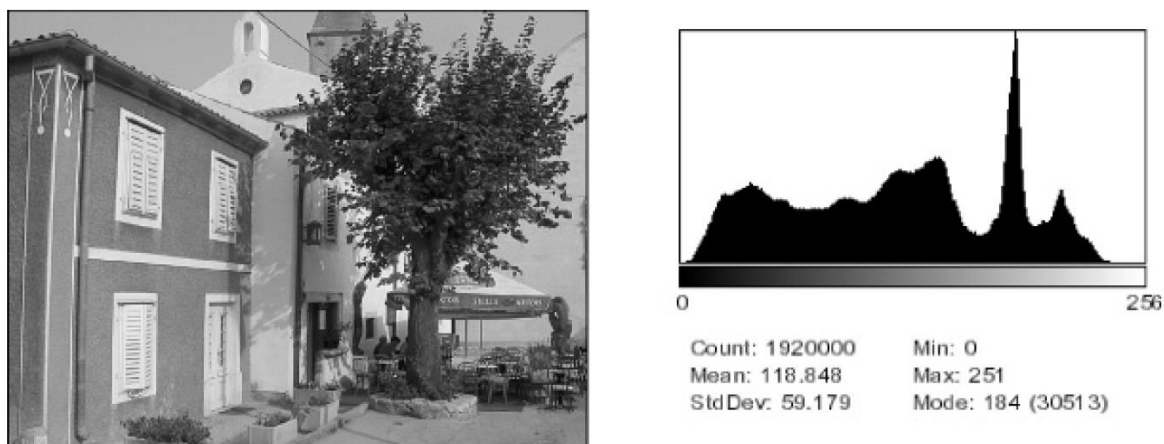


Figure 2-13. Sample grayscale image and its corresponding histogram.

Note that the majority of the photograph's pixels appear in light gray (intensity value of 184), as depicted by the global maximum near the right-hand side of the histogram, which "graphs" intensity values from black (intensity value of 0) to white (intensity value of 256) [41].

The photograph's histogram shows image-processing programmers and algorithms alike that the image contains fairly equal amounts of darker shades of gray and that a significant percentage (about 16%) of the image's pixels contain a specific intensity value of 184 (light gray) [41]. Histograms for color images represent either the corresponding grayscale image using a

weighted average<sup>11</sup> of the color channels' intensity values to determine the corresponding grayscale intensity for each pixel, or they display the intensity distribution for each color channel within the image (such as red, green, and blue). Usually, the latter type of histogram presents more information about possible saturation and exposure issues with the image [41].

### 2.3.3. Cumulative Color Histograms Depict Parts of Image Containing Ranges of Intensity Values

Cumulative histograms represent a sort of integration of the typical histogram. Each column in the cumulative histogram represents the *sum* of the number of pixels that contain an intensity corresponding to that column and the total number of pixels that contain a lower (darker) intensity value than the one corresponding to the column [41]. A concrete example appears in Figure 2-14 below:

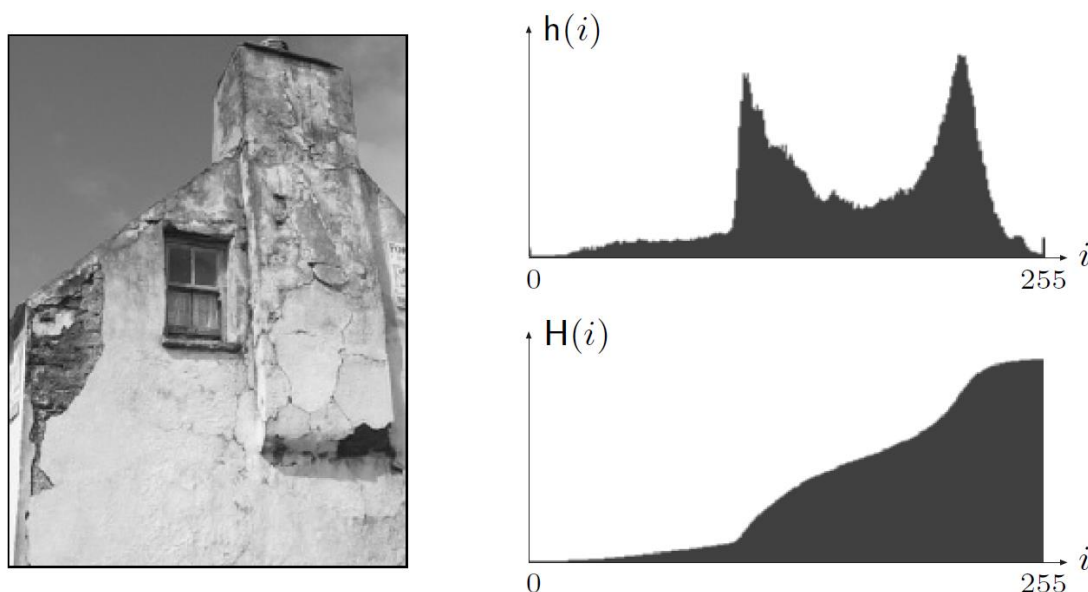


Figure 2-14. Sample grayscale image with its corresponding cumulative histogram. The slope of an image's cumulative histogram graph  $H(i)$  increases significantly at the intensity values containing a local maximum within the original histogram  $h(i)$ . In the image above, these "spikes" in slope occur at intensity values of about 100 and about 200 [41].

<sup>11</sup> For example, the International Telecommunication Union Radiocommunication Sector (ITU-R) presents suggested coefficients of  $R = 0.2125$ ,  $G = 0.7154$ , and  $B = 0.072$  for high-definition television sets and digital images in its Recommendation BT.709 (ITU-R Rec. BT.709). This set of coefficients reflects the relative apparent brightness of each component color (blue, for example, naturally appears much darker than red and green to the human eye) 41. Burger, W. and Burge, M.J. *Digital Image Processing: An Algorithmic Introduction Using Java*. Springer Science+Business Media, LLC, New York, New York, 2008.

As shown in the figure above, the given image contains a significant number of pixels with an intensity value of about 100, and another large set of pixels contain an intensity value of about 200.

#### 2.3.4. Binary Regions: Locating where Foreground and Background Meet

A *binary region* comprises a set of adjacent pixels that each contains the same intensity value within a binary image. To describe an image in terms of binary regions, image-processing software needs to determine the number of regions within the image and the set of pixels that reside within each region. Software can complete this region segmentation process, typically called *region labeling* or *region coloring*, in two primary ways: *flood filling* and *sequential region marking* [41]. I use sequential region marking to identify regions for the color-based feature-matching algorithm that my application uses (see section 4.7.4, “Identifying Presence and Absence of Rocks using Color-Based Feature-Matching”).

##### *Flood Filling Explores Regions Outward*

Flood filling takes a pixel containing the “foreground” color as the starting point and explores neighboring pixels until the algorithm has discovered the entire region that contains the original pixel [41]. Programmers should use iterative versions of this algorithm so that the program can store neighboring pixels that it has not yet explored on a heap stack, rather than the (very limited) call stack [41]. Also, breadth-first variants of the iterative flood-filling algorithm tend to create smaller stacks than their depth-first counterparts because the breadth-first variants tend to find the region’s edges at approximately the same time, minimizing the number of elements allocated to the algorithm’s stack [41].

##### *Sequential Region Marking Processes Each Pixel within Image, Assigns Codes to Regions*

Sequential region marking traverses the image from top-left to bottom-right in row-major order [41]. For 8-connected image regions (regions where two pixels are connected as long as they are adjacent vertically, horizontally, or diagonally), the algorithm assigns a given pixel to the same region as its left, top-left, top, or top-right neighbor. If an edge lies on each of those four neighboring pixels, then the given pixel becomes the first one in a new image region [41].

As the algorithm progresses, “image region collisions” can occur, where two neighbors of a given pixel appear to belong to two different image regions but in fact reside in the same image “super-region.” If the algorithm encounters such a collision, it notes its presence, then arbitrarily selects one of the neighbors’ image regions to assign to the pixel at the “collision point” [41].



During the second pass through the image, the algorithm examines these “collision points” and merges the image regions as necessary [41]

Burger and Burge claim that the breadth-first flood-filling algorithm provides the best option for large, complex images [41]. However, the images that I use for edge detection are not that large and contain few distinct regions, which makes sequential region marking a viable algorithm for me to implement.

### 2.3.5. Cumulative Histogram Matching: The Cornerstone for Classifying Images

Image-processing software can execute a *histogram matching* algorithm to align the intensity histogram of one image more closely with that of a given “reference image.” The algorithm completes this process by examining the cumulative histograms of the given image  $A$  and the reference image  $R$  (represented as  $P_A$  and  $P_R$ , respectively). Section 4.7.3, “Classifying Images based on Stereotypical Sunny, Cloudy Weather Conditions,” discusses how I use this algorithm within my application to categorize images based on sky conditions.

For each intensity value  $a$  in the original image ( $A$ ), the algorithm computes the proportion of pixels in the image that contain an intensity value of  $a$  or less. The algorithm represents this value as  $P_A(a)$  or  $b$ . The algorithm then examines the cumulative histogram of the reference image ( $R$ ) and determines the intensity value  $a'$  that contains a value of approximately  $b$ . The algorithm represents this second value as  $P_R^{-1}(b)$ .

Putting these two equations together, the algorithm can successfully map the intensity each intensity value  $a$  from the cumulative histogram of the original image onto an intensity value  $a'$  from the cumulative histogram of the reference image such that  $P_A(a) = P_R(a')$ . Solving for  $a'$  yields the general histogram matching equation:

$$a' = P_R^{-1}(P_A(a))$$

Figure 2-15 below shows a graphical representation of the algorithm's matching process [41]:

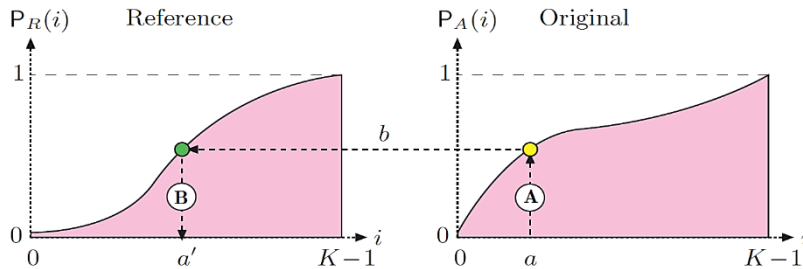


Figure 2-15. The general histogram matching algorithm: Mapping cumulative histogram intensity values. The algorithm takes a given intensity value  $a$  from the original image  $A$  (right) and determines the proportion of pixels within  $A$  that contain an intensity value of  $a$  or less. It then determines the intensity value  $a'$  within the reference image  $R$  (left) that contains that same proportion such that  $P_A(a) = P_R(a')$ . In this manner, the algorithm maps each  $a$  in  $A$  to a single value  $a'$  in  $R$  [41].

While the general histogram matching equation works wonderfully well in theory, most practical applications instead use a variation of this formula when processing two images containing discrete intensity histograms. The algorithm begins by forming the cumulative histogram of each image based on its respective intensity histogram. Since the original intensity histograms contain discrete values, the cumulative histograms remain non-continuous, non-invertible functions.

To compensate for this mathematical inconvenience, the algorithm discretizes the process of mapping the intensity values between the two images. In particular, the algorithm forms “blocks” for each intensity value  $a$  in the original image. Each block’s height represents the percentage of pixels in the original image that contain the intensity value  $a$ . The algorithm then takes each of these “blocks” and attempts to fit it under the cumulative histogram curve corresponding to the reference image. The algorithm “pushes” this block right to left under the curve until the value before the one that would cause the block’s height to exceed the value of  $P_R$  at that point. This process continues, with each block “stacked” on top of the blocks that correspond to lower intensity values of the original image. The resulting mapping of  $P_A(a) \rightarrow P_R(a')$  ensures that, for all intensity values  $a$  in the original image,  $P_A(a) \leq P_R(a')$ .

Figure 2-16 below illustrates this “block-stacking” process [41].

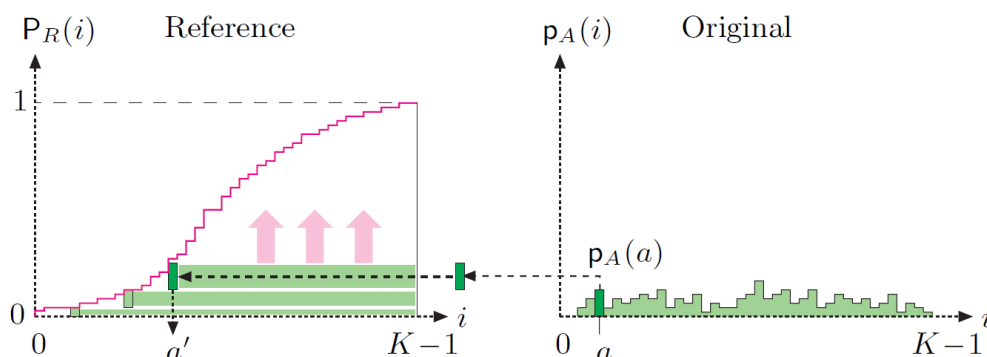


Figure 2-16. Discretization of the general histogram matching algorithm: “stacking” intensity value “blocks” from the original image onto the reference image.

The algorithm takes each intensity value  $a$  in the original image  $A$  (right) and creates a “block” of size  $p_A$ , or the proportion of pixels in the original image that contain an intensity value of exactly  $a$ . The algorithm then “pushes” this “block” from right to left over decreasing intensity values within the reference image  $R$  (left) until it reaches an intensity value  $b$  where the proportion of pixels in the reference image containing an intensity value of  $b$  or less ( $P_R(b)$ ) becomes less than the corresponding proportion for point  $a$  in  $A$  ( $P_A(a)$ ). The algorithm then maps the intensity value  $a$  with  $b + 1 = a'$ , or one more than the “threshold” intensity value for  $P_R$  [41].

### 2.3.6. Template Matching, A Process to Locate Shape-Based Image Features

When attempting to locate a feature of known composition within an image, programmers rely on template-matching algorithms. These programs take a *reference image* containing the desired feature and overlay this image on a larger *search image* of unknown contents. I use this algorithm within my application to perform the shape-based feature matching within an algorithm, as described within section 4.7.5, “Determining Presence or Absence of Lighthouses using Shape-Based Feature-Matching.”

The algorithm aligns the top-left corner of the reference image with that of the search image and compares the contents of the overlapping pixels. It then moves the reference image one pixel to the right, continuing until the right edge of the reference image has “fallen off” the edge of the search image. This process continues for each of the search image’s rows until the bottom edge of the reference image has “fallen off” that of the search image. Finally, the program determines the set of coordinates where the top-left corner of the reference image was located when the algorithm calculated a maximum amount of similarity between the two images [41]. A good method of computing the similarity between a reference image  $R$  and a search image  $I$  at position  $(i, j)$  involves taking the sum of the squares of the 2-dimensional Euclidean distance between each pair of corresponding pixels in  $R(r, s)$  and  $I(r + i, s + j)$ , as shown below:

$$\begin{aligned}
d_E^2 &= \sum_{(i,j) \in R} (I(r+i, s+j) - R(i,j))^2 \\
&= \sum_{(i,j) \in R} I^2(r+i, s+j) - 2 \cdot \sum_{(i,j) \in R} I(r+i, s+j) \cdot R(i,j) \\
&\quad + \sum_{(i,j) \in R} R^2(i,j)
\end{aligned}$$

In the above equations, the limits of each summation ( $(i, j) \in R$ ) represent the set of all points  $(i, j)$  that exist within the reference image  $R$ . The point of maximum similarity then matches the point where the square of the Euclidean distance between the two images reaches a minimum [41].

The middle term of this “base equation” contains a special name in statistics: the *linear cross correlation* between  $I$  and  $R$  [41]. Unfortunately, the linear cross correlation changes drastically as the intensity values within the search image  $I$  change. Therefore, many programs use a *normalized linear cross correlation*, which includes the effects of the intensity values within both the search image and the reference image. This adjustment changes<sup>12</sup> the squared Euclidean distance equation to:

$$d_E^2 = \sum_{(i,j) \in R} I^2(r+i, s+j) - \frac{\sum_{(i,j) \in R} I(r+i, s+j) \cdot R(i,j)}{\sqrt{\sum_{(i,j) \in R} I^2(r+i, s+j)} \cdot \sqrt{\sum_{(i,j) \in R} R^2(i,j)}} + \sum_{(i,j) \in R} R^2(i,j)$$

Note that the  $R^2$  term in the above equation is constant for a given reference image  $R$ . Also, the normalization process turns the  $I^2$  term into a constant, as well. Therefore, an algorithm can use only the middle term within the above equation and still compute an accurate metric of similarity between  $I$  and  $R$  [41].

The middle term of the above equation corrects for existing intensity values within  $I$  and  $R$  but loses its robustness when considering the relative differences in intensity values between  $I$

---

<sup>12</sup> Note that this modified equation also drops the constant factor “2” from the middle term, as it has no effect on the relative Euclidean distances for the different points that the algorithm inspects.

and  $R$ . An equation that introduces the average intensity value of  $I$  and  $R$  into the equation can solve this problem:

$$r = \frac{\sum_{(i,j) \in R} (I(r+i, s+j) - \bar{I}(r,s)) \cdot (R(i,j) - \bar{R})}{\sqrt{\sum_{(i,j) \in R} (I(r+i, s+j) - \bar{I}(r,s))^2} \cdot \sqrt{\sum_{(i,j) \in R} (R(i,j) - \bar{R})^2}}$$

...where...

$$\bar{I}_{r,s} = \frac{1}{|R|} \cdot \sum_{(i,j) \in R} I(r+i, s+j)$$

...and...

$$\bar{R} = \frac{1}{|R|} \cdot \sum_{(i,j) \in R} R(i,j)$$

The above equation is set equal to  $r$  because it in fact calculates the *correlation coefficient* between the two images, which statisticians will recognize. When  $r = 1$ , the two images contain identical intensity values across all pixels, and when  $r = -1$ , the two images do not contain any intensity values in common across their respective pixels [41].

Since the right-hand term in the denominator is in fact the number of pixels in  $R$  times the variance in that image's intensity values and the local average of the search sub-image  $I$  need not be computed until the end, it is possible for programs to calculate the correlation coefficient in  $O(ij)$  time [41]. More advanced algorithms can even assign weights to certain areas within the reference image so that, for example, matches with the center quarter of the image receive twice as much "value" from a correlation perspective [41].

Nevertheless, the correlation coefficient exhibits only suboptimal robustness when comparing a reference image to a search image that contains the desired feature at a different scale or rotation from the original value. *Affine matching* methods use localized statistical

features of images to compensate for any affine transformation differences between the two images [41].

### 2.3.7. Chamfer Feature-Matching, An Advanced Form of Template Matching

Algorithms cannot use direct comparison between a binary reference image and a corresponding search image because the limited color palette (foreground and background) creates discretized distance functions that contain many “false positive” maxima [41]. Instead, they use a function called the *distance transform*, which calculates the distance from a given pixel to the nearest pixel containing the foreground color. (If the pixel itself contains the foreground color, then the distance transform for that location is 0).

The *chamfer* algorithm, which Borgefors explores in his paper, calculates the distance transform (DT) for a given binary image by iterating through the image’s pixels once from top-left to bottom-right in row-major order, then again from bottom-right to top-left in row-major order. I use this algorithm to simplify the process of detecting lighthouses within my application, as discussed in section 4.7.5, “Determining Presence or Absence of Lighthouses using Shape-Based Feature-Matching.” In his paper, Borgefors analyzes several types of distance transform (DT) functions and determines the maximum distance error that each DT accumulates relative to the “ideal” Euclidian distance transform (EDT) function. In particular, he examines the proper numerical approximations to assign for distances between adjacent pixels, as shown in Figure 2-17 below:

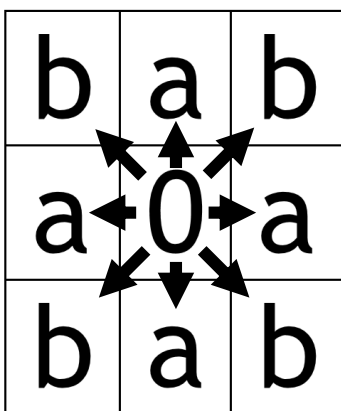


Figure 2-17. Approximation of Distances from a Pixel to Its 8-Connected Neighbors (3×3).

For each pixel in a binary image, it is a distance 0 away from itself, a certain distance  $a$  from its vertical and horizontal neighbors, and a different distance  $b$  from its diagonal

When using  $a = 3$  and  $b = 4$  instead of  $a = 1$  and  $b = \sqrt{2}$  to approximate 8-connected distances in a  $3 \times 3$  neighborhood, the maximum difference between this DT approximation and the “ideal” EDT becomes slightly smaller, especially for pixels fairly far away from the center reference pixel for the DT calculation. Therefore, Borgefors recommends using the chamfer 3-4 DT approximation in  $3 \times 3$  neighborhood situations [40].

Borgefors recommends using the chamfer 5-7-11 DT (with a  $5 \times 5$  neighborhood, as shown in Figure 2-18 below) when the algorithm needs to compute distances that nearly match the “correct” EDT function. He adds, however, that the chamfer 3-4 DT (with a  $3 \times 3$  neighborhood) can provide sufficiently accurate results when the program needs to minimize computational complexity or take imperfect binary image features (noise) into account [40].

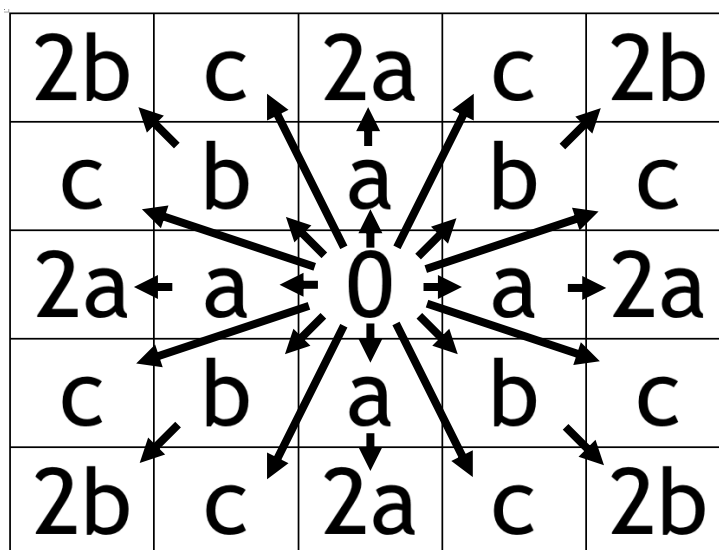


Figure 2-18. Approximation of Distance from a Pixel to Its Nearest 24 Neighbors ( $5 \times 5$ ).

For each pixel in a binary image, it is a distance 0 away from itself, a certain distance  $a$  away from its vertical and horizontal neighbors, a different distance  $b$  from its “short” (adjacent) diagonal neighbors, and a different distance  $c$  from its “long” diagonal neighbors (those that are one “chess knight move” away from the center reference pixel). Borgefors claims that the best integer approximation for  $5 \times 5$  neighbor situations is  $a = 5$ ,  $b = 7$ , and  $c = 11$  (adapted from [40]).

Since I am developing my application for mobile platforms where users expect quick response times, I have decided to sacrifice accuracy for computational speed and use the chamfer 3-4 DT. The features in some of the binary images I construct are somewhat noisy, anyway, which reduces the accuracy benefit that a chamfer 5-7-11 DT would provide.

### *Chamfer Value Propagation: How to Tell Where Nearest Foreground Neighbor Is?*

As the chamfer algorithm visits each pixel (in a 3×3 case), it checks the pixel’s left, top, top-left, and top-right neighbors during the first pass (the same set of neighbors as in the sequential region marking algorithm) and the pixel’s right, bottom, bottom-left, and bottom-right neighbors during the second pass. It then sets the pixel’s distance transform value to the minimum *chamfer value* among the four neighbors. This chamfer value is equal to the sum of the DT value for a neighbor pixel and its distance from the pixel currently being examined (one distance for vertical/horizontal neighbors, and a different distance for diagonal neighbors).<sup>13</sup> For pixels within a search sub-image that align perfectly with foreground pixels in the superimposed reference image, this chamfer value is 0 [41].

After computing the chamfer values for a search image  $I$ , an algorithm can use *chamfer matching* to calculate the similarity between  $I$  and a smaller reference image  $R$ . This process examines the chamfer values for all pixels within  $I$  that “overlap” the pixels in  $R$  containing a foreground color. The algorithm then takes the arithmetic mean of these chamfer values to determine the *chamfer match score*, which the literature traditionally abbreviates as  $Q$ :

$$Q(r, s) = \frac{1}{|FG(R)|} \cdot \sum_{(i,j) \in FG(R)} D(r + i, s + j)$$

...where  $D(r + i, s + j)$  denotes the chamfer value of the search image pixel located at  $(r + I, s + j)$  and  $FG(R)$  is the set of pixels in  $R$  that contain the foreground color. The point  $(r, s)$  within the search image that minimizes  $Q$  represents the location where the search image and reference image exhibit the most similarity.

The chamfer match score exhibits little robustness against affine transformations between  $I$  and  $R$  and remains quite sensitive to foreground “noise” (since the score relates directly to the presence of pixels containing the foreground color at specific locations), but it still works reasonably well for line images that contain relatively sparse foregrounds [41]. To reduce “false positive” results, programmers can instruct the algorithm to take the *root mean square* (rms) of

---

<sup>13</sup> The *chamfer value* can also use Manhattan (city-block) distances (2 for diagonal neighbors instead of  $\sqrt{2}$ ), but this approximation grossly overestimates the distance to foreground pixels on the same diagonal as the pixel currently being processed.



the chamfer values within  $I$  that correspond to the foreground-color pixels in  $R$  (shown below) instead of the arithmetic mean [41].

$$Q_{rms}(r, s) = \sqrt{\frac{1}{|FG(R)|} \cdot \sum_{(i,j) \in FG(R)} D^2(r + i, s + j)}$$

## 2.4. Space and Place: Embedded Comfort Within the Unknown

While many scholars have discussed the interrelationships between “space” and “place,” I have chosen to focus on two people who contribute to the cultural geography discipline: Henri Lefebvre and Yi-Fu Tuan. Lefebvre focuses on the distinction between consumption and production while Tuan emphasizes the difference between unknown apprehension and known safety in his work.

### 2.4.1. Lefebvre Compares Creation and Production, Tuan Focuses on Space and Place

Henri Lefebvre principally argues in his writing that a *creative work* portrays an irreplaceable and unique entity while a *produced object* represents a commodity that society can reproduce after performing a series of repetitive acts. Works and products both occupy a space, he explains to readers, but works fashion a space while products merely circulate within it. Lefebvre also reminds readers that even the most glamorous locations – the ones that have more human-built, creative works than most other places – ultimately serve a product-based purpose. For example, beautiful Venice exists because people have found the port to be a convenient location for trading *produced* goods. He argues that nature creates beings that appear spontaneously, without artificial staging, and that humanity has begun to overrun this unadulterated beauty by producing objects, where labor predominates, and by creating works, where labor is secondary (so that these works are less creative than their natural counterparts). [49]. Every space that humans occupy, therefore, transforms into a series of places that humans find more useful but also less exciting to experience.

Lefebvre continues by explaining that this *social space* forms out of past actions and permits or prohibits certain future events from taking place in that space, including both the production *and* the consumption of products. He further interprets this idea by stating that society cannot easily take the codes in a space and aggregate them into a “meta-code” because these codes are each contingent on the social practices between them and their respective inhabitants [49]. Humans might interpret these codes as indicators of separate identities among

societies, but in reality they are all related but incompatible *reproductions* of the same society. Similarly, people might consider each lighthouse as a separate entity within a particular section of the coastline it protects, but people can still reproduce these monuments. Nevertheless, each lighthouse still maintains its own aesthetic identity, allowing visitors to appreciate the space surrounding each monument as a creation that they can help shape.

In his works, Yi-Fu Tuan explains that a place adds a sense of identity to a space. Humans view open spaces as open, threatening, and representative of movement while they view contained places as closed, safe, and representative of pause [61]. People naturally gravitate toward comfort and familiarity when in a location, which they associate with closed, secure, fixed locations in an otherwise unfamiliar environment. For example, mariners look for a beam of safety and guidance from a lighthouse – a particular place – when surrounded by a space of dangerous coastal waters. When examining a scene, Tuan argues, people gaze at points of interest – *places* – within that scene or *space*, pausing for short amounts of time as they process each place within the space. People subconsciously interpret a location using this process because nature's stimuli are too overwhelming for humans to comprehend without placing these phenomena in an understandable context that human structures provide. Also, people cannot become emotionally attached to an abstract space the way they can to a concrete object or place [61]. After all, it is easier to imagine a physical object or location than a fluid, constantly changing idea.

Tuan warns architects, however, that a place must convey a general, humanistic message *and* a specific, historical one to transcend culture and avoid degenerating into cluttered space. Without the right guidance or maintenance, a place once esteemed for its permanence can become a source of irritation or even sadness within a society. One method of preserving a place's integrity is to represent its inhabitants' aspirations immediately [61]. Coastal residents continue to take pride in their local lighthouses because they have always protected their neighbors, many of whom once made a living in the unpredictable ocean. Another method of perpetuating a place's sense of lore is to associate its construction with a prominent movement. Tuan explains that nationalist movements in particular encouraged people to create patriotic works that had a collective identity to represent a particular nation-state [61]. The first set of lighthouses in the United States served to protect mariners, but these structures also represented the collective success of cities and ports along the new nation's coast.

#### 2.4.2. Emerging Prominence of Theories

When people occupy a space, they create personal landmarks to communicate their thoughts and ideals to other societies; if outside cultures respect the meaning of these places, they will not misinterpret a society's motives. Rodríguez de Castro (2012) agrees that Tuan has introduced moral, affect, and culture into studies of place [55]. This understanding helps people discover how best to protect and preserve monuments within a society's space for future generations to enjoy.

Once considered scholars for a niche discipline, Lefebvre and Tuan have both become more popular in recent decades because of growing concerns related to globalization and human activist movements. Schmid (2008) explains in his commentary on Lefebvre's ideas that more people reference the scholar's main work, *The Production of Space*, because "new space-time configurations" have developed during a period of unprecedented urbanization and globalization [56]. These societies, which rely more heavily on produced works, may disregard nature to a dangerous extent as they continue developing.

Tuan's work, on the other hand, appears in a movement called "humanistic geography," or the attempt to understand how humans shape the world in which they live. Entrikin and Tepple (2006) explain in their commentary that humans, as geographical agents, have moral and cultural goals along with their better-understood economic goals. They also remark that, since the humanistic geography movement took place simultaneously with the rise of neo-Marxism and increased social activism, many people over-classify humanistic geography as atheoretical and apolitical [44]. In fact, humans lay cultural claims to each space they conquer, building concepts of place within that space to convey their collective mindset symbolically.

By including references to signs and plaques within the "Photographs" screen of my application as well as efforts to preserve and protect a lighthouse within the "History" screen, I convey to users that these lighthouses represent places of real historical significance and that they should respect lighthouse societies' efforts to maintain these landmarks for decades and centuries to come.

### 3. Designing a Compelling Lighthouse Portal Using Audience Feedback

The most successful applications require a strong design first. This section discusses the design paradigms from the Android platform that I integrate into my application, along with several tenets regarding agile design and audience-based, iterative improvement of user interfaces. I then present the design of the app in its entirety before delving into the specific considerations and processes that I employed, such as iterative design, audience involvement, and ethical considerations.

#### 3.1. Android Design Guidelines Stress Simplicity, Usability

The “Design” section of the Android Developers website describes the best practices involved with creating a successful application for the platform, placing particularly strong emphasis on captivating users by appealing to their storied connections with different places and incorporating interaction standards from the operating system itself. According to the “Creative Vision” page on the Android Developers website, Android applications should provide pleasing layouts and transitions, include as simple a set of interaction patterns as possible, and encourage users to integrate the application into their daily routines and context-sensitive experiences [12]. Beyond this simplicity and ease of use, applications should evolve with the Android operating system, incorporating new user interface elements and presenting the correct type of feedback after users perform certain interaction gestures. For example, the “New in Android” page on the Android Developers website explains that, starting in Android 4.0, a long-tap opens a contextual action bar within most applications, rather than a contextual pop-up menu [25]. My application includes simple layouts, such as a grid-like table for displaying search results and an interactive timeline for presenting historical details for a specific lighthouse. In addition, it includes an action bar with a drop-down menu for selecting content screens, supporting users’ knowledge of this user interface element from Android 4.0.

##### 3.1.1. Simple Layouts Lead to Captivating Screens

Both the “Design” section of the Android Developers website and several Android design practitioners stress the importance of making each area of a screen’s user interface easy to understand, allowing a particular screen in an application to seem user-friendly when viewed from a high-level perspective. Greg Nudelman explains in his book that it is better to include multiple containers on a single screen than to make one container too long, particularly on screens containing a large amount of text [54]. The “UI Overview” page on the Android

Developers website expands upon this idea, encouraging developers to use a secondary action bar at the bottom of the screen to present additional options within a particular screen [33]. This division between the frequently-used and occasionally-used screen options prevents users from feeling overwhelmed with a screen immediately after it loads. While it's important to follow Android's design guidelines whenever possible, some developers have found that deviating from the platform's best practices actually *increases* the usability of certain applications. For example, Jason Mark explains in his blog entry "Four Android App Design Guidelines You Should Break" that action bar options should contain text, not just icons. These icons often present more confusion, especially to users who have not used the application before or for an extended period of time [51]. My application divides the frequently-used content screens, such as "Information" and "Photographs," into multiple sections so that users can better appreciate how these elements work together, as shown by the boxes drawn in Figure 3-1 on the following page.

Beyond presenting several elements on each screen, Android applications should also make these elements interactive. The "Design Principles" page on the Android Developers website recommends allowing users to interact directly with the elements on a particular screen, such as pictures and maps [14]. In addition, Nudelman assures developers that experienced Android users understand and apply the "act locally, think globally" mindset. They realize that action bar options and a screen's title relate to the information on that screen alone, and they know that they need to press the back menu button in the top-left corner of the screen to view another section of the application [54]. Several elements within my application respond directly to users' feedback, giving them a sense of accomplishment while using the software. These elements include the phone number and website on the "Information" screen and the photographs on the right-hand side of the "Photographs" screen. I have also incorporated the screen-dependent action bar options and a "back button" that returns users to the "Welcome" screen of the application, allowing them to "act locally" and "think globally."

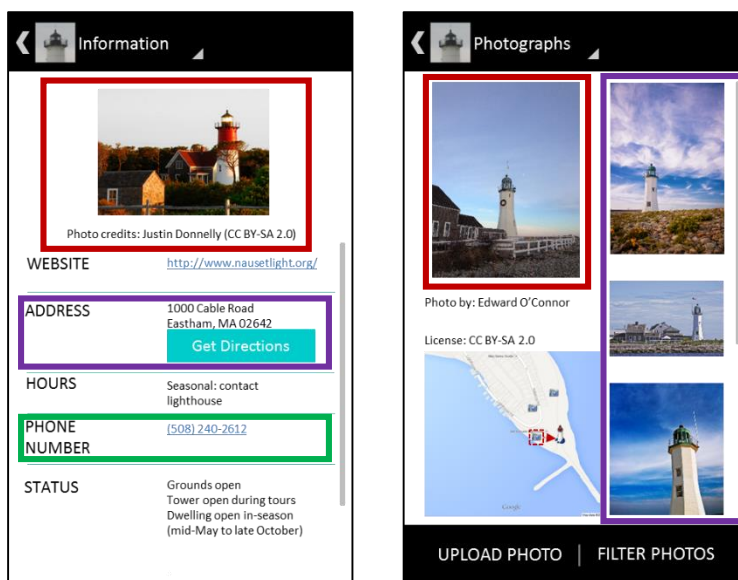


Figure 3-1. Multiple Elements within “Information” and “Photographs” Screens within Application.

The colored boxes each indicate separate interaction areas within the “Information” screen (left) and “Photographs” screen (right).

This chunking of content within my application prevents users from experiencing information overload and affords them a better sense of control and freedom within the software’s different components.

### 3.1.2. Keep Screens Simple, but How?

In designing the Lighthouse Navigator, I adhered to the best practices concerning simplicity and accessibility. Indeed, Android developers and design practitioners agree that Android applications should remain as simple as possible so that users can still interact with the software while distracted or in an unfamiliar environment. In particular, Nudelman advocates catering to users’ scattered minds by creating a space of interaction with the “real world” within an application [54]. The “Design Principles” page on the Android Developers website expands upon this idea, insisting that the application perform complex operations after users complete a simple interaction with a screen [14]. This feedback allows users to feel like they accomplish meaningful tasks quickly, providing an increased sense of satisfaction while using an application. For my application, I present an option for users to “search nearby lighthouses” and have the application figure out in real-time the set of lighthouses that reside closest to their respective current locations. I also include a variety of photographs of a particular lighthouse so that users can develop a sense of place before visiting the lighthouse in person. Both of these examples of user interaction appear in Figure 3-2 on the following page.

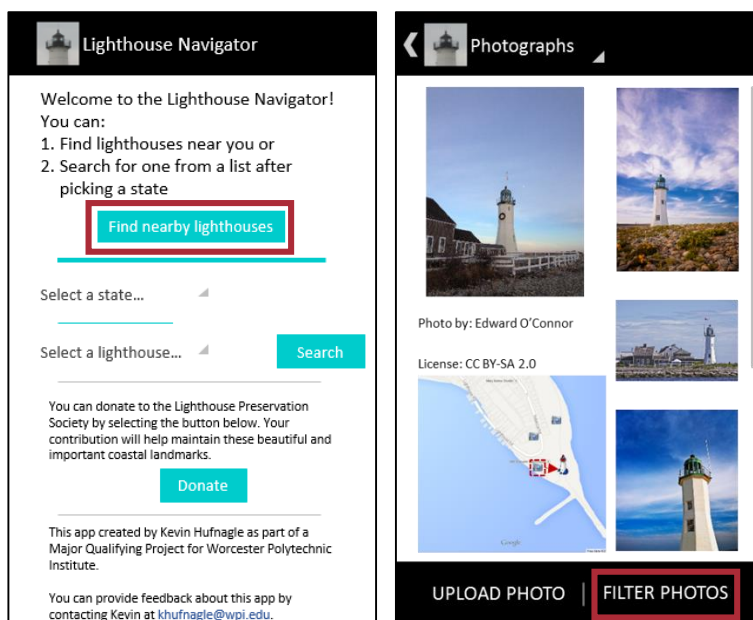


Figure 3-2. Areas of Screens Affording Real-Time User Interaction. The “Find nearby lighthouses” button in the “Welcome” screen of the application (left) and the “Filter Photos” button in the “Photographs” screen (right) both allow users to accomplish complex tasks with a single user-interface interaction, giving them a satisfying sense of power as they navigate through the application’s screens.

The application automatically determines the lighthouses that appear near the lighthouse, allowing users to focus on exploring the details of a specific lighthouse that interests them.

Sometimes, however, practitioners recommend design patterns that differ from Android’s best practices, leaving developers with a decision to make. For example, the “Application Structure” page of the Android Developers website recommends that developers incorporate exactly one navigation tool for viewing the different content screens<sup>14</sup> in an application: a set of tabs, a spinner, or a navigation drawer [6]. Mark, however, recommends using the tabs or spinner since these tools keep the user focused on the currently-open content screen [51]. Also, the “Navigating with Back and Up” page of the Android Developers website states that the “back button” should return users to the main screen of an application if several screens can bring users to a specific content screen, but Mark recommends against forcing users to “start over” within the application [24, 51]. For my application, I have chosen to use a spinner for navigating among the content screens since a group of four tabs would be cramped in a portrait orientation. Since

<sup>14</sup> For my application, these “content screens” include the “Information” screen, the “Photographs” screen, the “History” screen, and the “Reviews” screen. Unlike other screens in the application, each of these screens displays detailed information about a specific lighthouse.

users can access a given content screen from various other screens, I elected to stick with Android's simple solution of returning users to the "Welcome" screen upon selecting the "back button" in the top-left corner of a content screen.

### 3.2. Design Paradigms

In addition to following the guidelines that the Android platform offers, I incorporated tenets from the Agile development process into my design workflow and implemented Robert Johnson's idea of "audience-involved" communication to iteratively improve the design.

#### 3.2.1. Agile Development

Since I developed this application on a quite limited timescale, I used the Agile development method to add features to the product. This incremental, iterative approach allowed me to demonstrate a working version of my application at all times and integrate new features as soon as I developed them.

Developers have used the Agile method for over a decade, so it now has a fairly stable set of tenets. Kent Beck and others stress simplicity in this type of development process "[by] maximizing the amount of [unnecessary] work not done" and advocate frequent reflection to take appropriate action based on any shortcomings in the development process [38]. These mindsets lead to a steadier development pace and an increased ability to change the design of a piece of software, even late in the long-term development process. Tathagat Varma builds upon these ideas in his presentation entitled "The Joys of Designing Agile Solutions for New-Age Problems," stressing the need to consider relevant customers and to develop software that solves a specific, single problem that those customers have experienced [62]. Using this development mindset, Varma argues, the unit of iteration is the *validation of an idea through metrics* (emphasis added) [62]. Developers need to design and implement an idea of how an application might solve a problem for a set of customers, then test that idea using unit tests or, better yet, interactive customer feedback sessions. The deliverable for each of these iterations is a *minimum viable product* (MVP), which contains only the most essential features of the software necessary for obtaining feedback from prospective end-users. Once developers collect feedback from customers, they revise their MVP and re-release it for use in a future feedback session [62]. While creating my application, I presented designs, MVPs, then more robust MVPs to visitors of different lighthouses across New England to ensure that the application solved their information access problems.



In addition to developing my application using an iterative, customer-driven approach, I incorporated several other tenets of the Agile mindset in my project. I used Git version control for both the application itself and this report to document the history of my progress and to create backups so that I could access code or report details that I removed within a newer version. This version control system also served to avert disaster should my system or Google's servers fail at any point during the project. In addition, I created several versions of an Agile board, including a physical poster-board representation, to understand the goals and relative time commitments associated with each of my project tasks for a given week. Finally, I held regular meetings with both of my advisors, usually on a weekly basis, to serve as feedback sessions for the project as a whole. After all, my advisors served as key stakeholders for this project and deserved to offer suggestions regarding the focus of my research and implementation processes.

### 3.2.2. Audience-Involved: Designing with Iterative User Testing Throughout the Design Process

From the beginning of the application design process, I remained committed to having prospective users participate actively in the design's improvement. Robert Johnson, in his academic journal article "Audience Involved: Toward a Participatory Model of Writing," encourages a participatory, community model of communicating with audiences, specifically through writing. Rather than invoking *audience-addressed* communication – where the text changes audience members' thoughts as they consume it – or *audience-invoked* communication – where the audience is a fictional construct used in the writing processes – authors can incorporate *audience-involved* writing by allowing the recipients of their work to participate in the writing process [48]. In his article, Johnson focuses on how technical writers can complete an audience-involved documentation process to recognize a technology's users as part of the text. Figure 3-3 on the following page shows the relationships that form when writers use such a process:

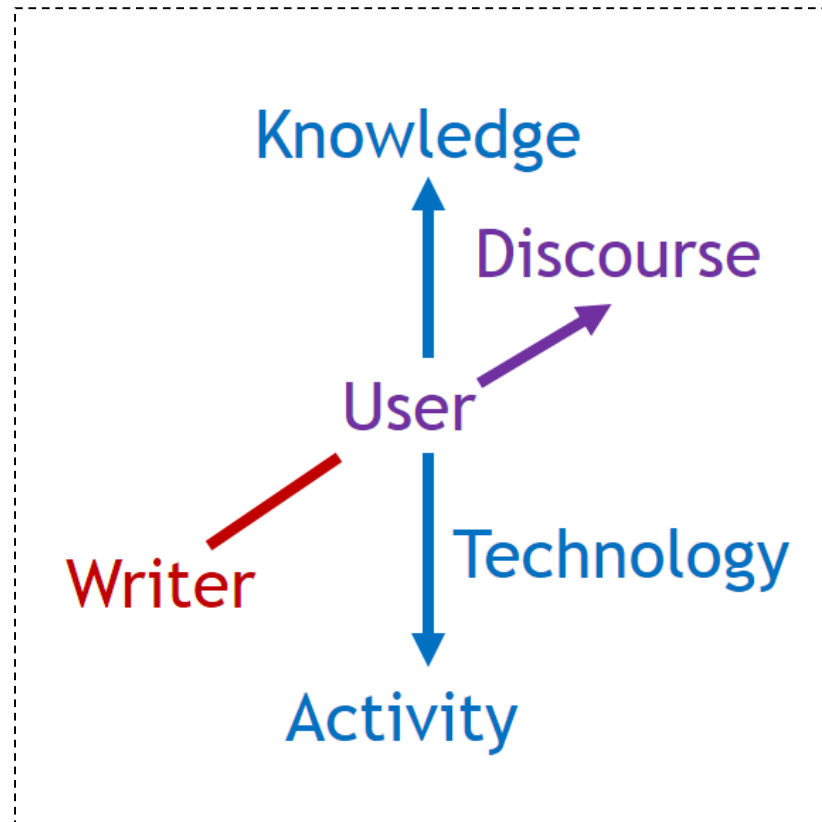


Figure 3-3. Relationships among Artifacts and Ideas while Using Audience-Involved Communication. By incorporating Robert Johnson's ideals within an *audience-involved* writing process, the writer works with the intended audience to develop a discourse. The audience can then consume this discourse more easily to acquire the knowledge necessary to use a piece of technology as a means for completing a life activity.

As the above figure shows, discourse becomes both the *result* of writers working with users and the *tool* with which users interact to acquire knowledge about technology when technical writers take an audience-involved approach to documentation. Users can then apply this newfound knowledge to the technology itself, creating an opportunity for them to complete activities with this technology more easily. Johnson warns readers that by not including users in creating or explaining technological innovations, they can enable technological determinism, in which users robotically generate knowledge with the technology rather than actively influence the knowledge that the technology can produce [48]. If users have no control over the ideas that technology can enable, it becomes increasingly difficult for technical writers and developers alike to convince users that this technology can benefit them.

Nudelman and Varma both apply Johnson's principles to application development. Nudelman advocates testing early and often on a viable audience in an appropriate context to design an application correctly using a problem-solving approach [54]. The audience can also

impart new knowledge about the application that the developer can incorporate in future versions. This mindset mirrors that of Varma and his MVP-feedback iterations discussed in the previous section. When developers and users work to improve a software product together, it becomes easier for this software to produce meaningful knowledge and serve users' needs.

### 3.3. Guiding Users through the Application's Different Screens

In this section, I analyze each of the different screens within the application from an end-user's point of view. I discuss the different components on each screen, my rationale for selecting different ways of presenting these components, and descriptions of the application's reaction to each type of user input.

#### 3.3.1. Screen Navigation Hierarchy

While this application does not contain that many screens, there is some complexity in how users can navigate among them. Figure 3-4 below shows the navigation hierarchy among the application's different screens. Users can access any other content screen by using the drop-down spinner in the action bar, or they can return to the "Welcome & Search" screen by selecting the "up" button in the top-left corner of the screen.

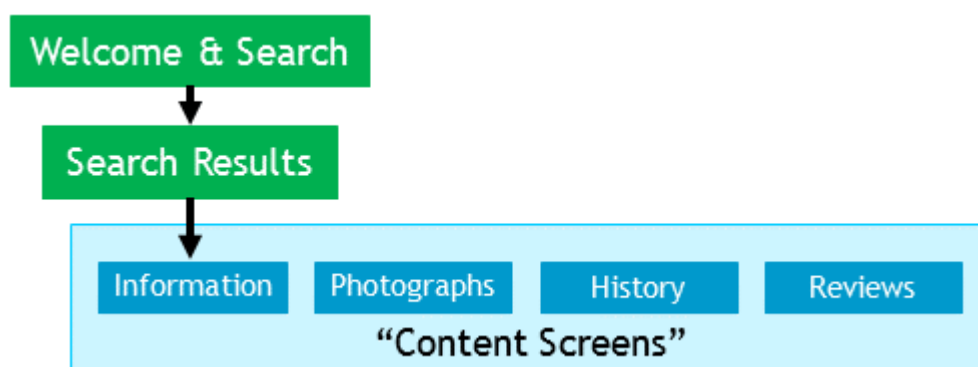


Figure 3-4. Relationship among Screens within Application.

Application users select a method they wish to use for searching, complete a search for a given lighthouse, and select a result to view detailed information about a lighthouse. This information appears across four separate screens (known as "content screens").

#### 3.3.2. Welcome & Search Offers Multiple Guided Search Options

The final design of the "Welcome & Search" screen within the application appears in Figure 3-5 on the following page and features two user-guided options for finding lighthouses that could interest them. This screen appears after the application finishes loading data about the lighthouses that the application supports and serves as the "main menu" screen for the application.

In order to acquaint users to the application as quickly as possible, I show the application's main functionality in the top half of this screen. I present a brief welcome message, followed by the two main search activities that they can complete on this screen. My professional writing advisor recommended that I include the ability to search for nearby lighthouses, and the Android application "US Lighthouses" features a state-based search, so I include both types of searching within my application. After speaking with lighthouse visitors, it seems like they would complete a search for nearby lighthouses more often, so I present that search option first. When users tap the "Find nearby lighthouses" button, this screen passes their respective current locations to the "Search Results" activity, and when they tap the "Search" button near the center of the screen, this screen passes the name of the lighthouse that the user has selected in the drop-down list to the same "Search Results" activity.

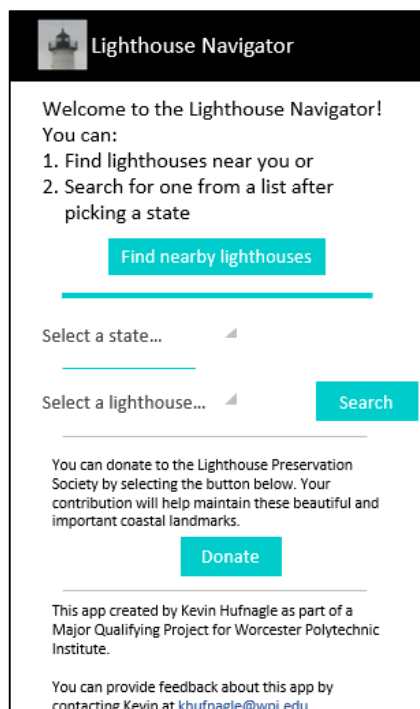


Figure 3-5. Final Design of "Welcome" Screen within Application.

As a way of thanking the Lighthouse Preservation Society for providing me with key information and resources related to New England lighthouses during the research phase of my project, I provide a button that allows users to donate to the organization on this screen. I include a brief description of the cause above this button to help users understand the purpose and

significance of their generous gesture. When users tap this button, the application redirects to a website that accepts donations for the Lighthouse Preservation Society.

Finally, I present the academic context of this application and invite users to provide me feedback about the application via email. The US Lighthouses application accepts users' comments and suggestions, so I thought that it would be appropriate to do so in my application, as well. When users select my email address, the application launches the "email" application that they have installed on their respective mobile devices.

### 3.3.3. Search Results Shows Lighthouse with Visual and Geographic Context

The "Search Results" screen displays each lighthouse's name, location, and picture together to help jog users' memory in a variety of ways and to help them associate each lighthouse's appearance with its name. The final design of the screen within the application appears in Figure 3-6 below. This screen appears after users choose to search for nearby lighthouses or after they select a state and the name of a lighthouse near the center of the "Welcome & Search" screen, then select the "Search" button.

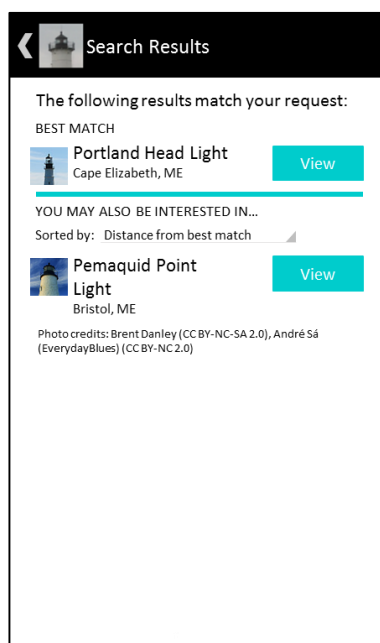


Figure 3-6. Final Design of "Search Results" Screen within Application

The name of the lighthouse that appears under the “Best match” heading matches the name of the lighthouse that users have selected on the “Welcome & Search” screen or the name of the lighthouses closest to the user’s current location. In this case, users have indicated that they wish to view more information about a lighthouse in Maine, namely Portland Head Light.

I present a left-facing arrow next to the application icon in the top-left corner. Users who tap this arrow can return to the “Welcome & Search” screen to search for a different lighthouse. This button allows users to recover if they have accidentally searched for a lighthouse other than the one that currently interests them.

The two major design components on this screen include the “best match” and a list of “good matches,” which I have adapted from an application for iOS devices (including the iPhone, iPad, and iPod Touch) called “Lighthouse Locator.” This iOS application displays the results for a lighthouse search on a map, indicating the total number of results that appear on the map. This ability to show multiple search results led me to think of several methods for displaying “good” matches for users’ lighthouse search queries that could appear under the “You may also be interested in...” heading. I decided originally to show three types of “good” search results: lighthouses near users’ current locations, distance from the lighthouse listed under “best match,” and similarity in name to the lighthouse listed under “best match.” I have kept only the latter two options in this final design, however, after expanding the “Welcome” screen to include nearby lighthouse search functionality, as discussed in the previous sub-section.

In both the “best match” and each “good match” on this screen, I include the name and location of a given lighthouse along with a small photograph of the landmark. Each search result in the “US Lighthouses” application includes these three elements, and the juxtaposition of this information helps users associate these aspects about a particular lighthouse, which they can recall during a trip to this lighthouse.

Finally, I include the credits for each lighthouse image that appears on this screen. I did not take any of these photographs, so I need to provide appropriate credit for these images. Also, I include only photographs whose licenses allow me to modify and adapt them, as I needed to crop details out of these images to obtain the correct aspect ratio and display only the upper half of each lighthouse tower.

### 3.3.4. Information Presents General Facts as Independent Units

The “Information” screen presents a series of clearly delineated facts, with links included as needed to simplify the information-gathering process even further. Figure 3-7 below shows the final design of the “Information” screen within the application. This screen appears after users select the “View” button next to a search result on the “Search Results” screen. The details that appear on this screen refer to the lighthouse that the user has chosen to view. In this case, users have chosen to view more information about Nauset Light in Eastham, Massachusetts. Users can also return to this screen from one of the other content screens by selecting the title of the currently opened content screen and choosing the “Information” option within the drop-down list. In this case, the details on the “Information” screen refer to the same lighthouse as the one featured on the previous content screen.

As in the search result screens, the “Information” screen includes an arrow in the top-left corner. When users select this arrow, they return to the “Welcome & Search” screen. While it could be convenient for users to return to the “Search Results” screen, the direct link to the “Welcome & Search” screen makes it easier for users to begin a new search for a different lighthouse. This simplicity in the application’s navigation also allows users to become familiar with the application more quickly, as they complete an entire search process each time they examine details for a particular lighthouse.

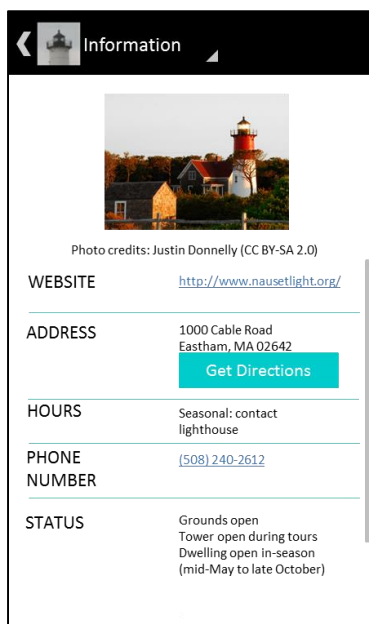


Figure 3-7. Final Design of “Information” Screen within Application

The top navigation area of the screen (known as the *action bar* on Android smartphones and tablets) features a heading for the screen, “Information,” that users can select in order to switch content screens. The arrow beneath and to the right of the screen heading provides a clue to users that they can tap on the screen name itself to view more information. When users select the “Information” heading, a drop-down list appears beneath the heading, showing the other content screens available in the application. Users can then choose which content screen they wish to open within the application by tapping on the name of the content screen – either “Photographs,” “History,” or “Reviews.” I use this drop-down list as the primary mechanism for switching among the application’s content screens instead of two other viable methods for showing the presence of other screens: (1) tabs beneath the action bar and (2) options that appear when users press the device’s menu button. The “Tabs” page of the Android Developers website recommends against using tabs for listing more than three content screens; since my application features four such screens, I cannot use this user interface element very effectively [32]. I also choose against displaying the other content screens on the bottom of the “Information” screen after the user presses the “menu” button for three reasons. By using a bottom context menu:

- The “Information” screen gives no visual cue that users can access other content screens;
- The options for viewing other content screens appear at the bottom of the screen, as far away from the name of this content screen as possible; and
- Most Android applications reserve the menu button for application-wide screens – such as help and settings – not for switching among individual screens.

The drop-down list method avoids these inconveniences and cognitive dissonances for users, making it easier for them to remember how to switch among the application’s content screens.

Beneath the action bar, I present a photograph of the lighthouse, along with its credits and license. This visual representation of the lighthouse reassures users that they have chosen the correct lighthouse to examine, and it presents a snapshot of the scenery surrounding the lighthouse so that users can identify it more easily when traveling to it.



The remaining elements of the screen each present a general fact about the lighthouse. These details include:

- The website that users can access to learn more about the lighthouse;
- The address of the lighthouse for land navigation purposes;
- The hours during which the lighthouse (or associated museum) is open;
- A phone number users can call to speak to a person who helps preserve the lighthouse;
- Whether visitors can access the lighthouse grounds, tower, and/or dwelling;
- The year when the lighthouse first opened;
- The height of the lighthouse's main tower; and
- The characteristic light pattern of the lighthouse, which appears at night or in foggy conditions.

I present this information in a table, with an information category in the left-hand column and the corresponding fact in the right-hand column. I separate each row of this table with a blue horizontal line in order for users to understand the pieces of information that belong to a particular category, especially for facts that contain multiple lines of text. I display a scroll bar to the right of these pieces of information to show users that some information does not appear on the screen at first and that they must scroll down to see some of the last few facts about a specific lighthouse.

I add special properties to some of the facts that appear on the “Information” screen. The “address” data contains a button named “Get Directions,” which users can select to open a maps application on their devices and navigate to that address from their respective current locations. The “US Lighthouses” application includes a “Directions” option on the action bar of its lighthouse content screen with similar functionality. The “phone number” listed on the screen appears as a hyperlink; when users tap on it, the phone application opens with this phone number pre-dialed. The user simply needs to press the “send” button on the phone keypad to place a call to the lighthouse. Finally, not all lighthouses have consistent hours, and some do not provide hours at all. In that case, I include the text “Seasonal – contact lighthouse” in the “hours” fact. This piece of information prompts users to call the lighthouse or visit its website to learn more about when they can visit.

### 3.3.5. Photographs Features an Interactive Gallery

The “Photographs” screen features both an interactive map and a gallery where users can view each of the images associated with a particular lighthouse, allowing them to appreciate the geographic and visual contexts of the lighthouse once again. The final design of the “Photographs” screen within the application appears in Figure 3-8 on the following page. Users can navigate to this screen from another content screen by selecting the name of that content screen, then by choosing the “Photographs” option in the drop-down list. The images that appear on the screen depend on the lighthouse that users have chosen to view. In this example, users have opted to view more information about Old Scituate Light in Scituate, Massachusetts.

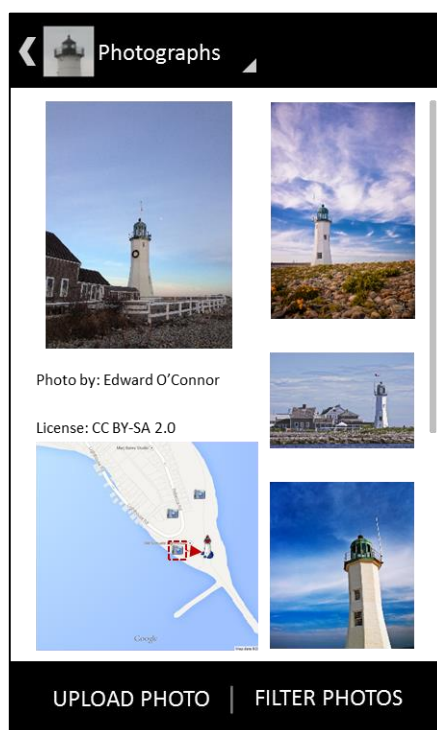


Figure 3-8. Final Design of “Photographs” Screen within Application.

As in the “Information” screen, the “Photographs” screen includes a left-facing arrow in the top-left corner and the screen’s name in the action bar. By selecting the arrow, users can return to the “Welcome & Search” screen within the application, and by selecting the name of the content screen, users see a drop-down list featuring the names of the other content screens. They can then choose the content screen they wish to view next by selecting it from this list.

The left-hand side of the main area on this screen features one of the images associated with the lighthouse that users have chosen to view in more detail. Some of the photographs that I

include in the application have unusual aspect ratios that do not match the one in this section of the screen. In these cases, I shrink the entire image so that its longest dimension fills the area of the screen and its shorter dimension contains extra padding. For example, the “main image” in the figure below contains extra padding along its horizontal axis. I employ this “fit to section of the screen” algorithm to avoid cropping the edges along the image’s longer dimension.<sup>15</sup> When users select an image along the right-hand side of the screen, it appears in the top-left corner of the screen in place of the image that resided there before. Beneath the image that appears in the top-left area of the screen, I include the photograph’s credits and licensing guidelines. These pieces of information update when users select another image along the right-hand side of the screen. Therefore, these credits always refer to the photograph that currently appears in the top-left area of the screen.

In the bottom-left area of the screen, beneath the “main image” and its associated credits, I include an interactive map that shows the location of the lighthouse as well as the relative locations of the photographs associated with this lighthouse. Users can select this map to open a full-screen version within a “maps” application installed on their respective devices. I include a lighthouse icon on the map that corresponds with the location of the lighthouse. I also show the approximate location of the image that currently appears in the top-left corner of the screen by adding a dashed red box around the icon on the map whose location most closely matches that of the “main image.” The “US Lighthouses” application includes a similar map for displaying the relative locations of lighthouses on a “Search Results” screen. By displaying collections of photographs in my application using such a map, I show users the places around the lighthouse that likely offer particularly scenic views, allowing them to enjoy their visit even more. I also include the direction that the photograph faces on this map. This element of the map mirrors a feature in the “Photo Tourism” research application by Snavely *et al.* (2006), which indicates the direction that that an image faces in addition to its location [58]. I indicate this direction with a red arrow pointing from the camera icon that represents the approximate location of the “main image” currently shown in the top-left corner of the screen. This map does not display each photograph’s relative location; each camera icon on the map represents a group of photographs taken at similar locations. I organize the images using this method because users can select these

---

<sup>15</sup> In fact, some photographs that I include in the application prohibit me from altering their contents in this way. I cannot crop any image with a license that includes “no derivatives,” for instance.

camera icons to activate a location-based image filter, which future versions of the application could support. By adding extra negative space among the icons shown on the map, users can select one of the icons without tapping a different icon by mistake.

The right-hand side of the screen shows a gallery of images that photographers have taken of the lighthouse currently being showcased. When users have activated a filter by tapping a camera icon or by selecting one from a list that appears after selecting the “Filter Photos” button, only the images that satisfy this filter appear in this area of the screen. In cases where this area of the screen cannot show each picture in the gallery at once – which happens most of the time when users interact with this screen – a scrollbar appears to inform users that they can scroll down to view additional photographs. I present a gallery in the main “Photographs” screen so that users can view more than one photograph of the lighthouse at once. This combination of visual elements gives users a better sense of the space and scenery surrounding the lighthouse, allowing this space to feel more like a place when they visit the lighthouse in person, even for the first time.

The bottom of the screen features two buttons that allow users to add a photograph to the gallery of images appearing along the right-hand side of the screen or to view subsets of this gallery. I originally presented these buttons as icons in the top-right corner of the screen, but after several studies with prospective users, I discovered that these users tend to look for possible actions to complete on a screen near the bottom of that screen and that they could recognize textual buttons more quickly and accurately than their iconic representations. When users select the “Upload Photo” button, another screen loads, allowing them to browse to the images that they have taken on their smartphone and select one to add to the gallery. By selecting the “Filter Photos” button, users see a pop-up dialog, where they can select one or more filters to apply to the gallery. These two buttons allow users to customize the types of images that appear in the gallery, allowing the application to cater to a particular user’s image-viewing preferences.

### 3.3.6. History Presents Lighthouse Facts One Year at a Time

I present historical details for a lighthouse one fact at a time, allowing users to browse a lighthouse’s narrative at their own pace. Figure 3-9 on the following page presents the final design of the “History” screen within the application. The details that appear on this screen depend on the lighthouse that users have chosen to examine in more detail. In this example, the screen shows historic details about Cape Cod (Highland) Lighthouse in Truro, Massachusetts.

Users can navigate to this screen from another content screen by tapping on the name of that content screen in the top action bar, then by choosing the “History” option in the drop-down list that appears.

As in the “Information” and “Photographs” screen, the “History” screen features a left-facing arrow in the top-left corner and the name of the screen in the action bar. Users can return to the “Welcome & Search” screen by selecting the arrow, and they can navigate to a different content screen by selecting the “History” screen title and choosing a screen from the drop-down list.

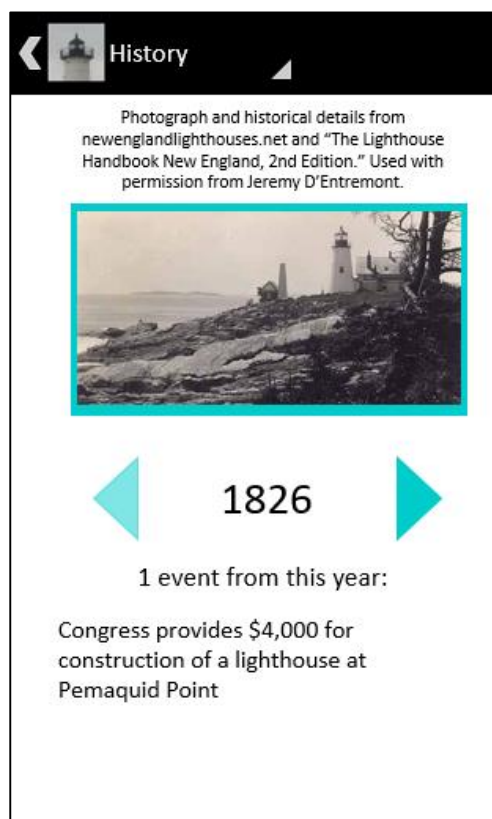


Figure 3-9. Final Design of “History” Screen within Application.

Beneath the action bar, I include an area of text that gives credit to Jeremy D’Entremont. Mr. D’Entremont kindly agreed to let me reference his research on this screen, and he deserves prominent recognition since he has worked for decades to compile different facts and images of lighthouses across New England. Beneath these credits, I include a historic photograph of the lighthouse from Mr. D’Entremont’s collection. I add a light blue frame around the image because most historical photographs appear in grayscale, and a white sky does not appear well, if at all, next to the white background on each of the content screens within my application. Most historic

photographs feature lighthouses in their original form, which helps users of the application appreciate the extent to which a given lighthouse has changed since its original construction.

Below this photograph, I include a year indicator with arrows on either side of the year provided. These arrows allow users to navigate through time in either direction to view each of the historical facts about the lighthouse that this application includes. If users have reached the first fact or most recent fact for a given lighthouse, one of the arrows appears faded to indicate that users should not expect the screen's contents to change should they select the faded arrow. After users have navigated to a given year, they see two pieces of information beneath it and the arrows:

- The number of events associated with a given year; and
- A set of descriptions describing the events themselves.

One of the subjects at a post-design usability study suggested that I include the number of events corresponding to each event marker, which motivated me to include it before the event descriptions. Some years correspond with more information than can fit in the bottom area of the screen all at once. When this problem occurs, a scroll bar appears next to the “number of events” heading and event descriptions to alert users that they need to scroll down to read about each of the events that took place during a specific year.

### 3.3.7. Reviews Present Visual Summaries of Visitors' Experiences

This screen shows icons symbolizing visitors' general opinion about the area surrounding the lighthouse, presenting the review sources in a glance-friendly format. The final design of the “Reviews” screen within the application appears in Figure 3-10 on the following page. The reviews that appear on this screen refer to the lighthouse that users have chosen to examine more closely. For example, the following figure shows reviews for the Cape Neddick “Nubble” Lighthouse in York, Maine. Users can navigate to this screen from another content screen by selecting the name of that content screen, then choosing the “Reviews” option in the drop-down list.

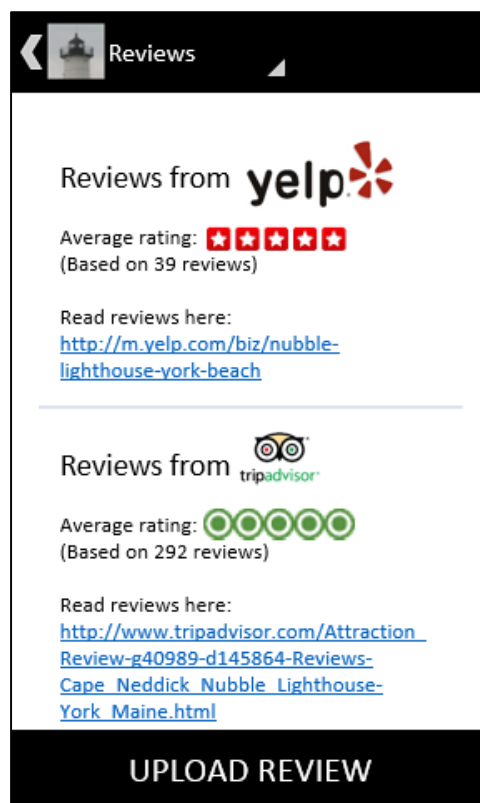


Figure 3-10. Final Design of “Reviews” Screen within Application.

As in each of the other content screens, the “Reviews” screen includes a left-facing arrow in the top-left corner of the screen and a heading in the action bar indicating the name of the screen. By selecting the arrow, users can return to the “Welcome & Search” screen within the application, and by selecting the “Reviews” heading, users see a drop-down list featuring the names of the other content screens. They can choose the content screen they wish to view next by selecting it from this list.

Underneath the action bar, I include summary information about reviews from two popular websites documenting traveling experiences: Yelp and TripAdvisor. Underneath each website’s logo, I present a visual representation of the average rating, which is easier for users to recognize quickly than a textual equivalent – such as 4.5 out of 5. To show the strength of the review score, I also indicate the number of people who have uploaded a review onto the website whose logo appears above the average rating. I then provide the URL for the page containing more information about the lighthouse location on the travel experience website so that users can select the link and view the web page itself. Because of Yelp’s stringent branding guidelines, I could not include actual reviews on this screen. a heading that indicates the website containing

the reviews that currently appear on the screen. Users can select the name of the website to view a drop-down list, where they can choose the name of a different website to view other reviews of the lighthouse. This version of the application can display reviews from Yelp or TripAdvisor. Presenting users with several sources for reviews helps reduce the biases inherent in each of the websites, such as the types of details that the reviews usually contain and the demographics of the travelers who post the reviews. Also, while Yelp encourages its reviewers to post about short visits to a location, TripAdvisor focuses more on extended stays at a particular destination, usually associating attractions with a particular lodging option.

Finally, users can select the “Upload Review” button at the very bottom of the screen to share their own experiences about visiting a particular lighthouse. The application asks users which website they wish to use for uploading a review, then takes users to that website to complete the uploading process. As with the buttons at the bottom of the “Photographs” screen, potential users of this application recognize action buttons best when they include text and when they appear at the bottom of the screen, which is why the “Upload Review” contains both of these user interface characteristics.

#### 3.4. Improving Design using Iterative, Audience-Based Process

The compelling, user-friendly screens described in the previous section required a great deal of interactive feedback and iterative improvement to create. I used a pre-design survey to direct the application’s initial design, then shared this design with prospective users at various lighthouses in New England to determine the interaction patterns I could include or change to maximize the application’s usability. Figure 3-11 on the following page illustrates the iterative, audience-driven process I completed to mold the application’s design into its final form.



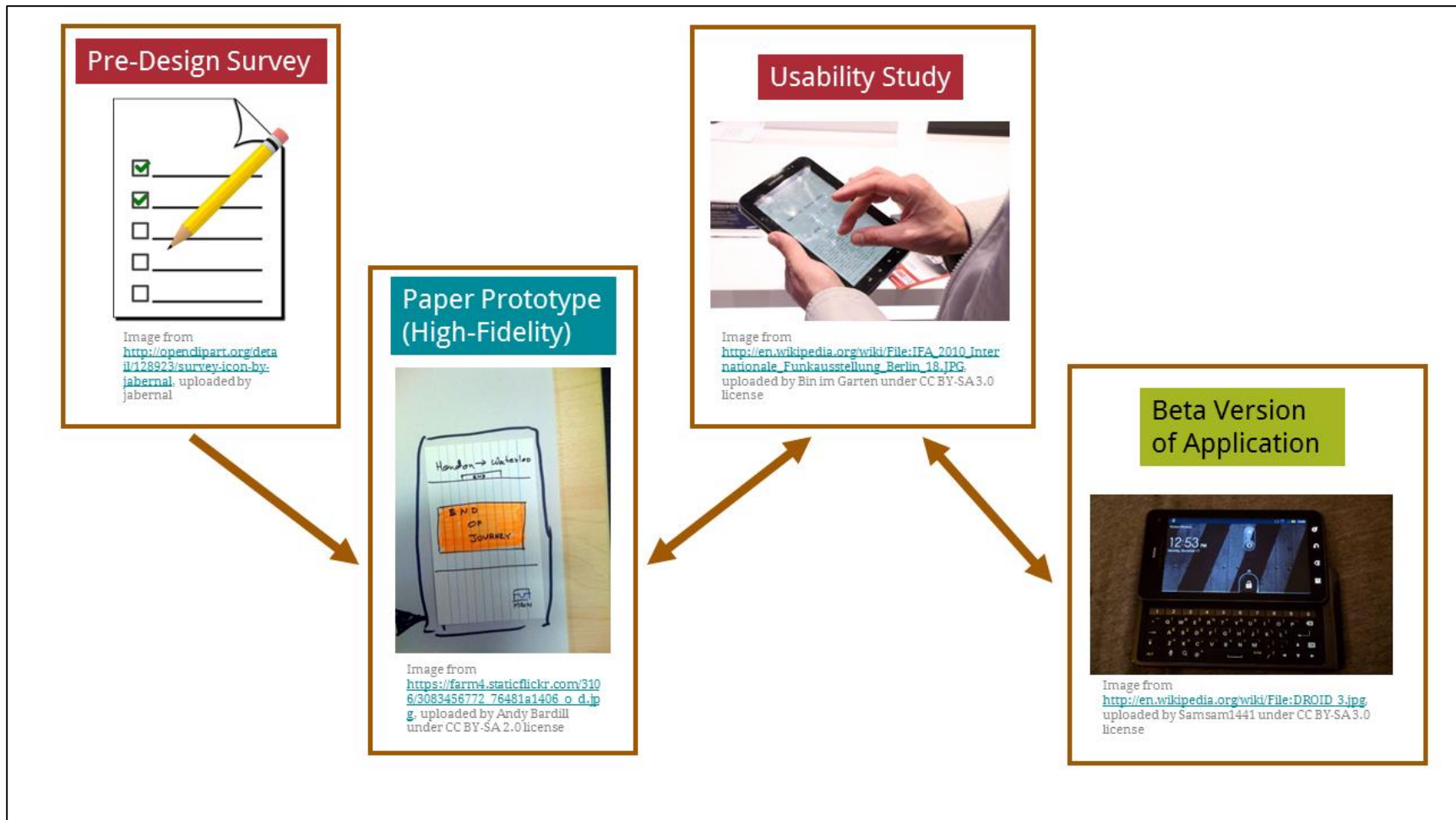


Figure 3-11. Application Design Process.

After creating a high-fidelity paper prototype of the application based on pre-design survey feedback, I completed several rounds of usability studies to improve the paper prototype iteratively. I then completed the same process after building the beta version of the application (exact implementation details described in next chapter).

### 3.4.1. Pre-Design Surveys Indicate Prospective Audience’s Interest in Application, Photographs

The feedback I received from the surveys at Cape Neddick (“Nubble Point”) Lighthouse in York, ME during early June 2013 (see Appendix D: Survey for Visitors of Cape Neddick Lighthouse) assured me that visitors expressed a significant amount of interest in using a lighthouse traveling application, especially since many of them own smartphone devices themselves. Of the 40 visitors who responded to the survey, 27 of them (68%) indicated that they own iPhone and Android devices. Furthermore, 19 of these 27 respondents (70%) who own an Android or iPhone expressed that they would be “very likely” or “extremely likely” to download my application. However, 17 of the 25 respondents (68%) who answered the question regarding application use frequency expressed that they would use it occasionally or less often. Therefore, these infrequent users would benefit from a user interface that offers enough simplicity and intuition for them to retain the application’s basic functionality. They could also take part in optional tutorials within the application to (re-)learn the essential pieces of functionality within the software.

These surveys also show that visitors to lighthouses remain cognizant of the visual space surrounding the lighthouse and appreciate opportunities to capture or view photographs. Of the 24 respondents who answered the question regarding lighthouse research activities, 17 of them (71%) indicated that they view photographs when discovering information about a particular lighthouse. Also, 30 of the 41 total respondents (73%) “always” take pictures when visiting lighthouses. Even more interesting, 19 of these respondents (46%) expressed that they visit a lighthouse “to appreciate the scenery that surrounds [it].” Clearly, these visitors appreciate the visual aesthetics of the lighthouse’s surroundings, so I realized after receiving these results that my application needed to offer users clear and plentiful options for interacting with photographs.

### 3.4.2. Initial Design Presented Hard-to-Use Search Options, Features in Photographs Screen

The initial version of the application design featured a less guided “Welcome” screen. In particular, the earliest version of this activity lacked an option for users to find lighthouses near their current location. Instead, I presented an input text box, allowing users to enter the name of a lighthouse they wish to view. Figure 3-12 on the following page shows this different set of options relative to the working beta version of the application.

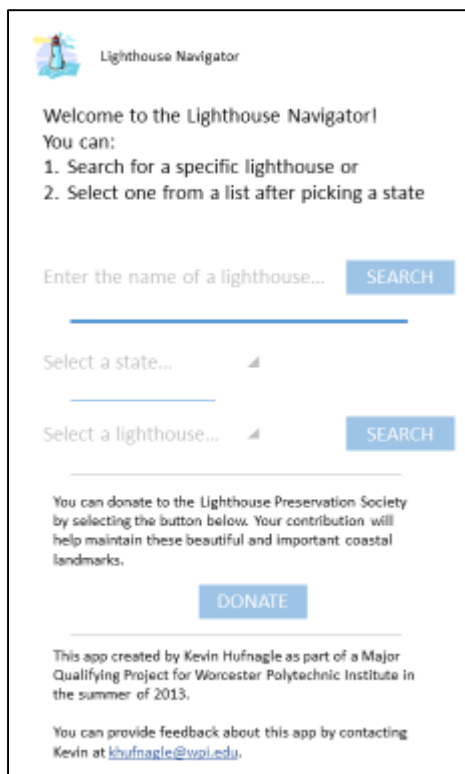


Figure 3-12. Initial Design of “Welcome” Screen within Application.

This interface element assumes that users have a good sense of lighthouse names, and when they travel to new lighthouses or ones they visit only occasionally, it’s possible for them to not know the official name of the monument.

Based on the responses I received from the pre-design surveys, I created a feature-rich “Photographs” screen within the initial design. Figure 3-13 on the following page depicts the earliest version of this activity. While creating a gallery with eight images visible at any given time would have offered more visual context to users, I soon discovered that Android devices’ screen width restrictions would prevent me from displaying more than one column of photographs at a time. In addition, I included several icons with a “clip art” style in the top-right corner of the screen. These icons, in fact, represent the different content-based activities within the application (Information, History, Reviews<sup>16</sup>). Upon interacting with prospective users in future studies and learning more about Android’s design paradigms, I discovered that – not only do these icons break the guideline of creating a menu-based activity selector element within the

<sup>16</sup> I called this activity “Stories” when I first designed the application, which gives the book icon a more sensible connotation.

interface – these icons lack the flat design that Android supports to such a great extent. These “clip art” symbols would not render well on smartphones, particularly ones with low screen densities.

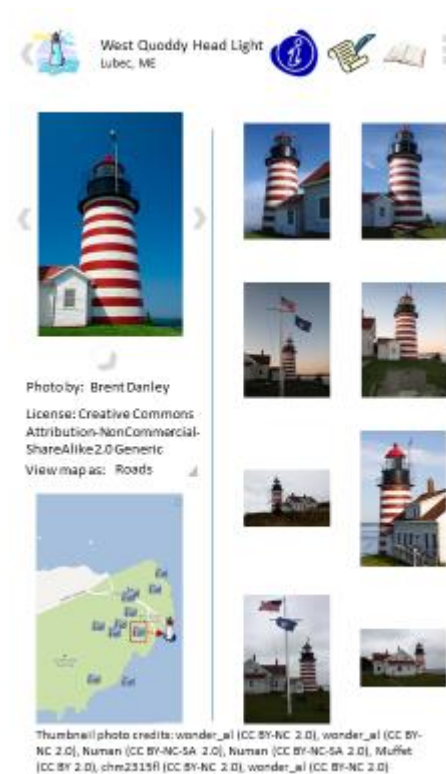


Figure 3-13. Initial Design of "Photographs" Screen within Application.

### 3.4.3. Post-Design Usability Studies Called for User-Guided “Welcome” Options, Limited Icons

In order to create valid and helpful feedback from the usability studies that I conducted, I set up a table to attract interest from visitors and created a script for guiding them through the different features of the application. I placed this “usability study station” outside of the lighthouse’s gift shop. This station featured a picture frame advertising the study, a clipboard for me to record subjects’ results, a binder containing the study script and written out questions for subjects, and a photo book in which I placed index cards containing printouts of my application’s design. While I refrained from reading the script verbatim in a robotic manner for most of the study sessions, I did rely on its structure to keep the presentation of the study consistent among subjects. This control over the studies’ progression offered more consistent feedback for me to consider while creating the next iteration of the application. More details concerning the usability setup and progression appear in Appendix F: Usability Study Materials.

My first study, conducted at Cape Neddick Lighthouse once again, focused on the application's high-level navigation and the interactive features that I offered within the "Welcome," "Information," and "Photographs" screens. Of the 6 subjects who took part in the study, 5 of them (83%) expressed that they would prefer to search for a lighthouse using a guided list than typing in the name of a lighthouse within the "Welcome" screen. In fact, many of them expressed disdain towards this manual entry option completely, with several of them suggesting that I include a "find nearby lighthouses" button. Based on this overwhelmingly consistent perception, I completed this user interface element switch. While these subjects appreciated the options available on the "Photographs" screen, 2 of them (33%) wished to see even more interactivity with the map; they envisioned the map expanding to fill the device's screen whenever they selected it from this activity. Since I could implement this feature surprisingly easily, I introduced this functionality in future versions of the application. Several subjects also requested that I add a "nearby attractions" field within the list of general facts about the lighthouse. I refrained from including this piece of information because it would have been difficult to categorize the attractions in a manner that made sense to users. I would have needed extra usability studies focusing on this feature alone.

The following study, which took place at Old Scituate Light in Scituate, MA during early-August 2013, focused on different types of interaction affordances within my application, particularly the icons that appeared within the "Photographs" screen. I soon discovered that the subjects demonstrated dismally low comprehension rates for these icons. Of the 10 subjects at this study, 2 of them (20%) could correctly identify the "upload photo" icon (a camera with an up-facing arrow next to it) and only 1 (10%) could identify the "filter photo" icon (a funnel). These results indicated indisputably that these buttons required text instead of icons. This study also focused on the possible methods for switching among the application's content-based activities, either a drop-down list at the top of the screen or a pop-up menu near the bottom. After examining these options, 8 of the 10 subjects (80%) indicated a preference for the drop-down functionality, which I introduced into the application once I finished developing this screen. As for the "Welcome" screen options, I showed the newly-placed "find nearby lighthouses" button alongside the list-based searching for states and lighthouses. The subjects expressed interestingly split opinions regarding this functionality; 3 of the subjects (30%) would select the "find nearby lighthouses" button, 4 of them (40%) would search with the list more often, and the other 3

(30%) would select an option based on the situation in which they used the application. Given this indecisiveness on the subjects' part, I kept both options in the "Welcome" screen.

#### 3.4.4. Post-Implementation Usability Studies Solidified Content-Based Screen Navigation

After implementing a beta version of my application, I conducted two further usability studies during the late summer months of 2013. These studies used a similar setup, except these subjects could interact directly with the application that I had installed on my Android device.

The first of these post-implementation studies took place at Highland "Cape Cod" Lighthouse in Truro, MA in late August 2013. This session focused on users' ability to master the application's high-level navigation functionality quickly. One of the most significant hurdles that these subjects faced involved transitioning from the "Information" screen to the "Photographs" screen. Each of the 7 participants attempted to select the photograph that appears near the top of the "Information" screen to complete the activity transition. Based on this perceived misunderstanding, I introduced a pop-up message informing users of the proper interaction strategy within future versions of this application. A more promising result, however, occurred when I instructed the subjects to view additional facts about the lighthouse. Every one of them understood that not all details about the lighthouse appear "above the fold" on the "Information" screen and instinctively scrolled down to view the remaining pieces of information. Therefore, I did not need to adjust the scrolling functionality within my application.

The final study for this project occurred several weeks later at Nauset Light in Eastham, MA. During this session, I tested the pop-up help message that I added to the "Information" screen and inquired about possible filters to include within the "Photographs" screen. Unfortunately, the cool weather and Sunday afternoon tour sessions limited the turnout to this study, but I still received valuable feedback from 4 hardy subjects. The "micro-tutorial" for navigating to the "Photographs" screen taught the subjects effectively, as 3 of them navigated to the second content-based activity with minimal assistance from me. These promising results assured me that this pop-up message offered useful guidance to new users who could become confused while attempting to switch among the content-based activities. These subjects also shared with me their interest in filters involving different parts of a lighthouse, weather conditions, time of day, and background content. I introduced some of these wishes into my image-processing algorithms, which classify images based on weather conditions and detect the

presence of rocks (arguably a piece background content) and lighthouse towers within images downloaded from Flickr.

### 3.5. Designing Universally Effective Application Involves Consideration of Market

In an ideal development environment, I could have focused exclusively on creating an interface that presents information and navigation elements that appeal most to the prospective audience. Market realities, however, forced me to develop interfaces that also conform to Android- and sponsor-specific guidelines. In particular, I needed to:

- Create an interface that can scale appropriately to the wide variety of screen sizes used in devices that support the Android platform; and
- Obey the branding specifications for any company whose information they plan on incorporating into the application.

#### 3.5.1. Presenting Screen Flexibility and Limiting Dependencies on Other Applications

In stark contrast to Apple’s mobile devices – such as the iPhone and iPad – Android devices lack a universal hardware design, forcing me to form an interface that strikes an optimal compromise between universality and scalability. While a limited personal budget prevented me from testing the application on an extensive variety of physical devices, I used various Android virtual devices to ensure that the application remains usable on multiple types of smartphones.<sup>17</sup> I maximized the baseline usability of my application by introducing graceful degradation on each screen. By using sizes of elements relative to the screen’s dimensions – rather than absolute dimensions – I ensured that the elements on the different screens of my application would scale appropriately to any Android smartphone screen size. I also used a scalable action bar – the collection of top and bottom bars on the application’s screens – that automatically introduces an “overflow” button when the button text cannot fit across the width of a user’s device. An ellipsis appears instead, and the user can select this ellipsis button to display a drop-down menu containing the names of the buttons that would have appeared on the action bar had more room been available.

In addition to introducing these scalable interface elements, I kept my application as self-contained as possible. Links to external applications included only a map-based application, such as Google Maps, and a browser application, such as Mozilla Firefox or Google Chrome. I added

---

<sup>17</sup> For the sake of limiting the project’s scope, I chose not to optimize my application for Android tablets. Using the principles discussed in this section, however, I could extend the application’s interface flexibility so that tablet users could also enjoy the Lighthouse Navigator application.

these links to my application with the confidence that most Android smartphones possess both a mapping application and a browser. When using these design principles to present a scalable interface that achieved independence from most external application, I gave myself more up-front work. I could then, however, add additional elements to the different content screens whenever user requests demanded me to do so, without fear that I would wreck the clean and consistent interface I had created.

### 3.5.2. Abiding by Branding Guidelines forms Important Ethical Foundation

Just as Android's hardware fragmentation guided my design of my application's interface, the branding guidelines of various websites – including Flickr and Yelp – and requests from Jeremy D'Entremont affected how I present the content within the application's different screens. In addition to presenting photographs that contain only Creative Commons licenses within my application, I included attribution to the photographer and the license itself near each image, abiding by the brand guidelines for both Flickr and Creative Commons. Yelp contains such a stringent content-based branding guideline that the website practically created the content layout on the “Reviews” screen of my application for me. In particular, I could not display numerical representations of the average review score, the reviews themselves, or reviews from another source before those from Yelp. While TripAdvisor's guidelines contain far fewer restrictions, I present reviews from this alternative website beneath those from Yelp in the same manner to keep my interface consistent.

After discovering the wealth of content related to New England lighthouses on Jeremy D'Entremont's websites, and contacting him for permission to use excerpts of his research about lighthouse facts and historical details, he kindly obliged and even allowed me to present the content in a manner that I had determined would look best within my application. This freedom in designing the content layout on the “Information” and “History” screens within my application complemented the regulatory structure of the “Photographs” and “Reviews” screens. When Android designers develop applications of their own, they need to keep the branding guidelines and content attribution requests of their sponsors in mind. Of course, an ideal scenario would occur when a company's employees create an application for that company; in this case, employees can rely exclusively on internal branding guidelines, rules they probably understand well.



### 3.6. Presenting the Modular Structure of the Android and Image-Processing Projects

After spending the last few sections discussing the application from the end-user's perspective, I focus on the developer's perspective of the project in this section. I present class diagrams illustrating the structure and relationships among the application's key modules and offer remarks on the most significant components within each piece of the application. I also present more extensive details about the application's internal data structures and logic in the following chapter.

#### 3.6.1. Android Application Complex Enough to Warrant Separate Class Diagrams

The Lighthouse Navigator application contains such a complex internal structure that a single class diagram would present an overwhelmingly confusing schematic of the relationships among the classes I have developed for the software. Therefore, I illustrate the application's structure across four class diagrams: the application logic within the user interface screens, the data structures for storing lighthouse information, the utility classes for completing downloading and parsing tasks, and the adapters for displaying complex user interface elements.

##### *Photographs Screen Dominates Application Logic Complexity*

While each screen within the Lighthouse Navigator application presents a significant amount of information, only the "Photographs" screen contains a substantially complex internal structure. Figure 3-14 on the following page indicates how the `PhotographsActivity` class serves as the gateway to the application's lower-level logic, particularly the background threads for completing the Flickr photograph downloading process.<sup>18</sup> In addition, both the `PhotographsActivity` and the `ReviewsActivity` reference the data structures corresponding with these screens' names, forming a sensible relationship between a category of lighthouse information and the screen on which these elements appear.

---

<sup>18</sup> Note that this class diagram omits relationships among the activities themselves; since I discuss the related interdependencies earlier in this chapter, presenting them in an illustration would create unnecessary clutter.

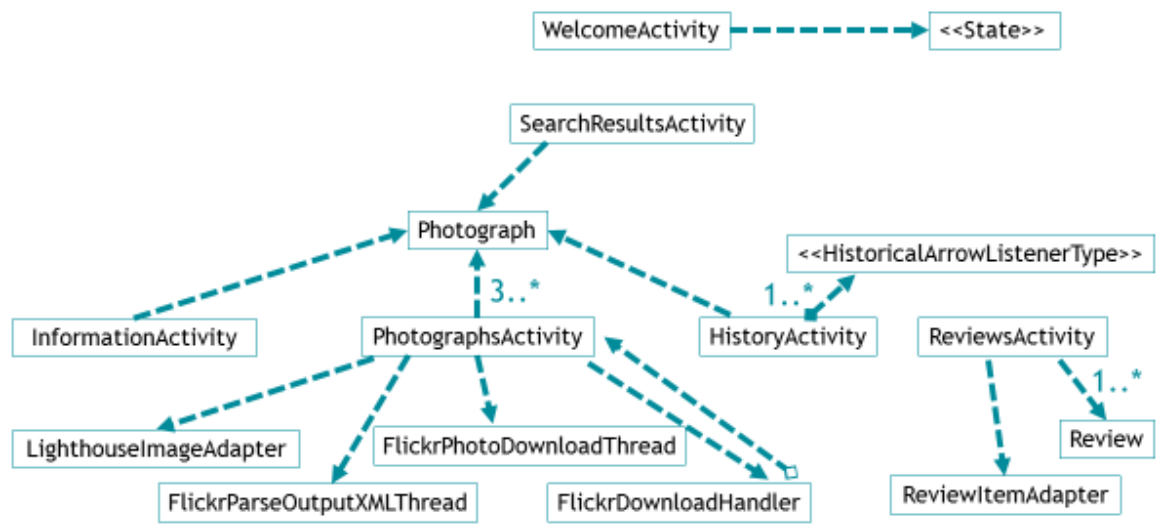


Figure 3-14. Class Diagram for Lighthouse Navigator User Interface Screens.

*Lighthouse Data Structure Contains All Other In-Memory Objects within Application*

Since the application uses lighthouses as a proof-of-concept implementation, the central in-memory data structure, `Lighthouse`, represents one of these seaside landmarks. As Figure 3-15 below shows, this data structure in fact contains all other objects, which in turn collect subsets of information regarding lighthouses, such as the physical or geographic address of a tower. Also, the `Photograph` class forms the only connection to the `ActivityVisible` enumeration within the project; the `Lighthouse` class in fact maps an `ActivityVisible` value to a set of `Photograph` objects to determine the screen on which these images appear.

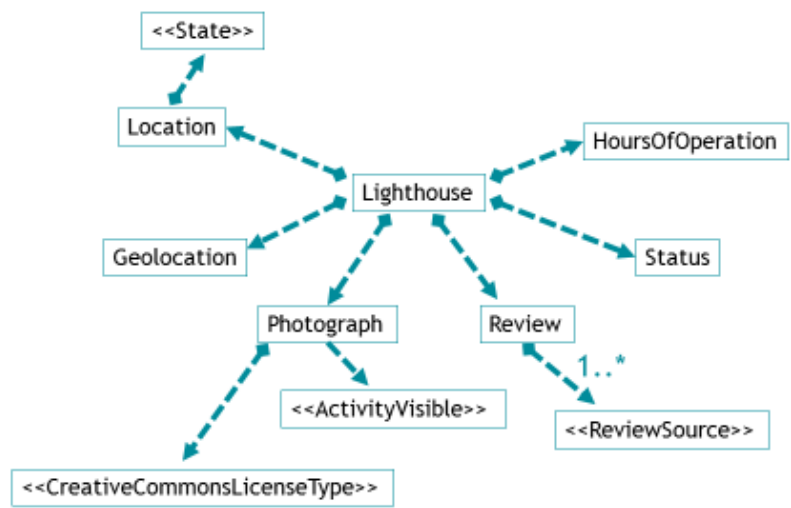


Figure 3-15. Class Diagram for Lighthouse Navigator Data Structures.

### Download Handler Serves as Main Utility Manager within Application

The application's utility classes, which provide the background downloading and parsing functionality for organizing lighthouse information and photographs from Flickr, rely on the FlickrDownloadHandler class to regulate their execution. The handler class in this project manages the application's multithreading functionality, in particular monitoring the progress of the FlickrPhotoDownloadThread and starting execution of the FlickrParseOutputXMLThread once the application has successfully received images and photograph metadata for all pictures stored on Flickr's servers. Besides depicting the central functionality of the handler, Figure 3-16 below shows that these utility classes reference Photograph and Lighthouse objects frequently, which seems sensible considering that these two data structure represent the application's focal points.

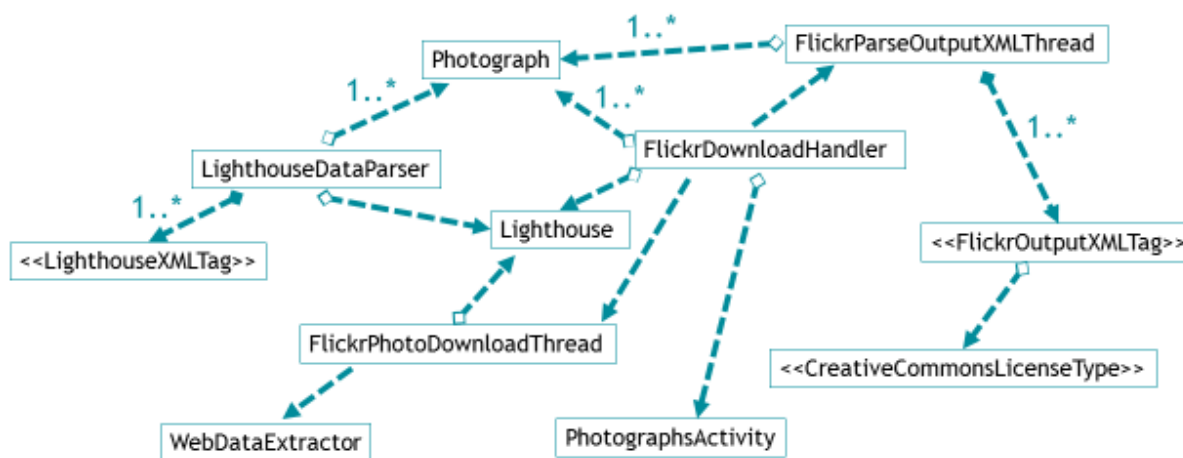


Figure 3-16. Class Diagram for Lighthouse Navigator Utility Classes.

### Image Adapters Stay Simple in This Application

For this project, I introduce two adapters into my application, which present collections of specific data structures and therefore retain simple internal structures. As Figure 3-17 on the following page illustrates, both of these adapters contain a collection-based field storing information about the data structures they display and little else. The ReviewImageAdapter class also references the ReviewSource enumeration, which allows the adapter to display information about the correct website within a specific location on the “Reviews” screen.

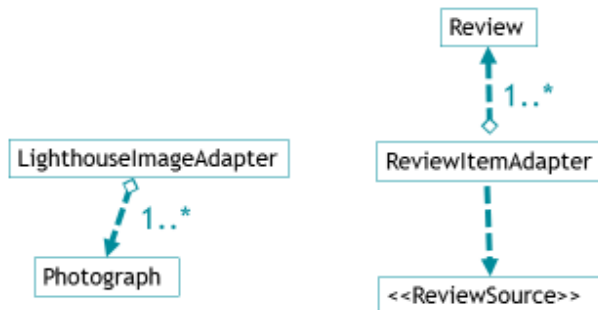


Figure 3-17. Class Diagram of Lighthouse Navigator Interface Adapters.

### 3.6.2. Class Diagram for Image-Processing Algorithms Includes Minimal Coupling

As discussed within section 4.7.1, “Discovering inability to Use ImageJ Directly with Android,” I could not integrate the image-processing algorithms I developed for this project into my application; as a result, I created a separate project that expresses the relatively simple internal structure that this collection of algorithms possesses. Figure 3-18 below shows that these image-processing algorithms remain mostly self-contained, allowing me to run and test them independently more easily. The only interdependency stems from convenience. The `LighthouseFeatureFinder` class, which performs a series of transformations on a test image of unknown contents to detect a specific feature shape, clamps the 90th percentile of intensity values to 255 during one step. Since the `ImageClassifier` class already creates a cumulative distribution function of intensity values, I used this method to locate, and clamp, the top 10% brightest pixels in a grayscale, transformed version of the test image for shape-based feature-matching purposes. I add more subtle program integrity by creating `ChamferNeighborDistances` and `ChamferAnalysisDirection` enumeration objects within the `FeatureShapeMatcher` class. I introduce these symbolic constants to counteract the algorithmic complexity that the shape matcher contains, improving the class’s maintainability.

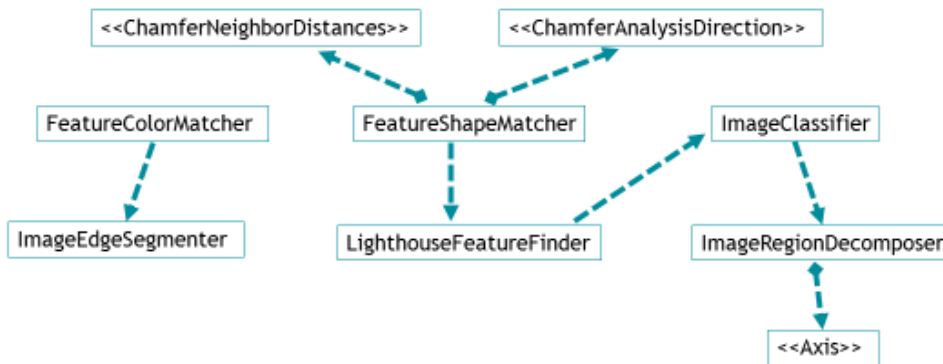


Figure 3-18. Class Diagram of Image-Processing Algorithm Logic Classes.

## 4. Structure and Strategies for Implementing Application

After completing a preliminary design for the Lighthouse Navigator application and testing its feasibility within a survey and two usability studies, I developed the application logic and data structures necessary for presenting information shown in the design storyboards. I soon discovered that this complex application would require a well-structured code hierarchy, so I separated the project's source code files into multiple packages based on their different roles. I used several built-in Android tools – including the SparseArray data structure and XmlPullParser interface – as well as my own data structures to achieve code clarity and simplicity. These in-memory data structures contained only text, however, so I created several adapters for converting the textual data to compound user interface elements that users would find visually pleasing. The photograph gallery within the application's "Photographs" screen represents a cornerstone feature, so I used Flickr's API to search for as many geotagged photographs of a particular lighthouse as possible and organize the information that Flickr embeds in search results. I then created a set of background threads to copy this data into the application's resource files. Flickr's elegant API inspired me to develop an XML file of my own; this document encapsulated the different pieces of lighthouse information that the application's different content-based screens present. To provide even more persistent data, I created a photograph cache that allows users to view photographs of a lighthouse from Flickr consecutive times without having to contact Flickr's servers to download the images each time. Once I organized the set of photographs for a particular lighthouse, I created a set of image-processing algorithms that filter the photographs by weather conditions, presence of rocks, and presence of lighthouse towers. Finally, I created several sets of training data to test these algorithms and evaluate their performance, as discussed in section 5.1, "Findings from Image-Processing Algorithm Testing." This chapter discusses each of the above implementation steps in more detail.

### 4.1. Preparing Development Environment: Eclipse and ADT

When developing the Lighthouse Navigator application, I used the Eclipse integrated development environment (IDE) to achieve a simpler and more customizable programming experience. Eclipse offers real-time tools that assist with Java development, such as syntax highlighting, method suggestions, and real-time compilation checking. These features allowed me to focus on solving the high-level design problems that I faced while developing the application. The software also includes a handy refactoring feature, which allowed me to change

the names and locations of the different classes that comprise the project with ease. Eclipse automatically updates all references to a class whose attributes change, encouraging me to use an appropriate set of class names and a programmer-friendly class structure at all times throughout the development process. Eclipse also allowed me to customize my project settings extensively. I could specify a specific set of formatting rules for my source code, which Eclipse applied every time I saved a source code file within my project. Therefore, I achieved a uniform style across each of these source files. Also, I adjusted the syntax structures that triggered compilation errors or warnings, preventing me from introducing subtle bugs into my project files. Such errors include memory leaks, unintended fall-through between cases of a `switch` statement, and attempted null-pointer access.

Eclipse offers an Android-based plug-in called the Android Development Tools (ADT), which offers the same real-time development features for Android that Eclipse has normally for Java in addition to several other useful debugging tools. By integrating ADT into Eclipse, I received useful real-time feedback as I developed my application, including warnings about deprecated methods and suggestions for methods to use within a particular context. ADT also includes a tool called the Dalvik Debug Monitor Server (DDMS), which emulates an Android device's virtual machine once it begins communicating with a computer running Eclipse and ADT. DDMS includes a tool of its own called LogCat. This feature, which appears in a pane within the Eclipse workbench, displays log messages sent to an Android device. I used LogCat to display console assurance messages, such as reaching a particular point in a method, and to view exception messages to help direct me to the source of any run-time errors I had introduced into my code during a programming session.

#### 4.2. Presenting an Overview of Lighthouse Navigator Application

A project as large and complex as this application requires a stable, easy-to-follow code structure. Therefore, I divided the project's source code into different packages based on each file's role. Similarly, I distributed the application's resource files across several folders. I also created a consistent activity (screen) hierarchy within my application so that users could understand the application's high-level navigational structure as quickly as possible. Finally, I needed to include several supporting projects to simplify my application's code and make the project as a whole more backwards-compatible. I describe each of these high-level elements of my source code within the sub-sections of this section.

#### 4.2.1. Dividing Source Code into Various Role-Based Categories

In order to achieve code understandability and scalability, I divided the source code files within my project into role-based packages. While each file contributes towards the application's functionality, each one encapsulates a specific category of responsibilities. Each package represents one of these categories. The role-based division of source files also makes it easier for other programmers and me to extend the functionality of this application in the future.

In particular, I divided the source code files within my project into the following five packages:

- *Base folder* (`edu.wpi.khufnagle.lighthousenavigator`) – Contains activity logic for each of the different screens that appears as users interact with my application;
- *Data* (`edu.wpi.khufnagle.lighthousenavigator.data`) – Contains custom data structures that I use for organizing and accessing in-memory information about the lighthouses that the application features. I discuss these data structures in more detail within section 4.4, “Creating New Data Structures for Organizing Run-Time Application Data;”
- *List* (`edu.wpi.khufnagle.lighthousenavigator.list`) – Contains custom enumerations for expressing categories of options more clearly within the other source code files of the project. I describe each of these lists within section 4.4.3; “Review Data Structure Organizes Visitors’ Experiences of Lighthouses;”
- *Util* (`edu.wpi.khufnagle.lighthousenavigator.util`) – Contains the utility classes needed for downloading and processing the lighthouse information that I retrieve from existing websites. I describe the operations within these classes in more detail within section 4.5, “Using Flickr’s API and Android Multithreading to Download Photographs;” and
- *View* (`edu.wpi.khufnagle.lighthousenavigator.view`) – Contains classes representing collections of user interface objects that I add as units within several screens of the application. I describe these custom user interface in more depth within section 4.4.5, “Creating Custom User Interface Objects to Encapsulate Lighthouse Information.”

The image-processing classes appear within a separate project in my source code since I provide only a proof-of-concept implementation demonstrating their functionality. I discuss this section of code in more detail within section 4.7, “Constructing Image Processing Algorithms with ImageJ.”

#### 4.2.2. Arranging Resource Files using Android Guidelines

I distribute the different types of visual resources featured in the application across several folders so that I can locate and update these files more easily and see the interdependence among the different files more clearly. This folder hierarchy also matches Android’s general guidelines. Each type of resource appears in a folder that describes the resource’s role within the project.

In particular, my application projects contains five different categories of resources. I represent each of these categories with a folder; a more detailed description of their contents appears below.

##### *Drawable*

This folder contains each of the images that appear on at least one screen within the application. These images include navigation icons like buttons and arrows as well as the photographs that the application can display without an Internet connection. This folder also includes several XML files that cause the application to display different versions of a particular image based on users’ actions, such as selecting a particular image.

I apply a specific set of naming conventions for the photographs of lighthouses that appear within this folder, which appear in Table 4-1 on the following page. The data structures that I create (and describe in section 4.4, “Creating New Data Structures for Organizing Run-Time Application Data”) depend on the images containing this *exact* naming structure.



Table 4-1. Naming Convention of Lighthouse Photographs Appearing within “Drawable” Resource Folder.

Role of Image	Naming Structure	Example
<b>Thumbnail within “Search Results” screen</b>	"searchresults_ <modified_name_of_lighthouse>.png"	“searchresults_pemaquidpointlight.png”
<b>Header image within “Information” screen</b>	"information_ <modified_name_of_lighthouse>.png"	“information_pemaquidpointlight.png”
<b>Gallery image along right-hand side of “Photographs” screen</b>	"photographs_ <modified_name_of_lighthouse>_ <flickr-id>.png"	“photographs_pemaquidpointlight_2708085451.png”
<b>Header image within “History” screen</b>	"history_ <modified_name_of_lighthouse>.png"	“history_pemaquidpointlight.png”

In the above table, “modified name of lighthouse” refers to the name of the lighthouse with all spaces removed and all characters converted to lower-case letters. Also, the Flickr ID refers to the unique ID number assigned to the photograph on Flickr’s servers; I introduce this value to ensure that all images within my application contain unique file names. I use a naming convention as consistent as possible so I can extend the application’s functionality more easily in the future.

This folder in fact appears as five separate ones within the application’s resource hierarchy. Each folder represents a different screen density. By including images of an appropriate size for each of these resolutions, I can ensure that each Android smartphone using my application will render these images correctly, without distortion or scaling artifacts.

### *Layout*

This folder features XML files that describe the visual structure of the different user interface elements on each screen within the application. These XML files closely resemble HTML files describe each page on a website. This folder also contains XML files describing the layout of the more complex user interface elements, such as the dropdown list allowing users to select a content-based activity and the list depicting lighthouse search results.

### *Menu*

This folder includes an XML file that describes the layout of the bottom action bar on the “Photographs” screen, specifying the location of each action button that appears within this user interface element.

### *Values*

This folder contains four XML files that specify constant values used within the XML files representing other project resources. These files include:

- *Colors* – Lists the names and values of each custom color used within the application, such as the background color for buttons on the different screens;
- *Dimens* – Lists the names and dimensions of the different text sizes and screen margins featured within the application;
- *Strings* – Lists the different pieces of text that appear within the application, including sub-headings within the different screens and paragraphs that appear regardless of the lighthouse users have chosen to view; and

- *Styles* – Lists the set of custom styles that I created for the different user interface elements on the application’s screens. These styles, which resemble CSS rules quite closely, allow me to make changes to a particular style and propagate these changes across each of the user interface elements that use that style. This file also makes the XML files found in the “Layout” folder less verbose and repetitive, since I can express a screen element’s appearance with a single style instead of a collection of attribute assignments.

## XML

The XML file in this folder lists the general facts, photograph details, historical events, and user review metadata for each lighthouse that the application features. I discuss this file in more depth within section 4.6.1, “Creating an XML File Structure that Allows for Simple Additions and Updates.”

### 4.2.3. Creating an Appropriate Metadata Structure

Within my application’s manifest file (`AndroidManifest.xml`), I select attributes for the application’s activities that simplify the screens’ interfaces and the navigation among the application’s screens. The “label” attribute, which usually presents the name of the screen to users, is blank for my application’s content-based activities because this name actually appears within the content-activity drop-down list that I include within the screen’s top action bar. Since I automatically mark the correct element within this list as active (and visible to users) based on the screen shown, this list element effectively forms the title of the activity instead. Each activity in the application descends directly from the “Welcome” activity, except for the “Welcome” activity itself. Users trigger this functionality by selecting the “up” button in the top-left corner of any screen within the application. This simple activity hierarchy gives users one-touch access to the application’s main menu on every screen, allowing them to become familiar with the application’s screen-based navigation more quickly.

In addition to presenting a clear, simple navigation system within my application, I require users to consent to as few permissions as possible when installing the application so that they can maintain their privacy and personal security on their phones. Nonetheless, some features within my application require special permissions, which I enumerate and describe below:

- `READ_GSERVICES` and `MAPS_RECEIVE` – Allows the application to display the location of a lighthouse within an embedded Google Maps object on the “Photographs” screen;
- `ACCESS_COARSE_LOCATION` and `ACCESS_FINE_LOCATION` – Grants the application permission to find lighthouses near the user’s current location;
- `ACCESS_NETWORK_STATE` and `INTERNET` – Allows the application to determine whether the user is connected to a network or Wi-Fi Internet service; if so, the application can use this connection to download photographs from Flickr for viewing within the application; and
- `WRITE_EXTERNAL_STORAGE` – Grants the application permission to store information related to Google Maps fragments on the smartphone’s external storage device.

Since some users of my application might use older Android smartphones than others, I allow my application to run on as many versions of the Android operating system as possible, maximizing its backwards-compatibility potential. I have designed my application to run on the newest version of Android (SDK level 19, or version 4.4 “KitKat”), but the application can also run on operating systems as old as SDK level 8 (version 2.2 “Froyo”). According to the Android Developers site, devices with an older operating system version than “Froyo” accounted for less than 1% of check-ins to Google’s servers in August 2013, so it’s safe to assume that my application is available to about 99% of the current Android smartphone user base [13]. In order for the application to use some of the newest user interface features on older phones, I use Android’s “support” libraries within my application project. I discuss the libraries that I include in more detail within the following sub-section.

#### 4.2.4. Introducing External Packages into Project

In order for older Android smartphones to render the user interface elements within my application correctly, I include several Java archive files (those with a `*.jar` extension) from Android’s support libraries as part of my application project. Version 4 of the support library (the one whose classes run successfully on all Android devices with an SDK level of 4 or greater) contains classes that allow Google Maps fragments to display correctly within application screens, such as the “Photographs” screen within the Lighthouse Navigator application. I use the files from version 7 of the support library even more extensively; this set of source code allows the action bar to appear at the top of each activity within my application. The action bar includes

the drop-down list, allowing users to select a specific content-based screen when viewing information about a lighthouse.

In addition to integrating some of Android's support libraries into my application, I also use version 2.4 of Apache's Commons IO library to express web-based input/output functionality more concisely. In particular, the `transferURLToFile()` method within the `WebDataExtractor` class of my application project contains a call to the Apache Commons IO method `copyURLToFile()`, allowing me to take the HTML source code from a website and transfer it to a file within the phone's internal storage with a single line of code.

### 4.3. Using Existing Data Structures and Packages to Enhance Application Efficiency

As I developed the Lighthouse Navigator application, I used several existing software data structures and packages. These tools simplified the application logic for storing and transferring data, allowing me to focus on the best methods for gathering and designing the information within the application instead. In particular, I used the `SparseArray` class for storing historical events and the `XmlPullParser` interface for processing XML documents containing structured lighthouse information; both of these tools reduce the amount of memory that my application consumes while running. I also included the Apache Commons IO package, which features a conveniently concise method for transferring web contents to a file within an Android device's internal storage space. I describe each of these software tools within the following subsections.

#### 4.3.1. Sparse Array Data Structure Trades Access Time for Efficient Memory Use

In the `LighthouseData` class, I use a special type of map called a `SparseArray` to maximize the space efficiency of the historical data that I associate with each lighthouse featured in the application. The structure and interface of this Android-specific collection mirror those of a `HashMap` in Java, but the data structure uses less memory space at the cost of a slight decrease in performance. According to the Android Developers guide for the `SparseArray` class, the class maps primitive `int` types, rather than the wrapper `Integer` class, to values of type `Object`. Since `SparseArray` keys do not need to be auto-boxed and can fit in four or five bytes (depending on the type of machine in use), they consume less memory. On the other hand, access to elements in a `SparseArray` is slightly slower because value access relies on a binary search and because element insertion or removal requires the traversal of a linear array structure. The Android Developers guide assures programmers, however, that the performance decrease for

`SparseArray` objects with less than a few hundred elements is at most 50% [59]. Since I associate only a half-dozen or so historical pieces of information with each lighthouse, I can safely assume that this data structure marginally reduces the application's overall performance.

#### 4.3.2. XML Pull Parser Offers Event-Driven Processing of Lighthouse Data

The XML PULL API offers an event-driven model for processing XML documents that offers convenience for programmers and memory efficiency for devices. The API processes one element of an XML file at a time, so I could create separate handlers for each type of element and build the appropriate in-memory data structures within my application more easily. This type of parser, like the Simple API for XML (SAX), stores the contents of the XML file as a single string [34, 57]. This storage strategy uses far less memory on the device than does the Document Object Model (DOM) method of processing an XML file, which creates entire `Node` and `NodeList` objects to represent different XML elements. This event-based API requires that I process the elements in the order that they appear within the file. Since the data structures I create within my application project mirror those found in the XML file anyway, however, this restriction does not adversely affect my ability to extract information from the file.

The XML PULL API offers a simple abstraction of well-formed XML documents to complement its good performance. According to the creator of the API, Aleksander Slominski, objects using this API recognize five different types of XML elements [57]:

- `START_DOCUMENT` – The beginning of the XML file, which parsers always select automatically when they begin processing a document;
- `START_TAG` – The opening tag of an XML element;
- `TEXT` – The contents between tags delimiting an XML element;
- `END_TAG` – The closing tag of an XML element; and
- `END_DOCUMENT` – The end of the XML file, indicating that the parser has now processed the entire document.

A well-formed XML document always contains `TEXT` elements between `START_TAG` elements and `END_TAG` elements. Also, tags nest properly, with all [`START_TAG`, `TEXT`, `END_TAG`] element groups contained completely within other element groups. Since I use well-formed documents within this project, I can use data structures that rely on the parser's ability to

process elements in sequence and nested elements (elements that appear within other elements) from the inside out.

Within my application, I use Android's implementation of the XML PULL API to download images from Flickr and organize the images' respective metadata in a convenient manner. In order to use the `XmlPullParser` interface, I create a new instance of the `XmlPullParserFactory` class. Since the Android implementation of this interfaces locates the appropriate parsing class automatically by searching through the device's `CLASSPATH` environment variable, I do not need to locate the class myself [35]. The parser examines the elements' start tags in the order in which they appear within the document, capturing each of the relevant attributes as needed. For example, when I begin processing the `START_TAG` of a `PHOTO` element, I need to capture the `id`, `url_z`, `ownername`, and `license` attributes that Flickr includes with this start tag. (An explanation of these attributes appears in section 4.5.1, "Structuring Calls to Flickr's API to Obtain Geotagged Images.") I use these attribute values to create an appropriate name for the image that I download from Flickr into the internal storage space of the Android device. Once I have processed the corresponding `END_TAG` event of the `PHOTO` element, I place these attributes into an in-memory data structure that I call a `Photograph`. I use this type of object within the `LighthouseImageAdapter` class (discussed in section 4.4.6, "Converting a List of Photograph Data into a Collection of Images") to maintain the connection between an image's metadata and its contents. As the parser continues to process the different elements within the XML document containing the Flickr API response, I place the elements currently being processed on a stack, using a custom object called a `FlickrOutputXMLTag` to represent each element and the plain text with which it corresponds. This stack allows me to extract information from nested elements in the proper order based on the document's structure.

Since my own XML file containing general lighthouse information appears as a resource file within my project, I use a similar process but a different implementation of the XML PULL API to extract information from this document. Instead of using the standard `XmlPullParser`, I use the `XmlResourceParser` to indicate that the XML document to parse appears within the project as a resource file. As I process the `END_TAG` event of a `PHOTOGRAPH` element within this document, I use the `getIdentifier()` method to associate the resource ID of an image (pointing to the image itself) with a `Photograph` object containing that image's metadata. This

process and its purpose mirror those of mapping downloaded Flickr images to their respective pieces of metadata.

#### 4.3.3. File Utilities from Apache Commons IO Drastically Simplify Extracting URL Contents

When retrieving image data from Flickr, including the images themselves, I rely on the `FileUtils` class in the Apache Commons IO library (version 2.4) to complete the web response transfer in a simple and efficient manner. According to the Apache Commons website, this utility library comprises six packages, including: reading files using Java's `Reader` classes, writing files using Java's `Writer` classes, filtering files, and comparing files. The library also includes a group of utility classes for performing operations, including the transfer of web data [8].

As I looked for assistance on downloading and saving web response data using Java, I stumbled upon a StackOverflow “answer” from Shengyuan Lu, who suggests using the Apache Commons IO library to complete the transfer to a local file in a single line of code using the `copyURLToFile()` method. Even handier, the users whose comments appear below Lu's answer remind developers to also include timeout parameters [43]. These time parameters prevent the method from blocking indefinitely by setting a maximum amount of time for connecting to the website and for extracting the website's data. By incorporating this library and its useful functionality into my project, I could express the extraction process from the website to a local file using simpler code and fewer exception-handling cases. This streamlined code decreased both developing and debugging time.

#### 4.4. Creating New Data Structures for Organizing Run-Time Application Data

The data structures and packages discussed in the previous section provided simple, elegant tools for storing and transferring data within my application, but I still created my own data structures that cater to the problem domain associated with this project: lighthouses and their photographic and experiential contexts. I introduced three large data structures – `Lighthouse`, `Photograph`, and `Review` – to encapsulate the different types of information about lighthouses so that I could access this domain-specific data within other areas of the application in as convenient and flexible a manner as possible. I also formed several enumerations to represent sets of categories more robustly. I describe these custom data structures in depth within the following sub-sections.



#### 4.4.1. Lighthouse Data Structure Contains Information for Content-Based Activities

The `Lighthouse` data structure forms an in-memory container for each piece of information available about a particular lighthouse that my application features. Some fields within this data structure remain simple enough to require primitive or standard-library typing. The name, phone number, website, and light pattern fields for a given lighthouse are strings since they contain alphanumeric characters and several other symbols, such as `(`, `)`, and `_`. I express the phone number as a string in order to create a standard format across all lighthouses visible within the application. Similarly, I use integers to express the year built and height fields of the lighthouse since these facts express exclusively quantitative information.

Other details about the lighthouse contain additional complexity and, therefore, require data structures of their own to express their contents clearly. Each lighthouse's address, geographic coordinates, hours of operation, and visitor access details reside in compound data structures. Each data structure contains a descriptive name and a set of embedded pieces of information:

- `Location` – Represents a physical address. Contains strings representing the street and city name and a value of the `State` enumeration class for the state;
- `Geolocation` – Represents a set of geographic coordinates. Includes two double-precision floating-point numbers, one for the latitude and one for the longitude. These coordinates indicate the lighthouse tower's precise location;
- `HoursOfOperation` – Represents a collection of information about the times of day and seasons when the lighthouse museum or visitors' center is open. Contains three strings: one for number of days during the week when the lighthouse is open, one for the time interval when the lighthouse visiting area is open (such as 8:00 AM – 5:00 PM), and one for the range of months when this area is open (such as May – September); and
- `Status` – Represents a collection of information regarding the extent to which visitors can explore the lighthouse. Include three strings: one indicating whether the general public can access the grounds surrounding the lighthouse, one indicating whether the general public can climb to the top of the lighthouse tower, and one describing miscellaneous facts and details, such as visitors' ability to eat meals in the lighthouse tower or stay for a night or week inside the keeper's cottage.

The Lighthouse object even includes several collections, which I express using maps and lists. I use a `HashMap` object to organize the set of photographs (or the “album”) containing images of the lighthouse. Each element within this album maps an enumeration value (of type `ActivityVisible`) representing an application screen to an `ArrayList` of `Photograph` objects describing the images that appear within that application screen. Each `Photograph` object includes metadata about an image as well as a reference to the image itself. To provide an ordered and organized collection of historical events that have taken place at the lighthouse, I place these data into a `SparseArray` object (described in more detail within section 4.3.1, “Sparse Array Data Structure Trades Access Time for Efficient Memory Use”). This object maps a year to an `ArrayList` of strings that describe the set of events that occurred during that year at the lighthouse. Finally, I combine the sets of information regarding visitors’ experiences of lighthouses from different websites into an `ArrayList` of `Review` objects. I use a list collection instead of a set because Yelp’s branding guidelines indicate that reviews from that website should always appear first. By using a list with information from Yelp as the first element, I can guarantee that Yelp will appear at the top of the screen every time the “Reviews” activity loads.

The Lighthouse object also contains a static utility method called `constructDirectoryFriendlyLighthouseName()`. This method transforms the format of a lighthouse name for use within a file name on a device’s internal storage drive or in the project’s resource files. In particular, it removes the spaces within the name of the lighthouse since computers do not always exhibit consistent behavior when processing file names that contain spaces. This method also converts all characters to their lower-case equivalents because Android runs a distribution of Linux; as such, file names are case-sensitive from a processing perspective.

#### 4.4.2. Photo Album Data Structure Encapsulates Photograph Information

The `Photograph` object incorporates the metadata associated with an image appearing within the Lighthouse Navigator application, such as the photographer and the Creative Commons license associated with the image. Unlike the Lighthouse object, the `Photograph` object uses mostly primitive and standard-library types to describe its attributes.

These attributes include:

- Strings to describe the Flickr username of the person who uploaded a photograph to Flickr, the URL to the website where the photograph resides, and the location within the Android device’s internal storage where other activities can locate the downloaded version of the photograph;
- A long integer representing the unique ID that Flickr’s servers have provided to a specific photograph;<sup>19</sup> and
- An integer representing the ID that the application has assigned a photograph that appears as a project resource. Note that this “resource ID” can – and usually does – differ from the “Flickr ID” of the same photograph.

Only the photograph’s license type appears within a special type, `CreativeCommonsLicenseType`. This enumeration associates the license type “code” that appears within Flickr’s API responses with the actual Creative Commons license to which it refers. By using this enumeration, I could create two methods within the `Photograph` object – `getLicenseType()` and `getLicenseTypeAbbreviation()` – to convert the enumerated license type into a descriptive string or an abbreviated version of this string, respectively. The output from these special accessor methods appears within the “Information” and “Photograph” screens of the application. For more information on Flickr’s integer-to-license-type mapping, see section 4.5.1, “Structuring Calls to Flickr’s API to Obtain Geotagged Images.”

Since some photographs already exist as image files on the device when users install the application, not all `Photograph` objects contain the same in-memory representation; as a result, I overloaded the object’s constructor to handle each type of photograph that the application features. I include a “base” constructor that contains all six object attributes as input parameters. All other constructors call this constructor implicitly. For photographs from Flickr that reside within the application project’s resource files, which users can access immediately after installing Lighthouse Navigator, I create a constructor that does not include parameters for the photograph’s source website, since its Flickr ID provides enough information, and for the photograph’s location within the device’s internal storage, since the photograph does not appear

---

<sup>19</sup> Integers in Java can contain only 9 base-ten digits; since some Flickr IDs contain more, I needed to use the `long` type instead.

there. After downloading photographs from Flickr within the application, I initialize the associated `Photograph` objects with a constructor with every field except the source website as the input parameters. As with the photographs that reside within the device's resource files, these "downloaded" images contain a Flickr ID, which serves as a unique reference to that photograph. Finally, the historical photographs need references only to the image's resource ID, the URL of the website where it appears, and the name of the owner (Jeremy D'Entremont in all cases). As a result, the constructor for creating historical photographs contains only those three fields as input parameters. I associate the photograph with its original website to provide appropriate attribution to the image when it appears within the "History" screen of my application.

#### 4.4.3. Review Data Structure Organizes Visitors' Experiences of Lighthouses

The `Review` object features information about user reviews from a particular travel experience website. I use only primitive types and a simple enumeration to express this review metadata. I use a pair of strings to represent the name of the attraction listed on the travel experience website that features the lighthouse as well as a code that uniquely identifies the attraction (used on TripAdvisor only). I also include the average rating for the lighthouse-based attraction inside a double-precision floating-point integer to account for half-points in some of the scores, and I place the number of reviews that users have uploaded in an integer. Finally, I present the source of the travel experience website within an enumeration called `ReviewSource`, which represents either "Yelp" or "TripAdvisor" for this version of the application.

The string-based values within the `Review` object help me construct the URLs that users see on the "Reviews" screen within the application. Table 4-2 on the following page shows the structure of these URLs, which I construct in the `generateURL()` method of the `Review` class:

Table 4-2. Structure of URLs for Travel Experience Websites Featuring Lighthouses.

Travel Experience Website	URL Structure	Example
<b>Yelp</b>	"http://m.yelp.com/biz/<attraction-name>"	http://m.yelp.com/biz/pemaquid-point-lighthouse-bristol
<b>TripAdvisor</b>	"http://www.tripadvisor.com/Attraction_Review-<attraction-code>-Reviews-<attraction-name>.html"	http://www.tripadvisor.com/Attraction_Review-g40533-d1102612-Reviews-Pemaquid_Point_Lighthouse-Bristol_Maine.html

Within the “Reviews” screen, users see these websites as links, allowing them to access them within the browser application on their respective devices to learn more about visitors’ experiences at a particular lighthouse.

#### 4.4.4. Enumerations Give Meaning to Lists within Application

The Lighthouse Navigator contains several sets of categorical information; since Java allows programmers to represent each element in this type of set as an integer with a symbolic name, I incorporate this feature into as many of these sets as possible. The symbolic names allow me to use `switch` statements instead of the slightly less-efficient `if` statements, and by using these categorical values as enumerations, I generate compilation errors instead of logic errors when I misspell a category’s name, making the application more robust and reliable. I list the name, purpose, location, and elements of each enumeration within Table 4-3 on the following page.

Table 4-3. Descriptions of Enumerated Lists within Application.

Enumeration	Containing Class	Purpose	Elements (Values)*
<b>ActivityVisible</b>	ActivityVisibilityList	Indicates the name of the screen within the application where a particular photograph appears.	SEARCH_RESULTS, INFORMATION, PHOTOGRAPHS, HISTORY
<b>ContentActivityList</b>	ContentActivityList	Maps each content-based screen within the application to the index of the drop-down list that appears within the top action bar on each of the content-based screens, allowing users to switch among these screens as desired.	INFORMATION (0), PHOTOGRAPHS (1), HISTORY (2), REVIEWS (3)
<b>CreativeCommonsLicenseType</b>	CreativeCommonsLicenseTypeList	Maps each integral code that Flickr assigns to a Creative Commons (2.0) license type to the name of that license type.	ATTRIBUTION_NONCOMMERCIAL_SHARE_ALIKE (1), ATTRIBUTION_NONCOMMERCIAL (2), ATTRIBUTION (4), ATTRIBUTION_SHARE_ALIKE (5), NO_RESTRICTIONS (7)

FlickrOutputXMLTag	FlickrParseOutputXMLThread	Contains the tag names of the elements that appear within the response from Flickr's API after requesting geotagged photograph information.	RSP("rsp"), PHOTOS ("photos"), PHOTO("photo")
<b>HistoricalArrowListenerType</b>	HistoryActivity	Lists how the "History" activity could navigate through the list of historical events after users select one of the arrows	PREVIOUS, NEXT, NO_ACTION
<b>LighthouseXMLTag</b>	LighthouseDataParser	Contains the tag names of the elements within the XML document that organizes <b>all</b> lighthouse information appearing within the application.	LIGHTHOUSES ("lighthouses"), LIGHTHOUSE ("lighthouse"), NAME ("name"), ADDRESS ("address"), STREET ("street"), CITY ("city"), STATE ("state"), COORDINATES ("coordinates"), LATITUDE ("latitude"), LONGITUDE ("longitude"), PHONE_NUMBER ("phone-number"), WEBSITE ("website"), HOURS ("hours"), FREQUENCY ("frequency"), TIME ("time"), SEASON ("season"), VISITOR_ACCESS ("visitor-access"), GROUNDS

		<pre> ("grounds"), TOWER ("tower"), OTHER("other"), YEAR_BUILT ("year-built"), HEIGHT ("height"), LIGHT_PATTERN ("light- pattern"), PHOTOGRAPHS ("photographs"), PHOTOGRAPH ("photograph"), ID ("id"), ACTIVITY ("activity"), OWNER ("owner"), LICENSE ("license"), SOURCE_WEBSITE ("source-website"), HISTORICAL_EVENTS ("historical-events"), HISTORICAL_EVENT ("historical-event"), YEAR("year"), DETAILS ("details"), REVIEW_SOURCES ("review-sources"), REVIEW_SOURCE ("review- source"), SOURCE_NAME ("source-name"), BUSINESS_NAME ("business- name"), ATTRACTION_CODE ("attraction-code"), AVG_RATING ("avg-rating"), NUM_REVIEWS ("num-reviews") </pre>
--	--	--



<b>ReviewSource</b>	Review	Represents the different travel experience websites from which I obtained lighthouse review information.	YELP("Yelp"), TRIP_ADVISOR("TripAdvisor"), NONE("")
<b>State</b>	StateList	Contains the different geographic states (within the United States) represented in the application, including the “select a state” placeholder that appears within the “Welcome” screen.	SELECT_A_STATE("Select a state"), MAINE("Maine"), NEW_HAMPSHIRE("New Hampshire"), MASSACHUSETTS("Massachusetts"), RHODE_ISLAND("Rhode Island"), CONNECTICUT("Connecticut")

\*Values indicated in parenthesis where applicable.

The “values” that appear in the right-hand column of the above table represent fields that a certain enumeration contains. This field allows me to switch between a symbolic name for an integer and the integer or string itself that the symbol represents as needed. This dual expression of the enumerations’ elements allows me to update an enumeration assignment more easily, particularly when processing XML documents.

In addition to using the above enumerations, I create a “fake” enumeration for use primarily within the `FlickrDownloadHandler` class. This “set of elements” assigns a descriptive symbol to integer codes of messages that the handler receives. Based on the code of the message, the handler can determine whether the application has finished downloading or parsing the XML document containing the Flickr API response. A more detailed discussion of the asynchronous downloading of Flickr photograph data appears in section 4.5.2, “Creating Background Threads to Complete Photograph Downloading Operation.”

#### 4.4.5. Creating Custom User Interface Objects to Encapsulate Lighthouse Information

While the custom data structures described in the above section present collections of information about lighthouses in a form that the application to process, these text-based structures would overwhelm users accustomed to a substantial amount of graphics within a smartphone screen’s layout. To address this need for a more graphically-friendly user interface, I created two adapter classes that convert the text-based information into user interface units that contain human-friendly graphics and layouts. These classes include the `LighthouseImageAdapter` class that converts a list of photograph details into a gallery of images within the “Photographs” activity and the `ReviewItemAdapter` class that arranges review details from a particular website into a standardized layout structure within the “Reviews” activity. I describe both of these classes in more detail within the sub-sections of this section.

#### 4.4.6. Converting a List of Photograph Data into a Collection of Images

The `LighthouseImageAdapter` class converts an array of photograph data into a user interface element comprising the images that the data represent. The class keeps close track of the photographs’ origin – whether they represent downloaded images from Flickr or static resource images – in order to render the photographs properly both initially and after users interact with the gallery’s photographs regardless of whether the smartphone possesses an

Internet connection. The class's constructor takes in not only the application context and the list of photographs to convert to images, but it also accepts a Boolean value indicating whether the application has downloaded the list of photographs from Flickr. If the smartphone running the application is connected to the Internet, then the photograph data does indeed refer to Flickr photographs, and the adapter should display the images stored in the lighthouse photograph cache. Otherwise, the adapter should display the thumbnails that appear within the `/res/drawable/` folder of the project. This Internet connectivity distinction determines the method used to display the image itself within the `getView()` method of the class. The application renders photographs downloaded from Flickr and stored in the smartphone's internal storage using the `setImageDrawable()` method while it uses `setImageResource()` instead to display images from the project's resource files when in offline situations. The adapter class also keeps track of the correspondence between the index of a photograph within the data array and the contents of the photograph itself. This connection between data structures allows the `GridView` object within the "Photographs" activity to display the correct image in the top-left corner of the screen after users select a specific one from the gallery on the right-hand side of that screen.

Besides displaying the photographs correctly, this class adds an aesthetic touch to create eye-pleasing whitespace around each image. In particular, I added a vertical padding of 10 density pixels and a horizontal padding of 5 density pixels to each photograph. Therefore, consecutive photographs within the gallery contained a total of 20 density pixels of padding, and the gallery itself contained a spacing of 5 density pixels between it and the content on the left-hand side of the "Photographs" screen.

#### 4.4.7. Displaying Review Details as a Single User Interface Unit

The `ReviewItemAdapter` class encapsulates the review metadata about a specific lighthouse from a certain website – such as the average review score and number of reviewers for Pemaquid Point Lighthouse on Yelp – into a single user interface object. This object in turn contains embedded groups of user interface objects as well as a connection with a layout structure defined in an XML file. The adapter's constructor references a resource ID. Unlike the photograph image adapter, this adapter associates its layout with a relative layout defined in the `/res/layout/` folder of the project. This XML resource file presents the structure for each set of user interface objects corresponding to a particular review source. The adapter can use the

names given to user interface elements within this XML file to access the elements directly, modifying these elements' contents according to the details within the currently active `Review` object. The top-most element of the “review source” list item contains a `TextView` object indicating the website from which I originally retrieved the reviews. This element features a compound drawable, an Android construct that allows me to attach the appropriate logo directly to the `TextView` itself. To ensure that the logo does not appear distorted, I first create a container for the logo that matches the size of the logo itself before adding this container as a compound drawable to the `TextView` object. To make the adapter class more maintainable, I assign constant integer variables to the dimensions of the Yelp and TripAdvisor logos that this class adds to the “Reviews” screen. Finally, I make sure to display the proper noun plurality within the `TextView` object that shows the number of reviews that form the average score. If a particular reviews website features only one review, then the text should say, “Based on 1 review,” not “Based on 1 reviews.”

#### 4.5. Using Flickr’s API and Android Multithreading to Download Photographs

Flickr offers an API for identifying geotagged photographs from a specified location and including a certain subject, such as lighthouses. I used this API within my application to identify and organize a set of images containing a specific lighthouse. Once I found the photographs that Flickr makes available for use within my application, I downloaded them to an Android device’s internal storage, where the application’s image adapter classes can easily find them for rendering. In order to complete such a complex, network-intensive task, however, I needed to execute a series of background tasks that run asynchronously relative to the application’s main thread with which the user interacts directly. In particular, I created a handler object to ensure that the application could download and process the correct photograph information from Flickr in the proper order. I describe the Flickr API protocol as well as the interdependence of the classes providing background threading within this section.

##### 4.5.1. Structuring Calls to Flickr’s API to Obtain Geotagged Images

When my application detects an Internet connection, it downloads geotagged, Creative Commons-licensed images of a lighthouse when users elect to view the “Photographs” screen for that lighthouse. I constructed a detailed request to Flickr’s API to select and download the correct set of photographs for a given lighthouse. The request uses Flickr’s `flickr.photos.search` API method to find geotagged photographs of lighthouses that contain a Creative Commons

license, rather than a full copyright restriction. Within the `FlickrPhotoDownloadThread` class, I assemble the URL to pass to Flickr's APIs. Figure 4-1 on the following page illustrates the call itself and describes the constituent components of the request. Once Flickr's servers process this request for photographs, it responds with an XML file containing information about each photograph satisfying the API request. An excerpt of a sample response from this API call appears in Figure 4-2 on the page following Figure 4-1.

#### 4.5.2. Creating Background Threads to Complete Photograph Downloading Operation

Since Android applications do not allow network access to occur on the main user interface thread and the platform's design guidelines strongly discourage programmers from introducing long-lasting blocking operations on this thread, I completed the Flickr photograph downloading and information parsing as two separate background, asynchronous threads. I created classes encapsulating the background downloading and processing functionality and named these files `FlickrPhotoDownloadThread` and `FlickrParseOutputXMLThread`, respectively. I could not use Android's built-in `AsyncTask` handler since the important progress dialog updates occur during events within the parsing thread, and `AsyncTask` objects cannot easily present updates on a background thread that a different background thread starts.

The application logic for executing these background threads begins in the `PhotographsActivity` class. First, the activity checks to see if the user is connected to the Internet or if the lighthouse being examined already exists within the photograph cache (discussed within section 4.6.2, "Adding a Cache to Preserve Existing Photographs for One Lighthouse"); if either condition is true, then the activity skips the asynchronous processing tasks completely as the application no longer needs to complete them. Otherwise, however, the activity checks to see if the downloading thread has already begun; if not, it loads a `ProgressDialog` user interface element within the "Photographs" screen, alerting users that the application has begun downloading images and their respective metadata from Flickr's servers into the device's internal storage. Finally, the activity creates the handler used to manage the background threads and allows the downloading thread to begin execution in the background.

### Request to Flickr's API

```
http://ycpi.api.flickr.com/services/rest/?
method=flickr.photos.search&api_key=<my-API-key>&
text=lighthouse&license=1,2,4,5,7&content_type=1&
has_geo&lat=<lighthouse-latitude>&lon=<lighthouse-
longitude>&radius=1&extras=url_z,owner_name,license
```

### Explanation of Request Query String

- `text = lighthouse` – Search for photographs that contain the word “lighthouse” within the title, description, and/or tags;
- `license=1,2,4,5,7` – Find photographs that contain only Creative Commons license that allow me to share the images freely with attribution, or photographs that have no restrictions at all (type 7);
- `content_type=1` – The response from Flickr should contain only photographs (no videos);
- `has_geo` – Implicitly set to true. Each photograph should be geotagged; `lat` and `lon` – The geographic coordinates at the center of the search circle for geotagged photographs, set to the location of the lighthouse tower within the Lighthouse object (discussed within section `\* MERGEFORMAT`)

```
radius=1
```

```
extras=url_z,owner_name,license
```

Figure 4-1. Structure of query to Flickr's API for retrieving geotagged, Creative Commons-licensed photographs of lighthouses.

### Response within Flickr's API

```
<rsp stat="ok">
  <photos page="1" pages="1" perpage="250" total="244">
    <photo id="7261109726" owner="7327243@N05"
      secret="12b7500630" server="7235" farm="8"
      title="Pemaquid Point Lighthouse - Maine"
      ispublic="1" isfriend="0" isfamily="0" license="5"
      ownername="Doughtone" url_z="http://
      farm8.staticflickr.com/7235/
      7261109726_12b7500630_z.jpg" height_z="640"
      width_z="480"/>
    [243 results omitted]
  </photos>
</rsp>
```

### Explanation of Response XML Elements

- `rsp stat="ok"` – The API responded with an “OK” code;
- `photos total="244"` – 244 photographs stored on Flickr’s servers satisfy the request criteria; and
- `photo id="7261109726" ... license="5" ownername="Doughtone" url_z="http://farm8.staticflickr.com/7235/7261109726_12b7500630_z.jpg" height_z="640" width_z="480"/>` – The first photograph in the results list contains a

Figure 4-2. Excerpt of sample geotagged photograph response from Flickr’s API.

While the `PhotographsActivity` class starts the background threads’ execution, most of the threading logic takes place within the `FlickrDownloadHandler` class, which manages thread execution and message handling. In particular, this handler ensures that the `FlickrParseOutputXMLThread` class does not begin execution until `FlickrPhotoDownloadThread` class has finished writing to the local `output.xml` file, which contains the response from Flickr’s API (as described in the previous sub-section). This

serialized threading procedure, while unconventional, ensures that the XML parsing thread processes the information related to the correct lighthouse, avoiding data inconsistencies within the application. The handler also receives a progress message from a `FlickrPhotoDownloadThread` object each time it finishes processing a `START_TAG` event for a `PHOTO` element within the Flickr response XML file. Once the handler receives this message, it relays it to the `ProgressDialog` object created within the `PhotographsActivity` class, allowing the text on this dialog to update and inform the user that the application has begun processing metadata for another photograph of the lighthouse. Once the application has finished parsing Flickr's XML response, the handler deletes the existing collection of `Photograph` objects, which refer to images stored in the application's resource files, and loads metadata for new `Photograph` objects from the photograph cache within the device's internal storage. This deletion process prevents the images that come installed with the application from appearing twice when users download images from Flickr. Finally, the handler restarts the `PhotographsActivity` in order for the application to render the newly downloaded images.

The background thread handler's features allow the threads themselves to contain relatively straightforward functionality, making these hard-to-debug threads less error-prone. The `FlickrPhotoDownloadThread` initiates the request to Flickr's API to locate relevant photographs on the website's servers – as described in section 4.5.1, “Structuring Calls to Flickr's API to Obtain Geotagged Images” – then copies the XML file that Flickr provides as a response to the device's internal storage using the Apache Commons IO package, as discussed in section 4.3.3, “File Utilities from Apache Commons IO Drastically Simplify Extracting URL Contents.” In order to prevent the parser thread from processing the wrong Flickr response, this downloading thread deletes the existing `output.xml` file and creates a new one with up-to-date information from Flickr's servers. Once the downloading thread has finished executing, the `FlickrParseOutputXMLThread` uses the `XmlPullParser` interface (described in section 4.3.2, “XML Pull Parser Offers Event-Driven Processing of Lighthouse Data”) to process the elements within Flickr's response XML file one element at a time. In particular, after this thread processes each of the attributes of a `PHOTO` element within the file, it uses the Apache Commons IO package to transfer the contents of the image itself to the device's internal storage directly, then notifies the `FlickrDownloadHandler` class that the application has finished processing another photograph and should update the `ProgressDialog` object within the



`PhotographsActivity` class accordingly. The orchestrated execution among these interdependent data-processing classes makes the application more responsive from users' perspectives as they witness the `ProgressDialog` updating on a regular basis. Users can even leave the "Photographs" screen and examine other pieces of information about the lighthouse while the downloading and parsing threads continue executing.

#### 4.6. Creating a Flexible Back End: XML File and Photograph Cache

The application requires persistent storage of lighthouse information to achieve consistency between use sessions as well as temporary storage of photographs from Flickr to offer flexibility for users lacking a consistent Internet connection. I created an XML file to store the different pieces of information about each lighthouse within the application; this semantically-rich document allowed me to update and add fields to existing elements of lighthouse data as necessary. With the application's photograph cache, users can view photographs for a particular lighthouse from Flickr multiple times consecutively without needing to download these images repeatedly, saving valuable processing time and reducing the need to rely on Internet connectivity for application functionality.

##### 4.6.1. Creating an XML File Structure that Allows for Simple Additions and Updates

My application project contains an XML file called `lighthousedata.xml`, which stores lighthouse information and metadata that the application uses. The document simplifies the information updating process and offers a convenient connection to the in-memory data structures with which the document and application interact. I created a `<lighthouse>` element within the document to represent each monument featured within the application and contain all relevant pieces of information about a particular lighthouse. By placing data about different lighthouses within separate XML elements, I can update the application to support new lighthouses without disturbing the manner in which the `LighthouseDataParser` class (discussed in section 4.3.2, "XML Pull Parser Offers Event-Driven Processing of Lighthouse Data") interacts with information regarding existing lighthouses. Within each `<lighthouse>` element, I created an element hierarchy for a particular lighthouse's facts. This hierarchy's structure closely mirrors the in-memory data structures that I created for storing lighthouse information as the application executes. (More detailed descriptions of these custom data structures appear in section 4.4, "Creating New Data Structures for Organizing Run-Time Application Data.") Furthermore, these inner elements' names match (or nearly match) the

names of the fields within the Lighthouse, Photograph, and Review classes. This correspondence between XML document contents and application class hierarchy allows me to change a piece of information for a given lighthouse with the confidence that the application’s screens will present this change correctly. Whenever I add an element representing a new piece of lighthouse information, I simply need to:

1. Create a new entry within the `LighthouseXMLTag` enumeration;
2. Add a field to the data structure corresponding with the new element’s parent element in the XML document, such as `Lighthouse`; and
3. Assign the contents of the new element (from step 1) to the new field of the data structure (from step 2) within the `LighthouseDataParser` class.

This simple updating process allows me to change the types of information that the application displays to users without having to complete an extensive refactoring process.

#### 4.6.2. Adding a Cache to Preserve Existing Photographs for One Lighthouse

To allow users of my application to access photographs more quickly and reliably, I created a cache for images that the application downloads from Flickr’s servers into the device’s internal storage. This cache stores photographs from Flickr featuring a single lighthouse, allowing the application to display these images instantly – without having to download them from Flickr again – upon subsequent requests to view these images. Therefore, users can search for a lighthouse they wish to visit and download the photographs of that lighthouse from Flickr while at home or in a location with a strong Internet connection, then view these images at the lighthouse itself, where a network connection may be unavailable.

The application uses a simple file transfer method and a parametric file naming system to create a cache quickly and preserve the images’ metadata. After the `FlickrParseOutputXMLThread` processes the attributes for a particular photograph from Flickr, the object transfers the image itself from Flickr to the cache using the Apache Commons IO package (described in section 4.3.3, “File Utilities from Apache Commons IO Drastically Simplify Extracting URL Contents”). The file name for the cached photograph contains its metadata, including a “directory-friendly” version of the lighthouse name (as described in section 4.4.1, “Lighthouse Data Structure Contains Information for Content-Based Activities”) as well as the photographer’s name with all spaces replaced with ‘~’ characters. This parametric

naming system allows other pieces of the application to access a cached image's metadata by examining a particular piece of its file name.

The application's main logic ensures that the cache's photographs appear only when appropriate, simplifying the `LighthouseImageAdapter` class that renders these images. The `PhotographsActivity` contains a Boolean method called `photoCacheExists()`, which determines whether the cache contains photographs for the lighthouse currently featured within the "Photographs" screen. This method creates the directory containing the photograph cache if it does not already exist within the device's internal storage, then checks to see if the cache is empty or contains photographs depicting a different lighthouse. If either of these two cases is true, then the method concludes that the lighthouse's photographs do *not* appear in the cache and instructs the application to call Flickr's API and download the photographs from the Internet, as described in section 4.5, "Using Flickr's API and Android Multithreading to Download Photographs." The `LighthouseImageAdapter` class renders a lighthouse's images within the "Photographs" screen after the application checks the presence of a cache and completes any necessary updates, so the adapter can access and display images from this cache whenever the application detects an Internet connection.

#### 4.7. Constructing Image Processing Algorithms with ImageJ

Some users of my application might enjoy viewing the entire gallery of photographs available from Flickr's servers, but others may want to view photographs that satisfy a particular criterion, such as those containing a lighthouse tower or those taken on a sunny day. I addressed this functionality request by creating three different image-processing algorithms. The `ImageClassifier` class assigns an image to a "stereotype" category, such as sunny or cloudy weather conditions. The `FeatureColorMatcher` algorithm identifies the presence of a specific range of grayscale intensity values within a photograph. Finally, the `FeatureShapeMatcher` class uses distance transform functions and chamfer match scores to determine the presence of a particular shape feature within an image. Before I describe each of these algorithms, however, I discuss an unfortunate situation regarding my inability to integrate these filters directly into my application and offer a high-level overview of the filtering project's structure.

##### 4.7.1. Discovering inability to Use ImageJ Directly with Android

A few weeks before I completed this project, I discovered that ImageJ's in-memory objects do not always load correctly within Android devices, preventing me from integrating the

image-processing algorithms that I discuss in this section into my application. ImageJ's base image-processing classes use Java's built-in `java.awt.Image` objects for organizing different graphics files. Android devices, on the other hand, store images as resource files and render them as `View` objects. Since these two representations of images contain some incompatibilities, it is currently impossible to embed native ImageJ code within Android projects.

Several solutions exist for mitigating this integration issue, but each has sizeable drawbacks. I could modify the lower-level classes of ImageJ to reference Android-compatible representations of images, allowing the device to access these objects efficiently. This process would take a considerable amount of time, however, as I would need to learn about the operations ImageJ uses to convert image bytes into Java objects and change this conversion process to a more Android-friendly one. I could also use different image-processing software, such as OpenCV for Android, which has already addressed the underlying issues with combining Java- and Android-based image objects. OpenCV uses C++, however, so I would need to make sure that I incorporated features that Android's Java-based libraries support, as well. Finally, I could implement a client-server model where my application runs on a client Android devices and contacts a server whenever users wish to complete photograph filtering operations. The major drawback with this solution involves the dependence on an Internet connection. While using the application in an offline setting, users could no longer complete filtering operations, hindering their ability to customize the visual context of the lighthouse they wish to explore within the application.

#### 4.7.2. Describing the Structure of the Image Processing Java Project

The image-processing filters that I created appear under the base package name `edu.wpi.khufnagle.ij` within a project entitled "ImageJProcessor." As with my Android application, I divide the classes used to support the different types of filters into different packages, each representing a different category. These packages include:

- *Algos* – Contains the main logic for completing the three different image-processing algorithms that I developed for this project;
- *Drivers* – Contains "test" programs that allow me to evaluate the algorithms' accuracy and execution time;
- *Error* – Contains custom Exception classes for handling run-time errors that occur from misusing the algorithm-based classes and their associated utility classes; and

- *Util* – Contains utility classes for completing lower-level operations within the different algorithms as well as classes representing in-memory data structures. I discuss these utility classes with the algorithms they support in the following sub-sections.

#### 4.7.3. Classifying Images based on Stereotypical Sunny, Cloudy Weather Conditions

`ImageClassifier` objects determine the relative similarity between two images by comparing their cumulative distributions of intensity values and determining the number of pixels that would need to change intensity values within one image to cause these two cumulative distributions to (nearly) match. The `ImageClassifier` constructor accepts a “test” image of unknown contents and a “reference” image with a known category. The algorithm then constructs two cumulative histograms, one for the test image and the other for the reference image. The y-axis value of each intensity value within these cumulative histograms indicates the proportion of pixels within the entire image that contain the given intensity value or less (darker). After creating the two histograms, I loop through each intensity value within the test histogram. For each value  $I_T$ , I determine the reference intensity value  $I_R$  such that the cumulative histogram function ( $cdf()$ ) of the test function marginally exceeds that of the reference image. That is, I find the maximum value  $I_R$  such that:

$$cdf(I_T) > cdf(I_R)$$

After determining the maximal value  $I_R$  that satisfies the above equation, I create a new histogram called an “adjusted test histogram.” I add the number of pixels at intensity value  $I_T$  within the original histogram to the number of pixels at intensity value  $I_R$  within the adjusted test histogram. This assignment ensures that the adjusted test histogram and the reference histogram contain as similar a pair of cumulative distribution functions as possible and measures the amount of “effort” needed to have the contents of the test image match those of the reference image. Finally, the algorithm compares the test image’s original histogram with its adjusted histogram, counting the total number of pixels that change intensity values between the two histograms. The algorithm then divides this value by 2 (since a simple swap between two intensity values counts as two changes in the above step) and divides this halved value by the total number of pixels in the test image. This proportion, which serves as the algorithm’s return

value, represents the similarity between the test image and the reference image; smaller numbers indicate a closer match.

To make this classifying algorithm more accurate from the driver's perspective, I employ a utility class called `ImageRegionDecomposer`, which divides an image into several (nearly) equally-sized sub-images. I use ImageJ's `setRoi()` method to select a region of interest within the original image to become a sub-image, then call the library's `crop()` method. This latter method creates a new image, which contains a subset of the pixels within the original image. These new images form the original image's "sub-images." The most complex part of this class involves selecting the sub-images that will contain extra pixels when the original image does not divide perfectly evenly. To complete this operation, I use a method called `calculateSubimageLengths()`, which assigns the center sub-images with the first leftover pixel, then works to the image's sides, favoring the image's right and bottom halves. Figure 4-3 on the following page offers an example of the image division process within this method.

Original image contains a width of 103 pixels; attempting to divide image into 5 sub-images

Index	0	1	2	3	4
# Pixels	20	20	20	20	20

Evenly distribute as many pixels as possible.

Index	0	1	2	3	4
# Pixels	20	20	<b>21</b>	20	20

Assign the first leftover pixel to the center sub-image (index 2).

Index	0	1	2	3	4
# Pixels	20	20	21	<b>21</b>	20

Assign the second leftover pixel to the sub-image immediately to the right of center (index 3, or +1 relative to the previous index).

Index	0	1	2	3	4
# Pixels	20	<b>21</b>	21	21	20

Assign the final leftover pixel to the sub-image immediately to the left of center (index 1, or -2 relative to the previous index).

Figure 4-3. Sample Distribution of Pixels across Several Sub-Images.

In order to favor the image's left and top halves, instead, I simply need to change the statement:

```
if (indexToAssignNextPixel <= numSegments / 2)
```

to

```
if (indexToAssignNextPixel < numSegments / 2).
```

Within my driver class, I decompose the images and apply a set of comparisons to determine the correct classification for a test image relative to several reference images featuring “stereotype” weather conditions. I call the `ImageRegionDecomposer` class to divide the test and reference images, then compare the contents within these sub-images. This image division process ensures that an unusual section of a test image does not cause the influence the driver into categorizing the image incorrectly. If the driver classifies a majority of sub-images within a particular “stereotype” category, then I consider the image as a whole to satisfy that “stereotype.” In the case of a tie, I examine the sub-image that is most similar to one stereotype and compare this value to the similarity between another sub-image and the other stereotype. The sub-image with the most similarity – that is, the sub-image whose adjusted test histogram contains the least number of intensity value adjustments relative to its corresponding original test histogram – determines the image's classification.

#### 4.7.4. Identifying Presence and Absence of Rocks using Color-Based Feature-Matching

I created a `FeatureColorMatcher` class that determines whether an image of unknown contents contains a sizeable amount of a specific color, expressed as a grayscale intensity value. The constructor accepts a color to find within an image, the acceptable error in the color value expressed as a proportion of the intensity value spectrum (0-255), and the minimum proportion of the image that the regions need to occupy within the image needed for the algorithm to conclude that a sufficiently large part of the image contains the desired color. Once an `ImageEdgeSegmenter` object (discussed below) divides the image into edge-separated regions, the color matcher calculates the average color within each region. If the color matches the desired color to within the given error tolerance, then the color matcher adds the number of



pixels in the region to the total number of pixels within the image containing the desired color. The class then computes the proportion of the image containing the desired color (total number of pixels of regions containing the desired color as the average color divided by the number of pixels within the image). If this proportion equals or exceeds the desired “minimum image proportion” value, then the algorithm concludes that the image contains features with the desired color.

I use an optimized version of the iterative neighbor-finding algorithm to segment an image into regions separated by its edges. The `segmentImage()` method of the class examines each pixel of the image sequentially in row-major order. At each pixel location, the method determines whether the left, top, top-left, or top-right neighbors of the pixel represent edges. If all four of these pixels contain edges, then the pixel being examined becomes the first pixel of a new image region. Otherwise, the pixel becomes part of the image region of the adjacent non-edge neighboring pixel. Within this method, I use two complementary Map objects. The first data structure maps an image region number to the set of pixels that the region contains, and the other structure associates an image pixel with the image region number that the class assigns to the pixel. This second Map object allows the method to determine the region number of a neighboring pixel in  $O(1)$  time, rather than  $O(n)$  time, significantly speeding up the image segmentation process. Unlike many examples of this iterative edge-find algorithm in the literature, this implementation completes only one pass through the image. As a result, the class treats different portions of a connected area as different regions. This optimization does not affect the color-matching algorithm, however, since it takes into account the *total* number of pixels containing the desired color. If the different sections of a connection component each contain an average color matching the desired one, the color-matching algorithm will detect this match regardless of whether the component appears as one region or several distinct ones within the “image segments” data structure.

The driver that tests this algorithm, which attempts to locate the presence of rock-colored features within lighthouse photographs, offers several parameters that I can adjust to affect the algorithm’s performance. These variables include:

- The RGB values of the color to detect within the image, which the `FeatureColorMatcher` class then converts to a grayscale intensity value;

- The color error tolerance that allows the algorithm to “count” near-matches as regions containing the desired color; and
- The minimum proportion of the image that needs to include the desired color – or one close to it – for the algorithm to consider the color as present within the image.

Section 5.1.2, “Color-Based Feature Matching Requires Delicate Balance of Parameters,” summarizes how changes to these parameters affect the algorithm’s accuracy.

#### 4.7.5. Determining Presence or Absence of Lighthouses using Shape-Based Feature-Matching

The `FeatureShapeMatcher` class detects the presence of a known feature within a test image of unknown contents. The constructor for the class accepts a test image of unknown contents, a reference “template” image that contains an area significantly smaller than that of the test image, and a value indicating the highest acceptable “chamfer match score ratio” (I describe this metric below) for the algorithm to conclude that the photograph features a lighthouse. The class’s `computeDistanceTransformValue()` method begins by completing two sequential passes through the image, once from top-left to bottom-right and another time from bottom-right to top-left. During both passes, the algorithm processes the pixels in row-major order. For each pixel, the method first assigns a ridiculously high distance transform value to the pixel (`Integer.MAX_VALUE`). The algorithm then determines whether the pixel contains a foreground color; if so, the pixel’s distance transform value becomes 0. Otherwise, the method checks the distance transform values of the pixel’s neighbors that the “pass” has already processed; for example, during the forward pass, the method checks the left, top, top-left, and top-right neighbors. The algorithm then calculates the sum of the neighbor’s distance transform value and the “cost” of reaching this neighbor from the current pixel. For this algorithm, I use distance costs of 3 for one city block (horizontal and vertical) and 4 for two city blocks (diagonal). If the minimum sum among the pixel’s neighbors is less than the pixel’s current distance transform value, the pixel’s value changes to match this “neighbor sum.” Once the method has assigned the proper distance transform value to each pixel within the image, the algorithm overlays the reference template image atop the test image at every possible location. For each overlay position, the algorithm determines the distance transform value of each pixel within the test image that lies “underneath” a foreground pixel within the reference image. The sum of these distance transform values, known as the *chamfer match score*, determines the relative similarity

between the reference template image and a specific sub-image of the test image. The algorithm locates the sub-image within the test image where this chamfer match score reaches a minimum, indicating a very close match. It then divides this match score into the maximum possible match score for that overlay position<sup>20</sup> and determines whether this quotient is less than or equal to the maximum allowed chamfer match score ratio. If the algorithm calculates a small enough actual ratio, it concludes that the image contains the feature depicted within the reference template image.




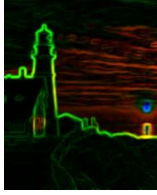


The shape-based feature-matching algorithm performs best when examining binary images, so I apply a series of transformations that exaggerate the edges within the test image and maximize the chances that the algorithm will find the presence of the desired feature when appropriate. Table 4-4 on the following page illustrates the process that I perform within the `LighthouseFeatureFinder` utility class to prepare the test image for feature-based comparison.

The transformed images depends only on the image's original contrast, not the colors it contains. Therefore, the transformation works equally well on light features against a darker background and dark features within a brighter scene.

---

<sup>20</sup> For a given test-reference image overlay, the chamfer match score reaches a maximum when one image contains one foreground pixel at its center and the other image contains one foreground pixel in a corner. In order to reach this corner, the reference image must “walk” diagonally  $n$  steps to an edge of the test sub-image, then follow that edge to the corner by taking another  $m$  steps. The distance transform value and the chamfer match score then become  $4n + 3m$ .

Table 4-4. Process of Transforming Test Image used in Shape-Based Feature-Matching Algorithm.

Step	Image	Description
0		Original test image.
1		Map the intensity range [175, 255] to [0, 255] for the red and blue channels.
2		Perform a $3 \times 3$ neighborhood smoothing twice.
3		Apply ImageJ's built-in Sobel edge-detection algorithm.
4		Convert the image to grayscale.
5		Assign the 90th percentile intensity values and above to 255 (increase contrast).

6		<p>Clamp all intensity value of 254 or less to 0.</p> <p>Test image now ready for comparison.</p>
---	---	---

#### 4.8. Preparing Image-Processing Algorithms for Testing

For this project, I wished to investigate the performance of the image-processing algorithms that I created and described in section 4.7, “Constructing Image Processing Algorithms with ImageJ.” In order to run as controlled of an experiment as possible, I sampled carefully a set of geotagged images of New England lighthouses as “training galleries” for the algorithms, fixed my system’s conditions as much as possible while executing the algorithms, and used simple, consistent metrics for measuring the algorithms’ accuracy and speed.

##### 4.8.1. Selecting Training Data for Algorithms

Since I plan on filtering images from my application with these image-processing algorithms, I used the set of all 1,080 geotagged images of New England lighthouses (as of February 2014) as my training data. For each algorithm, I created three different random samples of 50 images from this training set and applied the algorithm to each sample. Note that, since I completed the sampling events independently of one another, some images appear in multiple samples.

##### 4.8.2. Creating an Environment for Conducting Testing

As I tested the image-processing algorithms, I created as consistent a set of system environment conditions as possible. For testing, I used my Dell Studio 1737 laptop, which contains 3.99 GiB<sup>21</sup> of RAM and a dual-core, 2.66 GHz Intel® Core™ 2 Duo CPU T9550. I created and ran the programs within Eclipse Kepler Service Release 2 (build ID 20140224-0627). The contents of the training data folder did not change for the duration of the tests, and I closed all programs except for Eclipse and Microsoft Excel when running the image-processing programs.

<sup>21</sup> 1 GiB, a “gibibyte,” equals  $2^{30}$  bytes (as opposed to 1 GB, or “gigabyte,” which represents 1 billion bytes).

#### 4.8.3. Measuring Image-Processing Algorithm Effectiveness: Speed and Accuracy

I evaluated the different filtering algorithms by measuring their classification or feature-identification accuracies as well as their execution time as measures of algorithm performance and speed. In order to determine each algorithm's accuracy, I classified the different samples of training data by hand to develop a set of "correct" answers for the algorithms to find. I then compared the algorithms' classifications or feature recognition results with my own for each image to determine the methods' accuracies. For measuring execution time, I used Java's built-in `currentTimeMillis()` method. I called this method immediately before the algorithm began processing the first image – after the driver class set parametric variables and initialized training image sample IDs – and again after the program computed the algorithm's accuracy. I took the difference between these two timestamps to determine the number of seconds, to two decimal places, that elapsed as the algorithm completed execution. Since these algorithms require at least 250 milliseconds to process each training image sample, I determined that I did not need a higher-precision method of tracking system time.

## 5. Reflecting on Algorithm Performance and Application Usability

I ran a series of tests on the image-processing algorithms I developed for this project, evaluating each program's accuracy and speed and discovering the relationships between variations in algorithm input parameter values and these performance metrics. I also examined the improvements I made to the application's design, finding trends among pieces of feedback from the usability studies I conducted at several lighthouses across New England. This chapter presents a detailed description of these algorithm performance results and describes the three most important considerations for this application's design.

### 5.1. Findings from Image-Processing Algorithm Testing

As I implemented this project, I created three different image-processing algorithms for filtering the set of photographs that appears for a given lighthouse within the application. These programs include:

- An image classifier that determines whether an image contains sunny or cloudy skies;
- A color-based feature matcher that detects the presence of rock-colored details; and
- A shape-based feature matcher that detects the presence of lighthouse towers.

After developing these image-processing algorithms, I evaluated their performance by running the “driver” class corresponding with each algorithm, as discussed within section 4.7.2, “Describing the Structure of the Image Processing Java Project.” To complete these evaluations, I selected 50 random geotagged images of New England lighthouses from Flickr and used this set of images as input for each algorithm. I needed to check the algorithm's consistency and robustness, so I used two other sets of random geotagged images as additional input to the algorithm, averaging the results I received across all three trials. This section presents the accuracy and speed that each image-processing algorithm can achieve after tuning each program's input parameters to their respective optimal values.

#### 5.1.1. Image Classification Performs best with Moderate Image Decomposition

After running the image classification algorithm using different amounts of image decomposition and examining different “cutoff points” within these images, I discovered that a moderate amount of image decomposition offers additional accuracy and speed for this algorithm. Figure 5-1 on the page following this page illustrates the influence of image decomposition rates and cutoff amounts on the algorithm's accuracy across three separate sets of

training images. The algorithm performs fairly well under all tested conditions, easily surpassing the baseline accuracy of 50%, but it shows particularly strong results for cases involving the top half or so of the image and 4 or 5 decompositions along each axis. To help determine which set of parameters offers the optimal solution, I consulted the relationship between the number of sub-images created from the original image and the algorithm's execution time, as illustrated in Figure 5-2 on the following page. This graph shows a noticeably steep incline between 8 sub-images ( $4 \times 4$  decomposition with the top half considered) and 15 sub-images ( $5 \times 5$  decomposition with the top 60% considered). The 45% time penalty for using a  $5 \times 5$  decomposition strategy over its  $4 \times 4$  counterpart overshadows any marginal increase in accuracy that might occur. Therefore, I recommend using a  $4 \times 4$  image decomposition when classifying photographs based on weather conditions, examining only the sub-images corresponding to the top half of the original image.

This algorithm performs reasonably well given the proper input parameters, but interestingly, it could perform even better if these images did not introduce such a significant level of subjectivity into the classification process. Quite a few images within the training data sets contain sky conditions characteristic of both sunny and cloudy weather, leading to uncertain human classifications. The image-processing algorithm cannot resolve this ambiguity elegantly, either, which explains the "accuracy ceiling" of about 75% for this algorithm.



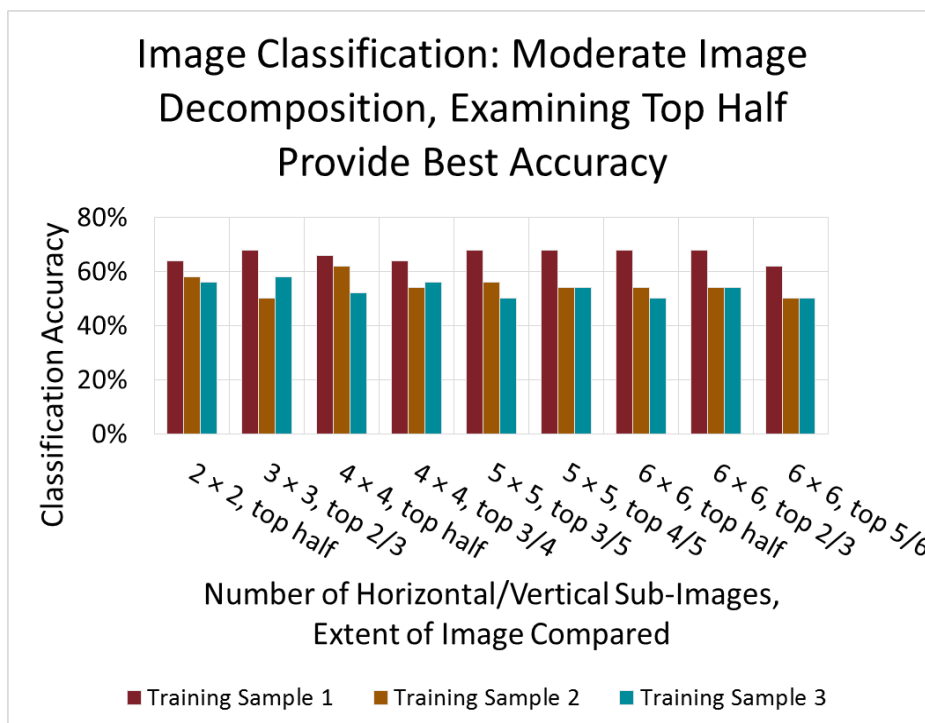


Figure 5-1. Effects of Image Decomposition, Percentage Inclusion on Algorithm

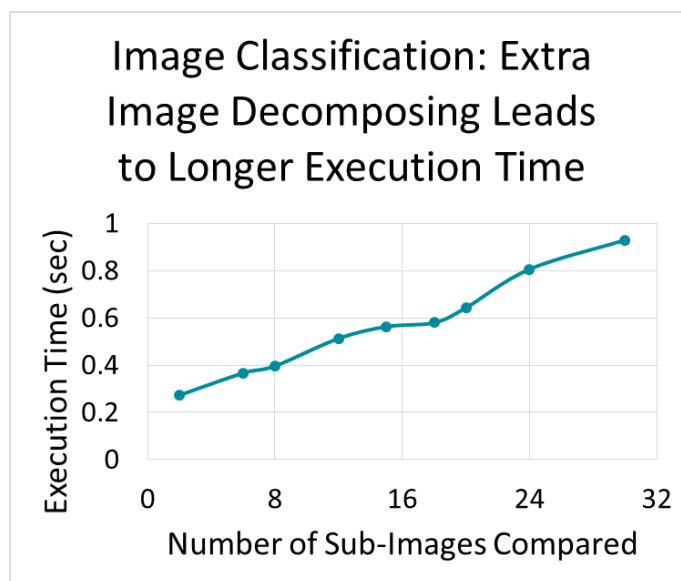


Figure 5-2. Effects of Sub-Image Complexity on Image Classification Execution Time.

### 5.1.2. Color-Based Feature Matching Requires Delicate Balance of Parameters

The color-based feature-matching algorithm contains the most complex and interdependent set of parameters, making it difficult to determine each variable's optimal value for typical cases. My strategy for finding the best set of parameters involved tuning one parameter at a time, finding the maximum possible algorithm accuracy for that parameter. As a proof of concept, this algorithm detects the presence of rocks by finding a characteristic grayscale intensity value within a given image. The sets of training images contain interestingly different proportions of photographs containing rocks, from 38% in the first set to 64% in the third set. These significant deviations likely cause the inconsistencies in algorithm performance across the different data sets.

I began evaluating the algorithm by examining the optimal grayscale intensity value for detecting rocks within images and discovered that intensity values between 40 and 70 (out of 255) yield increasingly higher detection accuracies, following by a precipitous drop in algorithm performance. As I tested different grayscale intensity values, I fixed the color difference tolerance – the proportion of the intensity value spectrum that an image region's average intensity value can differ from the desired intensity value and still consider the region a “matching segment” of the image – at 0.10, and I kept the minimum image proportion – the proportion of the image required to contain the desired intensity value (to within the color difference tolerance) for the algorithm to consider the rocks as “present” within the image – constant at 0.20. Figure 5-3 on the following page shows the relationship between intensity value and algorithm accuracy. Note that the third set of training data caused the algorithm to perform best at the maximum intensity value used for this experiment: 100. Since this set of photographs contains an unusually high percentage of “rock images,” the additional instances of those grayscale intensities within the set might have created fewer false positives on the upper end of the intensity value spectrum tested, driving the accuracy upward. Despite this potential advantage for the third data set, the algorithm still finds rocks within images most reliably when looking for a grayscale intensity value of 75.

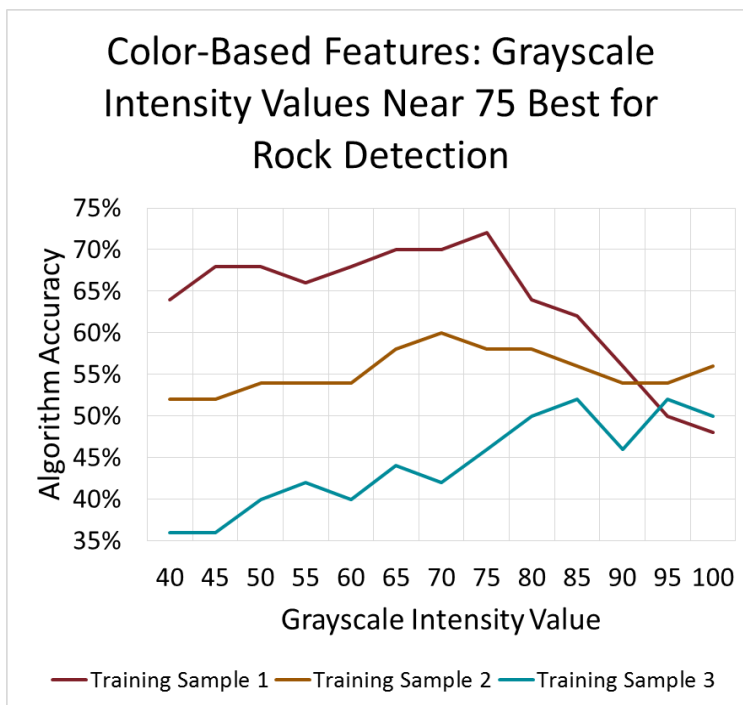


Figure 5-3. Changing Grayscale Intensity Value Affects Color-Based Feature-Matching Algorithm Accuracy.

With the grayscale intensity value now fixed at an optimal level, I then focused on the color difference tolerance, discovering that moderate values for this variable offer the best overall results. As Figure 5-5 on the following page shows, the third set of training data yielded optimal accuracy values for fairly large color difference tolerances, deviating curiously from the trend expressed in the other two sets. The better-behaved first set of training data, however, showed a relatively steep decline in algorithm accuracy once the color difference tolerance passed 0.15. Since I consider most tolerances higher than this value to be too permissive of intensity variations, anyway, I elected to choose a low-moderate value of 0.14 as the ideal tolerance margin for detecting the presence of rocks within images.

With two parameters for the color-based feature-matching algorithm now fixed at optimal values, I could turn to the final variable, the minimum proportion of the image that should contain the desired intensity value (or one close to it). Figure 5-4 on the following page shows an intriguing phenomenon; this graph presents a near-mirror image of the graphs for the other two parameters for the algorithm.

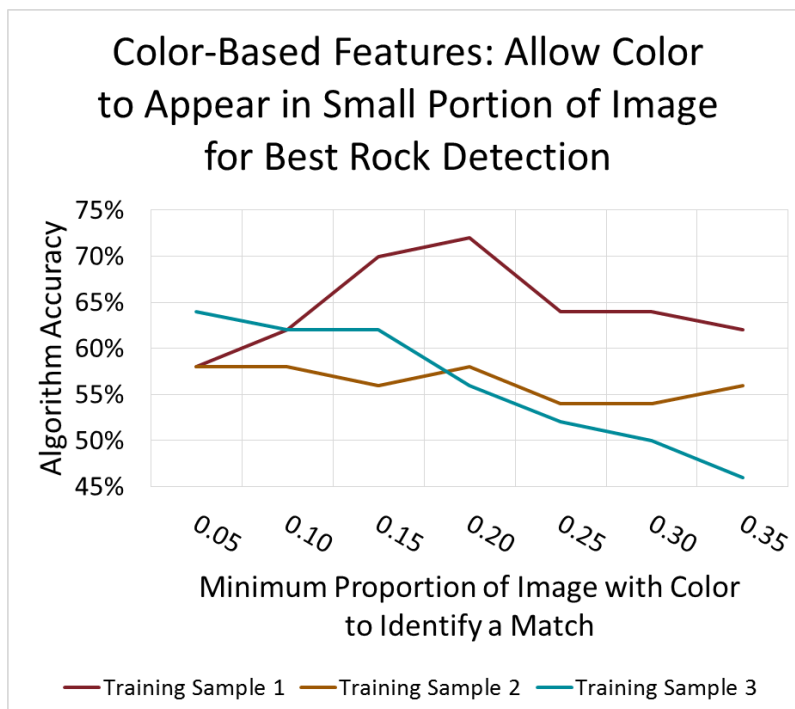


Figure 5-5. If Small Amount of Image Contains Color, Best to Consider it a Match

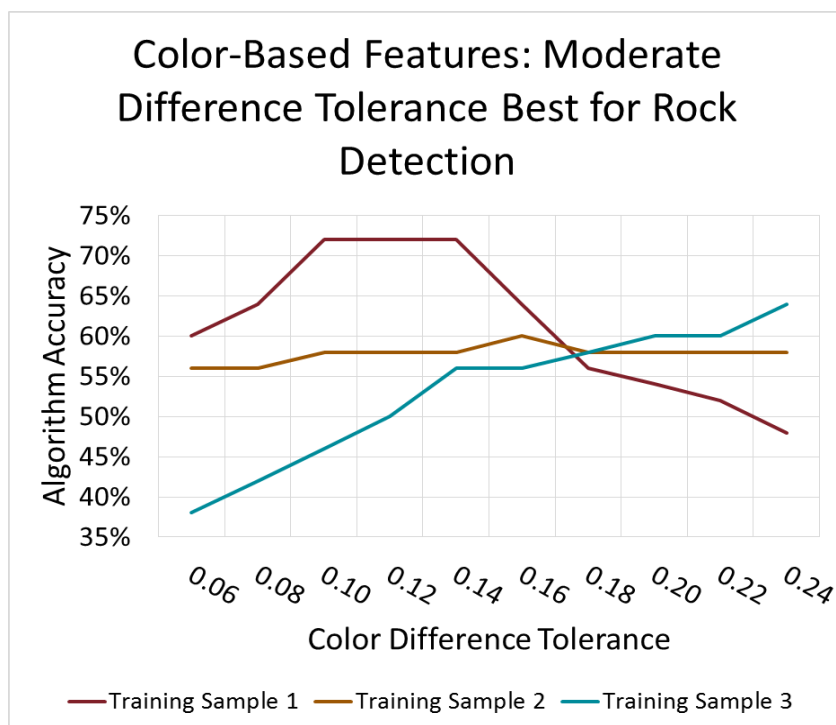


Figure 5-4. Moderate Color Difference Tolerance Best for Color-Based Feature-Matching Algorithm Accuracy.

This trend reversal could occur because the third set of training images, while plentiful in occurrences of rocks, contains mostly photographs depicting small areas of rocks. Therefore, when the algorithm applied higher “minimum image proportion” parameters on this third set, the program falsely assumed that certain images contain too few instances of rock-colored features for these structures to appear in the photographs. To mitigate this situation and minimize false positive results, I propose a fairly low minimum image proportion value, 0.15, for this type of color-matching algorithm. After completing all three parameter optimizations, I achieved a maximum average algorithm accuracy of 63%, yielding a p-value of about 0.064 since 51% of the training images contain rocks.

As I discovered the relationship between changing parameter values and accuracy for the color-based feature-matching algorithm, I also measured the program’s speed. Interestingly, this execution time – about 5 seconds for classifying 50 images – remained constant across all cases within each of the tests I administered. The third set of images yielded a marginal increase in execution time, perhaps because these images contained more complexity or a larger overall image size. Otherwise, however, I found no discernable association between changing parameter values and algorithm speed.

#### 5.1.3. Shape-Based Feature-Matching Algorithm Performs Best with Multiple Templates

Interestingly, the most complex image-processing algorithm, which detects the presence of a specific shape within images, such as lighthouse towers, produced the most consistent results. The transformations that I applied to each image, as discussed in section 4.7.5, “Determining Presence or Absence of Lighthouses using Shape-Based Feature-Matching,” likely yielded simpler, more homogeneous conditions for the algorithm, which led to the impressively consistent accuracy trends. Figure 5-6 on the following page depicts the well-formed relationship between the maximum chamfer match score – the maximum allowed deviation between the foreground pixel locations of a reference image and those of an overlaid test sub-image – and the algorithm’s accuracy when using a single reference template of medium size. In particular, the accuracy increased quite rapidly as the maximum allowed chamfer match score increased, then leveled off somewhat as the maximum score continued to increase. The algorithm’s false negative rate approached 0% for chamfer match scores of 0.007 or greater, which causes the diminishing returns on program performance. Since roughly 67% of images across the three sets of training images contain lighthouses, this algorithm also outperformed a “dummy” program

that blindly assumes the presence of a lighthouse in each photograph. A 73% average accuracy for the ideal maximum chamfer match score of 0.006 yields a p-value of about 0.23, a respectably low value for an algorithm as intricate as this one.

When using a single reference template to detect the presence of lighthouses within the various test images, the shape-based feature-matching algorithm's speed remained relatively constant. Figure 5-7 on the following page depicts this uniform result across different chamfer match scores. Note, however, that this algorithm's execution time – about 60 seconds for processing 50 images – represents more than an order-of-magnitude increase relative to the other two algorithms, indicating this program's significant complexity. The algorithm's speed remained consistently high since it needed to check the similarity between the reference template image and *every possible* contiguous sub-image of the test image that matched the reference template's size. The resulting large number of overlays – almost 20,000 for a  $20 \times 50$  reference image and a  $150 \times 200$  test image – causes the relatively slow execution speed.

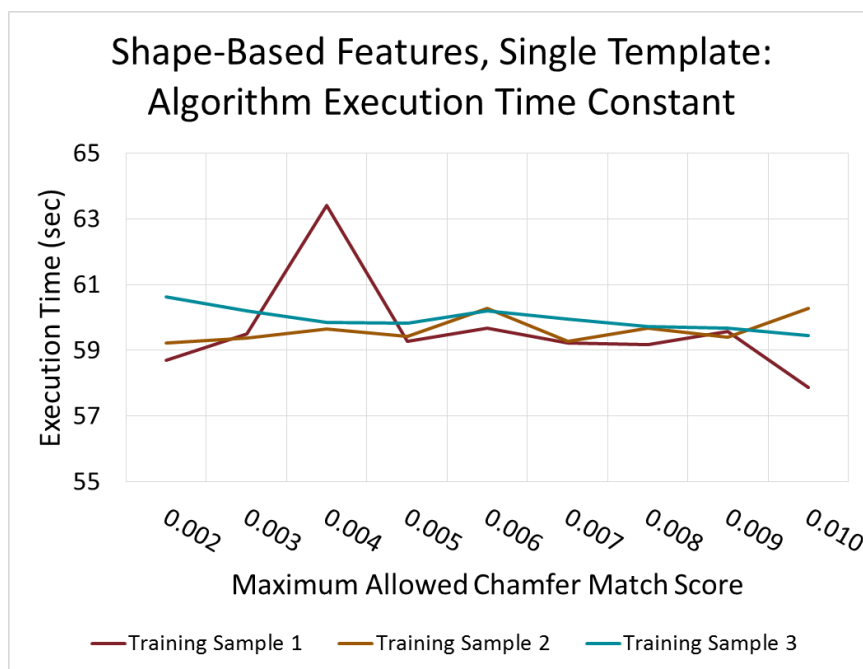


Figure 5-6. Execution Time of Shape-Based Feature-Matching Algorithm Constant when Using Single Reference Template.

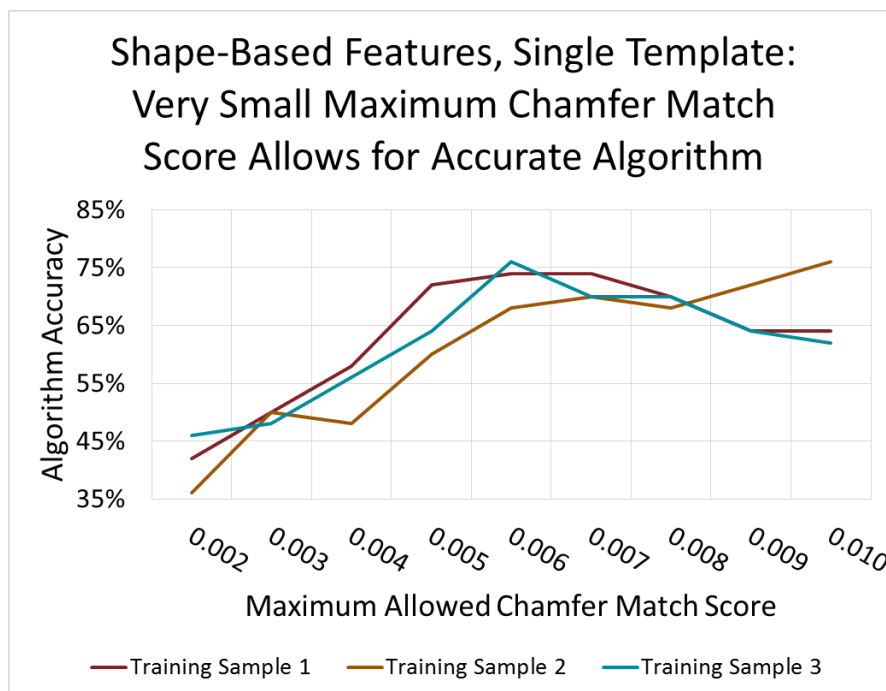


Figure 5-7. Very Small Chamfer Match Score Allowances Lead to Accurate Shape-Based Feature-Matching Algorithm.

When comparing images against two reference template images to detect a specific feature shape, the algorithm performed slightly better compared to the single-template case but required an even smaller maximum chamfer match score to achieve the optimal accuracy. Figure 5-8 on the following page shows that the multiple-template algorithm's accuracy diminished gradually as the maximum allowed chamfer match score increased beyond 0.005. At the slightly lower value of 0.003, the algorithm achieved its maximum average accuracy, 76%. The resulting p-value, about 0.11, indicates the slight increase in algorithm performance relative to using a single reference template. The increased accuracy occurred since the multiple-template version of the algorithm needs to detect the presence of a desired feature within an image using a small reference image *or* a large reference image. For images where the lighthouse consumes a significant area of the photograph, the comparison against a small reference image will fail, but the same type of comparison against a large reference image will succeed, informing the algorithm that the image does indeed contain a lighthouse. Since the algorithm requires only one successful comparison with a reference template to indicate the presence of a feature shape, the maximum allowed chamfer match score needs to decrease slightly. Otherwise, too many checks against the small reference template, which occurs first in the algorithm, would succeed, forcing

the algorithm to rely on comparisons with the large reference template to reject candidate images that contain features similar to those of a lighthouse but do not actually feature a lighthouse.

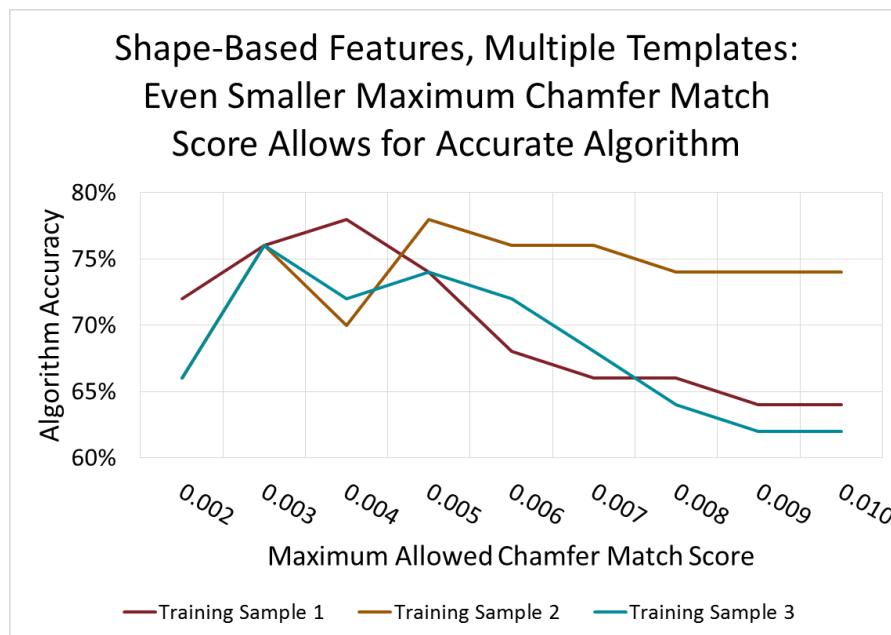


Figure 5-8. Even Smaller Chamfer Match Score Allowances Yield Accurate Shape-Based Feature-Matching Algorithm Results Given Multiple Reference Templates.

While the multiple-template version of the shape-based feature-matching algorithm yielded higher accuracies with lower maximum permissible chamfer match scores, its execution time decreased as the maximum chamfer match scores increased, creating a trade-off between speed and accuracy with this version of the algorithm. Figure 5-9 on the following page shows the gradual decline in execution time as I allowed the algorithm to accept less precise matches with the pair of reference templates.

Note too that the execution time for very low maximum allowed chamfer match scores exceeded that of the single-template variant by up to 65%. This trend in algorithm speed occurred because it uses short-circuit logic to evaluate the presence of a lighthouse within an image. In particular, if the algorithm determines that the feature depicted within the small reference template occurs within the test image, then it decides at that moment that the image contains a lighthouse, skipping the comparison against the large reference template altogether. As the maximum allowed chamfer match score increased, the algorithm performed short-circuit logic increasingly often, yielding significant time savings. Some of the speed enhancement also occurred because the small reference image takes less time to traverse than a medium-sized one,



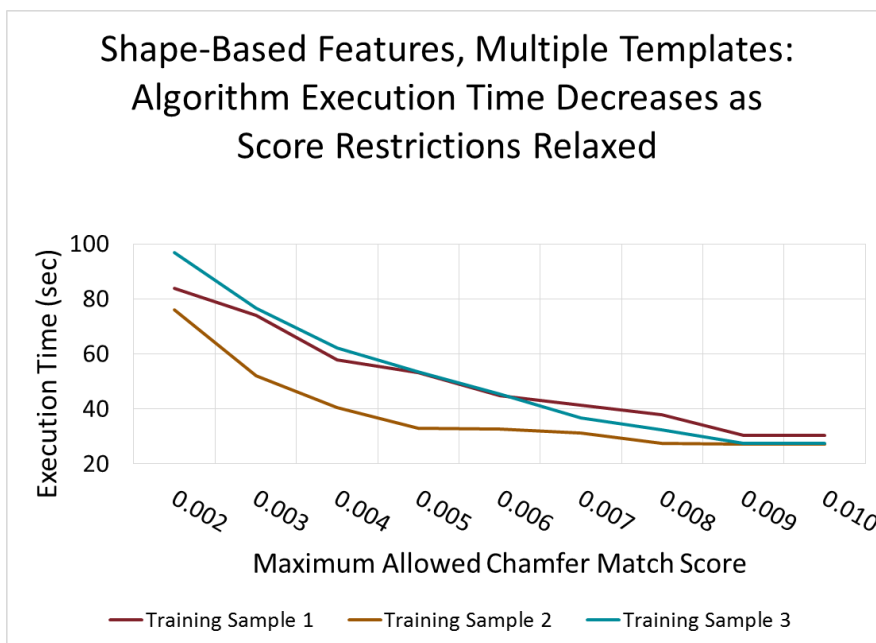


Figure 5-9. Shape-Based Feature-Matching Algorithm Using Multiple Reference Templates Takes Less Time to Execute as Chamfer Match Score Allowance Increases.

allowing the algorithm to complete all of the necessary comparisons with the test image more quickly. The smaller maximum chamfer match score values created a significant speed reduction for the algorithm because it needed to compare almost *every* test image with both reference templates. While comparisons with the small reference template occur fairly quickly, those with its larger counterpart take a significantly longer period of time to complete, so the combined time to complete the feature detection process with both reference template images accumulates quickly. In order to reduce these time-consuming operations, I recommend using a maximum allowed chamfer match score of 0.005 when using two reference templates. Even with this compromise for the sake of speed, the algorithm still achieved a reliably high accuracy (73%,  $p$ -value = 0.15) when tested with this maximum chamfer match score.

## 5.2. Presenting User-Friendly Elements in Mobile Application Increases its Appeal

When designing the Lighthouse Navigator application, I needed to determine not only how to present location-specific information, but I also needed to design the user interface carefully so that users could interact with my application seamlessly and readily absorb the information that it delivers. Without an aesthetically appealing interface, users can still learn a few aspects of a particular lighthouse, but they cannot fully appreciate how it interacts with the surrounding scenery nor will they retain the significance of the landmark itself. When designing

the Lighthouse Navigator application and presenting it to prospective users during usability study sessions, I found that users:

- Understand textual buttons better than their iconic counterparts;
- Appreciate occasional pop-up hints and subtle navigational cues for navigating the application's different content screens; and
- Prefer multiple avenues for accessing similar content within the application.

#### 5.2.1. Textual Buttons Easier to Understand Than Iconic Ones

Despite Android's strong penchant for iconic buttons – due in part to the lack of screen space offered on most of the platform's devices – users still find textual buttons far more user-friendly than an iconic representation of this button. The usability studies that I conducted at different lighthouses across New England revealed that users initially misunderstood most of the icons that I presented in early iterations of the Lighthouse Navigator application. Some of their initial interpretations of these icons deviated from their intended motif by a considerable margin, such as assuming that a funnel-shaped icon resembled a martini glass more than a filter. Even worse, when these users interpreted the icons, they could not easily “unlearn” their incorrect assumptions, so they tended to make similar interaction errors repeatedly across the application's different content screens. This lack of retraining ability frustrated users quickly and would likely cause them to reconsider opening the application during appropriate times during their traveling experiences in the first place. The textual-based buttons, on the other hand, offer language- and cultural-specific cues that users correctly interpreted almost instantly, allowing them to interact with a particular content screen easily and effectively. More importantly, users retained the proper interaction patterns and applied them to other content screens that offer similar user interface options, which allowed users to develop accurate and refreshingly simple mental models of the application that they could apply to future journeys to lighthouses.

While the textual buttons provide additional appeal for the application from each user's perspective, they hinder my ability to create truly global travel-based applications. Since text forces an application to use a specific language, I would need to localize the application. As any linguistics practitioner knows all too well, however, language-specific terminology does not always translate well to other cultures. Therefore, even if I offered a setting to change the application language, I could still confuse users who speak a more obscure language if I did not

understand the language's subtleties well enough to present a translation that fits the context of the screen. This misinterpretation of the application's functionality could cause users who identify with different cultures to create separate – potentially conflicting – senses of the space surrounding a lighthouse, which would hinder the application's ultimate goal of offering a universal appreciation for the unified yet unique feel that a seaside space should provide any world citizen.

#### 5.2.2. Users Appreciate Occasional and Helpful Navigational Cues

While users generally appreciated self-explanatory elements within the Lighthouse Navigator application, they still tended to appreciate navigational cues, even ones as intrusive as pop-up hints. One of the simplest methods I used for integrating interactive assistance into an application involved dimming navigational elements that users should not select, or ones that would have no effect on the information appearing within a screen upon user selection. For example, the “arrow” buttons within the “History” screen appear dimmed when users have reached either the oldest or newest historical fact corresponding to the lighthouse that they have chosen to view. Without this visual reminder, users would most likely believe (falsely) that they could navigate to facts older than the oldest fact provided or ones more recent than the newest fact provided and would become frustrated quickly when the application appeared to not update at all upon selecting the un-dimmed arrow. They would probably assume that the application had become unresponsive in some way and would attempt to close it.

Similarly, I made an effort to indicate clearly when user interface elements exist beyond the currently-visible edges of a given screen with a scroll bar. The lack of screen space on Android devices prevents all but the simplest screens from fitting entirely “above the fold.” By presenting a scroll bar immediately after a given screen loads, I helped users realize that additional details reside off the edge of the screen and invited them to tap and drag the screen with their finger or stylus to view this additional information. Without the scroll bar guidance, users might have missed important details related to a particular content-based screen within my application and complain about the seemingly incomplete information that it presented. They would then most likely look for a different application about lighthouses that could present the same information more effectively.

A more complex situation occurred when users interacted with an element on a screen that appeared when the screen loaded and responded to user input, but the result of this

interaction deviated from the user's expectations. A key example of this phenomenon within the Lighthouse Navigator application occurred when users viewed the "Information" screen after selecting a particular lighthouse to view. When users attempted to view photographs of the lighthouse, they instinctively selected the photograph that appears on the "Information" screen. In reality, however, this photograph appears simply to provide visual context for the textual information that appears beneath it. In order to direct users to the proper interaction element, I presented a pop-up textbox when users select this photograph on the "Information" screen, instructing them to select the word "Information" that appears in the top bar (known to developers as the "action bar") of the screen. Upon selecting this word, a list of other screens appeared, and users usually recognized that this list provides the interaction mechanism for switching among the application's content screens. This interaction element, while somewhat counterintuitive for most users, remained simple enough for them to remember to perform the same action on a different content screen, giving them control over the application quickly and affording a pleasant user experience. Application developers should introduce similar functionality for the interface elements that beta testers find difficult to use properly at first, but they should use this particular tool sparingly. Users tend to eschew – rather than embrace – frequent intrusive help messages that delay interaction within the application as it represents an annoyingly familiar parallel with pop-up advertisements on websites.

### 5.2.3. Present Empowering Choices for Users

When developing my application, I strove to present interfaces that give users choices for navigating to a particular piece of information whenever possible. I realized that, while text-based icons and navigational hints offer users good guidance on completing individual interactions within an application, they alone do not provide the holistic appeal that multiple interaction options offer. Besides rendering some of the aforementioned navigational cues unnecessary, this flexibility in high-level navigation gave users a stronger sense of control over using the application, making its use a more enjoyable, fun experience for them. Given the sheer volume of applications available to users on Android platforms alone, I would imagine that users remember applications they find easiest to use and turn to those applications more frequently when faced with a need to find information. By taking the time and effort to provide a navigationally flexible application, I make my application more appealing to users, which would most likely translate into more downloads and more positive feedback should the application

ever appear on the Google Play Store. The main menu of an application presented easiest and most obvious location for me to include multiple navigation options. During my usability studies, I found that users would search generally for nearby lighthouses or specifically for a given lighthouse they wished to visit equally often; some even explained that they would use both in the long-term, selecting the one that corresponded best to their particular needs at a given time. Therefore, I elected to keep both options available on the “Welcome” screen of the Lighthouse Navigator, which prospective users appreciated. I needed to guard against introducing ambiguous navigation patterns within my application however. In particular, I needed to ensure that users could “backtrack” out of a content screen in the same way regardless of how they navigated to that screen. I achieve this unified “back navigation” functionality by having the “up” button in the top-left corner of a content screen take users back to the “Welcome” screen. This rapid return to the main menu sidesteps the “Search Results” screen, which appears differently to users based on the option they select on the “Welcome” screen.

## 6. Reflecting on the Project's Takeaways and Future Trajectories

I achieved several key successes within this project, most notably creating a user-friendly application that lighthouse visitors express interest in downloading and implementing image-processing algorithms that exceed blind-chance expectations. I also succeeded in creating a simple and extensible back-end structure for storing lighthouse information and in developing a shared narrative experience across the application's content screens. Despite these achievements, I encountered difficulties that limit the application's current functionality. This chapter describes the application's usability but narrow culture focus, the image-processing algorithms' accuracy but narrow domain focus, and ideas for extending the application's functionality and projecting real lighthouse experiences in the future.

### 6.1. Application Attained Usability Goal, but Cross-Cultural Narratives Tough to Create

The application that I developed satisfied the usability goal I presented within the Introduction chapter, and prospective visitors expressed excitement about the application's eventual availability within the Google Play Store. This application satisfied users' needs since it presents features users care about, such as finding nearby lighthouses, displaying photographs of a given lighthouse, and offering directions to the grounds at or near a lighthouse. Prospective users could also view different aspects of the lighthouse – such as general facts and historical events – in an interface that they found easy to use. This level of control that users command over the application's interface allows them to create a shared narrative experience with other lighthouse visitors and with the societies responsible for maintaining these landmarks more easily, which will help them preserve the seaside monuments for generations to come.

As I designed and tested the application's user interface, I discovered a source of tension between creating a shared, cross-cultural experience and using interface elements that prospective users could comprehend. Ideally, the "Photographs" screen within the application would include only symbolic information that users from any demographic could understand and appreciate, allowing the narrative that the content-based screens create to transcend cultural boundaries. I found in my usability studies, however, that prospective users could not understand the purpose of the iconic buttons I attempted to use for the "upload photo" and "filter photos" operations. Therefore, I needed to resort to a culture-specific visual language – namely, textual buttons – to maximize my application's usability. This decision prevents me from transitioning this application into a different cultural domain without significant adjustments to the user

interface options. Thankfully, the Android platform provides built-in support for multiple languages, making the cross-cultural transition easier.

## 6.2. Image-Processing Algorithms Achieved High Accuracy, but Tough to Generalize

As discussed in the Results chapter above, all three image-processing algorithms exceeded the baseline “blind chance” accuracy expectations, so this feature successfully assists users who enjoy viewing specific types of photographs while remaining invisible to those who wish to view entire photograph galleries instead. To satisfy the computer science research requirements for this project, I devoted a great deal of focus and effort toward these algorithms as I developed the application. I needed to isolate several parameters for each operation in order to maximize each algorithm’s accuracy, which offered me insight into each program’s inner workings. For example, I discovered that color-based feature-matching algorithms perform significantly better when searching for a *range* of colors instead of a specific hue. Since prospective users indicated during the final usability study (at Nauset Light) and the survey from the Lighthouse Preservation Society that they would appreciate the ability to categorize photographs based on weather conditions and the amount of the lighthouse visible. Therefore, I can speculate that actual users would appreciate the image-processing algorithms that I have developed. Future usability studies could confirm that users enjoy these types of photograph filters.

The successful image-processing algorithms that I developed for this project nevertheless contain enough complexity to show that it is still difficult to classify images and detect features within them. These images possess a very temperamental structure; any transformations or recoloring applied to the image drastically reduces the performance of the algorithms used to analyze it. In order to create better-behaved algorithms, I needed to tune the parameters for each program carefully, then perform either a simplified operation on the original images (classify the top half of each image to detect weather conditions) or the original algorithm on simplified images (applying the transformations to a test image so that the shape-based feature-matching algorithm would detect lighthouse shapes successfully). Such simplifications cause the algorithms to work under narrow criteria; more mathematically intelligent methods for analyzing the image could help generalize these operations.

### 6.3. Completing App Functionality, Adding Filters, Investigating Simulacra in Future

The image filtering functionality remains disappointingly absent within the current version of the application, but developers could apply one of several models to introduce these operations in a future version. The simplest conceptual model involves creating middleware to associate `ImagePlus` and `ImageProcessor` objects from ImageJ's library with the representations of images that Android uses natively. This middleware would need to execute quickly and take up little storage space on users' devices, however, to keep the application from becoming too bloated. Furthermore, developers would need an extensive understanding of the low-level structures comprising both ImageJ image objects and Android image views to integrate this middleware successfully. A more complex but less programmatically daunting method features the client-server model. For this application, the image-processing code could execute as a Java program on a server machine, and the client Android application could contact the server whenever users requested a "filter photos" operation. The main drawback with this approach involves the need for consistent Internet connectivity. Users would not be able to filter photographs in an area containing no service. Another challenge lies in the ability to marshal and unmarshal the `Photograph` objects successfully so that the filtered subset of photographs computed on the server (Java) side matches the "gallery subset" displayed on the client (Android) side after completing a filtering operation.

The application also contains several pieces of incomplete functionality that future projects could address. The most significant unimplemented feature involves the ability to "find nearby lighthouses." The application would need to calculate the user's current location, then compute the set of lighthouses closest to that user. Developers would need to decide how to calculate location, using the simpler "as the crow flies" distance calculations or the more realistic "driving distance" calculations. The application could also display information about more lighthouses. Developers can find information regarding these seaside monuments easily by consulting Jeremy D'Entremont's book [1]. Once these developers identify the pieces of information they wish to add to the application, they can insert these data into the `lighthousedata.xml` document quickly and easily given its screen-based organization, and they can extend the existing `LighthouseDataParser` class to extract any new fields that they develop. Future developers could also incorporate the "donate" functionality that appears on the "Welcome" screen within the current application, setting up online accounts with organizations



like the Lighthouse Preservation Society. The application's users could then contribute directly towards maintaining the seaside landmarks. Finally, the application could introduce several more filters that prospective users have requested, such as classifying indoor and outdoor photographs and identifying the season depicted within a given photograph. These additional image-processing algorithms would allow users to view images containing only the features that interest them.

Finally, future research could also consider Jean Baudrillard's idea of simulacra, or a false sense of reality within copies of an artifact. This idea appears within the pieces of information that I present for the different lighthouses within my application, particularly the images that appear on the "Photographs" screen. These details form a *representation* of reality, rather than the reality itself. As users of the application consume this information, they could form false expectations of the lighthouse's visual and experiential contexts, leading to disappointment when the actual monument does not meet these ideals.

The application has already garnered plentiful support from the New England lighthouse community and demonstrates several modern approaches to analyzing images. Future improvements to the application would only further improve lighthouse visitors' ability to immerse themselves in a shared narrative experience of these seaside places.

## 7. References

1. "911 Wireless Services," Federal Communications Commission, n.d. From <http://www.fcc.gov/guides/wireless-911-services>, accessed June 10, 2013.
2. "About: Mission," Groundspeak, Inc., 2013. From <http://www.groundspeak.com/about.aspx#Mission>, accessed May 13, 2013.
3. "Accomplishments of The Lighthouse Preservation Society," Lighthouse Preservation Society, 2013. From <http://www.lighthousepreservation.org/accomplishments.php>, accessed June 26, 2013.
4. "The App Garden: API Documentation: flickr.photos.search," Yahoo! Inc., 2013. From <http://www.flickr.com/services/api/flickr.photos.search.html>, accessed September 9, 2013.
5. "The App Garden: Create an App," Yahoo! Inc., 2013. From <http://www.flickr.com/services/apps/create/apply/>, accessed September 9, 2013.
6. "Application Structure," Android, n.d. From <http://developer.android.com/design/patterns/app-structure.html>, accessed July 17, 2013.
7. "Boston City Guide," Google, 2013. From [https://play.google.com/store/apps/details?id=com.tripadvisor.android.apps.cityguide.boston&feature=search\\_result#?t=W251bGwsMSwyLDEsImNvbS50cmlwYWR2aXNvci5hbmRyb2lkLmFwcHMuY2l0eWd1aWRILmJvc3RvbiJd](https://play.google.com/store/apps/details?id=com.tripadvisor.android.apps.cityguide.boston&feature=search_result#?t=W251bGwsMSwyLDEsImNvbS50cmlwYWR2aXNvci5hbmRyb2lkLmFwcHMuY2l0eWd1aWRILmJvc3RvbiJd), accessed May 17, 2013.
8. "Commons IO," The Apache Software Foundation, 2014. From <http://commons.apache.org/proper/commons-io/>, accessed April 29, 2014.
9. "Cowbird: About," Cowbird, n.d. From <http://cowbird.com/about/>, accessed May 13, 2013.
10. "Cowbird: Culture," Cowbird, n.d. From <http://cowbird.com/culture/>, accessed May 13, 2013.
11. "Cowbird: Guide," n.d. From <http://cowbird.com/guide/>, accessed May 13, 2013.
12. "Creative Vision," Android, n.d. From <http://developer.android.com/design/get-started/creative-vision.html>, accessed June 11, 2013.
13. "Dashboards," Android, n.d. From [http://developer.android.com/about/dashboards/index.html?utm\\_source=ausdroid.net](http://developer.android.com/about/dashboards/index.html?utm_source=ausdroid.net), accessed May 4, 2014.
14. "Design Principles," Android, n.d. From <http://developer.android.com/design/get-started/principles.html>, accessed June 11, 2013.
15. "Dining at The Lighthouse Preservation Society," 2013. From <http://www.lighthousepreservation.org/dining.php>, accessed June 26, 2013.
16. "Free Travel Guides," TripAdvisor LLC, 2013. From <http://www.tripadvisor.com/TravelGuides>, accessed May 13, 2013.
17. "Geocaching 101," Groundspeak, Inc., 2013. From <http://www.geocaching.com/guide/default.aspx>, accessed May 13, 2013.
18. "Geocaching - The Official Global GPS Cache Hunt Site," Groundspeak, Inc., 2013. From <http://www.geocaching.com/>, accessed May 13, 2013.
19. "Geocaching: Trackables," Groundspeak, Inc., 2013. From <http://www.geocaching.com/track/default.aspx>, accessed May 13, 2013.

20. "Geotagging: Do More with your Images and Videos," Nikon Inc., 2013. From <http://www.nikonusa.com/en/Learn-And-Explore/Article/gwur7o4y/geotagging-do-more-with-your-images-and-videos.html>, accessed June 10, 2013.
21. "HISTORY Here," Google, 2013. From [https://play.google.com/store/apps/details?id=com.aetn.history.android.historyhere&feature=search\\_result#?t=W251bGwsMSwxLDEsImNvbS5hZXRuLmhpc3RvcnkuYW5kcm9pZC5oaXN0b3J5aGVyZSJD](https://play.google.com/store/apps/details?id=com.aetn.history.android.historyhere&feature=search_result#?t=W251bGwsMSwxLDEsImNvbS5hZXRuLmhpc3RvcnkuYW5kcm9pZC5oaXN0b3J5aGVyZSJD), accessed May 17, 2013.
22. "The Lighthouse Preservation Society," Lighthouse Preservation Society, 2013. From <http://www.lighthousepreservation.org/>, accessed June 26, 2013.
23. "National Lighthouse Day," American Lighthouse Foundation, 2010. From [http://www.lighthousefoundation.org/museum/natlighthouseday\\_info.htm](http://www.lighthousefoundation.org/museum/natlighthouseday_info.htm), accessed June 26, 2013.
24. "Navigation with Back and Up," Android, n.d. From <http://developer.android.com/design/patterns/navigation.html>, accessed September 3, 2013.
25. "New in Android," Android, n.d. From <http://developer.android.com/design/patterns/new.html>, accessed July 17, 2013.
26. "NYC Way -- Everything NYC," Google, 2013. From [https://play.google.com/store/apps/details?id=com.newnycway&feature=search\\_result#?t=W251bGwsMSwyLDEsImNvbS5uZXdueWN3YXkiXQ..](https://play.google.com/store/apps/details?id=com.newnycway&feature=search_result#?t=W251bGwsMSwyLDEsImNvbS5uZXdueWN3YXkiXQ..), accessed May 20, 2013.
27. "Opportunities for Investment," Lighthouse Preservation Society, 2013. From <http://www.lighthousepreservation.org/opportunities-for-investment.php>, accessed June 26, 2013.
28. "Panoramio -- Photos of the World," Google, 2012. From <http://www.panoramio.com/map#lt=28.000000&ln=-33.000000&z=15&k=1&a=1&tab=1&pl=all>, accessed May 13, 2013.
29. "Panoramio -- Tags," Google, n.d. From <http://www.panoramio.com/tags>, accessed May 13, 2013.
30. "Panoramio help: Understanding popularity in Panoramio," Google, n.d. From [http://www.panoramio.com/help/understanding\\_popularity](http://www.panoramio.com/help/understanding_popularity), accessed May 13, 2013.
31. "Reviews of Hotels, Flights and Vacation Rentals - TripAdvisor," TripAdvisor LLC, 2013. From <http://www.tripadvisor.com/>, accessed May 13, 2013.
32. "Tabs," Android, n.d. From <http://developer.android.com/design/building-blocks/tabs.html>, accessed July 17, 2013.
33. "UI Overview," Android, n.d. From <http://developer.android.com/design/get-started/ui-overview.html>, accessed July 17, 2013.
34. "XmlPullParser," Android, 2014. From <http://developer.android.com/reference/org/xmlpull/v1/XmlPullParser.html>, accessed April 29, 2014.
35. "XmlPullParserFactory," Android, 2014. From <http://developer.android.com/reference/org/xmlpull/v1/XmlPullParserFactory.html>, accessed April 29, 2014.
36. Yelp Elite Squad, Yelp Inc., 2013.
37. Yelp: About Us, Yelp Inc., 2013.
38. Beck, K., et al. Principles behind the Agile Manifesto, Agile Manifesto, 2001.
39. Bobbitt, R. "Photographers Have Several Camera Options for Geotagging Pictures with GPS Points," 2009. From <http://www.directionsmag.com/articles/photographers-have->

- several-camera-options-for-geotagging-pictures-with-gps-/122479, accessed June 10, 2013.
40. Borgefors, G. Distance Transformations in Digital Images. *Computer Vision, Graphics, and Image Processing*, 34 (3). 344-371.
  41. Burger, W. and Burge, M.J. *Digital Image Processing: An Algorithmic Introduction Using Java*. Springer Science+Business Media, LLC, New York, New York, 2008.
  42. Cristoforetti, A., Gennai, F. and Rodeschini, G. Home sweet home: The emotional construction of places. *Journal of Aging Studies*, 25 (3). 225-232.
  43. echoblaze, S.L. "How to download and save a file from Internet using Java?," Stack Exchange, Inc., 2014. From <http://stackoverflow.com/questions/921262/how-to-download-and-save-a-file-from-internet-using-java>, accessed April 29, 2014.
  44. Entrikin, J.N. and Tepple, J.H. Humanism and Democratic Place-Making. in Aitken, S. and Valentine, G. eds. *Approaches to Human Geography*, SAGE Publications, London, England, 2006, 30-41.
  45. Gardiner, M. Everyday Utopianism: Lefebvre and his critics. *Cultural Studies*, 18 (2). 228-254.
  46. Genachowski, J. Second Report and Order: In the Matter of Wireless E911 Location Accuracy Requirements, Washington, D. C., 2010.
  47. Hyland, J. Hufnagle, K. ed., 2013.
  48. Johnson, R.R. Audience Involved: Toward a Participatory Model of Writing. *Computers and Composition*, 14. 361-376.
  49. Lefebvre, H. *The Production of Space*. Blackwell Publishers Ltd., Oxford, United Kingdom, 1991.
  50. Luo, J., Joshi, D., Yu, J. and Gallagher, A. Geotagging in multimedia and computer vision -- a survey. *Multimedia Tools and Applications*, 51 (1). 187-211.
  51. Mark, J. "Four Android App Design Guidelines You Should Break," Fast Company Labs, 2013. From <http://www.fastcolabs.com/3012752/four-android-app-design-guidelines-you-should-break>, accessed July 17, 2013.
  52. Massara, F. and Severino, F. Psychological Distance in the Heritage Experience. *Annals of Tourism Research*, 42. 108-129.
  53. Nadal-Melsió, S. Lessons in Surrealism. in Goonewardena, K., Kipfer, S., Milgrom, R. and Schmid, C. eds. *Space, Difference, Everyday Life: Reading Henri Lefebvre*, Routledge, New York, New York, 2008, 161-175.
  54. Nudelman, G. *Android Design Patterns: Interaction Design Solutions for Developers*. John Wiley & Sons, Inc., Indianapolis, Indiana, 2013.
  55. Rodríguez de Castro, A. Tuan, Premio vautrin lud de geografía. *Didáctica Geográfica*, 13. 155-160.
  56. Schmid, C. Henri Lefebvre's Theory of the Production of Space: Towards a three-dimensional dialectic. in Goonewardena, K., Kipfer, S., Milgrom, R. and Schmid, C. eds. *Space, Difference, Everyday Life: Reading Henri Lefebvre*, Routledge, New York, New York, 2008, 27-45.
  57. Slominski, A. Quick Introduction to XMLPULL V1 API, High Performance Distributed and Parallel Systems Research, Department of Computer Science, Indiana University, 2005.

58. Snavely, N., Seitz, S.M. and Szeliski, R. Photo Tourism: Exploring Photo Collections in 3D. *ACM Transactions on Graphics -- Proceedings of ACM SIGGRAPH 2006*, 25 (3). 835-846.
59. SparseArray. Android, 2014. From <http://developer.android.com/reference/android/util/SparseArray.html>, accessed April 29, 2014.
60. Spencer, C. "How to Geotag Images," Digital Photography School, 2008. From <http://digital-photography-school.com/how-to-geotag-images>, accessed June 10, 2013.
61. Tuan, Y.-F. *Space and Place: The Perspective of Experience*. University of Minnesota Press, Minneapolis, Minnesota, 1977.
62. Varma, T. The Joys of Designing Agile Solutions for New-Age Problems, 24-7 Innovation Labs, Pune, India, 2013.

## Appendix A: Instructions for Installing Project Files (README)

Kevin Hufnagle (WPI username = khufnagle)

May 6, 2014

Major Qualifying Project – “Lighthouse Navigator” Android Application  
Guide to Setting Up Development Environment

### Table of Contents

1. Brief Overview of Application
2. Components to Install on Machine
3. Steps for Executing the “Lighthouse Navigator” Application
4. Steps for Executing the Image-Processing Algorithms
5. Steps for Gaining Access to Flickr APIs within Application
6. Steps for Gaining Access to Google Maps APIs within Application

### 1. Brief Overview of Application

This Android application enhances travel experiences to lighthouses across New England, both before and during trips. Within this app, users can view:

- General facts about different lighthouses;
- Photographs that people have taken of the lighthouses;
- Historical events that have taken place at the lighthouses; and
- Reviews of the lighthouse sites from visitors.

In particular, the application comes bundled with an advanced set of image-processing algorithms, which users could execute to filter the photographs that appear within the application.

### 2. Components to Install on Machine

- Java Development Kit (JDK) -- Includes Java Runtime Environment (JRE) as well as debugging and development tools.  
<http://www.oracle.com/technetwork/java/javase/downloads/index.html?ssSourceSiteId=otnjp>
- Android Software Development Kit (SDK) -- Includes Android’s APIs and Android SDK Manager, a portal for downloading additional Android tools.  
<http://developer.android.com/sdk/index.html?hl=sk>

- Eclipse -- IDE for Java and Android development.  
<https://www.eclipse.org/downloads/>
- Android Development Tools (ADT) Plug-In for Eclipse -- Offers more powerful Android development functionality within Eclipse.  
<http://developer.android.com/sdk/installing/installing-adt.html>
- Files for the application itself -- The most recent copy of my application.  
<https://github.com/Kbhredsox/lighthousenavigator>
- Android emulator or (better yet) a physical Android device -- For running the application.  
<http://developer.android.com/tools/devices/managing-avds.html>

### 3. Steps for Executing the “Lighthouse Navigator” Application

1. Go to GitHub and download the most up-to-date version of the application (by selecting the “Download ZIP” button along the right-hand side of the page. Remember where you save the ZIP file on your local machine.
2. Open Eclipse and select the workspace where you would like the application files to reside. (Create a new workspace if this is your first time using Eclipse on this machine.)
3. Navigate to **File > Import**. The “Import” dialog appears.
4. Choose **General**, then **Existing Projects into Workspace**. Click **Next**.
5. Near the top of this window, make sure the **Select archive file** radio button is enabled, then click **Browse** and find the ZIP file you downloaded in step 1. The projects associated with the application should appear in the “Projects” area of the window automatically.
6. Click **Finish**. Eclipse imports the project files. Note that, because of the project’s extensive size, this process could take a few minutes.
7. Expand the “LighthouseNavigator” project to view the application’s files.
8. Right-click the “LighthouseNavigator” project and select **Run As > Android Application**. The “Android Device Chooser” dialog appears.
9. Select the device or virtual device you wish to use to run the application, then click **OK**. The application’s “Welcome” screen should appear on the device you selected.
10. As you interact with the application, you will notice that the “Photographs” screen does not load correctly. Complete procedures (5) and (6) within this guide to fix this issue.

#### 4. Steps for Executing the Image-Processing Algorithms

1. Complete steps 1-6 from the above set of directions.
2. Expand the “ImageJProcessor” project, then expand the “src” folder and the `edu.wpi.khufnagle.ij.drivers` package.
3. Right-click the driver corresponding to the image-processing algorithm you wish to demo and select **Run As > Java Application**. The program’s output appears in the “Console” pane near the bottom of the Eclipse Workbench.

#### 5. Steps for Gaining Access to Flickr APIs within Application

**Warning:** It appears as though the method I used for authenticating with Flickr’s APIs has been deprecated. The steps below might still work, but it is advised to follow the most recent set of directions on Flickr’s website.

1. Obtain an API key by completing step 1 at the following website:  
<https://www.flickr.com/services/api/auth.howto.web.html>

You will need to log into your Yahoo!, Facebook, or Google account to continue.

2. On the page that loads after logging in, decide whether you wish to apply for a non-commercial key or a commercial one. (**Note:** Since I intend to offer this application for free to the general public, I applied for a non-commercial key.)
3. Another page loads, prompting you to enter the name of the app (“Lighthouse Navigator”) and explain in as much detail as possible how you are extending or using the functionality of the application.

In a little while, you should receive an email allowing you to access your API key *for this specific application only*.

4. Complete steps 1-7 for executing the Android application associated with this project if necessary. (If you have already loaded the project into a workspace, simply open Eclipse and navigate to that workspace.)
5. Open the `FlickrPhotoDownloadThread.java` file within the `edu.wpi.khufnagle.lighthousenavigator.util` package and navigate to line 95 within this file. This line should include the following section of a URL:

```
&api_key=3ea8366b020383eb91f170c6f41748f5
```

Replace this API key with the one you received from Flickr in step 3.



6. You should now be able to run the Lighthouse Navigator application (as explained in steps 8 and 9 within the procedure for running the application for the first time) and be able to download photographs from Flickr’s servers.

## 6. Steps for Gaining Access to Google Maps APIs within Application

1. Navigate to <https://console.developers.google.com>. Sign into your Google account (or create a new one) if necessary.
2. Click **Create Project** near the top-left corner of the page. The “New Project” pop-up dialog appears.
3. Within this dialog, enter a descriptive name for the project (such as “Lighthouse Navigator”) and change the product ID if desired. Enable or disable the checkboxes beneath these two text fields to your liking, then click **Create**. The project appears within your Google Developers Console dashboard.
4. Click the name of the project you just created, then select the **APIs & auth** option along the left-hand navigation bar. A list of APIs you can use should appear.
5. Scroll down to “Google Maps Android API v2” (being careful to *not* select “Google Maps Engine API” or something similar) and press the **OFF** toggle button directly to the right of the API label. The button should turn green and display **ON**.
6. Next, select the **Credentials** sub-option beneath the **APIs & auth** option along the left-hand navigation bar. The credentials page loads.
7. Under “Public API access,” select **Create New Key**. The “Create a new key” pop-up dialog appears.
8. Select **Android key** from the list of options. The “Create an Android key and configure allowed Android applications” dialog appears.
9. In the text box near the bottom of this dialog, enter the SHA1 certificate fingerprint corresponding to your app, followed by a semicolon, followed by your project’s package name (such as `edu.wpi.khufnagle.lighthousenavigator`).

You can find your SHA1 certificate by opening a terminal or command prompt and navigating to the “bin” directory under your Java directory (such as `C:\Program Files\Java\jdk1.7.0_51\bin`), then executing the following command:

```
keytool -list -v keystore <path-to-your-keystore> -alias
    androiddebugkey -storepass android -keypass android
```

The path to your keystore is `C:\Users\<your-windows-username>\.android\debug.keystore` by default (or `~/ .android/debug.keystore` on Linux machines).

Part of the output should display the MD5, SHA1, and SHA256 certificate fingerprints on consecutive rows. The SHA1 certificate, which should appear in the middle, is the fingerprint you want to enter into the dialog from the beginning of this step.

10. Click **Create**. The “Credentials” page should now show an area titled “Key for Android applications.” The top line within this area, labeled “API key,” contains the value you will need in a few steps.
11. Complete steps 1-7 for executing the Android application associated with this project if necessary. (If you have already loaded the project into a workspace, simply open Eclipse and navigate to that workspace.)
12. Navigate to the `AndroidManifest.xml` file, which should appear in the root directory of the LighthouseNavigator Java project folder. Within the `<application>` element of this XML file, there exists a `<meta-data>` element with a “name” attribute of `com.google.android.maps.v2.API_KEY`. Change the corresponding value to the API key that you generated in step 10.
13. You should now be able to run the Lighthouse Navigator application (as explained in steps 8 and 9 within the procedure for running the application for the first time) and be able to view maps from Google Maps within the “Photographs” screen successfully.

## Appendix B: Version History

Version Number	Date	Comment
21	May 6, 2014	Added class diagrams of application within Design & Methodology chapter
20	May 5, 2014	Completed Introduction and Conclusion chapters; revised entire report; added appendices
19	May 2, 2014	Completed Results chapter
18	May 1, 2014	Completed Implementation chapter; cleaned up Literature Review and Design & Methodology chapters
17	April 30, 2014	Added description of back-end data structures (XML file, photograph cache) within Implementation chapter
16	April 29, 2014	Added discussions about data structures and Flickr photograph acquisition within Implementation chapter
15	April 28, 2014	Added details about the application project's high-level layout; began discussing custom data collections within the Implementation chapter
14	April 21, 2014	Began drafting the Design & Methodology chapter
13	April 12, 2014	Added coherent headings to Literature Review chapter; converted all CS-based literature review topics to prose; made small adjustments to existing "Space and Place" prose to address Prof. deWinter's feedback; began converting "Existing Smartphone Applications" section outline to prose
12	April 11, 2014	Rewrote beginning of "Current Websites" section within Literature Review chapter to include more obvious claim-based structure; removed references to irrelevant details
11	April 9, 2014	Adjusted presentation of claims in "Current Websites" section of Literature Review chapter based on feedback from advisors
10	April 7, 2014	Revised "user-friendly elements" section of Results chapter
9	March 31, 2014	Created high-level draft of Results chapter
8	February 17, 2014	Added information from seminal article (Borgefors) about chamfer feature-matching distance transforms within Literature Review
7	February 12, 2014	Added information about template-matching algorithms to Literature Review chapter

<b>6</b>	February 11, 2014	Added descriptions about contour detection and elementary image region shape descriptions within Literature Review chapter
<b>5</b>	February 7, 2014	Edited and completed existing outline of cumulative histogram comparison method within Design & Methodology chapter
<b>4</b>	February 3, 2014	Added explanation of general and discretized cumulative histogram matching algorithms within Literature Review chapter
<b>3</b>	January 27, 2014	Added extensive description of grayscale histograms and the characteristic image properties and flaws apparent with them within the Literature Review chapter – also began to add paragraph on adjusting an image’s cumulative histogram to an arbitrary reference cumulative histogram
<b>2</b>	January 24, 2014	Added discussion of different file formats and background information on ImageJ to Literature Review chapter
<b>1</b>	January 22, 2014	Added explanations of <code>SparseArray</code> object and the <code>FileUtils</code> library within the Implementation chapter

## Appendix C: Interview with Jay Hyland

### Questions for Interview

1. In what direction are you currently taking preservation efforts? Are you looking to restore a specific set of lighthouses, make a certain type of improvement to all lighthouses, or some other initiative?
2. What types of people typically visit the lighthouses your organization helps preserve? Are they young/old, from the United States/international travelers, male/female, outdoor enthusiasts?
3. Why do people (such as those from question 2) visit lighthouses? What is their primary purpose for visiting lighthouses (aesthetic appeal, waterfront view, historical knowledge, cultural knowledge)?
4. Do you think these “main visitors” to lighthouses look up information about the landmarks before visiting them?
5. Do you think these “main visitors” to lighthouses record their experiences (written documents, pictures)?
6. Would these “main visitors” (or family/friends of them) be interested in using an Android (smartphone) application to tour these lighthouses virtually and learn more about their respective histories?
7. How should I present this application? Should I encourage users to preserve individual lighthouses, or should I cultivate a cultured knowledge of lighthouses and their importance as navigation tools?
8. What resources should I consult to learn more about this organization and about lighthouses in general?

## Interview Minutes

**Attendees:** Kevin Hufnagle, Jay Hyland (president/founder, Lighthouse Preservation Society)  
**Location:** Newburyport Rear Range Lighthouse, Newburyport, MA  
**Date:** April 3, 2013

- I. Improve Both Individual Aspects of Lighthouses and Lighthouse Network as a Whole
  - A. Look at “Accomplishments” page on website (e.g. grant from National Parks Service) – “mansion” category
  - B. Lots of matching on part of societies – grant program has proven to be **very** effective
    1. Only small government grant necessary for non-profit groups to begin contributing
  - C. Important first step was getting lighthouses on the National Register of Historic Places in 1980s
  
- II. History of Lighthouse Preservation Society
  - A. Created in 1983
  - B. Started out as grant to document lighthouses
  - C. Slide photographs
  - D. Survey of lighthouses around country – started with Pequot Lighthouse in New London, CT
  - E. Hyland originally involved with grants in Hull in 1982 (New England lighthouses) – got “pass” from U. S. Coast Guard to get in various buildings across the region
  - F. Lighthouses were being automated as part of national initiative to get them all automated by 1990
  - G. Hyland documented “what was going on” as automation process continues – noticed that places boarded up too tight (peeling paint from moisture content) – loom siding just a “band-aid” solution
    1. Realized lack of tractable plan, funds, mission for historic preservation
    2. Maintainers = college-age people – not very experienced with building upkeep (e.g. putting vinyl on all surfaces!?)
  - H. Spoke with a neighbor in his North Shore community (Harvard professor in government department; her father was part of Department of Transportation in Washington, D. C.)
    1. Encouraged Hyland to bring issue to Congress – she joined him in creating Board of Directors to lobby Congress to receive funding – went door-to-door in D. C.
    2. Brought about Congressional hearing – Admiral of Aids and Navigation cited \$7 million deficit in maintenance – story went “viral” (circa 1986)

- I. Lighthouse stamps (Howard Paine, artist = Howard Koslow from NJ)
  - 1. Series since 1980s
  - 2. Issue stamps that include an iconic lighthouse from each state
  - 3. New England lighthouses featured on 5 stamps (one per state) in summer of 2013
  
- III. My Application Merges Technology and Communication
  - A. *American Lighthouses* book = good reference
  - B. Keeper's log in Newburyport lighthouse
    - 1. Goes back three years
    - 2. Much like a guest book at a cottage/condo
  
- IV. Allure to Lighthouses – Visitors
  - A. Can depend on lighthouse (Newburyport one in particular is quite romantic, allowing wealthy couples (ages = mid-20s and older) to dine at the top)
  - B. People of all ages (children to retired individuals) visit museums for lighthouses
  
- V. Popularity of Lighthouses
  - A. Reasons for visiting
    - 1. Beauty (photographers, artists visit for this reason)
    - 2. Location
    - 3. Architecture
      - a. These “characters of the coast” are each a little different
      - b. Even standardized structures have different paint jobs!
      - c. Varied sequence of flashes, foghorns
    - 4. Romance (lighthouses in U. S. equivalent to castles in Europe)
    - 5. Lifestyles of keepers (as it relates to modern life)
    - 6. Spiritual
      - a. “Light in the darkness”
      - b. “Gives us direction”
      - c. Literature, hymns, poems reference lighthouses (e.g. Longfellow wrote about Portland Head Light)
  - B. Curricula based on lighthouses
    - 1. Navigation (distance from ship to lighthouse)
    - 2. History lessons (e.g. Bob Gallagher at Old Scituate Lighthouse in Scituate, MA)
  - C. First public works act of Congress (signed by George Washington)
    - 1. Considered to be the “First Great Work of the American People”
    - 2. Goal = always be in range of at least one lighthouse along U. S. coastline
    - 3. Sometimes, building them was dangerous (e.g. Minot Point in Cohasset, MA) – big engineering achievement to create a stable one there
  - D. Europe took inspiration and modeled their lighthouses after U. S. ones

## VI. Information Available to Visitors

- A. New England lighthouse guides
  - 1. Scans of paper brochures by state online (no longer printed because not cost-effective anymore) – listing of lighthouses
    - a. Keep in mind that at least half of lighthouses are off-shore – should my project focus on just the easily-accessible lighthouses? Probably...
  - 2. Virtual map of lighthouses (more informational, akin to Google Maps)
- B. Personal accounts
  - 1. Internet?
  - 2. Keeper's logs

## VII. Smartphone Use among Visitors

- A. Lots of visitors (Hyland estimates at least half) use smartphones – e.g. Hyland takes pictures of couples dining at Newburyport Rear Range Lighthouse using their smartphones

## VIII. Emphasis on Individual vs. General Information about Lighthouses in Application

- A. Discuss unique feel of individual lighthouses – break down by state at least
- B. Follow lead of lighthouse publications!

## IX. Tour Guides at Lighthouses

- A. Not many tour guides, but museums have them
  - 1. Stonington Harbor (Stonington, CT) – Louise Pitaway
  - 2. Portland Head Light (Portland, ME)
  - 3. **Is this application a way for me to inspire more lighthouse museums in New England?**

## X. Resources to Consult

- A. Films (Several about the Newburyport lighthouses) – careful with copyright!
- B. User videos on YouTube – careful with copyright!
- C. Two episodes of *Chronicle* from WCVB-TV – **cannot** post this anywhere (WCVB guards rights to episode decades later)
- D. Hyland willing to share his slides/photographs with me at no cost (from Dover, NH)



## Appendix D: Survey for Visitors of Cape Neddick Lighthouse



# WPI

May 25, 2013

Dear Sir or Madam,

For my senior project at Worcester Polytechnic Institute, I intend to develop an Android application that allows people who visit lighthouses in New England (such as yourself) to learn more about these landmarks. When using this application, you will be able to (1) view photographs of different lighthouses, (2) explore the history of these lighthouses, and (3) read about other visitors' experiences and stories from exploring these lighthouses.

This short survey will gauge your experience with smartphones and your interest in the application I am developing. I would appreciate you taking the time to complete and return it.

Once you have completed this survey, please place it in the bin labeled "Completed Lighthouse Surveys."

If you have any questions about this survey or my project, please feel free to contact myself (contact information listed at the bottom of this page) or my co-advisor, Professor Jennifer deWinter. You can reach her by phone at (508) 831-6679, by email at [jdewinter@wpi.edu](mailto:jdewinter@wpi.edu), or by mail at Attn: Prof. Jennifer deWinter, Department of Humanities & Arts, Worcester Polytechnic Institute, 100 Institute Road, Worcester, MA 01609.

Thank you for your time and interest in this project.

Sincerely,  
Kevin Hufnagle  
Worcester Polytechnic Institute  
Computer Science and Professional Writing, Class of 2013  
(617) 549-2933  
[khufnagle@wpi.edu](mailto:khufnagle@wpi.edu)

**Instructions:** For each question, fill the bubble(s) next to **EACH** response that applies to you. Some questions may require more than one answer.

There are 13 questions in this survey (only the final question is open-ended) and should take no more than 5 minutes to complete.

**The following questions ask you to describe your traveling and picture-taking tendencies while visiting lighthouses.**

1. What is the most common reason that you travel to a given lighthouse? (Fill in **ONE** response.)
  - To learn about the history of the lighthouse
  - To appreciate the architecture of the lighthouse
  - To appreciate the scenery that surrounds the lighthouse
  - To spend quality time with a family member, close friend, and/or significant other
  - To visit other landmarks near that lighthouse
  - To enjoy nearby dining and/or entertainment options
  - Other (please specify) \_\_\_\_\_
  
2. How often do you take photographs of lighthouses when you visit them?
  - Always
  - Most of the time
  - Sometimes
  - Rarely
  - Never

*If you answered “Never” for question 2, skip to question 4 on the following page. Otherwise, continue to question 3.*

3. What device do you typically use to take photographs of lighthouses?
  - Smartphone (iPhone, Android, Blackberry, etc.)
  - Digital camera
  - Film (non-digital) camera
  - Other (please specify) \_\_\_\_\_

**The following questions ask you to describe your access to smartphones and your use of them.**

4. Do you own or have regular access to a smartphone?

- Yes, an Android
- Yes, an iPhone
- Yes, a Blackberry
- Yes, another type of smartphone (please specify) \_\_\_\_\_
- Not sure
- No

*If you answered “No” or “Not sure” for question 4, skip to question 13 on page 5. Otherwise, continue to questions 5 and 6, then to the questions on the next page.*

5. Do you take your smartphone with you when you travel to a lighthouse?

- Yes
- Sometimes
- No

6. Can you access websites and download data from the Internet using your smartphone when you travel to a lighthouse?

- Yes
- Sometimes
- No

*Questions continue on the following page.*

**The following questions ask you to describe your behavior when visiting websites that contain information about locations and/or online photographs.**

7. How often do you visit websites that contain information about a lighthouse (such as Yelp or TripAdvisor) before or while you travel to that lighthouse?
- Always
  - Most of the time
  - Sometimes
  - Rarely
  - Never

*If you answered “Never” for question 7, skip to question 9. Otherwise, continue to question 8.*

8. What do you typically look at on websites that contain information about a lighthouse? (Fill in **ALL** responses that apply.)
- Hours of the lighthouse
  - Cost of visiting the lighthouse
  - Contact information for the lighthouse
  - Address/directions for the lighthouse
  - Visitors’ reviews of the lighthouse
  - Photographs of the lighthouse
  - Other locations similar to the lighthouse
  - Other (please specify) \_\_\_\_\_
9. How often do you visit websites that contain online photographs of a lighthouse (such as Google Images, Flickr, or Instagram) before or while you travel to that lighthouse?
- Always
  - Most of the time
  - Sometimes
  - Rarely
  - Never

**The following questions ask you to evaluate a lighthouse application that you could use on your smartphone.**

10. How likely would you be to download a **FREE** smartphone application that would allow you to view photographs, history, and user experiences of a lighthouse?

- Extremely likely
- Very likely
- Somewhat likely
- Not very likely
- Not at all likely

*If you answered “Not at all likely” for question 10, skip to question 13 on the following page. Otherwise, continue to questions 11 and 12.*

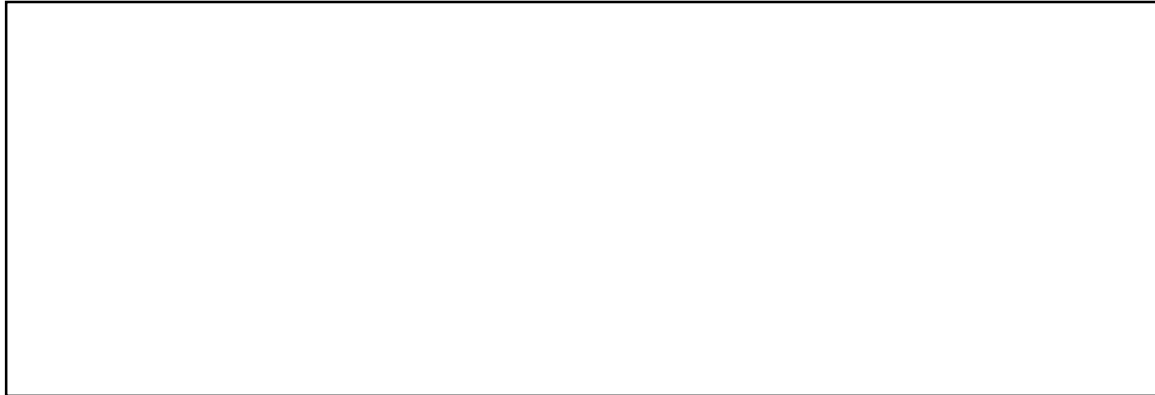
11. How often would you use the lighthouse application after installing it on your smartphone?

- Very often
- Often
- Occasionally
- Almost never

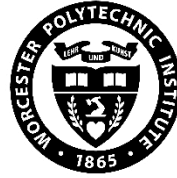
12. What features would you like to see in the lighthouse application? (Fill in **ALL** responses that apply.)

- View key information (location, hours, contact phone number)
- Show dining and/or entertainment options near a given lighthouse
- Search for a specific lighthouse
- Upload your photographs for a given lighthouse
- Dedicate separate sections of the application to photographs, history, and visitors' experiences
- Filter photograph results by different camera settings (resolution, date taken) within “photographs” section
- Change text settings (color, size) within “history” and “visitors' experiences” sections
- Upload your visiting experience(s) for a given lighthouse
- Other (please specify) \_\_\_\_\_

13. Thank you for taking the time to complete this survey. Please provide any additional comments or ideas in the box below. We will consider these responses during the design process to create an application that satisfies your needs.

A large, empty rectangular box with a black border, intended for the respondent to provide additional comments or ideas. The box is positioned centrally below the text and is currently blank.

## Appendix E: Survey for Lighthouse Preservation Society Members



# WPI

The Lighthouse Preservation Society  
11 Seaborne Drive  
Dover, NH 03820

March 3, 2014

Dear Sir or Madam,

For my senior project at Worcester Polytechnic Institute, I am developing an Android application that allows people who visit lighthouses in New England (such as yourself) to learn more about these landmarks. When using this application, you will be able to (1) view photographs of different lighthouses, (2) explore the history of these lighthouses, and (3) read about other visitors' experiences and stories from exploring these lighthouses.

This short survey gauges your use of smartphones and travel-based websites and has you evaluate the application design. I would appreciate you taking the time to complete and return it.

You can complete this survey electronically at the following link, or you can mail it back to the Lighthouse Preservation Society address (given at the top of this page).

<https://www.surveymonkey.com/s/lighthouse-app-feedback>

If you have any questions about this survey or my project, please feel free to contact myself (contact information listed at the bottom of this page) or my co-advisor, Jennifer deWinter.

You can reach her by phone at (508) 831-6679, by email at [jdewinter@wpi.edu](mailto:jdewinter@wpi.edu), or by mail at Attn: Prof. Jennifer deWinter, Department of Humanities & Arts, Worcester Polytechnic Institute, 100 Institute Road, Worcester, MA 01609.

**Please complete this survey by Thursday, April 3.** Thank you for your time and interest in this project.

Sincerely,  
Kevin Hufnagle  
Worcester Polytechnic Institute  
Computer Science and Professional Writing, Class of 2014  
(617) 549-2933  
[khufnagle@wpi.edu](mailto:khufnagle@wpi.edu)

Note: The original version of this cover letter appeared in a 12-point font.

**Instructions:** For each question, fill in the bubble(s) next to **EACH** response that applies to you. Some questions may require more than one answer.

There are 11 questions in this survey (only the final question is open-ended) and should take no more than 5 minutes to complete.

**The first question asks you to describe your access to smartphones.**

- Do you own or have regular access to a smartphone?
  - Yes, an Android
  - Yes, an iPhone
  - Yes, a Blackberry
  - Yes, another type of smartphone (please specify below)

---

  - Not sure
  - No

*If you answered “No” or “Not sure” for question 1, skip to question 11 on page 3. Otherwise, continue to question 2 below.*

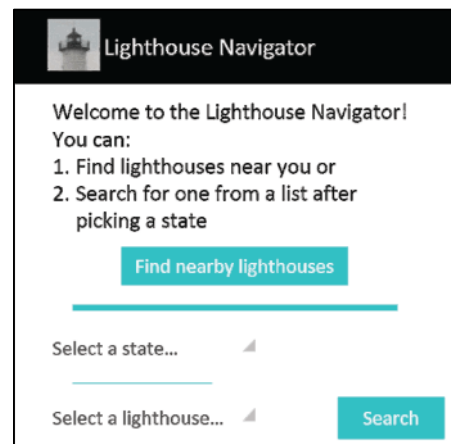
**The next two questions ask you to identify your use of websites related to travel.**

- Which of the following websites do you access most of the time before or during your trip to visit a lighthouse? (Fill in **ALL** responses that apply.)
  - Flickr
  - Google Images
  - Instagram
  - TripAdvisor
  - Yelp

- How often do you upload photographs of lighthouses onto Flickr for other people to see and share?
  - All the time
  - Often
  - Not very often
  - Never

**The next four questions ask you to describe your expected experience with the application I am developing.**

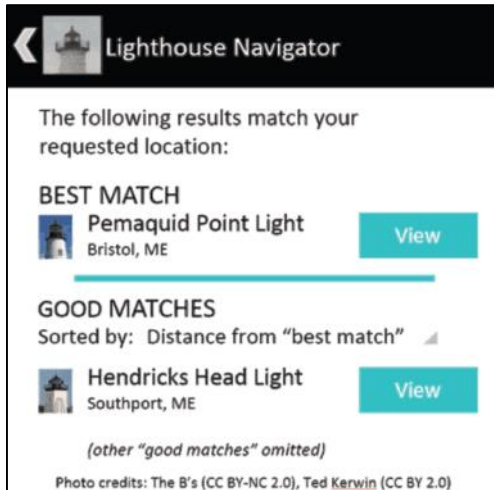
*Question 4 is based on the image below, which shows part of the “welcome” screen that appears within the application:*



- Which of the two search methods shown in the screen above would you consider using? (Fill in **ALL** responses that apply.)
  - Find lighthouses near you (first option on screen)
  - Search for a lighthouse from a list after picking a state (second option on screen)

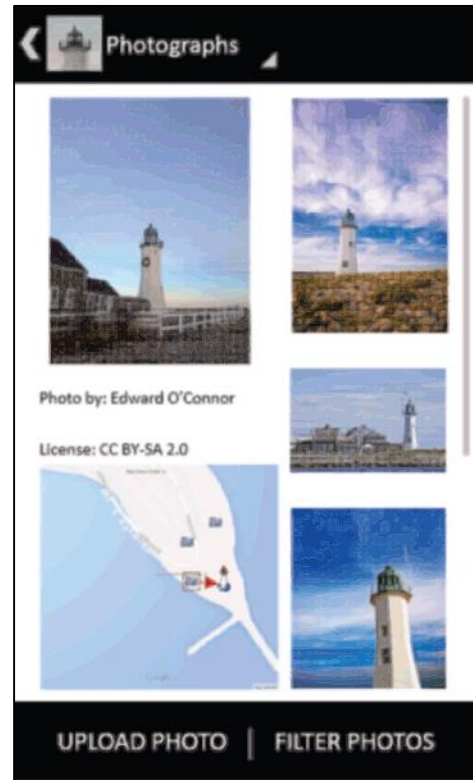


Questions 5 and 6 are based on the image below, which shows part of a sample “search results” screen that appears within the application:



5. In what way(s) would you expect to be able to sort the results that appear under the heading labeled “Good Matches”? (Fill in **ALL** responses that apply.)
- Distance from the lighthouse listed under “best match”
  - Similarity in name compared to the one listed under “best match”
  - Another method of sorting (please specify below)
- 
6. After selecting the “View” button to the right of a search result, which of the following screens should appear next? (Fill in **ONE** response.)
- General information about the lighthouse (address, hours, height)
  - Photographs of the lighthouse and a map showing where other people have taken pictures of that lighthouse
  - Historical facts about the lighthouse
  - Visitors’ experiences from exploring the lighthouse

Question 7 is based on the image below, which shows a sample “photographs” screen that appears within the application:



7. After selecting the “Filter Photos” button in the bottom-right corner of this screen, what options should appear on the screen? These options allow you to view photographs that fit specific criteria. (Fill in **ALL** responses that apply.)
- Time of day (day, twilight, night)
  - Time of year (winter, summer)
  - Location (inside, outside)
  - Type of background (water, land)
  - Amount of lighthouse visible (entire structure, tower only)
  - Another type of filter (please specify below)
-



## Appendix F: Usability Study Materials

### Preamble (Used in all usability studies)

Hi there, how are you doing today? Could I have a few minutes of your time to help me with a school research project about lighthouses?

Thank you for your willingness to participate in this study. During this session, I will be asking you several questions about your smartphone use and about the Android app that I am developing for my senior research project at Worcester Polytechnic Institute. The purpose of this project is to create an app that allows lighthouse visitors (such as yourself) the opportunity to (1) examine general information about lighthouses in New England, (2) view photographs of these lighthouses, (3) explore the history of these lighthouses, and (4) read about other visitors' experiences and stories from exploring these lighthouses.

I will be publishing this research online, and by participating, your responses may appear in this publication. Your response will remain anonymous, however, and I will not publish any personally identifiable information in the paper.

The purpose of this study in particular is to evaluate the usability of the app I am developing and to determine the pieces of information that should be most prominent within it. It should take no longer than 5 minutes to complete. You will not encounter any significant risks during this study, your participation is completely voluntary, and you may stop at any time.

Before we begin, do you have any questions for me?

Poster (Used in all usability studies)



**WPI**



Use a new Android app to  
guide you to lighthouses  
across New England!

Talk to this student to help create  
an app that allows you to explore  
more of these beautiful landmarks!

## Cape Neddick (“Nubble Point”) Lighthouse – July 19, 2013

### Script

We will begin the study with a couple of general questions about your smartphone use.

1. Do you use or have regular access to a smartphone?
  - a. If so, what type?
  - b. If so, how long have you used it?
  - c. If not, do you plan on gaining access to one within the next 6 months?

*If the answer to both (1) and (b) is “no,” the participant is disqualified from the study. I will say, “Thank you for your interest in this project. For the purposes of this study, however, we are looking for active or prospective smartphone users to evaluate this app. You are free to go and enjoy the rest of your time here.”*

2. What types of information do you look up, either online or through phone calls, before you travel to a lighthouse?

Excellent. I will now show you “paper” versions of several screens from the app that I am developing. For each screen that I show you, I will ask you a few questions about it. Feel free to interact with this paper design as if it were an actual app on your smartphone. Also, feel free to ask questions at any time as I show you the different screens.

*Show the participant the “splash screen” for the application.*

3. What is going on with the app when this screen appears?
4. How long would you expect this screen to appear on your smartphone?

Let’s assume that the app has finished loading and is now displaying this screen. *Show the participant the “welcome” screen for the application.*

5. What can you do from this screen?
  - a. Anything else?
6. Which type of search would you rather complete (enter the name of a lighthouse manually, or find its name in a list)?

Say you begin entering the name of a lighthouse and press the “p” key on your keyboard. *Show the participant the “welcome – search in progress” screen for the application.*

7. What should happen when you select one of the options that appears below the letter “p”?

You have finished typing the word “pemaquid” on the welcome screen and have pressed the “Search” button. Now the following screen appears. *Show the participant the “search results” screen for the application. Ask questions 8 and 9.*

8. Can you identify the lighthouse that best matches your search term?
9. What should happen when you click the “View” button?

Let’s say you searched for West Quoddy Head Lighthouse in Lubec, ME and pressed the “View” button in the search results screen that I just showed you. The following screen then appears. *Show the participant the “general information” screen for the application. Ask questions 10 and 11.*

10. What screen should appear if you tap the arrow next to the lighthouse icon in the top-left corner?
11. There is more information about the lighthouse that does not appear on this screen right away. How can you see this extra information?

Now, let’s say I tapped the camera icon (*point to the icon on the screen*) to view photographs of the lighthouse. The following screen appears. *Show the participant the “photographs” screen for the application. Ask questions 12-14.*

12. What should happen when you select each of the arrows that you see near the main picture? *Point to the picture in the top-left corner of the screen.*
13. Should the map appear more zoomed in, more zoomed out, or is it fine as it is on this screen?
14. It is possible to upload a photograph from this screen?
  - a. If so, how would you expect to complete this process? If not, why not?

When I want to view the history of a lighthouse, a screen similar to the one I am showing you now appears. *Show the participant the “history” screen for the application. Ask question 15.*

15. When you select a dot on the timeline, a fact about the lighthouse appears in the bottom part of the screen. Sometimes, a dot represents more than one fact. When you select one of these dots, one of two events could occur. *Show the two “timeline pop-up” history screens (screens 8 and 9 in the “Screen Design” section).* Either a pop-up dialog appears where you can select a fact to display in the bottom part of the screen, or both facts appear in the bottom part of the screen. (You may have to scroll to see the entire second fact.) Which event makes more sense to you?

Returning to our West Quoddy Head Light example from before, let’s look at a screen that displays visitors’ reviews of the lighthouse. *Show the participant the “reviews” screen for the application. Ask questions 16 and 17.*

16. It is possible to upload a review from this screen. How would you expect to begin this process?
17. What should happen when you select the “Vote Helpful” or “Respond” icon?

This concludes the app walkthrough part of the study. I will now ask several questions about the app as a whole.

1. How likely would you be to download this app?
  - a. If so, why would you use the app? Describe a situation where you would use it.
  - b. If not, why not?
2. How user-friendly is this app?

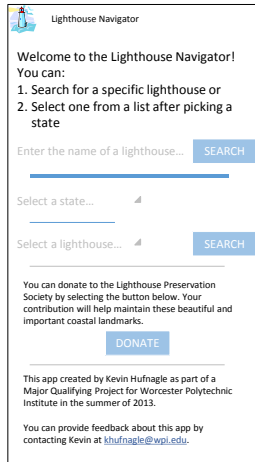
3. Do you have any additional questions for me?

Thank you again for your participation. You are free to go and enjoy the rest of your time here.

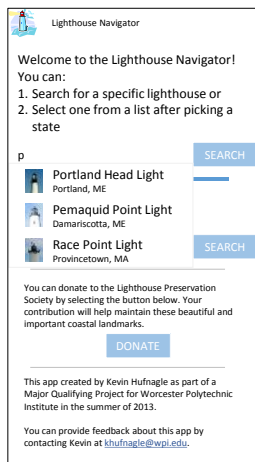
Paper Prototype Screens



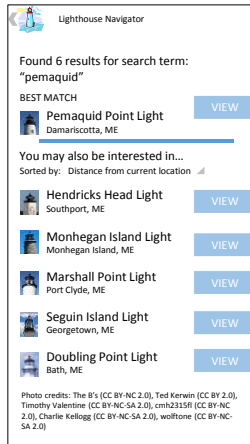
Splash Screen



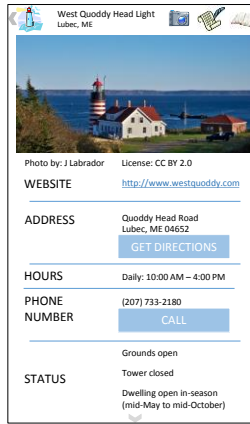
Welcome & Search



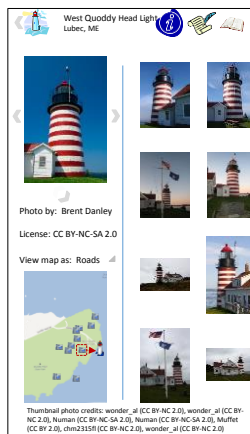
Search in Progress



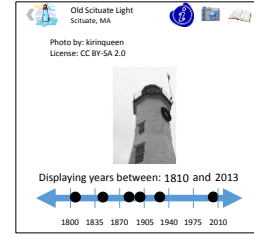
Search Results



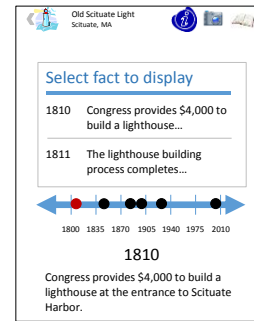
Information



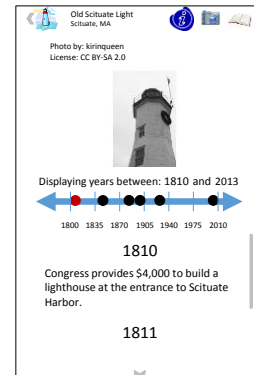
Photographs



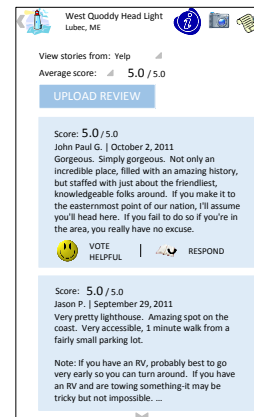
History



History – Pop-up



History – multiple facts



Reviews



## Response Sheet

Access to smartphone? Yes No Type \_\_\_\_\_ Time \_\_\_\_\_

Pieces of information examined before traveling to lighthouse:

*Splash Screen*

App behavior when screen appears:

Length of time that screen should appear:

Questions: Interaction:

*Welcome*

Things to do on screen:

Type of search to complete: Enter lighthouse name Find name in list

Behavior upon selecting auto-complete option:

Questions: Interaction:

*Search Results*

Identify “best match?” Yes No Action upon “View” select:

Questions: Interaction:

*Information*

“Back” screen: How to see extra info:

Questions: Interaction:

*Photographs*

Select arrows action: Map zoom: Zoom out Zoom in Just fine

Possible to upload photos? Yes No Action to complete/why not:

Questions: Interaction:

*History*

Interaction event: Pop-up dialog Both facts appear

Questions: Interaction:

*Reviews*

Process for uploading review:

Action upon selecting “Vote Helpful” or “Respond:”

Questions: Interaction:

Likelihood of downloading app:

Situation/why not:

User-friendliness:

Additional questions:

## Old Scituate Lighthouse – August 2, 2013

### Script

#### Pre-Survey Questions

We will begin the study with a couple of general questions about your smartphone use.

1. Do you use or have regular access to a smartphone?
  - a. If so, what type?
  - b. If so, how long have you used it?
  - c. If not, do you plan on gaining access to one within the next 6 months?

*If the answer to both (1) and (b) is “no,” the participant is disqualified from the study. I will say, “Thank you for your interest in this project. For the purposes of this study, however, we are looking for active or prospective smartphone users to evaluate this app. You are free to go and enjoy the rest of your time here.”*

2. How often do you visit lighthouses?
3. What types of information do you look up, either online or through phone calls, before you travel to a lighthouse?

#### Survey Questions

Excellent. I will now show you “paper” versions of several screens from the app that I am developing. For each screen that I show you, I will ask you a few questions about it. Feel free to interact with this paper design as if it were an actual app on your smartphone. Also, feel free to ask questions at any time as I show you the different screens.

*Show the participant the “splash screen” for the application. This screen appears for a few moments while the app loads information about local lighthouses. Now, let’s assume that the app has finished loading and is now displaying this screen. Show the participant the “welcome” screen for the application.*

4. Which type of search would you rather complete (find a nearby lighthouse, or search by state)? *Show the participant the “nearby search results” and “state-based search results” screens to show results from searching.*

Let’s say you searched for Old Scituate Lighthouse in Scituate, MA and pressed the “View” button in one of the search results screens that I just showed you. The following screen then appears. *Show the participant the “general information” screen for the application. Ask question 5.*

5. It is possible to see other pieces of information about this lighthouse, such as more photographs and historical facts. Android allows you to see these other screens either by selecting the title of the screen you are in and displaying a drop-down menu (*show the “general information drop-down” screen*) or by pressing the “menu” button and selecting an option from the bottom of the screen (*show the “general information menu” screen*). Which action would make more sense for you to take?

Now, let’s say I pressed the “Photographs” option. The following screen appears. *Show the participant the “photographs” screen for the application. Ask questions 6 and 7.*

6. What should happen when you select one of the icons in the top-right corner of the screen? *Point to the icons in the top-right corner of the “Photographs” activity screen.*
7. What should happen when you select each of the arrows that you see near the main picture? *Point to the picture in the top-left corner of the screen.*

When I want to view the history of a lighthouse, a screen similar to the one I am showing you now appears. *Show the participant the “history” screen for the application.*

Finally, let’s look at a screen that displays visitors’ reviews of the lighthouse. *Show the participant the “reviews” screen for the application. Ask questions 8 and 9.*

8. What should happen when you select the icon in the top-right corner of this screen? *Point to the icons in the top-right corner of the “History” activity screen.*
9. What should happen when you select the “Vote Helpful,” “Vote Unhelpful,” or “Respond” icons?

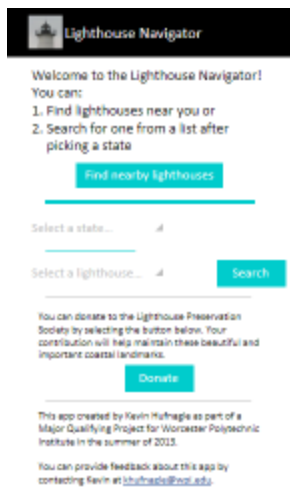
### *Post-Survey Questions*

This concludes the app walkthrough part of the study. I will now ask several questions about the app as a whole.

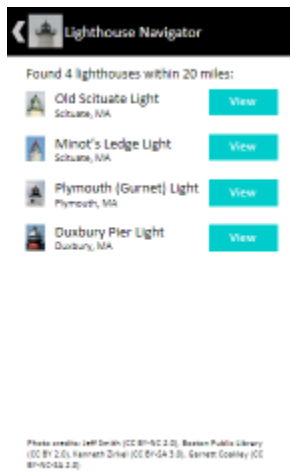
1. How likely would you be to download this app?
  - a. If so, why would you use the app? Describe a situation where you would use it.
  - b. If not, why not?
2. What further improvements to the app would you suggest? *If no response right away, ask about the icon design and color scheme.*

Thank you again for your participation. You are free to go and enjoy the rest of your time here.

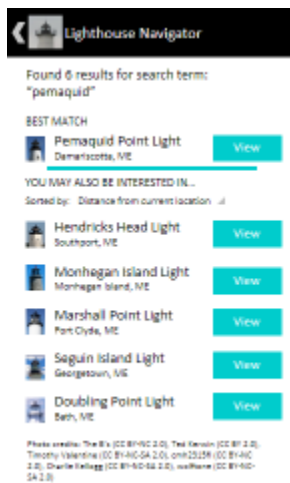
Paper Prototype Screens



Welcome Screen



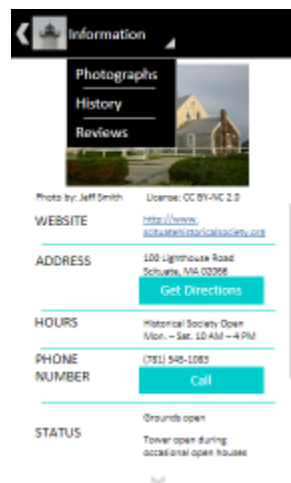
Nearby Search Results



State-Based Search Results



Information



Information (Drop-Down)



Information (Menu Button)



Photographs



History



Reviews



## Highland (“Cape Cod”) Lighthouse – August 31, 2013

### Script

#### Pre-Study Questions

We will begin the study with a couple of questions about your smartphone and computer use.

1. Do you own a computer (that isn’t a smartphone, such as a desktop, laptop, or tablet)?
2. If so, how many hours per day do you spend using it?
  - a. Less than 30 minutes
  - b. At least 30 minutes but less than 1 hour
  - c. At least 1 hour but less than 2 hours
  - d. At least 2 hours but less than 4 hours
  - e. At least 4 hours
3. Do you own a smartphone?

*If the answer to question 3 is “no,” the participant is disqualified from the study. I will say, “Thank you for your interest in this project. For the purposes of this study, however, we are looking for active smartphone users to evaluate this app. You are free to go and enjoy the rest of your time here.”*

4. What type of smartphone do you have?
5. How long (in months) have you used a smartphone that is the same type as your **current** smartphone (e.g. if you currently have a Samsung Galaxy S4, how long have you used Android smartphones in general)?
6. About how many minutes per day do you use your smartphone?
  - a. Less than 30 minutes
  - b. At least 30 minutes but less than 1 hour
  - c. At least 1 hour but less than 2 hours
  - d. At least 2 hours but less than 4 hours
  - e. At least 4 hours
7. What is your age, to the nearest decade (20s, 40s, 70s, etc.)?

Finally, I will ask a few questions about your typical lighthouse travel plans.

8. Did you plan your trip to this lighthouse?
  - a. If so, how many days in advance did you plan it?

9. Did you look up any information about the lighthouse before visiting today?
  - a. If so, what types of information did you look up? *There can be multiple answers.*
    - i. Location
    - ii. Hours
    - iii. Age or history
    - iv. Public vs. private
    - v. Other
  - b. If so, how many websites did you visit to find this information?
  - c. If so, how many phone calls did you make to find this information?
  - d. If so, did you find all the information you were looking for?
10. How many times have you visited a lighthouse so far this year?

### Study Tasks

Excellent. I will now present to you the app that I am developing on my Android smartphone. I will give you several tasks to complete within this app. I will watch as you interact with the app and try to complete each task. Please know that there is no one “right” way to complete some of the tasks that I will present you. Feel free to ask questions if you get lost or stuck while trying to complete a task.

*Open the app on my smartphone and navigate to the “Welcome” screen for the application.*

*Have the participant complete the first three tasks below.*

1. Determine the lighthouses in Maine that you can learn more about in this app.
2. Identify the lighthouse that is closest to your current location.
3. Navigate to a screen that shows general information about the lighthouse that is closest to your current location. *The lighthouse being emphasized here is notated as “this lighthouse” in future tasks.*

*If the participant cannot navigate to the “Information” screen successfully, open it for them and note on the response form that the application’s usability is poor within the “Welcome” and/or “Search Results” screen(s). Have the participant complete the next two tasks.*

4. Identify the height of this lighthouse.
5. Navigate to a screen that shows pictures other people have taken of this lighthouse.

*If the participant cannot navigate to the “Photographs” screen successfully, open it for them and note on the response form that the application’s usability is poor within the “Information” screen. Have the participant complete the next four tasks.*

6. Determine the number of photographs of this lighthouse that the app includes.
7. Try to upload a photograph of the lighthouse that you have taken on your smartphone. *I specify “try to upload” since the uploading functionality itself is still a work in progress.*
8. Show me how you would filter the photographs that appear on this screen (for example, by time of day or by the direction that the photographer is facing).
9. Navigate to a screen that shows historical facts about the lighthouse.

*If the participant cannot navigate to the “History” screen successfully, open it for them and note on the response form that the application’s usability is poor within the “Photographs” screen. Have the participant complete the next task. (There is no specific task on this screen since it is still a work in progress.)*

10. Navigate to a screen that shows visitors’ reviews for this lighthouse.

*If the participant cannot navigate to the “Reviews” screen successfully, open it for them and note on the response form that the application’s usability is poor within the “History” screen. Have the participant complete the final two tasks.*

11. Try to upload a review of the lighthouse. *I specify “try to upload” since the uploading functionality itself is still a work in progress.*

12. Return to the main menu of the app.

### *Post-Study Questions*

This concludes the app navigation part of the study. I will now ask several questions about the app as a whole.

1. How likely would you be to download this app?
  - a. Very likely
  - b. Likely
  - c. Unlikely
  - d. Very unlikely
2. *Ask if the participant answers (a) or (b) for question 1.* How often would you use the app?
  - a. Every week
  - b. Less than once per week but at least once per month
  - c. Less than once per month but at least once per year
  - d. Less than once per year
3. *Ask if the participant answers (a) or (b) for question 1.* In which of the following situations would you use it? *There could be multiple answers.*
  - a. While in the middle of a road trip (at most one day in any one location)
  - b. While on vacation in a specific location (at least two days in a specific location)
  - c. As inspiration for art (photography, painting, etc.)
  - d. Other
4. *Ask if the participant answers (c) or (d) for question 1.* Why would you not download this app?
5. What further improvements to the app would you suggest?

Thank you again for your participation. You are free to go and enjoy the rest of your time here.



## Response Form

*Demographics*

Own computer? Y N      Time spent \_\_\_\_\_      Age \_\_\_\_\_      Gender M F  
 Own smartphone? Y N      Type \_\_\_\_\_      Time owned (mos) \_\_\_\_\_      Time spent \_\_\_\_\_  
 Plan trip? Y N      Info. looked up \_\_\_\_\_      # Websites \_\_\_\_\_      # Calls \_\_\_\_\_      # Trips \_\_\_\_\_

<i>Welcome and Search Results</i>	Completed	Completed with help	Not completed
Determine lighthouses in Maine	4	3	2
Identify nearby lighthouses	4	3	2
Navigate to “Information” screen	4	3	2

Interaction notes:

*Information*

Identify lighthouse height	4	3	2	1
Navigate to “Photographs” screen	4	3	2	1

Interaction notes:

*Photographs*

Determine number of photographs	4	3	2	1
Upload photograph	4	3	2	1
Filter photographs	4	3	2	1
Navigate to “History” screen	4	3	2	1

Interaction notes:

*History*

Navigate to “Reviews” screen	4	3	2	1
------------------------------	---	---	---	---

Interaction notes:

*Reviews*

Upload review	4	3	2	1
Return to main menu	4	3	2	1

Interaction notes:

Likelihood of downloading app \_\_\_\_\_      How often will app be used \_\_\_\_\_  
 Situations for using app \_\_\_\_\_      Why unlikely to download app \_\_\_\_\_  
 Further improvements:

## Nauset Lighthouse – September 8, 2013

### Script

#### Pre-Study Questions

We will begin the study with a couple of questions about your smartphone and computer use.

1. Do you own a computer (that isn't a smartphone, such as a desktop, laptop, or tablet)?
2. If so, how many hours per day do you spend using it?
  - a. Less than 30 minutes
  - b. At least 30 minutes but less than 1 hour
  - c. At least 1 hour but less than 2 hours
  - d. At least 2 hours but less than 4 hours
  - e. At least 4 hours
3. Do you own a smartphone?

*If the answer to question 3 is “no,” the participant is disqualified from the study. I will say, “Thank you for your interest in this project. For the purposes of this study, however, we are looking for active smartphone users to evaluate this app. You are free to go and enjoy the rest of your time here.”*

4. What type of smartphone do you have?
5. How long (in months) have you used a smartphone that is the same type as your **current** smartphone (e.g. if you currently have a Samsung Galaxy S4, how long have you used Android smartphones in general)?
6. About how many minutes per day do you use your smartphone?
  - a. Less than 30 minutes
  - b. At least 30 minutes but less than 1 hour
  - c. At least 1 hour but less than 2 hours
  - d. At least 2 hours but less than 4 hours
  - e. At least 4 hours
7. What is your age, to the nearest decade (20s, 40s, 70s, etc.)?

Finally, I will ask a few questions about your typical lighthouse travel plans.

8. Did you plan your trip to this lighthouse?
  - a. If so, how many days in advance did you plan it?

9. Did you look up any information about the lighthouse before visiting today?
  - e. If so, what types of information did you look up? *There can be multiple answers.*
    - i. Location
    - ii. Hours
    - iii. Age or history
    - iv. Public vs. private
    - v. Other
  - f. If so, how many websites did you visit to find this information?
  - g. If so, how many phone calls did you make to find this information?
  - h. If so, did you find all the information you were looking for?
10. Including this trip, how many times have you visited a lighthouse so far this year?

### Study Tasks

Excellent. I will now present to you the app that I am developing on my Android smartphone. I will give you several tasks to complete within this app. I will watch as you interact with the app and try to complete each task. Please know that there is no one “right” way to complete some of the tasks that I will present you. Feel free to ask questions if you get lost or stuck while trying to complete a task.

*Open the app on my smartphone and navigate to the “Welcome” screen for the application.*

*Have the participant complete the first two tasks below.*

1. Determine the lighthouses in Maine that you can learn more about in this app.
2. Navigate to a screen that shows general information about the lighthouse that is closest to your current location. *The lighthouse being emphasized here is notated as “this lighthouse” in future tasks.*

*If the participant cannot navigate to the “Information” screen successfully, open it for them and note on the response form that the application’s usability is poor within the “Welcome” and/or “Search Results” screen(s). Have the participant complete the next task.*

3. Navigate to a screen that shows pictures other people have taken of this lighthouse.

*If the participant cannot navigate to the “Photographs” screen successfully, open it for them and note on the response form that the application’s usability is poor within the “Information” screen. Have the participant complete the next four tasks, along with two questions.*

4. Determine the number of photographs of this lighthouse that the app includes.
5. Try to upload a photograph of the lighthouse that you have taken on your smartphone. *I specify “try to upload” since the uploading functionality itself is still a work in progress.*
6. Show me how you would filter the photographs that appear on the screen so you can see certain types of photographs that people have taken of this lighthouse.
7. What types of filters would you find useful within this application?

8. One type of filter would entail placing camera icons on the map in the bottom-left corner of this screen. Were you to select these camera icons, only photographs from that general area around the lighthouse would appear along the right-hand side of the screen. *I will be illustrating this idea through gestures so that the following question makes sense to visitors.* On a scale of 1 to 4 (with 1 being “not at all useful” and 4 being “very useful”), how useful would you find these camera icons?
9. Navigate to a screen that shows historical facts about the lighthouse.

*If the participant cannot navigate to the “History” screen successfully, open it for them and note on the response form that the application’s usability is poor within the “Photographs” screen. Have the participant complete the next task. (There is no specific task on this screen since it is still a work in progress.)*

10. Navigate to a screen that shows visitors’ reviews for this lighthouse.

*If the participant cannot navigate to the “Reviews” screen successfully, open it for them and note on the response form that the application’s usability is poor within the “History” screen. Have the participant complete the final two tasks.*

11. Try to upload a review of the lighthouse. *I specify “try to upload” since the uploading functionality itself is still a work in progress.*
12. Return to the main menu of the app.

### *Post-Study Questions*

This concludes the app navigation part of the study. I will now ask several questions about the app as a whole.

1. How likely would you be to download this app?
  - a. Very likely
  - b. Likely
  - c. Unlikely
  - d. Very unlikely
2. *Ask if the participant answers (a) or (b) for question 1.* How often would you use this application while planning a trip to a lighthouse or traveling to one?
  - a. Almost always
  - b. Often
  - c. Not often
  - d. Almost never

3. *Ask if the participant answers (a) or (b) for question 1. In which of the following situations would you use it? There could be multiple answers.*
  - a. While in the middle of a road trip (at most one day in any one location)
  - b. While on vacation in a specific location (at least two days in that specific location)
  - c. As inspiration for art (photography, painting, etc.)
  - d. Other
4. *Ask if the participant answers (c) or (d) for question 1. Why would you not download this app?*
5. What further improvements to the app would you suggest?

Thank you again for your participation. You are free to go and enjoy the rest of your time here.

## Response Form

*Demographics*

Own computer? Y N      Time spent \_\_\_\_\_      Age \_\_\_\_\_      Gender M F  
 Own smartphone? Y N      Type \_\_\_\_\_      Time owned (mos) \_\_\_\_\_      Time spent \_\_\_\_\_  
 Plan trip? Y N      Info. looked up \_\_\_\_\_      # Websites \_\_\_\_\_      # Calls \_\_\_\_\_      # Trips \_\_\_\_\_  
 Found all information looking for? Y N

<i>Welcome and Search Results</i>	Completed	Completed with help	Not completed	
Determine lighthouses in Maine	4	3	2	1
Navigate to “Information” screen	4	3	2	1

Interaction notes:

*Information*

Identify lighthouse height	4	3	2	1
Navigate to “Photographs” screen	4	3	2	1

Interaction notes:

*Photographs*

Determine number of photographs	4	3	2	1
Upload photograph	4	3	2	1
Filter photographs	4	3	2	1
Navigate to “History” screen	4	3	2	1

Types of Useful Filters:

Usefulness of camera icons on map:

Interaction notes:

*History*

Navigate to “Reviews” screen	4	3	2	1
------------------------------	---	---	---	---

Interaction notes:

*Reviews*

Upload review	4	3	2	1
Return to main menu	4	3	2	1

Interaction notes:

Likelihood of downloading app \_\_\_\_\_      Freq. of using app during lighthouse travel \_\_\_\_\_

Situations for using app \_\_\_\_\_      Why unlikely to download app \_\_\_\_\_

Further improvements: