2014

A Creative Exploration Through a Mobile Application Interface

Julia Canning Carter
Bard College

A Creative Exploration of Sensors Through a Mobile Application Interface

A Senior Project submitted to
The Division of Science, Mathematics, and Computing
of
Bard College

by
Julia Carter

Annandale-on-Hudson, New York
May, 2014

Abstract

My senior project is a mobile application for iPhones and iPads. The application explores the devices' sensors and displays information visually at the level of the user interface to encourage the user to contemplate the role that their smartphone has in their life and consider how they interact with these small rectangular devices that they carry with them everywhere they go. The application uses touch gestures to allow the user direct manipulation of objects, accelerometer data to display unintuitive gravitational effects on the screen, and a background of a camera display to add an element of augmented reality to the program. These three types of user input work together to present a dynamically changing image to the viewer. The project also explores standardized interface features within iOS and their importance as mobile computing becomes an inescapable aspect of modern life for many people.

Contents

Abstract **1**

Acknowledgments **5**

1 Introduction **6**

2 Related Work **7**

3 Mobile Computing **10**

4 Software and Tools **13**

5 Project Goals **15**

6 Touch Gestures **19**
6.1 The Role of Touch Screens 19
6.2 Apple Standards 20
6.3 Implementation 22
6.4 Tools 24

7 Accelerometer **26**
7.1 Accelerometer Introduction 26
7.2 Apple Standards 27
7.3 Implementation 28
7.4 Tools 29

8 Camera **31**
8.1 Camera Background 31

8.2 Apple Standards 31
8.3 Implementation 32
8.4 Tools 33

9 Conclusion **36**
9.1 Future Modifications and Additions 36
9.2 Success of Project 37

10 Appendix **38**
10.1 ViewController.m 38
10.2 OrbView.m 39

Bibliography **45**

List of Figures

5.0.1 A diagram for the structure of the application. 17
5.0.2 An example instance of the application. 17
5.0.3 A screenshot from the application. 18

7.1.1 Diagram of accelerometer axes of the iPhone[10]. 27

8.3.1 A screenshot from the application. 35

Acknowledgments

I would like to thank my adviser Rebecca Thomas for her support and guidance throughout my senior project. I would also like to thank the entire Computer Science department for their encouragement.

1
Introduction

My senior project is an abstract iPhone and iPad application that uses the devices' sensors at the level of the user interface to encourage the user to contemplate the role that their smartphone has in their life and consider how they interact with these small rectangular devices that they carry with them everywhere they go. I chose to do this project because I have noticed that the degree to which people use and rely on their smartphones has become overwhelming for many Americans. I think that the mobile phone phenomenon is particularly interesting because the majority of users only understand their phones at surface level via the user interface. Most available smartphone applications serve the purpose of providing the user access to another preexisting form of technology or media. For example, people can use phones to watch television, read books or newspapers, or access tools like calculators and maps. I thought it would be interesting to make an application whose main focus was the idea of 'the app.' This caused my project to take a creative turn in conjunction with computer science.

2
Related Work

After I came up with my idea and began researching related work, I came across a number of relevant projects done by other people. For example, the artist Rafael Rozendaal is a "visual artist who uses the internet as his canvas." [19] He publishes some of his art in the form of websites, so each piece has its own URL. When a user visits one of his websites, they see a scene he has put together and can often interact with the page via mouse movement and clicks. I liked the concept of having the user contribute to the art in such a way that each time the page loaded, active participation would affect how the page performed. I tried to incorporate the notion of user participation in my own app, but I used some of the additional features of computing that are unique to mobile devices.

In addition to this particular web artist, I noticed a trend of musicians releasing increasingly interactive music videos this year. For example, the band Arcade Fire released an interactive music video for their song "Reflektor" in September of last year [2] [6]. The video integrates mouse movement, mouse clicks, and even the computer's webcam at certain points throughout the song. Similarly, in late November, musician Bob Dylan

rereleased his famous song "Like a Rolling Stone" with an interactive video allowing the user to change the channel of a digital television within their web browser [7] [8]. He offered multiple channels with different scenes of well-known people and characters lip-synching the classic song. This too enabled the viewer to have personal control over their experience. They could choose which version of the song they wanted to watch and change their mind on a whim with a single click. They could watch through it dozens of times, each time having a slightly different experience than before.

In February 2014, the band Radiohead released a mobile application called "PolyFauna" with the design company Universal Everything [18]. Upon its release, the band described the application as coming "from an interest in early computer-life experiments and the imagined creatures of our subconscious. Your screen is the window into an evolving world. Move around to look around." [17] In the PolyFauna application, the user can drag their finger on the screen, which spawns features of an environment. If the phone physically rotates, the environment on the screen rotates along with the device, allowing the user to explore. The imagery is abstract, yet reminiscent of nature, and the background music is from one of the band's songs.

Radiohead's application was much more similar to my senior project than the previous examples because it uses the same platform, the mobile device, and it includes user-sensor interaction. The developers utilize both touch gestures and the device's gyroscope to change direction within the app. Radiohead's application was released months after I began my senior project, and there are certainly some similarities in intention. However, my application explores different sensors in different ways. It could have been discouraging that a world-famous band released a professionally developed, abstract, artistic mobile application at the same time I was trying to create my own, but instead I decided that

their application could help contribute to validation of my ideas about mobile applications becoming a viable form of art.

3
Mobile Computing

Jon Agar, a professor at University College London, discusses the impact of mobile phones and smartphones in his book **Constant Touch**. The first part of his book gives a history of mobile phones and their importance, and his newest edition expands on that topic with the addition of smartphones. He makes the distinction between the two in that a smartphone is essentially a computer in addition to a phone. Previously mobile phones had expanded human means of communication, but when the phone also gained computing ability, its power grew. Agar has a model of rings of personal technologies [1]. The outer ring includes items that are personally owned but largely immobile, such as desktop computers. The next ring contains laptop computers, which are designed to be moved around, but are still a burden and require effort to transport. Agar calls the third ring, the ring that encompasses smartphones, that of "intimate" technologies. He writes, "These are portable but are carried without exertion. They are kept close to the body. They are so useful or important or engaging that we don't register their weight. Very few technologies make it to the inner ring," such as clothes, shoes, glasses, and now smartphones [1].

Smartphones are essentially general-purpose computers that can fit inside your pocket. Their portability contributes to their popularity, and in modernized countries their presence is highly visible. They have many functions. Agar elaborates:

> The things I use my smartphone for are just as diverse, perhaps more so, as they were in the case of my desktop computer. I check my mail, send texts, update a Facebook status and skim through tweets. I browse, using Safari, websites that carry everything from national news and sport to London ornithology. I share my pictures on Flickr, listen to music on the iPod, and play Angry Birds, Blitz and Welder. Under dark clear skies I can hold the iPhone up and read the stars using Starmap Pro. When bored I flick left and right through my apps to find something to do. I even, occasionally, use it to make a phone call [1].

Mobile computing is interesting because of its increasing prevalence in daily life. Many people spend many hours of their day touching their phones. People often rely on their smartphones for critical information. Smartphones have replaced maps, dictionaries, address books, and endless other products. Furthermore, every year mobile devices continue to become more powerful and have more capabilities. Competition between designers, manufacturers, and distributors, particularly Apple and Android, helps contribute to the growing abilities of smartphones.

I chose to design my senior project application for iOS products for a few different reasons. The first is that I happen to own an iPhone, so I am much more familiar with its interface than I am with Android. This also enabled me to test my application on my own hardware without having to acquire additional devices. Another reason is that although Apple did not invent the smartphone, the iPhone was the first widely used smartphone and it helped define many of the features that all smartphone users have come to expect [1]. Furthermore, as Agar discusses, Apple has a large degree of control over the quality and content of its products. I thought that it would be interesting to look at Apple's standards for development because people who use iOS products often develop certain expectations of their devices. Certain gestures take on very specific meanings, which become second

nature to their users. People who use Apple products accept the restrictions that Apple imposes "in exchange for the implied guarantee of quality." [1] Apple has earned a loyal group of consumers because of their emphasis on sleek design and consistent, user-friendly interfaces. Apple users are guaranteed a device that works well in ways they learn to expect. Google's Android, the main competitor of Apple's iOS, exists on the other end of this spectrum between freedom and consistency within a smartphone.

The Android operating system uses a kernel based on Linux, which is completely open source. Agar writes, "Google's bet was that Android's open nature would make it nimble, quicker to develop new applications for, and appealing to the broader community of coders."[1] Google's bet paid off, and Android phones are actually more numerous than iPhones today. A major difference between the brands is that Android users can completely customize their phone's features to suit their specific needs, if they desire to do so. Android also offers different options for hardware, unlike iOS. The diversity, flexibility, and freedom that Android offers are highly valuable features for a mobile operating system. However, I wanted to be able to reflect on concrete design standards with my project, so iOS was a more fitting choice for my purposes.

4
Software and Tools

In terms of development, iOS and Android both provide many tools and libraries for programmers to use. As might be expected, given the goals of each company, Apple has more restrictions for its development environment. For creating iOS applications, the programmer must develop with Apple software called Xcode. In the last couple of months I have started to program Android applications and I find their developer tools to be much more user-friendly than Apple's. Additionally, the familiarity of the programming language Java and the open source nature of Android make it easier to learn. In retrospect, it is possible that I would have been able to incorporate more features into my application if I had programmed in Android because my largest obstacle for this project was learning how to program for iOS.

Learning to program for Apple mobile devices involved learning a new programming language, Objective-C, and becoming familiar with a new integrated development environment, Xcode. Furthermore, over the duration of my senior project Apple released a major software update for iOS, and Xcode correspondingly underwent various updates.

These updates did not affect the functionality of my program, but they did add an adjustment period for me to get used to the new programming environment. Likewise, many of the tutorials I used to teach myself how to make iPhone applications had been published a few years ago, and Xcode had changed significantly in the time since then, so tutorials were often hard to follow.

The most notable obstacle I had was with Xcode's interface builder. The intention of the interface builder is to allow the programmer the ability to design the visual look of their application by dragging and dropping elements onto a layout. The idea of the interface builder is a good one but I personally found it difficult to navigate between the interface builder and other code, particularly because this was the feature that seemed to have evolved the most between different versions of Xcode. I had a hard time finding tutorials that created interfaces via hand coding, rather than via the interface builder, so this was probably the biggest practical challenge of my project. To overcome the problems I had with Xcode's interface builder, I eventually decided to avoid using it altogether.

Again in retrospect, the Eclipse IDE for Android development also offers an interface builder, and I find their version much more intuitive than Apple's. For Android, the programmer can graphically design the layout but they also have access to a corresponding XML file so that they can finesse the visual layout by modifying the code and set up links between these interface graphics and the main Java program. Additionally, the way to 'inflate' the Java code with this layout was much more straightforward than my experience with Xcode.

5
Project Goals

The main goal of my project was to have an iPhone application that allowed the user to interact with it for the sake of exploration, using as many of the device's sensors as possible. Creatively, I thought it would be interesting to offer users a program that differed from the vast majority of other mobile applications because it did not offer a functional purpose, or have any defining features of a game. I thought it might be meaningful to have people try to experience an app that made them realize the nature of their interaction with their cell phone, and remind them of the power of the device through its sensors. I think that the numerous sensors mobile devices possess give them a lot of potential to be a type of robot that we carry around with us everywhere we go. I think it is fascinating that in under than a decade (the first iPhone came out in 2007) human behavior has evolved along with smartphones. Therefore it could be beneficial for a person to spend a few minutes with my application and possibly consider how odd it is that they spend a significant amount of time running their hands over a small piece of glass, angling the device at different degrees, or looking into a camera. I am interested in exploring the ways in which

people interact with their devices and with the outside world through their mobile devices.

My original idea for my application was to have two or three different types of environments that the user could reach by performing certain gestures. Each of these different environments would then allow the user to interact with different sensors of the device, and visually see results of their actions. However, I had a harder time learning to program for iOS than I had anticipated, so rather than having different 'environments' for each type of sensor, I combine the effects of three sensors into one environment. My application uses touch gestures, the accelerometer, and the camera to produce visual output for the user in hopes of stimulating their thought process about mobile devices.

The resulting program has three main classes (Figure 5.0.1). At a high level of abstraction (Figure 5.0.2), the ViewController class controls what happens when the application runs. This class also manages the output of the device's camera. It overlays the camera view with an instance of a custom class, OrbView. The OrbView class handles touch gestures and the accelerometer. The OrbView class utilizes another custom class, called Orb. The OrbView can spawn new Orbs and it handles their movement via touch and gravity. See Figure 5.0.3 for an example screenshot from the application with a few active orbs. Because the application makes constant use of the device's sensors, a screenshot is a limited representation, but helpful nonetheless.

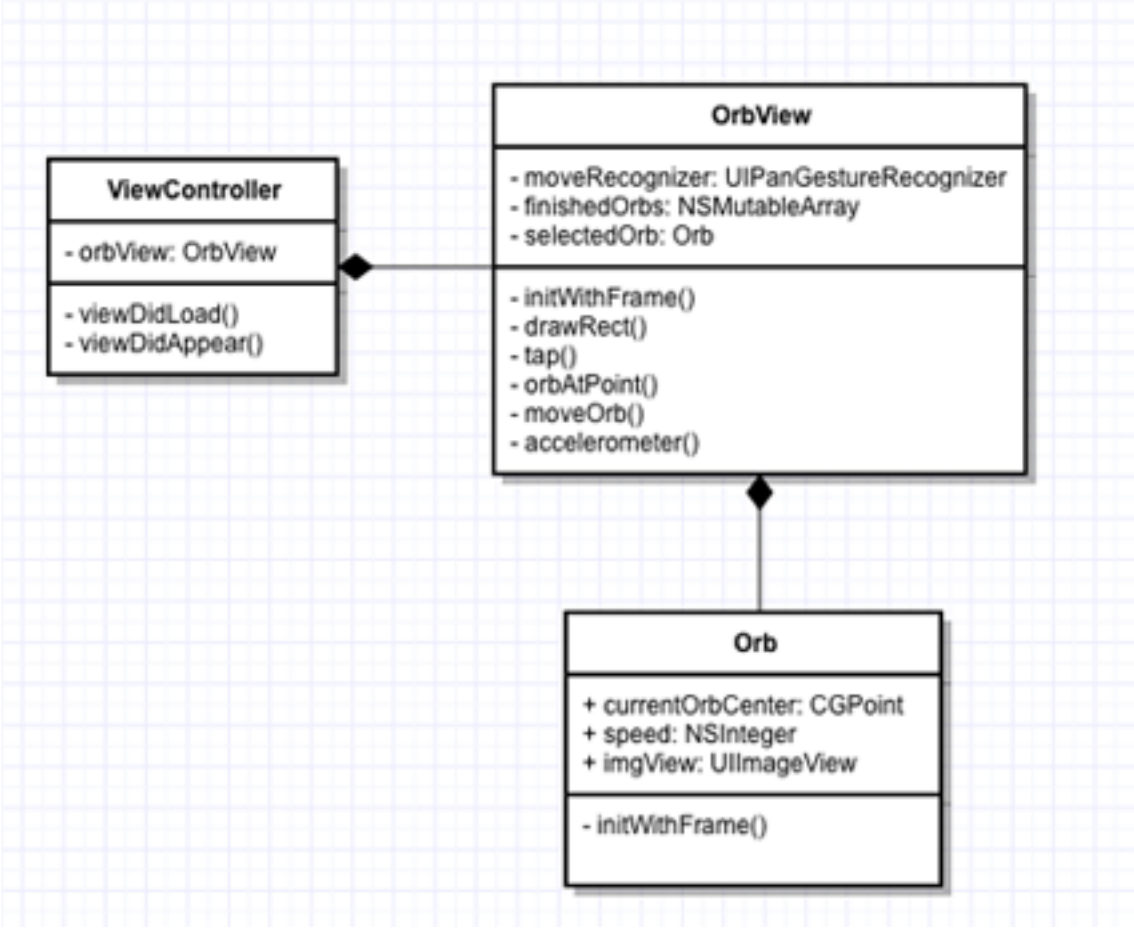

Figure 5.0.1. A diagram for the structure of the application.

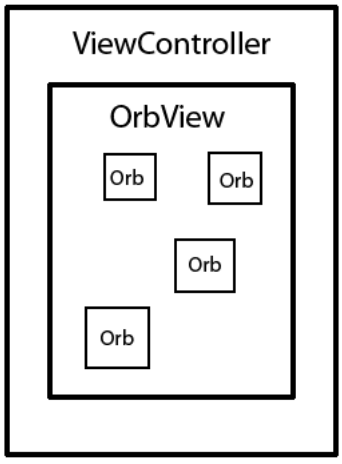

Figure 5.0.2. An example instance of the application.

Figure 5.0.3. A screenshot from the application.

6
Touch Gestures

6.1 The Role of Touch Screens

The iPhone was the first major commercial success for a smartphone with a touchscreen [4]. In 2005, Apple bought the company FingerWorks, which had skills, knowledge, and patents for powerful touchscreen technology, which Apple would use in 2007 with the first iPhone. Jon Agar provides an explanation of the iPhone's touchscreen technology:

> The phone is sensitive to live fingers and not inert objects. All this cleverness is possible because the iPhone screen has several smart layers. On the outside is tough glass. Sitting on the glass is a nearly invisible grid of fine electrical wires. The lines are about a millimeter apart. One line carries an electrical charge, while the other detects the slight disturbance caused in the electrical fields, as your finger, acting like a weak capacitor, swipes the screen. By arranging the lines in a grid the position of your finger, and whether it's moving in a particular direction, can be measured [1].

The computer inside the smartphone then interprets finger movements and responds accordingly. Touchscreens that respond quickly and accurately to the input of a user's fingers have changed the way that many people interact with technology on an everyday basis.

I recently traveled outside of the country and certain small aspects of my trip helped me realize how omnipresent touchscreens are becoming. At the John F. Kennedy International

Airport, you can now order your food via a touchscreen. In fact, it was the only way to order food in my outgoing terminal. Foreign passengers who may not speak English need only tap on pictures of what they would like to buy, and a chef makes it for them. The touchscreen menu did not need to provide any directions for use; people used it correctly according to their intuition.

On the other hand, my flight had in-flight entertainment options for each passenger in front of their seat. When I first tried to access the entertainment, my instinct was to touch the screen. It did not respond, but I touched it again anyway because there were no other visible controls. The second time, the screen did respond, but when I tried to continue on to the next screen, my taps went unrecognized again. After a frustrating few minutes, I realized there was also a remote control in my armrest, so I did not have to suffer through the plane's sloppy touchscreen technology. Later in my flight, the person seated next to me (who had been asleep during my entertainment struggles) woke up and I watched her go through the same steps as I had earlier. I gave her the clue about the hidden remote control, but she decided to keep struggling along with the touchscreen. A similar occurrence happened on my flight home. My personal observations combined with current technological trends imply that touchscreens are becoming so omnipresent that people automatically assume they are going to work in a way they are accustomed to.

6.2 Apple Standards

The touchscreen is the most common and direct way through which individuals interact with their iPhones. The iOS Developer Library contains documentation for iOS Human Interface Guidelines. The section entitled "Interactivity and Feedback" emphasizes the importance of gestures for the user: "Using gestures gives people a close personal connec-

tion to their devices and enhances their sense of direct manipulation of onscreen objects. People generally expect gestures to work the same in all the apps they use."[11] The official documentation lists eight distinct types of gestures: tap, drag, flick, swipe, double tap, pinch, touch and hold, and shake. Each of these gestures has a specific meaning within iOS. For example, people always use a single tap for selection, whereas they expect a double tap to zoom in or out, depending on the current configuration of the screen.

In addition to outlining the standard gestures and their meanings, Apple also provides developers with several things to avoid. They advise that the programmer should not associate standard gestures with nonstandard meanings. Intuitively, this makes sense. It would be very frustrating for the same action to cause different results under similar circumstances. Likewise, Apple suggests that developers should not create custom gestures to perform actions that are already covered by the standard gestures within iOS. All of these guidelines exist to enhance the user's experience. Users grow so accustomed to the meanings of gestures that they often perform them automatically, without considering what they are doing. Panning the screen of a smartphone to scroll through lists becomes second nature, quite like turning the page of a book from the edge in order to read.

An evident goal of standardizing the meaning of particular gestures is that users can develop intuition for using their technology. Apple writes, "It's best to use standard gestures because people don't have to make an effort to discover them or remember them." This follows with the thinking of Donald Norman in his book, **The Design of Everyday Things**. Norman, much like Apple, emphasizes the importance of user-centered design. He writes, "The human mind is exquisitely tailored to make sense of the world. Give it the slightest clue and off it goes, providing explanation, rationalization, understanding."[16] People who use iPhones and iPads generally have a level of familiarity with their products;

they know how to achieve desired effects because the gestures to do so are similar between almost all applications. If a person spends any nontrivial amount of time using an iOS device, they will likely develop an intuitive understanding of the effects of their touch gestures within the system. Apple emphasizes the importance of using gestures in a standard way because their consumers have already formed associations between gestures and actions. Relative chaos would ensue if developers used a swipe gesture to zoom, or a pinch to undo.

Most of Apple's touch gestures can be mapped logically to their effects. For example, tapping a button on the screen mimics tapping a button in the real world. Something new happens: some sort of confirmation, noise, or transition. The user also expects and receives feedback for their interactions with a smartphone, assuming the application is designed to Apple's standards. Touchscreen gestures have a strong link to specific meanings within iOS. Touch is the most obvious sensor that modern smartphones possess, and it grants the user access to the vast majority of the phone's functionality. The notion that touch gestures give users a "close personal connection" to their devices is linked to the constant visual feedback users receive from moving their fingers in certain ways on their cell phones. In the final version of my senior project application, I use two of Apple's distinct touch gestures: taps and drags.

6.3 Implementation

In general, I decided to follow Apple's guidelines for touch gestures within my senior project. In my project, when the user taps the screen, an image (that I will henceforth refer to as an "orb") appears on the screen at the location of the tap. The orb then moves around according to the accelerometer, which will be discussed later. The user can also move the orb around the screen by placing a finger on the screen and dragging it around.

The orb will follow along with the finger until it is released. Additional single taps spawn more orbs, each of which can also be dragged around at the user's will.

The most obvious way that my application differs from Apple's standards is the initial tap to spawn the first orb. In general, iOS users tap preexisting objects for feedback. However, in my app the user must tap empty space in order to make the object appear. In informal test trials for my application, people tend to be initially confused by a 'blank' camera view (the camera will also be discussed later) but after a few seconds of confusion (and perhaps curiosity), their next instinct is to tap the screen. Often, after realizing that their tap created a round object that floats around the screen, their next instinct was to try to grab the orb.

Earlier prototypes of my project included some nonstandard touch gestures. I included swipes that changed the background and double taps to modify transparency, for example. However at that time, I had not implemented the ability to drag the orbs. For my midway presentation, I showed the group a version of my application that had implemented the accelerometer to move the orbs, but did not allow the orbs to be dragged via touch gestures. Every single person who tried using my app also tried to drag the orbs. I did not gather formal results of their opinions, but I observed what I interpreted as boredom and an immediate loss of interest upon realizing they could not 'interact' with the program, even though there were still other ways that they could have interacted with the app. It seemed to me that the link between specific touch gestures and expected behavior was critical and, consequently, too risky to omit from my app.

Therefore, I decided that if I were going to include touch gestures, I would need to conform to most of Apple's standards. The gesture of dragging an object on a screen is

much more widespread than just within iOS, and this performance feature adds a sense of comfort and understanding to my application even for people who are not familiar with iPhones. My orbs respond to drag gestures as the user expects, by following the user's finger.

6.4 Tools

I had a relatively simple time implementing tap gestures to spawn orb objects. Apple provides gesture recognizer objects, called UIGestureRecognizers, for such tasks, and their documentation was straightforward [13]. It was more of a challenge to get the image file to attach to the abstract orb object, but I was able to do so after some practice with custom image views in iOS.

Implementing dragging and dropping for the orbs was a more complicated process for me, as a beginner iPhone programmer. The textbook **iOS Programming: The Big Nerd Ranch Guide** helped me countless times in my process of learning to program iPhones. In Chapter 13, the book provides a tutorial for creating line objects of user-defined lengths that the user can then drag around, resize, and delete [5]. I followed this tutorial to learn the basics of gestures that are more complex (such as dragging). However, I had to modify the code significantly to suit my needs. For example, the book used line objects, which are predefined in an iOS library. I designed my own 'orb' objects and modified the book's drag and drop technique so that my program could search through an array of orbs that had already been spawned to check if the user was trying to drag any of them. The general process for my implementation of drag and drop is as follows: upon the detection of a drag gesture on the screen, check a square area around the point of the gesture to see if an orb is nearby. If so, while the gesture continues, keep track of the finger

location and modify the center of the orb accordingly, and then remove the old image of the orb and redraw. Later, when I added the accelerometer features, I continued to modify drag and drop so that the accelerometer would not affect an orb for the duration of a drag.

7
Accelerometer

7.1 Accelerometer Introduction

According to the iOS Developer Library,

> The accelerometer is actually made up of three accelerometers, one for each axis: x, y, and z. Each one measures changes in velocity over time along a linear path. Combining all three accelerometers lets you detect device movement in any direction and get the device's current orientation. Although there are three accelerometers, the remainder of this document refers to them as a single entity [10].

The x-axis of the accelerometer for Apple products traverses left-to-right, if looking at the screen of the device. In the same position, the accelerometer's y-axis is the vertical axis from the top of the phone to the bottom. The z-axis runs perpendicular to the surface of the phone. See Figure 7.1.1 for a diagram of the axes. Over the last few years, many popular iPhone applications have used the accelerometer in different ways. For example, in the early days of iPhones, many users would download apps that simulated drinking beer or milk. As the user tried to 'drink' from the device, the cup on the screen would lose liquid, or if they shook the phone, foam would appear on the beverage. Many games also use the accelerometer to add tilting features for a different type of challenge.

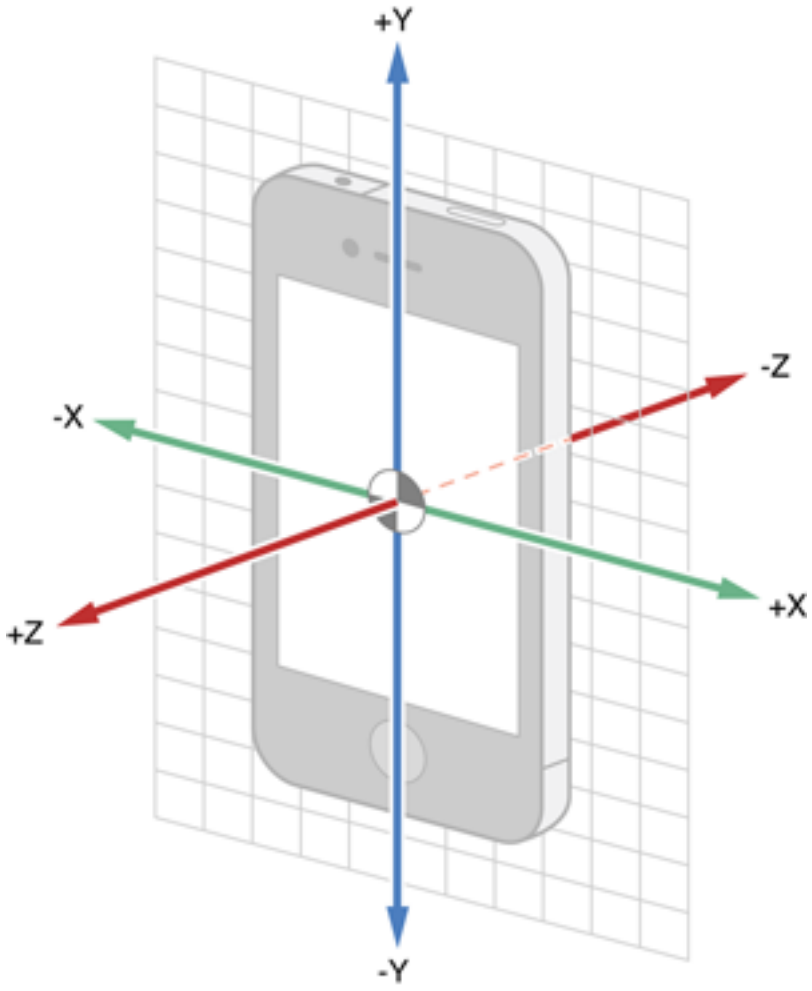

Figure 7.1.1. Diagram of accelerometer axes of the iPhone[10].

7.2 Apple Standards

Apple has far fewer guidelines and restrictions for use of the accelerometer than it does for touch gestures. The primary concerns regarding the accelerometer within iOS exist at a lower level: handling of the orientation of the device. The iPhone application developer rarely needs to check for device orientation via the accelerometer; the interface builder within XCode offers a variety of ways to handle different device orientations. If the app developer still needs more specific data, Apple then suggests that they use the "UIDevice" class for situations in which only general orientation details are required. iOS also uses the

device's accelerometer at lower level to detect Shake-Motion events. In iOS, shake-motions can signify "undo," so it is often important for the device to respond to shakes in a timely manner. Apple provides "The Core Motion Framework" to aid the developer in accessing raw accelerometer data and gyroscope data.

7.3 Implementation

Like my implementation of touch gestures, my implementation of the iPhone's accelerometer went through different stages. At my midway presentation, the accelerometer was the only sensor that controlled the motion of the orbs. However, even after adding the ability to drag the orbs around, I still wanted to use the accelerometer to move the orbs.

My first implementation used the accelerometer data from the x and y-directions to directly affect the x and y-positions of the orbs on the screen. I arbitrarily set a maximum number of orbs to be spawned (6), and programmed each orb to have a random speed. I also designed the program so that every other orb that generates would have a negative y-acceleration, so that at any given time, about half of the orbs would be headed in an opposite, and counterintuitive, direction compared to the others. Except for the fact that half the orbs were going "up," in this implementation, the orbs seemed to be following the rules of gravity. When the device laid flat on a surface, no orbs would move. When the device was in a vertical position, the orbs almost gave an illusion of a windy snowstorm. Furthermore, in my original implementation, orbs would get stuck at the borders of the frame, so I modified my code to make the orb appear at the opposite end of the screen whenever it hit a border, to give the illusion of a torus connecting opposite sides of the device's screen.

After I had the accelerometer working the way I intended, I started to play around with other values I could glean from the accelerometer data. The combination that appealed to me the most was using data from the x-field of the accelerometer to control the x-direction of the orbs, and data from the z-field of the accelerometer for the y-direction of the orbs on the screen. The visual effect of this data swap is counterintuitive but still smooth and interesting. Now, when the device lies on a flat surface, the orbs move. When the device is perfectly vertical (which, practically, does not occur often or for long), the orbs give the illusion of a lack of gravity; they stop moving. Of course, all angles of existence for the device can contribute to the flow in which the orbs move around, but the new accelerometer implementation is more interesting to look at and more fun to play with now that the orbs move in an unexpected way.

7.4 Tools

In order to improve my understanding of the iPhone's accelerometer, I sought out online tutorials because my textbook did not cover this sensor. The first tutorial that I learned from was a basic implementation that simply displayed the accelerometer data in labels on the screen for each of the three directions [3]. This was very useful to me because I could watch the data change in each direction while moving the phone around. Ultimately, I chose to use the x and z fields of the acceleration data to manipulate my orb objects because the orbs only move in 2d space.

I used a different tutorial that helped me learn how to move objects around the space of the screen with accelerometer data [15]. This tutorial used an iOS framework called Core Motion, but I ended up only using raw accelerometer data in my final project. I had to do many things differently from this tutorial, like adding the accelerometer informa-

tion to my OrbView class, rather than in the ViewController class that the tutorial uses. Nevertheless it was a useful tool to help me conceptualize what I wanted to accomplish with the accelerometer. The "Event Handling Guide for iOS: Motion Events" of the iOS Developer Library also helped me understand and then utilize the accelerometer for my application [10].

8
Camera

8.1 Camera Background

Cameras have existed in mobile phones for over a decade. My first cell phone did not have a camera, but now, about eight years later I take about a dozen photographs every day on my iPhone. Social media companies (and mobile applications) like Instagram have skyrocketed in popularity because of the iPhone's camera. Many smartphone users take advantage of the camera they carry around every day and therefore they are able to document, and maybe sometimes over-document, their lives.

8.2 Apple Standards

Apple provides easy access to its devices' cameras for developers. The "UIImagePickerController" class allows the programmer to access either the camera(s) or the camera roll, a list of previously saved photographs [9]. It is very simple to choose between the front camera and the rear camera, to show or hide controls, and to open the photo library inside of an iOS application. Unlike for touch gestures, Apple does not have strict standards for

use nor any suggested recommendations to follow in their documentation for the camera.

The first iPhone had only a rear-facing camera. Since then, Apple has actively improved cameras as one of the main updates of each new version of the iPhone. They have updated both the hardware and software over time. A present-day iPhone 5S has both front and rear cameras, can use flash (and automatically detect whether flash is necessary), take videos, take panoramas, and more. A major feature of iOS 7, released this past fall, was the ability to access the camera quickly, even when the device is locked. From the lock screen, the user can either swipe up or tap a little camera icon in order to take a picture very quickly.

8.3 Implementation

I use the iPhone camera in my senior project by using the camera view as the background for an overlay of user-generated orbs. There are two separate running versions of my application: one uses the device's regular camera, and the other uses the front-facing camera. In both versions, the camera view exists as the background in order to combine the external world with the virtual orb world within the screen.

The version of the application that uses the front-facing camera provides a creative expression of literally displaying the user while they explore the program. If the user holds the device vertically in front of them, the camera shows them their own face, with the orbs moving at a very slow speed on top of their features. The intention of confronting the user with their own image for this version of my application was to encourage them to consider their role as a consumer of technology. An artistic goal of the project is for the viewer to become more aware of the device's interface and their own role within this interface. By

literally facing themselves while they engage with touch gestures and the accelerometer, the user actually becomes a part of the application. If the user does not hold the device in front of their face, but rather engages with it while it rests flat on a surface, the user will then see an image of their own hand if they try to touch any of the orbs. The effect remains the same, yet now the visual focus shifts to the sense of touch. The version of my application with the front-facing camera is actually more interesting in the case when the device is at rest, because in the other version the camera becomes obscured by the surface below.

The other version of my application uses the rear-facing camera. In this version, the background of the program shows through to the world outside, or what is in front of the user. The intention with this version was to simulate a virtual reality with the orbs floating on top of the image of reality as a reminder to the viewer that they are looking through an interface that is looking through a lens. The antigravity effects that the orbs display when the device is vertical become even stranger when they overlay the normal world. At any angle (except when the camera is obscured) the screen will display a real, but eerie gravitational version of the real world. In particular, I designed the orb image to look somewhat similar to the way the lights in the RKC reflect on the ground, thinking that there might be an interesting visual effect during the senior project poster session (see Figure 8.3.1) if anyone faces the camera towards the floor.

8.4 Tools

Creating a view of the camera was a relatively simple task thanks to Apple's iOS Developer Library. The library contains a class called "UIImagePickerController" that can instantiate both cameras, hide camera controls, take pictures, and more [12]. I made the

background of my OrbView class transparent and tried to place this view on top of the camera view, but I had trouble getting the two views to simultaneously coexist and function properly. Eventually I found a tutorial that covered creating an overlay for a camera object [14]. The tutorial was from 2009 and slightly hard to follow because much has changed within iOS and XCode in just a few years. However, by reading the tutorial I realized I had overlooked two key missing components that I needed to make my program work properly. The first was a property called "cameraOverlayView" for the UIImagePickerController object. The second was that the picker needed to be called in a method called viewDidAppear, rather than viewDidLoad method, which is how I had previously loaded all other program segments. After I fixed these two problems, I was able to add an OrbView above the UIImagePickerController just as I had originally intended.

Figure 8.3.1. A screenshot from the application.

9
Conclusion

9.1 Future Modifications and Additions

If I had more time to work on the project I would continue to add features to my application. I think the addition of sound effects or music could contribute a lot to the entertainment value of the program. Likewise, ideally I would have given the user an ability to change the effects of the app. For example, if they set up the orbs in a particular configuration, they could change the way the program handled acceleration, or the color scheme could change to reflect a certain action. I would have also liked to use the gyroscope sensor to cause the orbs to rotate in their space on the screen, but I would need much more practice with programming graphics and animation in order to achieve that effect. In retrospect, the idea of making an application that uses all of the sensors except touch to control the events of the program would have been an interesting problem, but it also would have been extremely difficult to do well.

9.2 Success of Project

I think that the final version of my application is effective in its goals. Compared to earlier versions, people I have shared the final version with generally spend at least a few minutes playing around and exploring the application. As the developer, it is interesting for me to watch the differing ways that people interact with it. Some try to turn their interaction with the orbs into a game where they arrange them in different visual patterns. Others seem to enjoy watching the accelerometer work to its full extent to cause the orbs to zoom around the screen chaotically. I think the program is visually appealing, and the smooth motion of the orbs can be soothing even if the program is left idling on a desk. The degree of freedom that the program offers the user allows each individual to have a personal experience with the application. The inclusion of the camera helps combine the screen of the device with reality. I think the interaction of the three sensors that I used works well and it effectively combines user expectations with counterintuitive surprises.

10
Appendix

10.1 ViewController.m

```
//
//  ViewController.m
//
//  Created by Julie Carter on 4/7/14.
//  Copyright (c) 2014 Julie Carter. All rights reserved.
//

#import "ViewController.h"
#import "OrbView.h"
#import "Orb.h"
#import <AVFoundation/AVFoundation.h>


@interface ViewController ()

@end

@implementation ViewController


- (void)viewDidLoad
{
    [super viewDidLoad];

}


- (void) viewDidAppear:(BOOL)animated {
```

```
    // get width and height of current device screen
    CGFloat width = [UIScreen mainScreen].bounds.size.width;
    CGFloat height = [UIScreen mainScreen].bounds.size.height;


    // create an instance of OrbView of the correct width and height
    OrbView *overlay = [[OrbView alloc] initWithFrame:CGRectMake(0, 0,
     width, height)];


    // Create a new image picker instance:
    UIImagePickerController *picker = [[UIImagePickerController alloc]
     init];


    // Set the image picker source:
    picker.sourceType = UIImagePickerControllerSourceTypeCamera;

    picker.cameraDevice = UIImagePickerControllerCameraDeviceFront;
    // for rear camera use: UIImagePickerControllerCameraDeviceRear;


    // Hide the camera controls:
    picker.showsCameraControls = NO;
    picker.navigationBarHidden = YES;

    // Insert the overlay:
    picker.cameraOverlayView = overlay;

    // Show the picker:
    [self presentModalViewController:picker animated:YES];

    [super viewDidAppear:YES];
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

@end
```

10.2 OrbView.m

```
//
//  OrbView.m
//
//  Created by Julie Carter on 4/7/14.
```

```
//  Copyright (c) 2014 Julie Carter. All rights reserved.
//


#import <AVFoundation/AVFoundation.h>
#import "OrbView.h"
#import "Orb.h"


@interface OrbView ()

@end



@implementation OrbView

- (id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self)
    {
        // Initialization
        // initialize empty mutablearray of orbs
        self.finishedOrbs = [[NSMutableArray alloc] init];

        self.multipleTouchEnabled = YES;

        // set opacity to allow transparency for camera view
        self.opaque = NO;
        self.backgroundColor = [UIColor clearColor];

        // create and add gesture recognizer for single taps
        UITapGestureRecognizer *tapRecognizer = [[UITapGestureRecognizer
          alloc] initWithTarget:self action:@selector(tap:)];
        [self addGestureRecognizer:tapRecognizer];


        // create and add gesture recognizer for pan movements
        UIGestureRecognizer *moveRecognizer = [[UIPanGestureRecognizer
         alloc] initWithTarget:self action:@selector(moveOrb:)];
        [self addGestureRecognizer:moveRecognizer];


        // initialize accelerometer
        [[UIAccelerometer sharedAccelerometer]
         setUpdateInterval:(1.0 / 100)];
        [[UIAccelerometer sharedAccelerometer]
         setDelegate:self];

    }
```

```
    //return the view
    return self;
}


// main draw method. redraws each orb in its updated location as needed
with [self setNeedsDisplay]
- (void)drawRect:(CGRect)rect
{
    // remove the images from their old location
    for (Orb *orb in self.finishedOrbs) {
        [orb.imgView removeFromSuperview];
    }
    // add to new location
    for (Orb *orb in self.finishedOrbs)
    {
        orb.imgView = [[UIImageView alloc]
         initWithFrame:CGRectMake(orb.currentOrbCenter.x - 50,
         orb.currentOrbCenter.y - 50, 100, 100)];

        UIImage *image = [UIImage imageNamed: @"bloob.png"];
        orb.imgView.image=image;
        orb.imgView.userInteractionEnabled = YES;

        [self addSubview:orb.imgView];

    }
}


// handler for single taps
-(void) tap:(UIGestureRecognizer *) gr
{

    if ([self.finishedOrbs count] < 6) // arbitrary max # orbs = 6
    {
        CGPoint location = [gr locationInView:self];
        Orb *orb = [[Orb alloc] init];
        orb.currentOrbCenter = location;
        orb.speed = 1 + arc4random() %5;  //randomly set speed int

        [self.finishedOrbs addObject:orb];
        [self setNeedsDisplay];
    }
}


// returns an orb if there is an orb nearby to the point of a tap
-(Orb *) orbAtPoint:(CGPoint) p
{
    // Find an orb close to p
```

```
    for(Orb *orb in self.finishedOrbs)
    {
        CGPoint center = orb.currentOrbCenter;

        //create a test rectangle around each orb to search
        CGRect testRect = CGRectMake(center.x -50, center.y - 50,
         100,100);

        if (CGRectContainsPoint(testRect, p))
        {
            return orb;
        }
        else
        {
            // no orb at point
        }
    }
    return nil;
}

// handler for pan motions
-(void) moveOrb:(UIPanGestureRecognizer *) gr
{
    if (gr.state == UIGestureRecognizerStateBegan)
    {
        CGPoint point = [gr locationInView:self];
        self.selectedOrb = [self orbAtPoint:point];

        // accelerometer will not affect orb for duration of pan
        self.selectedOrb.speed = 0;
    }
    if (gr.state == UIGestureRecognizerStateChanged)
    {
        CGPoint translation = [gr translationInView:self];

        CGPoint center = self.selectedOrb.currentOrbCenter;
        center.x += translation.x;
        center.y += translation.y;

        self.selectedOrb.currentOrbCenter = center;
        [self setNeedsDisplay];

        [gr setTranslation:CGPointZero inView:self];
    }
    if (gr.state == UIGestureRecognizerStateEnded)
    {
     // accelerometer effects return upon release
        self.selectedOrb.speed = 1 + arc4random() %5;
        [self setNeedsDisplay];
        [gr setTranslation:CGPointZero inView:self];
```

```
    }

}


- (void)accelerometer:(UIAccelerometer *)accelerometer
didAccelerate:(UIAcceleration *)acceleration
{

    CGPoint accel;
accel.x = acceleration.x;
accel.y = acceleration.z;


    // find the bounds of the screen
    float maxX = self.frame.size.width;
    float maxY = self.frame.size.height;

    // modify moveFactor to change rate of motion
    float moveFactor = 1;


    for (Orb *orb in self.finishedOrbs)
    {
        moveFactor = moveFactor*-1;
        float movetoX = orb.currentOrbCenter.x +
         (accel.x * moveFactor);
        float movetoY = orb.currentOrbCenter.y +
         (accel.y * moveFactor);



        CGPoint center = orb.currentOrbCenter;
        if (movetoX > 0 && movetoX < maxX) // if within bounds
        {
            center.x += orb.speed*moveFactor*accel.x;
        }
        if (movetoX <= 0) // if at left border of screen
        {
            center.x = maxX;
        }
        if (movetoX >= maxX) // if at right border of screen
        {
            center.x = 0;
        }


        if (movetoY > 0 && movetoY < maxY) // if within bounds
        {
            center.y += orb.speed*moveFactor*accel.y;
```

```
        }
        if (movetoY <= 0)   // if at top of screen
        {
            center.y = maxY;
        }
        if (movetoY >= maxY)    // if at bottom of screen
        {
            center.y = 0;
        }

        orb.currentOrbCenter = center;  // update center value

        [self setNeedsDisplay];     //redraw


    }


@end
```

Bibliography

[1] Jon Agar, *Constant Touch, A Global History of the Mobile Phone*, Icon Books, Ltd., London, 2013.

[2] Arcade Fire, *https://justareflektor.com*.

[3] Brandon Cannaday, *iPhone Tutorial: Reading the Accelerometer, http://tech.pro/tutorial/968/iphone-tutorial-reading-the-accelerometer*, August 5, 2009.

[4] Nicole Cohen, *Timeline: A History of Touch-Screen Technology, http://www.npr.org/2011/12/23/144185699/timeline-a-history-of-touch-screen-technology*, December 26, 2011.

[5] Joe Conway, Aaron Hillegass, and Christian Keur, *iOS Programming: The Big Nerd Ranch Guide (4th Edition)*, Big Nerd Ranch Guides, February 2014.

[6] Erin Coulehan, *See Yourself in Arcade Fire's Interactive 'Reflektor' Video, http://www.rollingstone.com/music/news/see-yourself-in-arcade-fires-interactive-reflektor-video-20130909*, September 9, 2013.

[7] Bob Dylan, *video.bobdylan.com/desktop.html*.

[8] Andy Greene, *Bob Dylan Goes Interactive in 'Like a Rolling Stone' Clip, http://www.rollingstone.com/music/news/bob-dylan-goes-interactive-in-like-a-rolling-stone-clip-20131119*, November 19, 2013.

[9] iOS Developer Library, *Camera Programming Topics for iOS: Taking Pictures and Movies*.

[10] _____, *Event Handling Guide for iOS: Motion Events*.

[11] _____, *Interactivity and Feedback*.

[12] _____, *UIImagePickerController Class Reference*.

[13] _____, *UITapGestureRecognizer Class Reference*.

[14] jjob, *iPhone Camera Overlay App With Custom Button Example, http://www.musicalgeometry.com/?p=821*, December 10, 2009.

[15] John Morrison, *iOS: Getting Started with Accelerometer Data, https://www.captechconsulting.com/blog/john-morrison/ios-getting-started-accelerometer-data*, November 29, 2012.

[16] Donald Norman, *The Design of Everyday Things*, Basic Book, New York, 1988.

[17] John-David Penn, *Radiohead Releases "PolyFauna" App*, February 11, 2014.

[18] Radiohead, Nigel Godrich, Stanley Donwood, and Universal Everything, *http://www.universaleverything.com/projects/polyfauna/*.

[19] Rafael Rozendaal, *http://www.newrafael.com*.