Spring 2016

Master’s Project

Computer Science Department

California State University, **Dominguez Hills**

MACSEISAPP: AN EARLY EARTHQUAKE WARNING SYSTEM

A Project

Presented

to the Faculty of

California State University Dominguez Hills

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

by

Shayan Mehrazarin

Spring 2016

PROJECT: MACSEISAPP: AN EARLY EARTHQUAKE WARNING SYSTEM

AUTHOR: SHAYAN MEHRAZARIN

APPROVED:

This project is dedicated to my parents, Mohammad Reza Mehrazarin and Gilda Moshrefi, my brother, Dr. Shebli Mehrazarin, and my sister-in-law, Saman Karimi-Mehrazarin, who have supported me throughout my educational career.

ACKNOWLEDGEMENT

I would like to acknowledge and give special thanks to the distinguished faculty members who served on my committee: Dr. Bin Tang (chair), Dr. Mohsen Beheshti, and Dr. Jianchao (Jack) Han. As my advisor, Dr. Tang provided detailed guidance and encouragement throughout the development of this project. Additionally, I am thankful to have been supported financially throughout the development of this project in part by Dr. Tang's National Science Foundation Grant CNS – 1419952.

TABLE OF CONTENTS

LIST OF TABLES

PAGE

LIST OF FIGURES

ABSTRACT

Most MacBook and MacBook Pro notebook computers manufactured since 2006 are equipped with a device called the Sudden Motion Sensor, or SMS. The purpose of this sensor is to help protect the MacBook's Hard Disk Drive (HDD) in the event that the MacBook is accidentally dropped or experiences heavy shaking and vibration. Aside from protecting the MacBook, the Sudden Motion Sensor has been used for various purposes, especially as a seismograph. In this project, I design and implement MacSeisApp, an earthquake early warning system that utilizes the MacBook's Sudden Motion Sensor. MacSeisApp, inspired by both the *Quake Catcher* [1] and an existing seismograph application called *SeisMac* [2], is an application that not only displays a basic seismograph, but also aims to notify other users of earthquake activity in nearby cities using Apple's Push Notifications (APN) via a dedicated server. Written in both the C and Objective-C programming languages using the XCode Integrated Developer Environment (IDE), MacSeisApp utilizes the same open-source library, *smslib* [3], used in SeisMac in order to detect the vibrations via the Sudden Motion Sensor and translate it into points to be plotted as a seismograph. If there is shaking detected, it will draw some spikes, and once a spike is detected, MacSeisApp will communicate with a dedicated server. The dedicated server then executes a script to communicate with Apple's Push Notification Service server, which then processes the request and sends an alert via Push Notification to other MacBooks nearby. Additionally, a sound plays as the notification is displayed.

CHAPTER 1

INTRODUCTION

Most MacBook and MacBook Pro notebook computers manufactured since 2006 are equipped with a device called the Sudden Motion Sensor, or SMS. The purpose of this sensor is to help protect the MacBook's Hard Disk Drive (HDD) in the event that the MacBook is accidentally dropped or experiences heavy shaking and vibration. Since Solid State Drives (SSD) do not spin like the traditional HDD, MacBook models that feature a SSD do not come with the Sudden Motion Sensor equipped. Aside from protecting the MacBook, the Sudden Motion Sensor has been used for various purposes, especially as a seismograph. An application called *SeisMac* [2] uses the sensor to transform the user's MacBook into a seismograph by displaying real-time, three-axis acceleration graphs. SeisMac uses *smslib* [3], an open-source library that is written using both the C and Objective-C programming languages.

SeisMac along with the *Quake Catcher* [1] inspired the development of a new application called MacSeisApp, which not only displays a basic seismograph, but also aims to notify other users in nearby cities of earthquake activity using Apple's Push Notifications (APN) via a dedicated server. Written in both the C and Objective-C programming languages using the XCode Integrated Developer Environment (IDE), MacSeisApp utilizes the same open-source library used in SeisMac in order to detect the vibrations via the Sudden Motion Sensor and translate it into points to be plotted as a seismograph. If there is shaking detected, the program will draw some spikes, and once a spike is detected, MacSeisApp will then communicate with a dedicated server based at California State University, Dominguez Hills. The server at California State University, Dominguez Hills then communicates with Apple's server through the successful execution of a script written in the PHP scripting language. This script will first check if two different users executed this script within a four-second period. If this condition is true, then this script continues to send the request to Apple's server; otherwise, the script will abort if this condition is not met and it will not send the request. Apple's server then processes the request if received and then sends an alert via Push Notification to other MacBooks nearby. Additionally, a sound will play along with the notification as it is being displayed. The general idea and flow of events in MacSeisApp is illustrated in Figure 1.1.

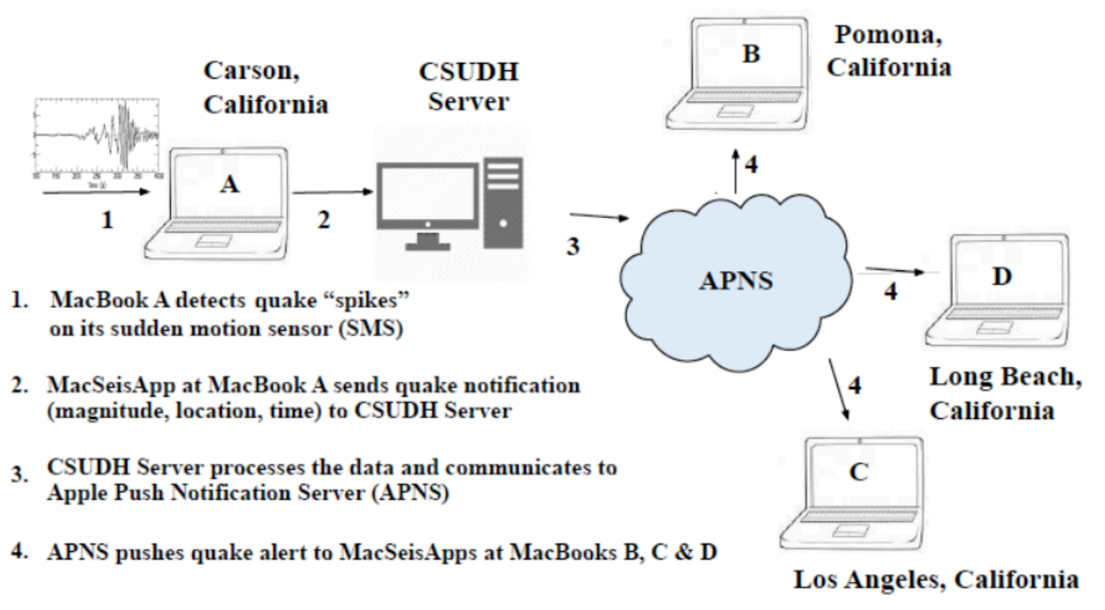

Carson, California
CSUDH Server
B
Pomona, California
A
1
2
3
4
APNS
4
D
4
Long Beach, California
C
Los Angeles, California
1. MacBook A detects quake "spikes" on its sudden motion sensor (SMS)
2. MacSeisApp at MacBook A sends quake notification (magnitude, location, time) to CSUDH Server
3. CSUDH Server processes the data and communicates to Apple Push Notification Server (APNS)
4. APNS pushes quake alert to MacSeisApps at MacBooks B, C & D

Figure 1.1: The General Structure and Idea of MacSeisApp

CHAPTER 2

BACKGROUND

2.1. Background on Apple's Sudden Motion Sensor (SMS)

Apple had first introduced the Sudden Motion Sensor, or SMS, to their MacBook and MacBook Pro lines of notebook computers starting in 2006. The Sudden Motion Sensor is Apple's proprietary gyroscopic sensor and the main purpose of its presence in the MacBook and MacBook Pro is to protect the MacBook's spinning hard disk drive (HDD) and to minimize damage just before it hits a hard surface when falling off of a table or other elevated surface. The Sudden Motion Sensor does this by sending a signal to the hard disk drive as soon as the sensor starts detecting a lot of unusual or violent motion, which is measured along the x, y, and z axes in *gravities (g)*. In response to this signal, the hard disk drive stops spinning temporarily.

In recent years, however, Apple has introduced solid-state drives, or SSDs, into more of their MacBook models. Unlike the traditional spinning hard disk drives, solid state drives do not consist of any spinning or moving parts internally, just like in USB flash drives or SD cards. Due to the lack of hard disk drives in these newer MacBook models, these newer models are consequently not equipped with sudden motion sensors.

2.2. Background on Apple's Push Notification Service (APNS)

Apple's Push Notification Service (APNS) was first introduced in iOS version 3.0 in 2009 and later introduced in Mac OS X "Lion" version 10.7 in 2011 as a means to provide an alternative to the existing *local* notification functionality built into the respective operating systems. According to Apple's developer program documentation [4], push notifications are commonly referred to as *remote* notifications.

Push notifications are significant in that they give developers the capability to send notifications to end-users, even when the application is not active or running. Over the years, push notifications have been used to notify users in various scenarios, such as:

- New or unread messages
- Weapon unlocks or invitations in games
- Severe weather alerts (i.e. tornado warnings, flash flood warnings), and
- Friend requests in social media applications.

The way Apple's Push Notification Service works is as follows. First, the developer has to set up his own server that supports scripting protocols, such as the PHP scripting language. The script will then have instructions to communicate with Apple's own dedicated server for push notifications using a secure connection via a secure layer protocol, which in the case of MacSeisApp is the Secure Sockets Layer (SSL) protocol. Finally, Apple's push notification server will then verify the authenticity of the developer's server security certificates, and if the certificates are valid, then Apple's server will *push* the notification to the user's device. The general idea and flow of the push notification is demonstrated in Figure 2.2.1 [4].

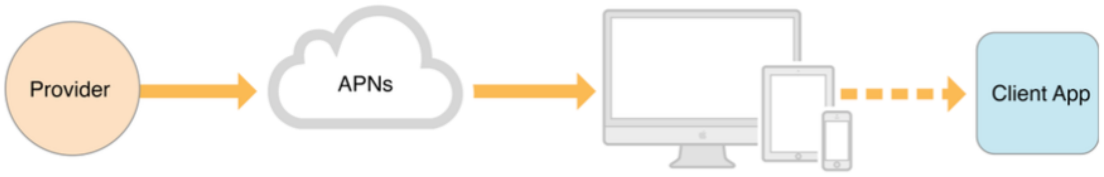


Figure 2.2.1: Structure and Flow of Apple's Push Notification Service [4]

SSL is a commonly used security protocol that creates an encrypted link between a web server and the end-user's web browser and ensures transmitted data remains private. SSL was widely used by developers and endorsed by Apple up until October 2014, which was when the Padding Oracle On Downgraded Legacy Encryption (POODLE) exploit on SSL version 3.0 was discovered by a group of security researchers from Google. As a result of this discovery, Apple forbids developers from using the SSL protocol for push notifications when publishing it to the Mac App Store. As an alternative security protocol, Apple recommends that developers use the Transport Layer Security (TLS) protocol for push notifications, which is a protocol that ensures privacy between communicating applications and their end-users on the Internet. However, there is an exception to this restriction, where developers are still permitted to use SSL certificates for testing and debugging purposes while in development or *sandbox* mode.

CHAPTER 3

METHODOLOGY

3.1. Development Model and Software Architecture

The development of MacSeisApp closely follows the *waterfall model*. The waterfall model is a developmental approach where a series of key events are performed sequentially in various phases. The waterfall model that is used in the development of MacSeisApp is illustrated in Figure 3.1.1

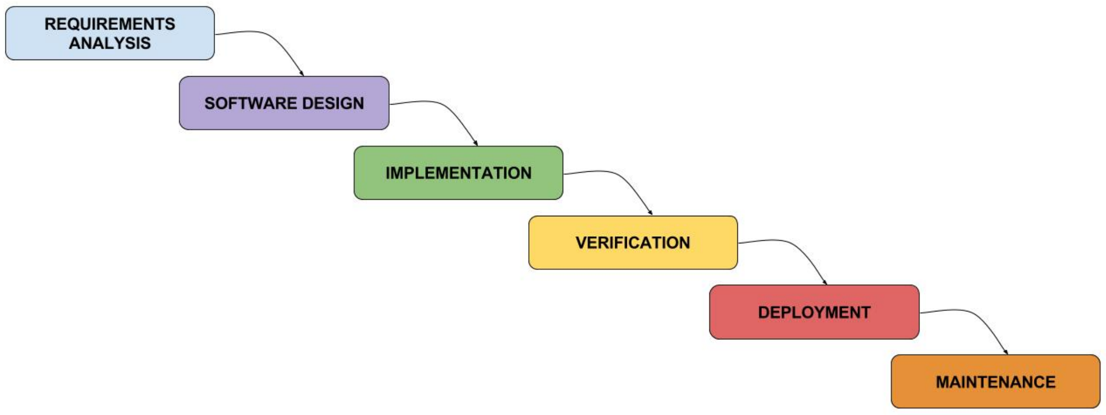


Figure 3.1.1: The Waterfall Model for MacSeisApp

The first phase in the development of MacSeisApp is *requirements analysis*, where the requirements for the project is provided and developed. The next phase in the development of MacSeisApp is *software design*, where the design for the project is developed, which involved studying the design of SeisMac. The third phase in the development of MacSeisApp is *implementation*, which involves programming and writing source code for the project. The fourth phase in the development of MacSeisApp is *verification*, where the application is tested to ensure it works as intended with minimal bugs or glitches. The final phases in the development of MacSeisApp are *deployment* and *maintenance*. Deployment involves the distribution of the software to end-users, and maintenance involves resolving bugs in the software that are discovered or reported along with keeping the application up to date.

The *software design* phase in the waterfall model is one of the most significant phases in the development of MacSeisApp. In particular, it involves drafting a layout of the software's structure, commonly in the form of software architecture design. Software architecture lays out the high level structures of a software and is typically broken down into the *front-end* and *back-end*. The *front-end* is the interface that clients and end-users are presented with, and the *back-end* is the server interface that is visible to developers. Figure 3.1.2 illustrates the software architecture of MacSeisApp.

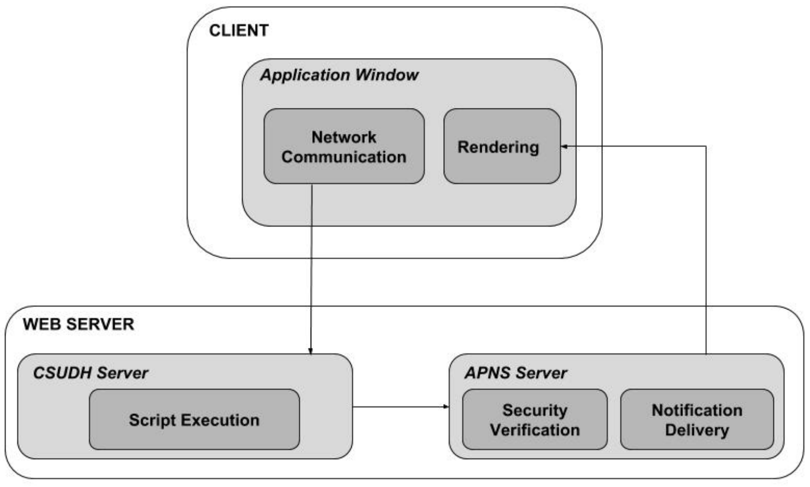
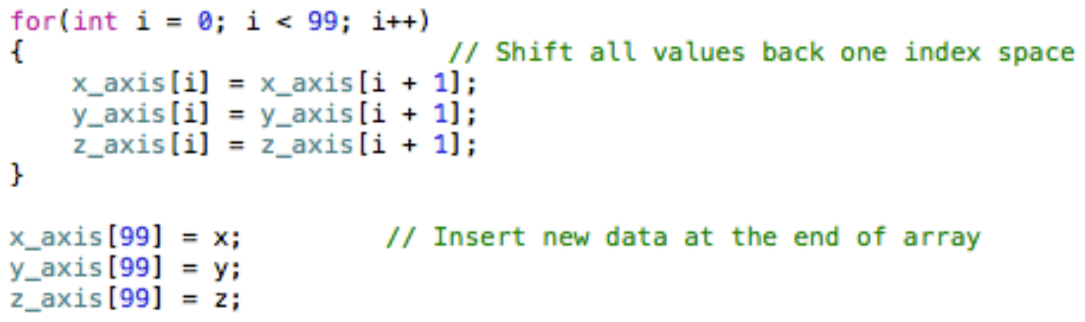


Figure 3.1.2: The Software Architecture of MacSeisApp

The software architecture for MacSeisApp consists of the *client* function and the *web server* function. The *client* is simply the interface that is presented to the user, which performs *network communication* and *rendering*. The network communication is used to link the *client* interface with the *web server* interface, while the rendering is used to display the seismograph interface and update it once every 0.1 seconds. The *web server*, on the other hand, consists of the server at California State University, Dominguez Hills and Apple's Push Notification Service server. The architecture in Figure 3.1.2 shows that the *client* first communicates with *CSUDH Server*; *CSUDH Server* then communicates with *APNS Server* through *script execution*; and finally *APNS Server* communicates with the *client* using *notification delivery* as long as *security verification* is successful. *Security verification* is the functionality that checks the authenticity and validity of the *CSUDH Server*'s security certificates.

3.2. The Role of the Sudden Motion Sensor in MacSeisApp

The Sudden Motion Sensor is responsible for collecting motion data once every 0.1 seconds while the MacSeisApp application is running, even if the user's MacBook is stationary. The motion data that is collected is then translated by the open-source library *smslib* into values of type *float* or *double*. The collected motion data can then be accessed by using the various provided accessor functions that are written in the C programming language, and the data for each of the three individual axes, *x, y,* and *z*, can be used in both code written in C and code written in Objective-C.



```
for(int i = 0; i < 99; i++)
{                              // Shift all values back one index space
    x_axis[i] = x_axis[i + 1];
    y_axis[i] = y_axis[i + 1];
    z_axis[i] = z_axis[i + 1];
}

x_axis[99] = x;          // Insert new data at the end of array
y_axis[99] = y;
z_axis[99] = z;
```

Figure 3.2.1: Inserting and Shifting Data Collected from SMS in Array

MacSeisApp then takes this data and stores it in three separate arrays of type *double* with a size of one hundred elements. The data is then plotted by MacSeisApp to the application window with each axis on its own separate line with a black background to depict a seismograph. The seismograph is then updated every time a new piece of data is collected from the Sudden Motion Sensor, which once every interval of 0.1 seconds. A total of one hundred plots of data are displayed at a time on each axis with the points interconnected with standard lines. The code for this scenario is shown in Figure 3.2.1.

3.3. Calculating the Magnitude of Seismic Activity From Data

Earthquakes are commonly measured based on the Richter scale. The data that is collected from the SMS for each of the three axes, however, is measured in *gravities (g)*, where the *x*, *y*, and *z* axes range from -1.0g to 1.0g. As gravity is a form of acceleration, *gravities* can easily be converted to *meters per square second* (m/s^2), where 1g is equal to 9.81 m/s^2.

Table 3.3.1: Relationship Between Gravity/Acceleration and Magnitude [5]

Richter Scale Value	**Approximate Gravity (*g*)**	**Approximate Acceleration (*m/s^2*)**	**Approximate Mercalli Scale Value**
5.0	± 0.03	± 0.29	V
5.4	± 0.05	± 0.49	VI
6.1	± 0.1	± 0.98	VII
6.6	± 0.25	± 2.45	VIII
7.3	± 0.5	± 4.91	X
8.0	± 0.75	± 7.36	XI
≥ 8.2	±1	± 9.81	XII

The severity of an earthquake is also assessed by the Mercalli scale. The magnitude of an earthquake can be approximated on the Richter scale using a combination of a rating on the Mercalli scale and acceleration or gravity values (shown in Table 3.3.1) [5].

The seismograph interface for MacSeisApp displays the data in the form of gravities (*g*). By using the conversions from Table 3.3.1, MacSeisApp can determine the approximate Richter scale value for magnitude 5.0 and above as long as the value of gravities (*g*) from both the *x-axis* and *y-axis* are greater than 0.03g. The Richter scale value is later used in the notification message in MacSeisApp in the event that an earthquake of magnitude 5.0 and above is detected.

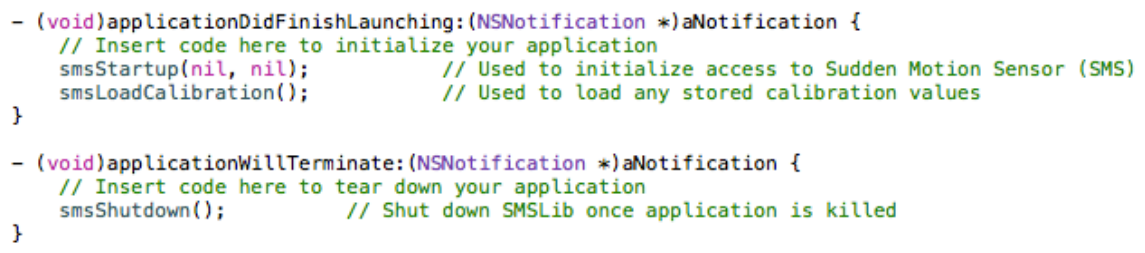
Additionally, MacSeisApp calculates the average gravities value for the last one hundred values collected from each axis and takes the absolute value of the difference between the newest value and the average. The purpose of this calculation is to reduce the likelihood of “false alarms” and to account for cases where the MacBook is placed on an uneven surface.

CHAPTER 4

DESIGN & IMPLEMENTATION

4.1. An Inside Look at MacSeisApp

The MacSeisApp application has been developed with various programming languages and tools. Specifically, it has been developed using a combination of Objective-C and C code and utilizes the Xcode Integrated Development Environment (IDE). This is actually possible due to the fact that the Objective-C programming language is derived from the C programming language. Additionally, it is very simple to distinguish between the code that is written in Objective-C versus the code that is written in C. One major difference in Objective-C that is immediately noticeable is that every project written in Objective-C contains a class called the *Application Delegate*, or commonly abbreviated as *AppDelegate*. The AppDelegate essentially handles the tasks that need to be done "behind-the-scenes" or "under the hood", commonly by the operating system.

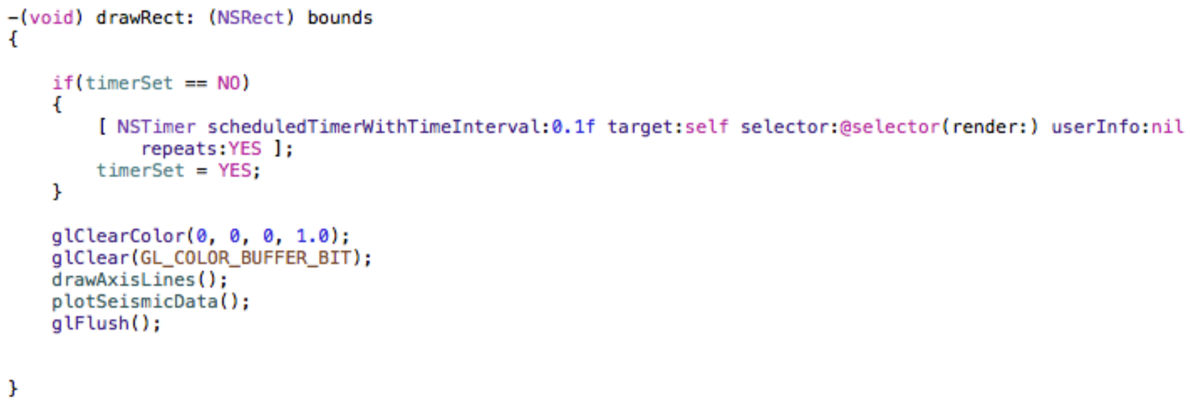


```
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    // Insert code here to initialize your application
    smsStartup(nil, nil);           // Used to initialize access to Sudden Motion Sensor (SMS)
    smsLoadCalibration();           // Used to load any stored calibration values
}

- (void)applicationWillTerminate:(NSNotification *)aNotification {
    // Insert code here to tear down your application
    smsShutdown();          // Shut down SMSLib once application is killed
}
```

Figure 4.1.1: The General Structure of an AppDelegate

The AppDelegate will always have two functions with a return type of void: *applicationDidFinishLaunching* and *applicationWillTerminate*. Inside these two functions, one would find some code similar to what is shown in Figure 4.1.1. The *applicationDidFinishLaunching* function will consist of any code that the developer wants to be executed as an application is starting up. The *applicationWillTerminate* function, on the other hand, will consist of any code that the developer wants to be executed right before the application fully shuts down.



```
-(void) drawRect: (NSRect) bounds
{

    if(timerSet == NO)
    {
        [ NSTimer scheduledTimerWithTimeInterval:0.1f target:self selector:@selector(render:) userInfo:nil
            repeats:YES ];
        timerSet = YES;
    }

    glClearColor(0, 0, 0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    drawAxisLines();
    plotSeismicData();
    glFlush();


}
```

Figure 4.1.2: The Combined Use of Objective-C and C Code in a Function

Aside from the code written in Objective-C, there is a need to utilize the C programming language in order to take advantage of the OpenGL framework [6]. OpenGL is a widely used graphics library first introduced in 1992 that allows developers to incorporate custom graphics into their applications, such as colored lines and shapes. The OpenGL library is needed in MacSeisApp in order to display the seismograph interface in real time. For example, the function that handles rendering new data on the seismograph interface once every 0.1 seconds has a mixture of Objective-C code and OpenGL code written in C, which is demonstrated in Figure 4.1.2.

4.2. Designing the Seismograph Interface

The seismograph interface for MacSeisApp was made possible thanks to the OpenGL library. Although written in C, OpenGL works seamlessly with code written in the Objective-C programming language. This is especially beneficial in the sense that it allows the developer to use various functions that come built in with the Mac OS X Software Development Kit (SDK).

In order to ensure that the data and the seismograph interface is updated once every 0.1 seconds, an object of type *NSTimer* from Objective-C is used. The timer, in particular, is used for the following three functions:

- Shifting existing content in array to the previous index, and data at index 0 is overwritten
- Adding new data to the end of the array at index 99
- Redrawing the interface each time with the new contents of the array

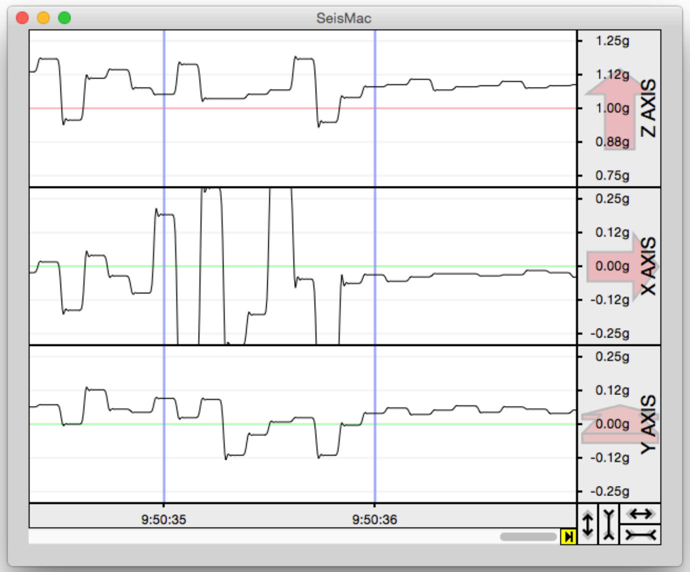


Figure 4.2.1: Seismograph Interface for *SeisMac* [2]

The seismograph interface of MacSeisApp was primarily influenced by the seismograph interface of SeisMac. A comparison of the seismograph interfaces between the two applications can be seen in both Figure 4.2.1 for SeisMac [2] and in Figure 4.2.2 for MacSeisApp. One can immediately notice when running on a newer MacBook Pro equipped with an Intel Core i5 processor that the seismograph interface for MacSeisApp looks a lot smoother with regards to the lines as opposed to SeisMac, where the lines look a bit clunky. This is likely because SeisMac was originally designed and optimized to work with the PowerPC and Intel Core 2 Duo processors that were common in older MacBook and MacBook Pro models. Additionally, SeisMac has not been updated in many years to include optimizations for newer Intel processors such as the Intel Core i5 and Intel Core i7.

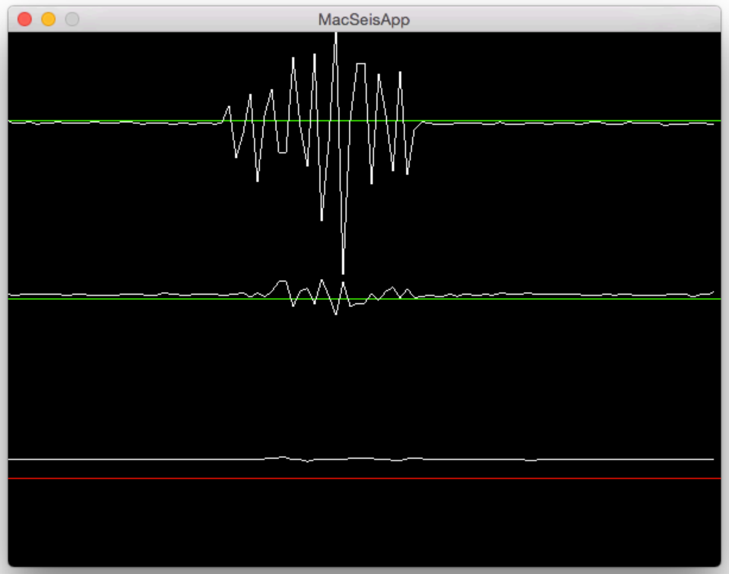


Figure 4.2.2: Seismograph Interface for *MacSeisApp*

4.3. The Use of Notifications in MacSeisApp

When it comes to grabbing an end-user's immediate attention, there are several options that the developer can take advantage of. Some options that were previously considered for MacSeisApp include email and Apple's *iMessage* messaging service. Due to the nature of MacSeisApp, neither one of those two options were able to be utilized as they require the end-user to click on a button to send the email or iMessage, which would not be practical in this case. Instead, MacSeisApp utilizes Apple's Push Notification Service (APNS) as it requires no interaction by the end-user to initiate the sending or receiving of notifications.

Furthermore, push notifications are especially used to grab the end-user's immediate attention, as they are delivered instantly. Push notifications are very effective in that there is at most one second of latency or "lag", even in the worst-case scenario where the end-user has a poor Internet connection. With MacSeisApp, a push notification would be generated and delivered to users in nearby areas and communities if the seismograph interface shows a "spike" on either the x, y, or z axes for two or more users within a four-second interval. This is handled by a smart seismic activity detection algorithm in the PHP script on the server at California State University, Dominguez Hills.

If at least two users in one city detect some shaking activity within a four-second interval that has similar data across the x, y, and z axes, then this smart seismic activity detection algorithm would send a push notification request to Apple's Push Notification

Server. This way, if at least two end-users detect “spikes” on their seismograph interface with a similar pattern, then very likely some legitimate seismic activity has occurred and a push notification will be received by all end-users within that area. Otherwise, if only one out of two or more end-users detects some activity, then very likely this activity was artificial, resulting from a slamming door or other artificial movement.

A push notification for MacSeisApp contains a message composed of the magnitude of the earthquake (based on the Richter scale), the location, the date, and the time of the earthquake.

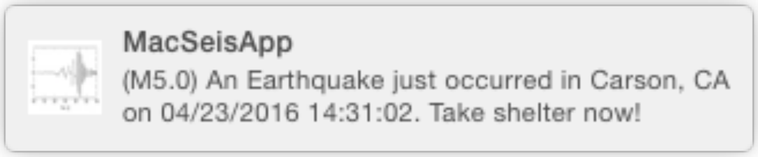
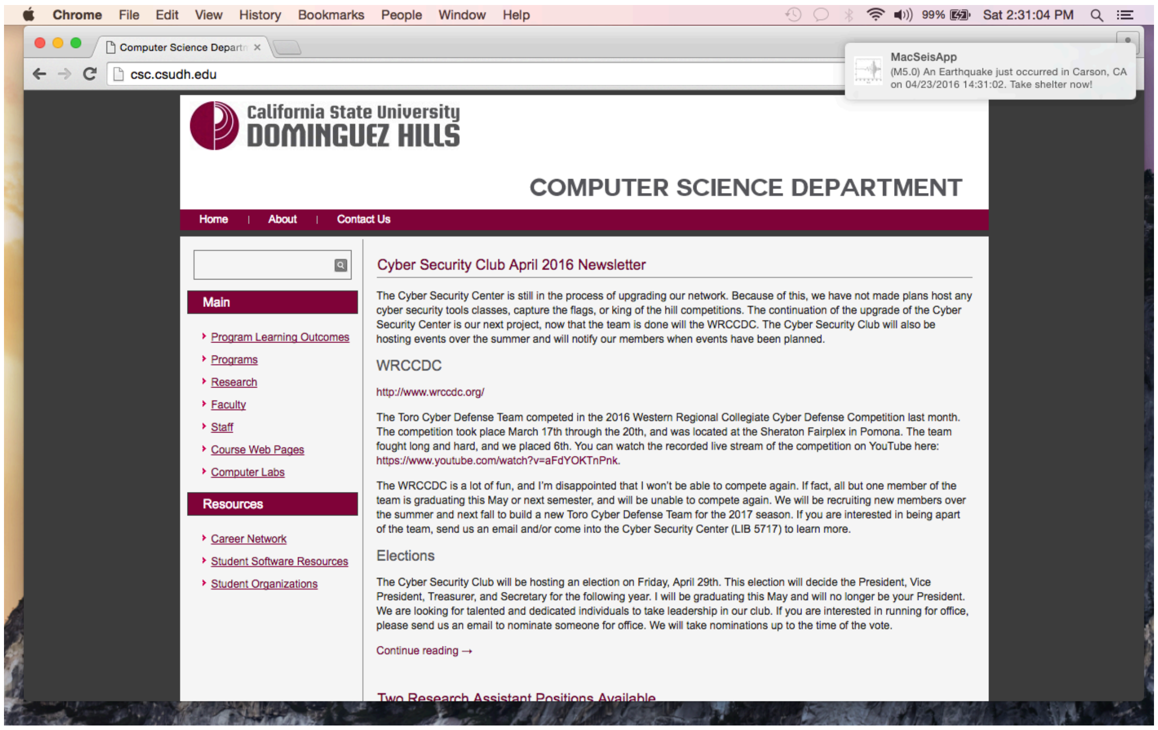

MacSeisApp
(M5.0) An Earthquake just occurred in Carson, CA on 04/23/2016 14:31:02. Take shelter now!

Figure 4.3.1: A Typical Push Notification from MacSeisApp

For example, there are two users running MacSeisApp located in the city of Carson, California; one user is located at California State University, Dominguez Hills and a second user is located at an apartment complex near the South Bay Pavilion shopping center. Both users detect a “spike” on the seismograph interface when running the application within a four-second interval, and the PHP script on the California State University, Dominguez Hills verifies that it has been executed by two different users. This script then proceeds to send a push notification request to Apple’s push notification server, which generates and sends a push notification similar to the one shown in the image in Figure 4.3.1 to other users in nearby cities, such as Torrance, Long Beach, Los Angeles, Irvine, and Riverside. The users in the nearby cities will then receive the push notification immediately within one second and have it show up on the top-right hand corner of their screen, similar to what is shown in Figure 4.3.2. Those users are guaranteed to receive the push notifications just seconds before the seismic wave reaches them. This is because data and network speeds are much faster than the speeds at which seismic waves travel through the ground.


Chrome File Edit View History Bookmarks People Window Help
99% Sat 2:31:04 PM
Computer Science Depart
csc.csudh.edu
California State University DOMINGUEZ HILLS
COMPUTER SCIENCE DEPARTMENT
Home | About | Contact Us
Main
Program Learning Outcomes
Programs
Research
Faculty
Staff
Course Web Pages
Computer Labs
Resources
Career Network
Student Software Resources
Student Organizations

MacSeisApp
(M5.0) An Earthquake just occurred in Carson, CA on 04/23/2016 14:31:02. Take shelter now!

Cyber Security Club April 2016 Newsletter

The Cyber Security Center is still in the process of upgrading our network. Because of this, we have not made plans host any cyber security tools classes, capture the flags, or king of the hill competitions. The continuation of the upgrade of the Cyber Security Center is our next project, now that the team is done will the WRCCDC. The Cyber Security Club will also be hosting events over the summer and will notify our members when events have been planned.

WRCCDC

http://www.wrccdc.org/

The Toro Cyber Defense Team competed in the 2016 Western Regional Collegiate Cyber Defense Competition last month. The competition took place March 17th through the 20th, and was located at the Sheraton Fairplex in Pomona. The team fought long and hard, and we placed 6th. You can watch the recorded live stream of the competition on YouTube here: https://www.youtube.com/watch?v=aFdYOKTnPnk.

The WRCCDC is a lot of fun, and I'm disappointed that I won't be able to compete again. If fact, all but one member of the team is graduating this May or next semester, and will be unable to compete again. We will be recruiting new members over the summer and next fall to build a new Toro Cyber Defense Team for the 2017 season. If you are interested in being apart of the team, send us an email and/or come into the Cyber Security Center (LIB 5717) to learn more.

Elections

The Cyber Security Club will be hosting an election on Friday, April 29th. This election will decide the President, Vice President, Treasurer, and Secretary for the following year. I will be graduating this May and will no longer be your President. We are looking for talented and dedicated individuals to take leadership in our club. If you are interested in running for office, please send us an email to nominate someone for office. We will take nominations up to the time of the vote.

Continue reading →

Two Research Assistant Positions Available

Figure 4.3.2: The Placement of the Push Notification on the Screen

CHAPTER 5

CONCLUSION & FUTURE WORK

5.1. Conclusion

The development of the MacSeisApp application has been a very promising experience in the sense that an existing concept has been refined and further expanded for novel applications with the potential to someday save the lives and property of the general public. Combining this novel concept with the various technologies and third-party libraries available today makes it possible to make this application a reality.

This is especially the case with MacSeisApp, where its development was first inspired by the SeisMac application that can be downloaded free of charge. Thanks to the introduction of Apple's Sudden Motion Sensor in 2006, Apple's Push Notification Service in 2009, and the *smslib* and *OpenGL* libraries, MacSeisApp has evolved from a new, refined idea to become real product. The original goal of MacSeisApp was to refine and expand upon the seismograph interface found in SeisMac by utilizing the real-time notification system via push notifications. Although the original vision of the project has been realized, it is important to note that the development of MacSeisApp is an ongoing process, as there are several more features that can be integrated at a later date, which is discussed further in the next section.

5.2. Future Work and Vision

MacSeisApp is an ongoing project that is full of potential, with several features that can be integrated into the MacSeisApp application in the future. One such feature includes having a counterpart application for iOS where end-users can also receive push notifications on their iPhone or iPad devices as well. Additionally, this counterpart application would also be able to display the latest five to ten earthquake events to the end-users. This could become a very effective way to instantly notify a large population of users over multiple platforms.

Another feature that may be added in the future is an options or settings panel where the end-user can enable or disable the push notifications functionality. This can also include the option to enable push notifications within a certain radius from the end-user's current location, such as within a 50 or 100-mile radius. This way, an end-user located in the San Francisco Bay Area, for example, can essentially opt-out of receiving push notifications for earthquake activity that is reported by end-users in the Los Angeles area.

Finally, a third feature that can be integrated to the MacSeisApp application in the future is an improved version of the smart seismic activity detection algorithm that differentiates between artificial or man-made shaking and legitimate seismic activity. One way that the existing version of this feature can be improved is to have the script running on the dedicated server at California State University, Dominguez Hills run an additional algorithm that is able to obtain all shaking activity data from all users and compare it. Although MacSeisApp is able to require reports from two different users within a four-second interval before a push notification is triggered, the current algorithm does collect any seismic data from the user. This can be problematic with regards to accuracy in that one user may report a spike on the *x*-axis while the other user reports a spike on the *y*-axis, which would not necessarily be considered related seismic activity. Additionally, if the data collected from each user has a similar pattern on each axis within this four-second interval, then the dedicated server will proceed with sending the push notification to all users of MacSeisApp. This feature would greatly help protect the integrity of the MacSeisApp application and to significantly reduce, if not, eliminate "false alarms."

REFERENCES

REFERENCES

[1] E. Cochran, J. Lawrence, C. Christensen, and R. Jakka. The quake catcher network: Citizen science expanding seismic horizons. Seismological Research Letters, 80(1):26–30, 2009.

[2] Griscom, Daniel. "SeisMac 3.0." *Suitable Systems - SeisMac 3.0*. Suitable Systems, 2012. Web. <http://www.suitable.com/tools/seismac.html>.

[3] Griscom, Daniel. "SMSLib." *Suitable Systems - SMSLib*. Suitable Systems, 2012. Web. <http://www.suitable.com/tools/smslib.html>.

[4] "Local and Remote Notification Programming Guide." *Apple Push Notification Service*. Apple, Inc. Web. <https://developer.apple.com/library/mac/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html>.

[5] "The Richter and Mercalli Scales." *The Geography Site*. National Grid For Learning, 26 March 2013. Web. <http://www.geography-site.co.uk/pages/physical/earth/richt.html>.

[6] "OpenGL Overview." *OpenGL - The Industry's Foundation for High Performance Graphics*. The Khronos Group. Web. <https://www.opengl.org/about/>.

APPENDIX

SOURCE CODE

```
//
//  AppDelegate.h
//  MacSeisApp
//
//  The header file for the AppDelegate
//
//

#import <Cocoa/Cocoa.h>

@interface AppDelegate : NSObject <NSApplicationDelegate>


@end
```

```
//
//  AppDelegate.m
//  MacSeisApp
//
//  The AppDelegate executes any code needed at startup and shutdown of application
//
//

#import "AppDelegate.h"
#import "smslib.h"

@interface AppDelegate ()

@end

@implementation AppDelegate

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    // Insert code here to initialize your application

            // Used to tell OS that this app wants to receive push notifications
    [NSApp registerForRemoteNotificationTypes:(NSRemoteNotificationTypeBadge |
                NSRemoteNotificationTypeSound | NSRemoteNotificationTypeAlert)];

    smsStartup(nil, nil);     // Used to start access to Sudden Motion Sensor (SMS)
    smsLoadCalibration();     // Used to load any stored calibration values
}

- (void)application:(NSApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
    NSLog(@"My token is: %@", deviceToken);
}

- (void)application:(NSApplication *)application
didFailToRegisterForRemoteNotificationsWithError:(NSError *)error
{
    NSLog(@"Failed to get token, error: %@", error);
}

- (void)applicationWillTerminate:(NSNotification *)aNotification {
    // Insert code here to tear down your application
    smsShutdown();          // Shut down SMSLib once application is killed
}

@end
```

```
//
//  ViewController.h
//  MacSeisApp
//
//  The header file for the ViewController
//
//

#import <Cocoa/Cocoa.h>
#import "smslib.h"
#import "Seismograph.h"

@interface ViewController : NSViewController
{
    sms_acceleration accel;
    Seismograph* glView;
}

@end
```

```
//
//  ViewController.m
//  MacSeisApp
//
//  The ViewController handles displaying the application window and its contents
//
//

#import "ViewController.h"

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    // Do any additional setup after loading the view.

    glView = [[Seismograph alloc] init];

                        // Fire collectSMSData every 0.1 seconds
    [NSTimer scheduledTimerWithTimeInterval:0.1f
            target:self selector:@selector(collectSMSData:) userInfo:nil repeats:YES]
}

- (void) collectSMSData:(NSTimer*)timer   // Collect data from Sudden Motion Sensor
{
    smsGetData(&accel);
    insert_data(accel.x, accel.y, accel.z);  // Call method to insert data into array
}

- (void)setRepresentedObject:(id)representedObject {
    [super setRepresentedObject:representedObject];

    // Update the view, if already loaded.
}

@end
```

```
//
//  Seismograph.h
//  MacSeisApp
//
//  The header file for the Seismograph
//
//

#import <Cocoa/Cocoa.h>

@interface Seismograph : NSOpenGLView
{
    @public
    NSOpenGLPixelFormat* pixelFormat;
    NSOpenGLContext* glContext;
}

- (void) drawRect: (NSRect) bounds;

void insert_data(double x, double y, double z);

void plotSeismicData();

@end
```

```
//
//  Seismograph.m
//  MacSeisApp
//
//  The Seismograph class handles the algorithm of collecting data and plotting
//    the collected data as a seismograph, updating once every 0.1 seconds
//

#import "Seismograph.h"
#include <OpenGL/gl.h>
#include <GLUT/glut.h>

@implementation Seismograph

        // Global variables needed
double x_axis[100];
double y_axis[100];
double z_axis[100];
NSRect theBounds;
bool timerSet = NO;
bool spikeDetected = NO;
int spikeCount = 100;
int onlyOnce = 1;

void insert_data(double x, double y, double z)  // Inserts data into each axis array
{
    double avg_x = 0;
    double avg_y = 0;
    double avg_z = 0;

    for(int i = 0; i < 99; i++)
    {                           // Shift all values back one index space
        x_axis[i] = x_axis[i + 1];
        y_axis[i] = y_axis[i + 1];
        z_axis[i] = z_axis[i + 1];

        avg_x += x_axis[i];
        avg_y += y_axis[i];
        avg_z += z_axis[i];
    }

    avg_x /= 99.0;
    avg_y /= 99.0;
    avg_z /= 99.0;

    if(fabs(x - avg_x) > 0.03 && fabs(y - avg_y) > 0.03)    // Earthquake above M5.0
    {                                                       //  detected
        NSString* magnitude = @"";

        if(fabs(x - avg_x) >= 1 || fabs(y - avg_y) >= 1)
        {
            // Above Magnitude 8.2
            magnitude = @"8.2%2B";      // %2B is used to represent a + for a URL
        }
```

```
else if(fabs(x - avg_x) >= 0.75 || fabs(y - avg_y) >= 0.75)
{
    // Magnitude 8.0
    magnitude = @"8.0";
}
else if(fabs(x - avg_x) >= 0.5 || fabs(y - avg_y) >= 0.5)
{
    // Magnitude 7.3
    magnitude = @"7.3";
}
else if(fabs(x - avg_x) >= 0.25 || fabs(y - avg_y) >= 0.25)
{
    // Magnitude 6.6
    magnitude = @"6.6";
}
else if(fabs(x - avg_x) >= 0.1 || fabs(y - avg_y) >= 0.1)
{
    // Magnitude 6.1
    magnitude = @"6.1";
}
else if(fabs(x - avg_x) >= 0.05 || fabs(y - avg_y) >= 0.05)
{
    // Magnitude 5.4
    magnitude = @"5.4";
}
else
{
    // Magnitude 5.0
    magnitude = @"5.0";
}

spikeDetected = YES;

if(onlyOnce == 1)
{
    onlyOnce = 0;

    // Create the request that is to be sent to the server
    NSMutableURLRequest *request =
        [[NSMutableURLRequest alloc] initWithURL:
            [NSURL URLWithString:
                [@"http://apns.csudh.edu/~apns/simplepush.php?mag="
                   stringByAppendingString:magnitude]]];

    [request setHTTPMethod:@"GET"];
    [request addValue:@"getValues" forHTTPHeaderField:@"METHOD"];
            // Selects the task the server will perform

    // Start an NSURLConnection with the request
    NSURLConnection *connection = [[NSURLConnection alloc]
                                   initWithRequest:request delegate:nil];
    if(!connection)
    {
```

```
                NSLog(@"Connection Failed");
            }
        }
    }

    x_axis[99] = x;         // Insert new data at the end of array
    y_axis[99] = y;
    z_axis[99] = z;
}

void plotSeismicData()      // Plots data collected from SMS for each
{                           //  index of respective arrays for the
    double pos = -1.0;      //   x, y, and z axes

    glColor3f(1.0f, 1.0f, 1.0f);
    glBegin(GL_LINES);
    {
        for(int i = 0; i < 99; i++)
        {
            glVertex2d(pos, x_axis[i] + 0.67);
            pos += 0.02;
            glVertex2d(pos, x_axis[i + 1] + 0.67);
        }
    }
    glEnd();

    pos = -1.0;

    glBegin(GL_LINES);
    {
        for(int i = 0; i < 99; i++)
        {
            glVertex2d(pos, y_axis[i]);
            pos += 0.02;
            glVertex2d(pos, y_axis[i + 1]);
        }
    }
    glEnd();

    pos = -1.0;

    glBegin(GL_LINES);
    {
        for(int i = 0; i < 99; i++)
        {
            glVertex2d(pos, (z_axis[i] - 1) - 0.67);
            pos += 0.02;
            glVertex2d(pos, (z_axis[i + 1] - 1) - 0.67);
        }
    }
    glEnd();
}
```

```
void xAxisLine()        // Draws a horizontal line for the x-axis
{                       //  towards the top of the window
    glColor3f(0.0f, 1.0f, 0.0f);
    glBegin(GL_LINES);
    {
        glVertex2d(-1.0, 0.67);
        glVertex2d(1.0, 0.67);
    }
    glEnd();
}

void yAxisLine()        // Draws a horizontal line for the y-axis
{                       //  in the middle of the window
    glColor3f(0.0f, 1.0f, 0.0f);
    glBegin(GL_LINES);
    {
        glVertex2d(-1.0, 0.0);
        glVertex2d(1.0, 0.0);
    }
    glEnd();
}

void zAxisLine()        // Draws a horizontal line for the z-axis
{                       //  towards the bottom of the window
    glColor3f(1.0f, 0.0f, 0.0f);
    glBegin(GL_LINES);
    {
        glVertex2d(-1.0, -0.67);
        glVertex2d(1.0, -0.67);
    }
    glEnd();
}

void drawAxisLines()    // Container function that calls each of the
{                       //  functions to draw the horizonal lines for
    xAxisLine();        //   the x, y, and z axes
    yAxisLine();
    zAxisLine();
}

-(void)render:(NSTimer*)timer   // Uses timer to refresh the window and
{                               //  render current data available
    [self setNeedsDisplay:YES];
}

-(void) drawRect: (NSRect) bounds   // Handles drawing the window for seismograph
{
    if(timerSet == NO)
    {
        [ NSTimer scheduledTimerWithTimeInterval:0.1f target:self
                           selector:@selector(render:) userInfo:nil repeats:YES ];
        timerSet = YES;
    }
    if(spikeDetected)       // Spike has been detected, makes background
    {                       //  red color for limited amount of time
        glClearColor(1, 0, 0, 0.3);
        spikeCount--;

        if(spikeCount == 0)
        {
            spikeDetected = NO;
            spikeCount = 100;
            onlyOnce = 1;
        }
    }
    else                    // Normal conditions, display default
    {                       //  black background until spike is detected
        glClearColor(0, 0, 0, 1.0);
    }

    glClear(GL_COLOR_BUFFER_BIT);
    drawAxisLines();
    plotSeismicData();
    glFlush();
}

@end
```

```
//
//  simplepush.php
//  MacSeisApp
//
//  The PHP script for the server, which includes the smart seismic activity
//    detection algorithm
//  Note: Asterisks (‘*’) are used in place of sensitive information
//

<?php

// First, keep track of how many times this script is run
$counterFile = ("ctr.txt");
$count = file($counterFile);
$count[0] += 1;
$fp1 = fopen($counterFile, "w");
fputs($fp1, "$count[0]");
fclose($fp1);
$isEarthquake = False;

$tmpVal = $count[0];
if($tmpVal == 1)      // This instance is the only one,
{                     //  wait to see if there is another instance
    sleep(4);
    $count = file($counterFile);

    if($count[0] == $tmpVal)    // No other instance, reset count
    {                           //  back to 0, do not send APNS
     $fp1 = fopen($counterFile, "w");
     fputs($fp1, "0");
     fclose($fp1);
    }
    // Otherwise, there is another instance, APNS triggered
}
elseif($tmpVal == 2)    // This is the second instance, APNS
{                       //  can be triggered
    $isEarthquake = True;
    $fp1 = fopen($counterFile, "w");
    fputs($fp1, "0");
    fclose($fp1);
}

if($isEarthquake)
{
     // Put device tokens here (without spaces):
     //      [NOTE: Actual tokens are not displayed for Appendix]
     $deviceToken = array('**********************************',
                          '**********************************');


     // Put your private key's passphrase here:
     //      [NOTE: Actual passphrase is not displayed for Appendix]
     $passphrase = '*******';
     // Get server date and time
     date_default_timezone_set('America/Los_Angeles');
     $date = date_create();

     // Put your alert message here:
     $message = '(M' . htmlspecialchars($_GET["mag"]) .
         ') An Earthquake just occurred in Carson, CA on ' .
         $date->format('m/d/Y H:i:s') .
         '. Take shelter now!';

////////////////////////////////////////////////////////////////////////////////

     $ctx = stream_context_create();
     stream_context_set_option($ctx, 'ssl', 'local_cert', 'ck.pem');
     stream_context_set_option($ctx, 'ssl', 'passphrase', $passphrase);

     // Open a connection to the APNS server
     $fp = stream_socket_client(
         'ssl://gateway.sandbox.push.apple.com:2195', $err,
         $errstr, 60, STREAM_CLIENT_CONNECT|STREAM_CLIENT_PERSISTENT, $ctx);

     if (!$fp)
         exit("Failed to connect: $err $errstr" . PHP_EOL);

     echo 'Connected to APNS' . PHP_EOL;

     // Create the payload body
     $body['aps'] = array(
         'alert' => $message,
         'sound' => "warning.caf",
         'badge' => 1
         );

     // Encode the payload as JSON
     $payload = json_encode($body);

     // Build the binary notification
     foreach ($deviceToken as $dt) {
         $msg = chr(0) . pack('n', 32) . pack('H*', $dt) .
                    pack('n', strlen($payload)) . $payload;

         // Send it to the server
         $result = fwrite($fp, $msg, strlen($msg));

         if (!$result)
            echo 'Message not delivered' . PHP_EOL;
        else
            echo 'Message successfully delivered' . PHP_EOL;
     }

     // Close the connection to the server
     fclose($fp);
}
```