3-Dimensional Weather Visualisation

Project Report

Sarah Nimitz
Duke Forsyth
Andrew Knittle

May 4, 2016
Blacksburg, VA, 24061

Course: CS 4624 Spring 2016
Instructor: Dr. Edward Fox
Client: Zachary Duer, zachduer@vt.edu

Table of Contents

Table of Figures

Table of Tables

I. Executive Summary

Abstract

A detailed description in the creation of a polling and parsing system for keeping track of severe weather warnings, as delivered by the National Weather Service, and an interface to allow the user to view a representation of Doppler radar data in three dimensions. This describes the roles of the team members, the work accomplished over the Spring 2016 semester, and the methods by which the team accomplished this work.

II. Introduction

Objectives

This project's primary function is to give dynamically created 3-dimensional visualizations of severe weather events. Normally, the data collected by a Doppler radar is not accurately portrayed in 2-dimensions, as Doppler radar collects information at a slight upward angle and a 2-dimensional representation cannot easily convey information based on altitude as well as latitude and longitude. In order to prevent the loss of this information, this project aims to create a 3-dimensional virtual reality through which severe weather events can be more accurately monitored.

Meteorologists could use this project to study severe weather events in a 3-dimensional viewing field. The Weather Channel has expressed interest in this project's success, and as such the project must be suitable for presentation and be able to accurately and easily convey information regarding severe weather events. Sarah Nimitz, Duke Forsyth, and Andrew Knittle work on two portions of the larger project: the frontend and the backend.

Frontend

While the client is working on the virtual environment itself, the team is working on developing the interface to control the user's position inside the virtual environment so that the user can have a different perspective. This interface will allow the user to move a virtual camera forward, backward, or turn.

Backend

The client has requested that the group work on a system to poll the National Weather Service's severe weather statements from the College of DuPage [16], parse them for important information, and store the warnings. These warnings will also be updated as new statements are released or updated, so that duration, type, and location of the warnings are correct for rendering. The client's program will then utilize several provided function calls in order to reference important warning information.

Team Roles

The team has broken into two subteams in order to tackle the two fairly independent portions of the project the client has assigned. These will be known as the "Frontend" and "Backend" teams. The Frontend team is in charge of the changes to the user's viewpoint into the rendered weather system and the Backend team is in charge of gathering relevant weather statement information from the requirement given from the frontend.

Sarah Nimitz

Member of the Backend team. She is responsible for the parsing of each of the severe weather statements and collecting the relevant weather warning information into objects that can then be referenced by the rendering program. This information includes warning ID, warning duration, warning type, warning geometry, and warning location. She will assist in the polling of the server if necessary. She is also in charge of editing and finalizing documentation, due to her background in English and writing.

Duke Forsyth

Co-member of the Backend team. He is responsible for the listening to and polling of the severe weather statement database [16]. A listener must be made so that upon a severe weather statement published it is collected by the program and parsed into the correct format. He will assist in the parsing of each severe weather warning text file if necessary. He is also in charge of collecting and compiling information based on the team's contributions and meeting times, due to his previous experience.

Andrew Knittle

The sole member of the Frontend team. He will make a user interface (UI) that allows the user to dictate movement on a 2-dimensional plane inside a 3-dimensional environment. The UI will consist of four different buttons (forward, backward, left and right) which alter the corresponding X- and Y-coordinates of the information rendered. The UI will also display the user's latitude and longitude relative to the 3-dimensional scene. The UI will be represented on a touchscreen tablet or phone. Most, if not all, of the code will be done in C# as it is the primary language used in Unity. Since Andrew is working alone on this section of the project, there is no practical reason to use any code sharing software such as Git or GitHub.

III. Users' Manual

User Interactions

The user will be able to utilize the system from the frontend. Using an augmented or virtual reality headset, the user is able to visualize the weather system in a 3-dimensional environment. A user interface is available on a tablet in order to control the available 3-dimensional representation that the user can view in the space. This allows them to pan left or right and generally adjust the view of the weather system.

By polling the available weather warnings, the user will be able to assess what weather system may interest them most. They are able to assess these warnings either in real time or by requesting a previous point in time with the radar information and warnings available at the time. The user is then able to explore the 3-dimensional Doppler radar information in order to assess the weather system and its attributes.

Terminology

Augmented Reality (AR)	Provides an onscreen display without limiting the user's vision via goggles/glasses [11].
The Cube	A four-story high ICAT laboratory in Virginia Tech's Moss Center for the Arts, equipped with real-time audio/visual rendering, augmented reality, and various other creative and scientific tools and instruments [2].
DirectX	A collection of application programming interfaces (APIs) for handling tasks related to multimedia, especially game programming and video, on Microsoft platforms [3].
Doppler Radar	A system that sends out several 360 degree sweeps of varying angles from a single site that can track precipitation and the motion of precipitation [4].
Institute for Creative Arts and Technology (ICAT)	An organization at Virginia Tech focused on the interdisciplinary studies between education, technical studies, and the creative arts [1].
The National Weather Service	The official U.S. governmental source for U.S. weather, marine, fire and aviation forecasts, warnings, meteorological products, climate forecasts and information about meteorology [5].

OpenGL	A cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics [6].
SlimDX	A free open source framework that enables developers to easily build DirectX applications using .NET technologies such as C#, VB.NET, and IronPython [7].
Unity	A flexible and powerful development platform for creating multi-platform 3D and 2D games and interactive experiences [8].
Virtual Reality (VR)	A display that fully engulfs the user's vision with an artificially created physical scene [12].
The Weather Channel	A weather channel owned by a consortium (including The Blackstone Group, Bain Capital, and NBCUniversal) that provides a national and local weather forecast for cities, as well as weather radar, report, and hurricane coverage [9][10].

Table 1. A list of terms and definitions to assist in understanding the project.

IV. Developers' Manual

Application Overview

Business Process

This project is not, at this time, intended for sale or commercial use. The primary way in which this project would affect the business process would be the usage of this system by meteorologists for academic purposes. In the current system, a meteorologist is able to view weather systems in a 2-dimensional presentation. This loses a lot of information in the process, or at least makes it a hassle to get data from certain specific points of a weather system (e.g. the center of a storm).

Our client's project would alter this system fundamentally. Instead of a system that can only easily represent information in 2-dimensions, this system will allow the weather to be visualized in 3-dimensions. An area similar to ICAT's Cube allows the meteorologist to visualize a weather system in the space around himself, and he can walk through the system or view the system from all possible angles, moving the viewing section via a tablet. Our project will take note of all severe weather events occurring at a point in time to assist the individual controlling the geographic location being simulated in choosing the most interesting area, whether by relevance to the project at hand or by the most activity.

User Roles and Responsibilities

The Academic Meteorologist is someone who studies weather systems. They can utilize the 3-dimensional space in order to better understand the weather phenomena at work. They will assess what severe weather events are occurring at a certain point in time and use the Unity-based UI to manipulate the 3-dimensional visualization to focus on the geographical location they wish to study.

The National Weather Service Worker is the person who monitors the weather outside of our project and produces a severe weather warning. This is published, collected on the College of DuPage's server [16], and is then polled by our program to better assess areas of interest for the user.

The Code Maintainer is the programmer who continues to work on the project after its initial completion. They are to perform all code maintenance. They will find and solve any incompatibilities and any other type of issue that arises in the system. The Maintainer will also implement new features or remove old ones as the project evolves.

Interactions with Other Systems

Severe Weather Event Poller

Upon a severe weather event occurring, the National Weather Service publishes a severe weather statement detailing the location, the type, the duration, the identification number (ID), and other important information regarding each severe weather event. In these statements, they may also provide updates for previous severe weather events, such as a change in location or an extension in a warning.

These statements are collected on a server hosted by the College of DuPage [16]. Their file system sorts the statements by time of publication and by type. Each file follows the naming format “YYYYMMDDHH” where the four-digit year is followed by the two-digit month, the two digit day, and the two digit hour in Zulu time of publication. The files are further divided by their suffix representing what type of statements are stored there-- for example, 2016022501.SVS will hold all severe weather statements published in the first Zulu hour of February 25, 2016 while 2016022501.SVR will hold all severe thunderstorm warnings published in the first Zulu hour of February 25, 2016.

Severe Weather Statement Parser

The Severe Weather Statement Parser is fed individual severe weather statements. This information is then stored in a severe weather statement object for reference, as well as the referenced warning.

These files can then be parsed at a later time in order to reference warning information to identify when the warning was active, previous states of the warning, and a text compilation of all severe weather statements associated with the warning.

Warning Processing Controller

When the Severe Weather Event Poller pulls each file from DuPage’s server, the controller breaks it down into its individual statements, and passes it to the Severe Weather Statement Parser.

An API is then used to connect our collection of severe weather statements to a system that renders the severe weather warnings and emergencies onto the 2-Dimensional Weather Renderer. The severe weather objects are pulled in order to provide the necessary information to accurately create these geometries and provide the user information regarding what these geometries represent.

2-Dimensional Weather Renderer

By polling the information collected by a Doppler radar, an accurate weather visualization of weather systems is created. By requesting information from the Warning Processing Controller, it is able to overlay its Doppler data with the warning geometry of active weather warnings, as well as allow the user to view all available data for each warning by interacting with it. Using user input, the 2-Dimensional Weather Renderer is able to inform the Unity Weather Renderer what Doppler data to render.

Unity Weather Renderer

Using information provided by the 2-Dimensional Weather Renderer, the Unity Weather Renderer takes 3-dimensional Doppler radar data and creates a 3-dimensional representation at the location requested. This information, upon being rendered, is left as an environment for the user to explore using the Unity Interface.

Unity Interface

Per request of the client, the interface is rendered onto a tablet. This device will be the main interface for navigating the 3-dimensional space of the weather system. The interface itself will be displayed on a touchscreen tablet while the user uses an accompanying pair of AR and/or VR goggles. Moving information to and from the Unity Weather Renderer, this interfaces provides navigational part of the project, allowing the user to interact and move around with the 3-dimensional weather.

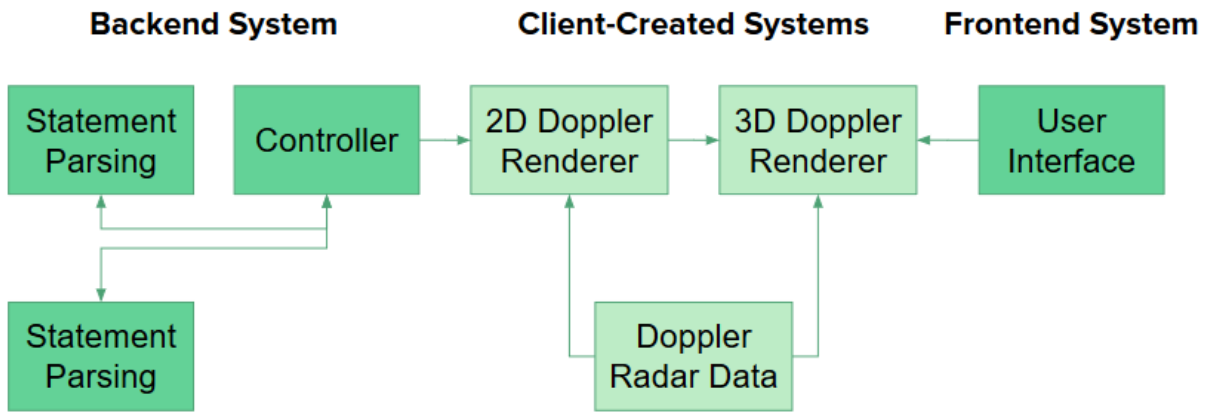

Figure 1. The flow of information between the frontend, backend, and client-created systems.

Production Rollout Considerations

This prototype is due by the middle of May 2016. The Weather Channel intends to come to the Virginia Tech campus and view the 3-dimensional weather visualization. If the project is done to a complete, or near-complete, state by the time this visit occurs, it is within the realm of imagination that The Weather Channel may consider using this project for their own uses.

By the time this visit occurs, the team would have to have a working prototype for The Weather Channel to use. Should The Weather Channel be reasonably impressed with our project and the progress the team has made, and they decide to use it, it is likely that several other meteorological specialists would pursue gaining access to the project as well. For this reason, the team must focus on getting a working and reasonably successful prototype ready for the May visit.

Functional Requirements

Statement of Functionality

The project is to have a 3-dimensional visualization of weather phenomena. The clients have broken this project into two separate tasks:

Frontend: An interface with the Unity game engine that the user will be able to use to pan across the current area and navigate around the weather phenomena. The client is then going to use our interface to communicate with the Unity engine and update the user's field of view to gain different perspectives of the 3-dimensional scene.

Backend: A subsystem has been created to poll the College of DuPage's servers [16] for needed severe weather statements published by the National Weather Service. The information will then be parsed and sent to the 2-dimensional representation system via SlimDX. This information will be represented in a 2- and 3-dimensional manner as per user requests. The information will be stored and logged**.** In order to allow the user to review older statements, and due to the miniscule storage space required to store these text files compared to what is required for the rendering, the backend has been delegated to store all relevant severe weather statements from the beginning of execution to the termination of the program per request of the client.

Reporting: Our group has been recording our meetings with the client and with each other on an Excel file. The file has a start and end time so that we have a rough overall idea of what the group as a whole is doing. We are currently setting up a personal timesheet for each person so that we will have a proper understanding of who did what and we currently have it set for bi-monthly intervals.

Scope

The frontend will have the Unity engine work with the group's interface, which will then in turn communicate with the backend. The backend will poll the National Weather Service's statements and parse the needed information from both the weather statements and radar information from the given area. After getting the required information, the backend will then send this information to the frontend for the Unity engine to then create a 3-dimensional representation of the information requested.

V. Design

Overview

This project has been designed into two parts for the team to tackle. First, the Frontend of the project will allow the user to manipulate their view of the 3-dimensional representation of a storm system. The Backend of this project will poll severe weather statements from the National Weather Service in order to provide an accurate list of all active weather warnings and their details.

Frontend

The team designed an interface that allows panning through the 3-dimensional environment created by the client. The 3-dimensional object's shape and position will be streamed to Unity and movement of the point-of-view (POV) will be done via an interface built in Unity. The information feeding into the interface will update every five minutes as data comes from a live radar feed.

The client chose to use Unity and to utilize C# as the language for this project.

Unity was chosen for its extensibility. Creating ports to various platforms and OSs is incredibly simple and the API is easy to use after a little practice. There is a dialogue box that appears after finalizing a project to choose which platform to deploy the project to such as Windows, Mac, Unix, Mobile, etc. This is incredibly useful since the scale of the Unity interface can be used on nearly any modern OS and on various devices with different degrees of hardware limitations.

The highest priority from the client was the panning interface. Fortunately, Unity's API is very extensible with built in functionality of camera control (panning, rotating, etc.). Most of the code will be done in C#. The interface itself will be used on a tablet in tandem with either a virtual reality (VR) or augmented reality (AR) headset. In either scenario there will be two instances of Unity running: one for the interface on the tablet (the primary focus of the Frontend team) and one for the visualization.

The next objective is to develop specific versions for VR and AR. The interface will be a general purpose interface, but the interface could be altered to better suit the needs of either VR or AR.

Backend

In order to collect all current severe weather statements, the polling system requests from the College of DuPage's server a collection of all severe weather statements every five minutes [16]. This is to mirror the Unity's ability to update every five minutes.

The collected severe weather statement text files (see Figure 2) are then parsed using regular expressions to find the following information: type of warning, warning ID, geographical

location, geometry of the warning, duration of the warning, publication time of the associated statement, movement of the system, and updates if any should come up.

```
347
WUUS52 KMLB 150033
SVRMLB
FLC061-150130-
/O.NEW.KMLB.SV.W.0029.160415T0033Z-160415T0130Z/

BULLETIN - IMMEDIATE BROADCAST REQUESTED
SEVERE THUNDERSTORM WARNING
NATIONAL WEATHER SERVICE MELBOURNE FL
833 PM EDT THU APR 14 2016

THE NATIONAL WEATHER SERVICE IN MELBOURNE HAS ISSUED A

* SEVERE THUNDERSTORM WARNING FOR...
  CENTRAL INDIAN RIVER COUNTY IN EAST CENTRAL FLORIDA...

* UNTIL 930 PM EDT

* AT 833 PM EDT...A SEVERE THUNDERSTORM WAS LOCATED OVER BLUE CYPRESS
  LAKE...MOVING EAST AT 15 MPH.

  HAZARD...60 MPH WIND GUSTS AND QUARTER SIZE HAIL.

  SOURCE...RADAR INDICATED.

  IMPACT...HAIL DAMAGE TO VEHICLES IS EXPECTED. EXPECT WIND DAMAGE
         TO ROOFS...SIDING...AND TREES.

* LOCATIONS IMPACTED INCLUDE...
  SEBASTIAN...FELLSMERE...BLUE CYPRESS LAKE...POINTE WEST AND VERO LAKE
  ESTATES.

PRECAUTIONARY/PREPAREDNESS ACTIONS...

FOR YOUR PROTECTION MOVE TO AN INTERIOR ROOM ON THE LOWEST FLOOR OF A
BUILDING.

&&

LAT...LON 2756 8060 2756 8062 2771 8087 2782 8080
      2777 8049 2764 8047
TIME...MOT...LOC 0033Z 292DEG 10KT 2775 8076

HAIL...1.00IN
WIND...60MPH

$$

15
```

Figure 2. An example of a weather statement, published on 14 April 2016.

The only types of warnings that are stored by the polling program are flash flood warnings, severe thunderstorm warnings, and tornado warnings (further broken down into tornado warning, tornado reported, or tornado emergency). These warnings are published by the National Weather Service.

This information is then used to create a severe weather statement object, and either used to create or to update a warning object. Upon a warning being created, a file is created in a local directory titled “Warnings.” Within the Warnings directory, the files are further divided by year. Each folder titled by year contains all weather statements published within that year and parsed by the team’s program. A third directory titled “Unlinked” contains all severe weather statements not associated with a warning or associated with a warning the parser does not support. The file names following the style “SSSSMMDD” such that the four-letter representation of the station from which the warning was published is followed by the two-digit month and the two-digit day. The file itself contains a list of all unlinked severe weather statements published that day, sorted from most recent to oldest.

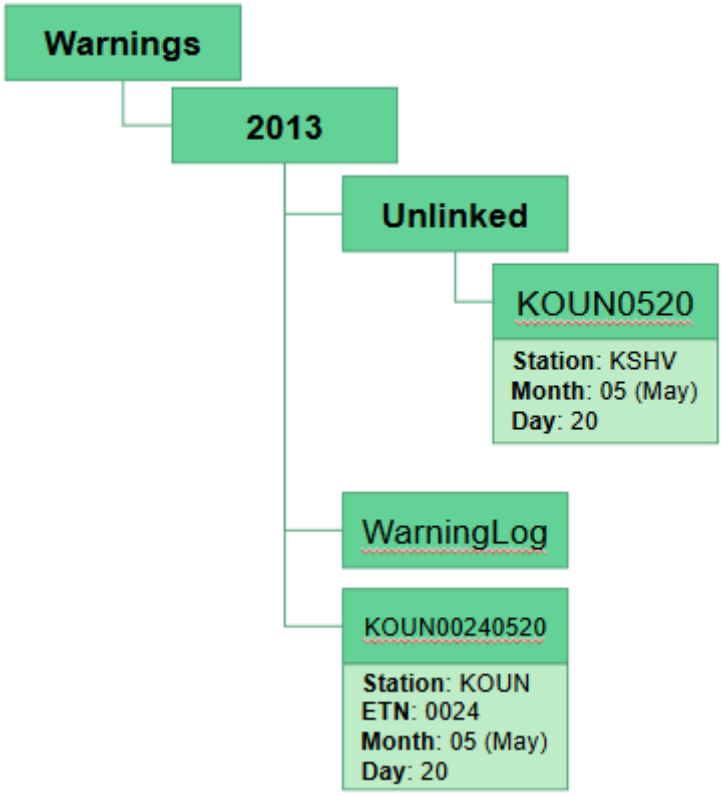

Figure 3. The file architecture for storing the parsed severe weather statements and their related warnings.

The warning files themselves, sorted within the directory by year, follow the naming style “SSSS####MMDD” such that the four-letter station the warning is associated with is followed by the event tracking number (ETN) that identifies the warning, the two-digit month of the warning starting, and the two-digit day of the warning starting. A final file titled “Warning Log” sits in each year directory, containing a list of all weather warnings and their associated start and end times, sorted by end time.

Due to the miniscule size of these warning files compared to all of the other objects in the program, namely the rendering, we have been asked to never trash collect these objects to allow for an individual stepping back in time to view previous warnings. The warning objects are

recreated from these files when necessary, parsed as if they were collected from the DuPage server, and active warnings are stored within the controller of the backend of the program. In this manner, the backend controller is also able to produce a list of all warnings active at a specific previous point in time or during a particular range of time.

The most important, and most referenced, information in this severe weather statement is the geometry of each warning. This is provided in a series of geological points that reference the corners of a polygon. This polygon is used to render a warning onto a 2-dimensional map in SlimDX, similar to the current television standard (see Figure 4). The exact color of the rendered warning is determined by the SlimDX program, referencing the type of warning that is being rendered. However, the geometry itself is pulled from the backend's controller.

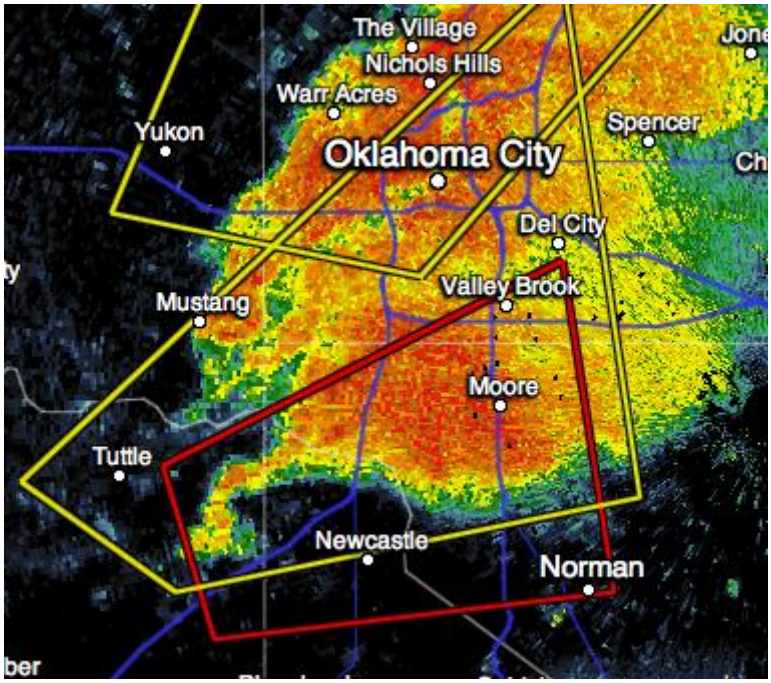

Figure 4. An example of a severe weather warning rendered onto a map.

The backend controller is also is responsible for determining the geometry of the actual warning box. This entails taking into account the stroke of the warning's border, as well as the overlap of each edge of the warning border.

In SlimDX, 2-dimensional radar information is rendered for ease of access. On top of that, the severe weather warnings are rendered after pulling the geometry of both the warning and the warning's borders.

Various other information is requested of the backend controller by SlimDX, including the general warning information for the user to read, the warning's duration, the warning type, and the warning's severity.

Should a severe weather statement be published to the College of DuPage's website without a warning, we are to process it as its own severe weather statement object; however, this should not happen in practice.

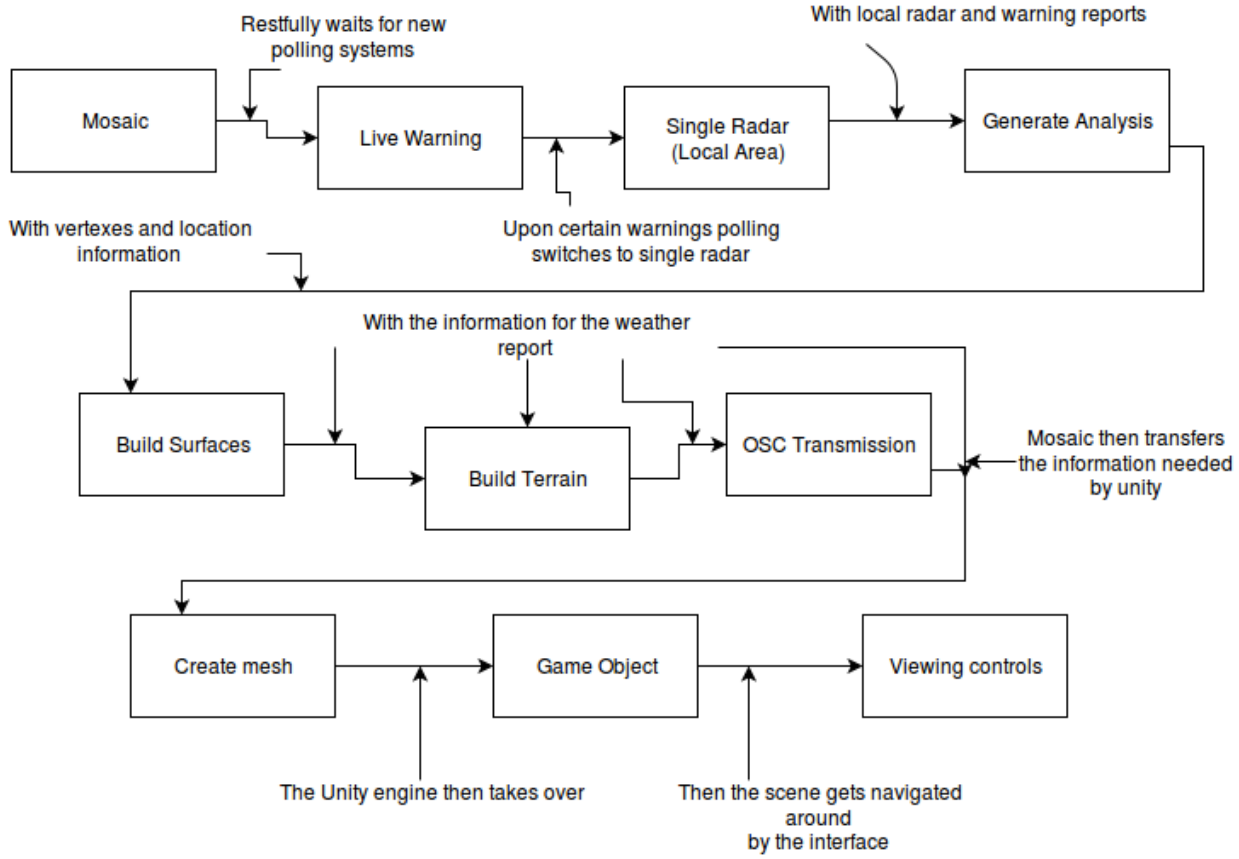

Figure 5. The full interaction of information requests and responses in the totality of the system.

VI. Prototype Summary

Frontend

Current Completed Tasks

The design of the interface has been finalized. The positioning of the text and buttons are designed for incredibly simplistic controls to make the use of the app easy. There are a total of five button inputs that the user can use: one for each direction as well as a button in the center. The buttons themselves take up the entire screen for so that users who have difficulties seeing the screen can still, intuitively, figure out how to use the application. Each button takes up a very large portion of the screen so that someone who cannot fully see the screen be able to guess where on the screen a button for a direction would be (moving forward is at the top of the screen, moving to the left makes up most of the left side of the screen, and etc.). However, the buttons have little functionality attached to them. The buttons can be clicked but nothing changes if they are clicked. The menu screen is now being rendered properly by the app so the user can properly see all of the buttons.

Current Functionality

All of the user inputs are displayed on the screen. The menu for the buttons appears on startup of the application, so as soon as the app is launched the controls are ready to be used. Each button contains a description displayed on them, indicating to the user what each is designed to do (the forward button says “Forward”, the left button says “Left”, etc.). All of the buttons are clickable, so user input is being registered by the app. An important note is that current use of the app is being done via the Unity editor, and instead of using a finger on a touchscreen, button pressing is being done by clicking with the mouse.

Remaining Tasks

Adding functionality to each button. The button on the left hand side must move the user along the negative x axis, the button on the right hand side must move the user along the positive x axis, the button on top must move the user along the positive y axis, the button on the bottom must move the user along the negative y axis, and the button in the center must move the user back to the origin point. Each time the user changes position the program must write out the position to a file. Once testing of functionality at this point is complete porting over the software to a tablet/touchscreen phone will begin. It is possible that minor UI designs will need to be changed to accommodate for different screen resolutions. However, it will be hard to predict exactly how the UI will need to be changed until the port is ready.

Estimated Time Remaining

The Frontend should be near completion by early May. Setting up functionality for the buttons should only take about a week. Writing out the file that contains the User’s current position should only take a few more days after. Testing will likely take another week to complete.

Finally porting the software over to the tablet/touchscreen phone and testing functionality will be seen through to expected deadline.

Backend

Current Completed Tasks

In order to fully understand the process required to parse for information within each warning, focus was first put on parsing information from a severe thunderstorm warning's syntax. Using the knowledge gained from that, all information relevant to the creation of any severe thunderstorm warning, flash flood warning, or tornado warning has successfully been parsed from its severe weather statement using various regular expressions in C#.

Using this information, the team is able to create a severe weather statement object, as well as, if a new warning is created, the warning that is associated with it. This information is also correctly recorded into its relevant file location-- either within the warning file, within the warning log, or within the unlinked statements folder. This information is then able to be parsed from those files again, enabling the team to recreate the warnings.

Current Functionality

The project currently contains a web crawler class, a polling class based on the links grabbed by the web crawler, and a reverse class that as per the client's request reverses the order of the statements. The polling class is called every five minutes, as per our client's request since the unity model is updated roughly at the same time, by a webClient downloadFile function [18] after the web crawler hands all the URLs as needed from the College DuPage weather system [16].

This file is then given to the controller, which breaks the files down into severe weather statements. Any that have been published since the previous check are then sent to the parser.

The parsing is completed by utilizing the language C# and the Regex class, most notably the Regex.Match and Regex.Split calls. The input string is broken up by double newlines, represented either by the characters '\r\n\r\n' or by the characters '\n\n.' The resulting array of strings, now broken by solid strings of words, are then parsed for certain patterns and passed into helper functions.

The second element is the first term to be parsed, specifically looking for key terms to identify what type of statement has been input. These key terms include "SEVERE THUNDERSTORM", "SEVERE WEATHER STATEMENT", and "FLASH FLOOD WARNING".

Also within this second element, regardless of warning type, the parser is able to identify the publication time of the statement. This can be found in the format '[TIME] [AM/PM] [TIMEZONE] [WEEKDAY] [MONTH] [DAY] [YEAR].' This means that the parser can pull the first word and convert it to a number. The second word can be tested for a match with either 'AM' or 'PM' and if it matches 'PM' then 1200 is added to the number found within the first

word. The third term is compared to a list of all abbreviations for timezones used by the United States in order to correct the time to a twenty-four hour Zulu time (UTC+0). The fifth term is compared to all month abbreviations using a switch-case, and the sixth and seventh terms are transformed to integers for the day and year.

The first element contains both the station who published the statement as well as the ID number of the warning associated, if any, and is passed to the respective helpers. Using the Regex '\\.([A-Z]{4})\\.' the station is discovered (in the pattern .XXXX. where X is a capital letter). Using the Regex '\\.([0-9]{4})\\.' the warning identification number is discovered (in the pattern .####. where # is a number).

Most warning types contain several bullet points, and one bullet point is dedicated to the expiration time for a new warning. This is in the specific format '* UNTIL [TIME] [AM/PM] [TIMEZONE]'. This acts much like the publication time, however there are exceptions for some warnings that use the terms 'MIDNIGHT' or 'NOON' to refer to 1200 AM or 1200 PM respectively. These are checked for, and processed accordingly. Using the final word in this bullet point, whether using the standard timestamp or using 'MIDNIGHT'/'NOON', the timezone will be determined and the final time shall be adjusted to be an accurate timestamp in Zulu time from any timezone used by the United States. The day, month, and year of the expiration time are pulled from the publication date, as the next instance of this time after its publication is when the warning shall expire.

If the warning being parsed as a tornado warning, the warning is tested for the keywords 'CONFIRMED', 'REPORTED', and 'EMERGENCY' in order to determine if the tornado warning is some different type.

By using the key term 'LAT...LON' the parser can find the list of warning vertices. What follows is a list of longs alternating between latitude and longitude. Taking the numbers in pairs, a list of PointF objects are created to represent the various warning vertices. By using the key term 'TIME...MOT...LOC' the parser can determine when to stop taking pairs of numbers, as this term marks the beginning of the next section.

Finally, using the key term 'TIME...MOT...LOC', the parser is able to find the motion of the storm system. This is in the format 'TIME...MOT...LOC [ZULU TIME] [DEGREE] [SPEED] [COORDINATE(S)].' Zulu time is specifically in the format '####Z' where '####' is a twenty-four hour time, so the term is collected and the 'Z' is removed. Degrees are given in the format '###DEG' where '###' is the degree of motion from North, so the term is collected and the 'DEG' is removed. Speed is given in the form '##KT' where '##' is the speed in knots, so the term is collected and the KT is removed. The final term, the coordinates, can list more than a single pair, so all following terms are collected and divided into pairs similarly to how the warning vertices are parsed.

Upon creation of a warning, a new file associated with the warning is created. Statements updating this warning are concatenated to the beginning of its respective file, separated from other statements with '=========='. This allows statements to be more easily parsed from this file. Warning start and end times are also added to a log file in the same location. Severe weather

statements without an associated warning are put into an unlinked collection within a folder at this location. Further details can be found in the backend design section on page 15.

Remaining Tasks

The severe weather statements have a difficult syntax to crack, as they can be passed in relation to any type of warning and must be identified by an associated weather station and event tracking number (ETN). Specifically, because it updates information of an existing warning, the warning object must be accessed or recreated and checked for warning type. From there, the statement must be checked for several different types of information updates, then the warning must be changed to reflect this new information.

An additional function required by the team's program is to adjust the vertices' locations to assist the client in rendering a visual outline of the various warning vectors. This would include three separate lines per pair of vertices: an inner outline, the colored stroke, and an outer outline. The only notable exception to this is a tornado emergency, where there will be five: an inner outline, one colored stroke, a middle outline, a second colored stroke, and an outer outline.

The client has requested that, should enough time remain after these requirements are completed, that the team approach the parsing and processing of marine warnings. As marine warnings are rare and largely outside of the client's research interest, this is only a want, not a requirement.

Estimated Time Remaining

Three days will then be needed in order to add support for stepping back to a previous point in time, complete with the warnings that were active at the time. One day will be necessary to figure out a formula in order to correctly create new vertices for warning outline.

For the polling system, the storage process for the new filing style will take a day, also the handling of information to the controller must be complete which is explained below.

Also fine tuning of the Controller class for the parsing and polling systems will take at most 2 days to fine tune and complete fully.

VII. Lessons Learned

Timeline

Frontend

18 March 2016	Finalize a proper layout of the UI.
25 March 2016	Finish implementing the logic of button controls.
29 March 2016	Verification of code correctness and implementation deadline.
01 April 2016	Finish implementing proper output from user input via the UI. Output is saved to a file to be read by the client.
08 April 2016	Verification of proper output from user input via the UI deadline.
13 April 2016	Cumulative testing of the entire frontend is complete, and desired functionality designated by the client is complete. From here additional features may be requested by the client and from this point onward the timeline may change for additional goals.
18 May 2016	Visit from The Weather Channel

Table 2. The projected timeline for the frontend at the beginning of the project.

Backend

11 March 2016	Complete severe weather statement parsing using regular expressions in C#, for Severe Thunderstorm Warning
Week of 14 March 2016	Meet with client to discuss progress and further details regarding visit from The Weather Channel.
18 March 2016	Complete implementation of all other warning types, including Tornado Warning and Flash Flood Warning
21 March 2016	Complete implementation of warning and severe weather statement objects.
25 March 2016	Complete implementation of helper functions used to hook into client's existing programs. Test all functionality to this point.
01 April 2016	Complete pull of individual severe weather statement from server, parsing

	into object. Additionally check updating pre-existing weather warnings.
08 April 2016	Complete collection of active warnings, and collection of all warnings that have been issued during the program's run time.
22 April 2016	Complete calculation of warning rendering, to account for border thickness, overlap, and outline.
29 April 2016	Allow user to step back through time and view previous warning states as they were during a given time.
06 May 2016	Implement other warning types such as marine warnings for the user to observe and track.
18 May 2016	Visit from The Weather Channel

Table 3. The projected timeline for the backend at the beginning of the project.

Backend Goalposts

Parsing Severe Weather Statement Files

The most important part for this portion of the project to succeed is the ability for the program to be able to read and understand the severe weather statements. In this manner, the program will first have to be able to correctly use regular expressions to find and extract this information correctly. Using the standards set forth by the National Weather Service for their severe weather statements found in [15], patterns must be discovered and utilized to correctly find this information.

Severe Weather Warning Objects

Once the above information has been correctly parsed, it must be stored in such a manner that the rendering program it is assisting is able to quickly reference any data it requires for a given weather warning. The program will take the parsed information and first identify whether the severe weather statement is referencing a new weather warning or if it is updating a previous one. Once that has been identified, either a new warning object will be created, or the old weather warning object will be found and updated.

Warning Object Communication Functionality

After the objects are created, they must be able to communicate with the renderer. In this case, a series of getters must be created in order to allow for the renderer to recover the correct information for a given warning.

Weather Statement Database Polling

In order for the program to gather the relevant weather statements, the program must be able to pull the severe weather statements whenever the National Weather Service publishes them. The

client has supplied the team with a database that collects these severe weather statements. Our program must be able to listen to this database and, upon a new severe weather statement being published, download and save the statement for parsing. At the beginning of execution, the program will also download and parse all previous weather statements so that all active weather warnings are recorded and accounted for.

Active Weather Warning Tracker

For the renderer's usage, the program must keep track of which warnings are currently active. For this, the program will be constantly updating each warning for its expiration time, and upon reaching this expiration time removing the warning from the collection of active warnings. This is independent of the full collection of weather warnings; for the user to be able to view previous selections of time, the entire collection of severe warning statements must be retained independently.

Weather Warning Rendering Geometry

The program has been requested by the client to contain a helper method using the geometry of the severe weather warning in order to correctly render the border for the user. In this case, three lines must be rendered for each edge (e.g. an outer black outline, an inner black outline, and the solid color the border shall appear as). An example of this can be seen in D2. This involves taking an input of the border's requested thickness and adjusting the known weather warnings geometry's coordinates in order to correctly render these three lines.

Frontend Goalposts

Layout

The layout for the tablet must be relatively straightforward for the user. It needs to be intuitive and relatively simple to use. When in use with a VR/AR display (e.g. goggles) the user's view of the tablet itself will be obscured, so the layout and size of the buttons on the UI need to reflect this restraint.

Logic

Unity's toolset makes setting up buttons to display on the tablet's screen much easier. Since each of buttons will perform similar functions (one-directional movement), most of the code for the buttons will utilize inheritance. Each button press will move the user in either X or Y directions. There will also be fifth button that returns the user to their original position.

Final Testing

After implementation of both the layout and the logical aspects associated with it, the frontend user interface must be tested extensively so that all portions act according to the team's intent as well as the user's expectations. Detailed further in the Testing section, this includes modular testing of input commands via an interactable surface, as well as general Unity debugging.

Tools and Resources

Frontend

The frontend team utilized the Unity game engine as per the client's request. With a Unity installation comes with its own IDE, to write various scripts with. Unity also comes with an extensive API that will be utilized heavily.

Backend

The Backend team utilized a private GitLab repository in order to collaborate on the coding process [13]. Per request of the client, and also to be compatible with code already existing within the client's project, the project utilized the language C#. Parsing for information in the severe weather statements will make heavy use of regular expressions, as that will allow us to specifically search through a string for a certain pattern of characters [14].

Testing

Frontend

Testing will be done through various iterations of the Frontend (completion of UI, testing through the Unity editor, porting the program to tablet, etc.) with each section getting a corresponding test.

UI testing was done very early on to see if someone with obscured vision could still use a touch screen interface. Testing was done on a notebook with drawings of, at the time, four button interface and having a third party try to use the drawing as a real UI. The outcome was favorable, and led to the creation of the fifth, middle button that brought the user back to the origin. Once the design of the interface was finalized, development in the Unity editor began.

Once the UI was created in the Unity editor, testing began within the editor. Testing consisted of running the application for the UI and then simply validating the functionality of each button. Each button was clicked on in the editor, and checked to make sure each button moved the user in the correct direction. Testing through the editor is different from an in environment test since each button is clicked by the mouse and not a finger pressing down on a screen. For this reason, testing was done on a touchscreen device later in development of the project.

The next phase of testing was to make sure that after every button press in the UI the current position of the user was saved to a file. The file simply stated where the user was in the simulated environment. Testing consisted of checking this file after several different button presses within the editor. After being satisfied with the results, testing moved forward to the final phase.

To make sure that UI made in the Unity editor would work on a touchscreen device as intended, the program was ported over to an iPhone 5S. The iPhone was chosen for sake of convenience and because the operating system is similar between most Apple phones and tablets. The tests in

this phase identical to the in-editor tests. Even though the port generated by the Unity editor to the touch screen device superficially seemed accurate, a full round of functionality testing was done. After pressing buttons on the screen and matching the results made within the editor, the final phase of testing was complete.

Backend

Within the Backend half of the project, there are several main portions that must be tested. The regular expressions written in C# are the most extensively tested. With an abundance of old severe weather statements available through various online resources, it is easy to collect a series of warnings in order to meet all of these criteria. For this reason, real severe weather statements were read into the parser as a series of tests.

The first thing to be tested was the identification of the different types of warnings: Severe Thunderstorm Warning, Flash Flood Warning, Tornado Warning, or Marine Warning. Tornado Warnings are further broken down into Tornado Warning, Tornado Reported, and Tornado Emergency. This is the most important aspect to test the parsing of, as being able to determine the type of warning has a direct impact on what parsing calls are made on the warning to decipher the rest of the information. An incorrect identification could result in the parser producing erroneous results by attempting to find information not available in a given warning.

The results led to the correct identification of all types of warnings. However, due to lack of implementation, Marine Warnings were fed into the same result as severe weather statements without an associated warning. This led to some interesting behavior when parsing severe weather statements based on that station and ETN in later testing, in which some additional checks had to be made to catch this case.

From there, the Event Tracking Number (ETN), reporting weather station, and time of statement delivery must be deciphered correctly. This must then be run against all other warnings in order to find if the ETN and weather station combination is unique. If it is unique, a new warning must be successfully created. If this combination already exists, then the existing warning must be updated.

This test was a simple check using File.Exist() in order to see if an existing warning has created a file by that name. After a simple syntax correction, tests were successful in identifying the existence of warnings existing with that ETN.

The rest of the information including expiration time, movement of the weather system, and warning geometry must be parsed and turned into PointF objects for use in the client's program. All of this must be correctly turned into a Warning object, and dumped into a data folder for later reference. The array keeping track of all active warnings at the current time must accurately be altered upon the addition of removal of a warning or the passage of time.

Warning objects were tested using their properties, and built in get and set functions. Several different severe weather statements creating warnings were parsed into the backend controller and the input values were compared to the values provided by the property's inbuilt get function.

Tests were successful. Testing warning objects after a severe weather statement updates them has not yet been tested.

The text dump files must also be read correctly. This is tested in tandem with the system in place for querying a point in time for active warnings. This list of warnings must accurately determine the state of the warning and weather system at the specific time queried, despite possible updates since that point in time.

The actual operation for gathering the weather statements must be tested. The portions that must be tested are polling the server to get the actual statements, gathering the updates to each of the weather statements as they occur, passing only the portions of the files that are required, and noticing when a file is complete and the weather phenomena will not be updated further.

To test the actual polling of the server, the only test created is simply verifying that the file that was intended to be polled from the server was within our log system. With a Diff checker (difference checker), it was verified that the copy that was polled was the same as the server that was polled.

The approach to check the length of the file has been changed due to the anomaly described below. The current approach simply polls the server and grabs all the files that are currently on the server should any files match our files in storage they are overwritten in the proper procedure (e.g. reversing the entries within the document as per clients request). The process takes more time but this time is miniscule since the database set is under a hundred entries at any one time.

The last case that was not able to be completed correctly was if our file length and the server file length is the same. It is assumed that the file had no changes but there maybe a case where the weather report is an error and the error is corrected but the file length is the exact same as before in this case our file is not correctly updated as of yet. To solve this issue the procedure was changed as described above.

A major issue the team encountered was the realization that the files we pulled from [16] was a collection chronologically of all statements delivered by the National Weather Service within a single hour. For this reason, functionality had to be adjusted in order to handle multiple warnings receiving updates from a single file. In addition, tests had to be held in order to assure that warnings' updates were correctly identified and read from this compilated file. The approach was changed to divide up the file into the statements before feeding it to the parser, allowing each statement to be checked for a warning ETN and warning type individually. Through testing, we determined that this has solved the issues related to how the College of DuPage sorts the severe weather statements.

VIII. Acknowledgements

We thank Trevor White (tsw90@vt.edu) of Virginia Tech for his help in understanding weather systems and their related warnings, as well as for allowing us to work on the polling portion of this project.

We also thank Zachary Deur (zachduer@vt.edu) of Virginia Tech for his help in utilizing Unity, and for allowing us to participate in the frontend portion of this project.

We thank Edward Fox (fox@vt.edu) of Virginia Tech for giving us the opportunity to work with our client and learn a new programming language along with interacting with databases and with Unity.

IX. References

[1] ICAT, "ICAT overview,". [Online]. Available: https://www.icat.vt.edu/content/icat-overview. Accessed: Feb. 20, 2016.

[2] ICAT, "Cube,". [Online]. Available: https://www.icat.vt.edu/content/cube-0. Accessed: Feb. 20, 2016.

[3] "DirectX," in *Wikipedia*, Wikimedia Foundation, 2016. [Online]. Available: https://en.wikipedia.org/wiki/DirectX. Accessed: Feb. 20, 2016.

[4] National Oceanic and Atmospheric Administration, "Doppler radar (online tornado FAQ),". [Online]. Available: http://www.spc.noaa.gov/faq/tornado/doppler.htm. Accessed: Feb. 20, 2016

[5] NOAA and N. W. Service, "About NOAA's national weather service," NOAA's National Weather Service. [Online]. Available: http://www.weather.gov/about. Accessed: Feb. 20, 2016.

[6] "OpenGL," in *Wikipedia*, Wikimedia Foundation, 2016. [Online]. Available: https://en.wikipedia.org/wiki/OpenGL. Accessed: Feb. 20, 2016.

[7] "SlimDX Homepage,". [Online]. Available: https://slimdx.org/. Accessed: Feb. 20, 2016.

[8] Unity, "Unity - game engine, tools and multiplatform,". [Online]. Available: https://unity3d.com/unity. Accessed: Feb. 20, 2016.

[9] D. Holloway, "IBM buys weather channel product and tech business," in *Media*, TheWrap, 2016. [Online]. Available: http://www.thewrap.com/ibm-finalizes-deal-for-weather-channel-product-and-tech-business/. Accessed: Feb. 20, 2016.

[10] The Weather Channel, "National and local weather forecast, hurricane, radar and report," The Weather Channel, 1995. [Online]. Available: https://weather.com/. Accessed: Feb. 20, 2016.

[11] "Augmented Reality," in *Wikipedia*, Wikimedia Foundation, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Augmented_reality. Accessed: Feb. 20, 2016.

[12] "Virtual Reality," in *Wikipedia*, Wikimedia Foundation, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Virtual_reality. Accessed: Feb. 20, 2016.

[13] "Code, Test, and Deploy Together with GitLab Open Source Git Repo Management Software," in *GitLab*. [Online]. Available: https://about.gitlab.com. Accessed: Mar. 2, 2016.

[14] "Regular expression," in *Wikipedia*, Wikimedia Foundation, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Regular_expression. Accessed: Mar. 2, 2016.

[15] Strager, Christopher S. *WFO SEVERE WEATHER PRODUCTS SPECIFICATION*. Silver Spring, MD: National Weather Service, 23 May 2014. PDF.

[16] Warning Server, College DuPage, 2016. Available: http://warnings.cod.edu/. Accessed: Mar. 3, 2016.

[17] "Microsoft Hololens," in *Microsoft*. [Online]. Available: https://www.microsoft.com/microsoft-hololens/en-us. Accessed: Mar. 3, 2016.

[18] "WebClient.DownloadFile Method", in Microsoft.[Online]. Available: https://msdn.microsoft.com/en-us/library/system.net.webclient.downloadfile%28v=vs.110%29.aspx