ABSTRACT

Title of thesis:	AN IMMERSIVE EXPERIENCE: VISUALIZING LARGE-SCALE CLIMATE DATA USING VIRTUAL REALITY AND INFRARED HAND-TRACKING TECHNOLOGY
	Theodore Corrales, Erin Estes, Kevin Ho, Austin Hom, Mughilan Muthupari, Justin Pan, Justin Shen
Thesis directed by:	Dr. Stephen Penny
	Department of Atmospheric and Oceanic Sciences

Recently, interest in understanding Earth's climate has risen in light of anthropogenic climate change. However, effective climate data visualization tools for studying climate remain largely outdated. We proposed the use of virtual reality to more effectively visualize climate data, implemented a prototype climate visualization tool using Unreal Engine, and conducted a focus group to gain expert insight and advice for evaluating and improving our visualization tool. Our regional view displayed the temperature of the Chesapeake Bay, surrounded by topographic data in one cohesive visualization, while our global view transformed and displayed climate data on a virtual globe using a perceptually-uniform color texture and incor-

porated an animated particle field to visualize vectorized data. Finally, we used the Leap Motion Controller to facilitate interaction with the visualizations through hand gestures. Overall, participants found that three-dimensional visualizations were more intuitive than two-dimensional visualizations and suggested areas for further improvement in the future.

AN IMMERSIVE EXPERIENCE: VISUALIZING LARGE-SCALE CLIMATE DATA USING VIRTUAL REALITY AND INFRARED HAND-TRACKING TECHNOLOGY

TEAM DIVA

THEODORE CORRALES, ERIN ESTES, KEVIN HO,
AUSTIN HOM, MUGHILAN MUTHUPARI,
JUSTIN PAN, JUSTIN SHEN

Dr. Stephen Penny

GEMSTONE
UNIVERSITY OF MARYLAND

COMMITTEE

Dr. Stephen Penny
Dr. Anna Borovikov
Dr. Timothy Canty
Dr. Tse-Chun Chen
Mr. Oscar Hendrick
Mr. Jeffrey Wayne Henrikson
Mr. Patrick Meyers
Dr. Mason Quick

Thesis submitted in partial fulfillment of the requirements of the Gemstone Honors Program, University of Maryland, 2019

ACKNOWLEDGEMENTS

Our project could not have been possible without the help of various individuals; we must take the time to thank them all. The first is Dr. Stephen Penny, our mentor, whose vision inspired our project and whose insights paved our way forward. He provided us with helpful suggestions week-to-week on immediate steps in our project. Next is our librarian, Dr. Kelley O'Neal — she had immeasurable impact over our papers, posters, and presentations. We would like to thank our discussants, whose voices have greatly shaped our project. We would also like to thank FedCentric and UMD Libraries for awards that provided resources with which we could develop. Thank you to everyone who took the time to attend our focus group; your inputs provided us with insight to the user base for our tool, allowing us to use your critiques as guidance during our iterative development process. Additionally, we would like to thank Ms. Jill Smith, our GEMS104 mentor, for her great advice on team dynamics and internal goals. Finally, we would like to thank the entire Gemstone staff, especially Dr. Frank Coale, Dr. Kristan Skendall, and Dr. Vickie Hill, for making this experience possible.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

2D	two-dimensional
3D	three-dimensional
AOSC	Atmospheric and Oceanic Sciences
API	application programming interface
AR	augmented reality
AVHRR	Advanced Very High Resolution Radiometer
AWS	Amazon Web Services
CAD	computer assisted design
CFSR	Climate Forecast System Reanalysis
CIE	International Commission on Illumination
D.C.	District of Columbia
GCCV	Global Climate Change Viewer
GPU	graphics processing unit
GRIB	GRIdded Binary
GUI	graphical user interface
HDF	Hierarchical Data Format
HI	haptic icon
IPCC	Intergovernmental Panel on Climate Change
IRB	Institutional Review Board
L3DT	Large Scale 3D Terrain Generator
NASA	National Aeronautics and Space Administration
NCAR	National Center for Atmospheric Research
NCEP	National Centers for Environmental Prediction
NetCDF	Network Common Data Form
NOAA	National Oceanic and Atmospheric Administration

PC	personal computer
UMD	University of Maryland
U.S.	United States
RCCV	Regional Climate Change Viewer
RGB	red, green, and blue
sRGB	standard RGB
VAPOR	Visualization and Analysis Platform for Ocean, Atmosphere, and Solar Researchers
VR	virtual reality

Part I

BACKGROUND

1

INTRODUCTION

A recent report from the Intergovernmental Panel on Climate Change (IPCC) has sounded the alarm that climate change is occurring faster than anyone, including top scientists, ever expected (Allen et al., 2018). Previous scientific estimates of the amount of warming that can be tolerated are far more conservative than once thought. Consequently, improving the general public's understanding of climate change and its underlying factors is of utmost importance. Only with this understanding can broad support be garnered for major preventative actions.

Scientists need improved tools to process and understand large volumes of satellite and computer modeled data. Since the capacity to collect and store climate data has improved dramatically over the last three decades, new problems have emerged that climate scientists must tackle. How should this large quantity of data be stored? How should it be processed? How can such large volumes be meaningfully displayed and analyzed so that scientists can gain new insights, understanding, and knowledge from the data? This last question is the focus of our research, as visualization software products for climate and weather data have not kept up with the rapidly advancing technology or growing data volumes.

Current data visualization techniques mostly fall into two broad categories: two-dimensional (2D) maps, which are most common, and three-dimensional (3D) globes, which can be ei-

ther physical or virtual. Two-dimensional maps have inherent limitations associated with dimensionality reduction (e. g. visualizing 3D spatial data in 2D). Such maps have difficulty with displaying high dimensional data (e. g. combined temperature, humidity, and velocity across a broad spatial area at multiple height levels). Virtual globes also suffer from the projection of 3D spatial data onto 2D, since there is no true sense of depth on traditional computer screens. Three-dimensional physical globe models require additional equipment that require large costs for materials, equipment, and installation, which make them relatively inaccessible to most research scientists. Many current visualization tools are specialized, and do not adapt to different datasets which may be better viewed in varying contexts (e.g. high resolution local data, global data, and animated data).

We propose a solution to this problem: interactive virtual reality (VR) weather and climate data visualization. By leveraging recent advances in VR technology, we develop software that is both interactive and immersive, allowing climate scientists to view and work with data in ways that were previously not possible. By utilizing cutting-edge hardware, we enable interactive hand gestures to replace traditional mouse and keyboard interactivity in an effort to provide a more streamlined and intuitive interface.

We have two primary research questions that lead to two main areas of focus for development:

1. *Technical performance:* How can we best design a variety of robust data pipelines to preprocess and visualize climate data in an interactive virtual environment?

2. *Intuitive user interaction:* What are the most intuitive methods to allow a user to interact with climate data in a virtual environment?

Throughout the iterative development of our prototype, these questions serve as a guide for our design choices as we aim to balance capability with ease of use. The specifics of our implementation are outlined next.

To address these questions, we appoint three technical development subteams. The three teams focus on (1) developing a data preprocessing pipeline, (2) creating user interfaces, and (3) building visualizations for high-resolution data of specific areas of interest. Membership in the subteams adapts as needed. The VR prototype combines development efforts from all three subteams: users may interact with both a floating 3D model globe and a local view in virtual reality, allowing the user to interface with the visualizations via hand gestures. The data displays perceptually-uniform color scheme representations of climate data projected onto either the globe or local region. Moving particles, which represent wind vector fields, are implemented as an overlay to provide an additional dimension of data. Focus groups with surveys are conducted to gauge professional and community interest and gain valuable feedback for further development.

The results of our prototype and focus group provide a proof-of-concept that VR is a viable method for visualizing 3D climate data on both global and local scales. Additionally, we suggest a novel framework for user interaction within a VR environment. However, before the tool can be fully deployed at a production scale, more development is needed. The tool should be able to visualize multiple types of data formats. Additionally, alternative methods should be investigated to ensure the system is scaleable to large datasets. Finally, the user interface is restricted to a few built-in gestures; future work should develop additional modes of interaction.

The rest of the thesis is organized as follows: In Chapter 2 we provide a literature review related to the current state of climate visualization research and technologies. In Chapter 3 we discuss our methodology, including how we process, visualize, and interact with the data, as well as our approach for conducting the focus group. In Chapter 4 we report the overall results of the project. Finally, we conclude with a discussion of the results in Chapter 5.

2

LITERATURE REVIEW

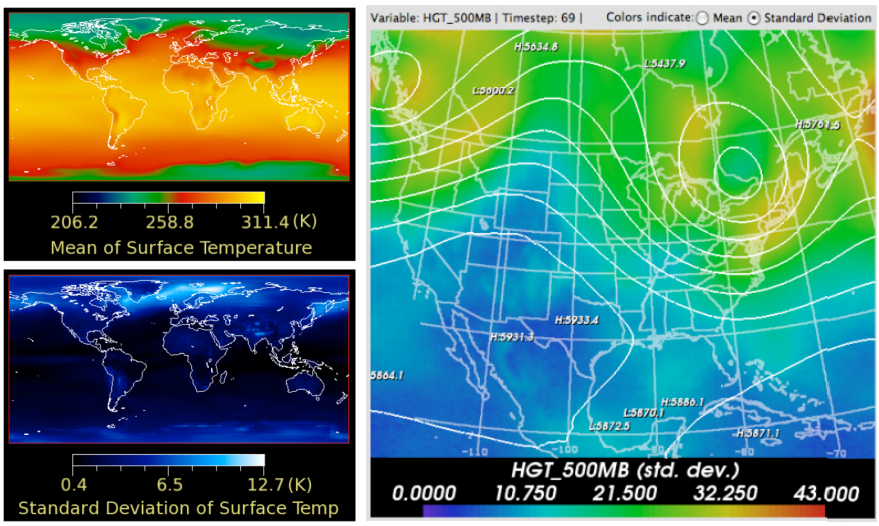
2.1 CURRENT CLIMATE DATA VISUALIZATION METHODS

We can categorize current methods of visualizing large-scale climate data into either 2D or 3D renderings. A 2D rendering displays data on a flat plane and uses colors or glyphs to depict values across each latitude and longitude. A 3D rendering displays data on a 3D globe which can be rotated to see all data points. Examples of 3D renderings include Google Maps and World Wind (Liu et al., 2015; Zhang et al., 2016).

There are strengths and weaknesses to each approach. A 2D rendering requires much less processing power than a 3D rendering. However, a 2D rendering (Alder et al., 2013; Potter et al., 2009; Teuling et al., 2011; Wickham et al., 2012) can visualize only a small number of parameters, while a 3D rendering (Liu et al., 2015; Zhang et al., 2016) can visualize volumetric data and promotes easier analysis of certain trends. To our knowledge, neither type of visualization has been tested to determine which is more effective for conveying climate information. This is an area of concern, since general studies of data visualizations indicate a correlation between aesthetic appeal and comprehensibility (Filonik and Baur, 2009). In the remainder of this chapter, we introduce various existing methods currently used to visualize climate data.

2.1.1 *Two-Dimensional Renderings*

Examples of 2D renderings include color maps and glyph maps. These use colors and symbols, respectively, to demonstrate variations in one or more variables over a large geographic area. Color maps typically vary from blue to red to indicate low to high levels of a parameter, respectively. Potter et al. (2009) first display the mean and standard deviation of surface temperature on individual color maps (Figure 1a). They combine these into one display, using color to indicate mean and contour lines to indicate standard deviation (Figure 1b).



(a) (b)

Figure 1: These maps allow users to display mean and standard deviation of surface temperature (a) individually by using independent color maps and (b) together by using a color map with contour lines (Potter et al., 2009)

However, this approach displays a single variable at a time, so users can only gather a limited amount of information from the maps. Other color maps have been developed that can display multiple variables through a two-color gradient. Research-

ers from the Institute for Atmospheric and Climate Science in Switzerland developed one such bivariate color map to display temperature and relative humidity (Figure 2) (Teuling et al., 2011).

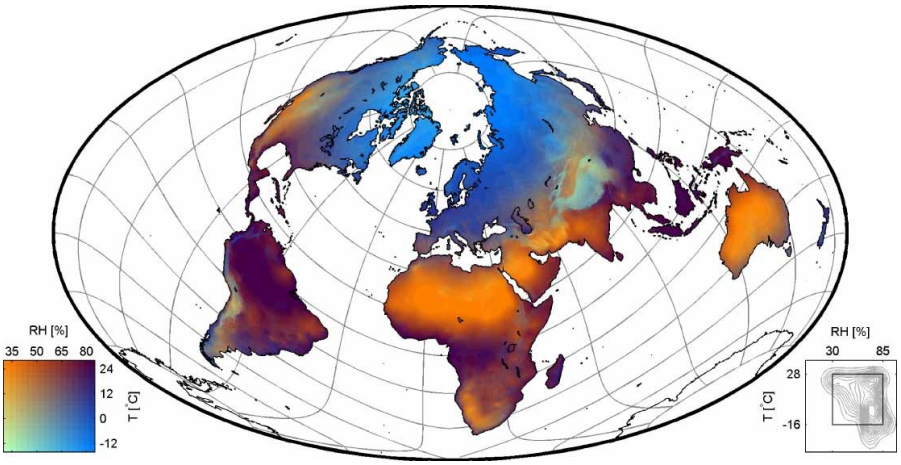
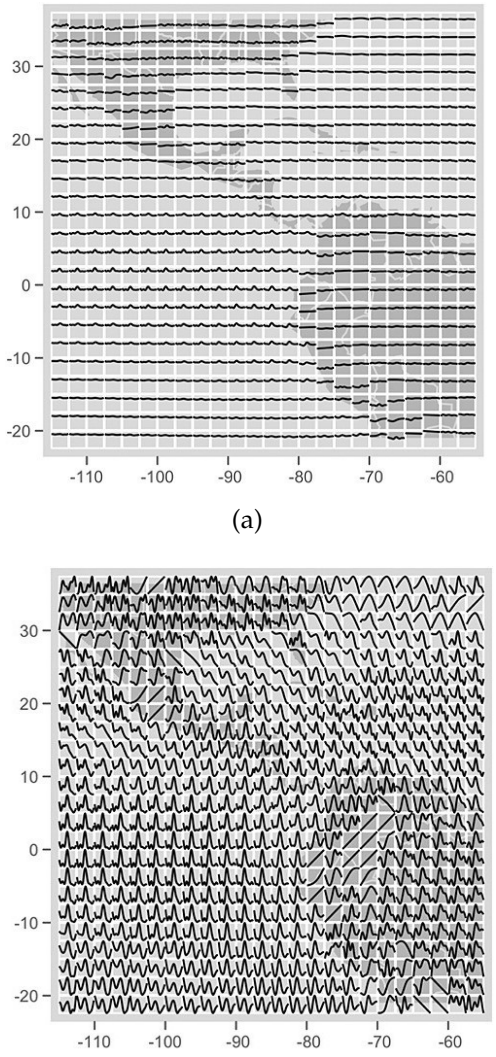


Figure 2: A bivariate color map indicates temperature and relative humidity by using a two-color gradient. The results are meant to be intuitive: the Sahara desert has a sandy color that indicates hot and dry, the Arctic has a deep blue that indicates cold with high relative humidity, the tropical regions are clearly distinguishable, etc. (Teuling et al., 2011)

Few existing 2D maps in the literature display more than two parameters at a time, and often require unintuitive approaches to do so (Potter et al., 2009; Teuling et al., 2011). Furthermore, very few strategies have been developed for 2D maps that can demonstrate changes in a parameter over time. One approach to displaying changes in parameters over time is the glyph map. This type of map was developed by Wickham et al. (2012) to display changes in a single variable over space and time by placing tiny graphs all across a world map, with each graph using the same scale (Figure 3). While glyph-maps allow users to view changes in a parameter over time and help highlight aberra-

tions in the data, they are less intuitive and can be difficult to decipher, especially for those unfamiliar with them.



(a)

(b)

Figure 3: Two glyph-maps representing the same temperature dataset for one year. Map (a) utilizes a global temperature scale, map (b) utilizes a local temperature scale (Wickham et al., 2012)

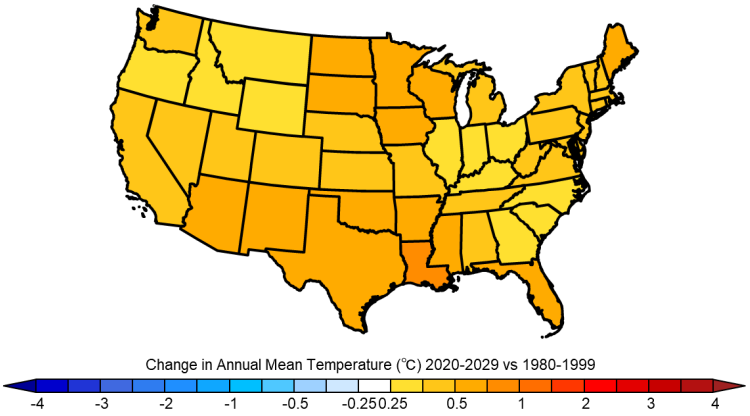
.

Alternative examples of 2D maps that account for time are the Global Climate Change Viewer (GCCV) and the Regional Climate Change Viewer (RCCV). Both of these visualization tools were developed by Alder et al. (2013) from the United

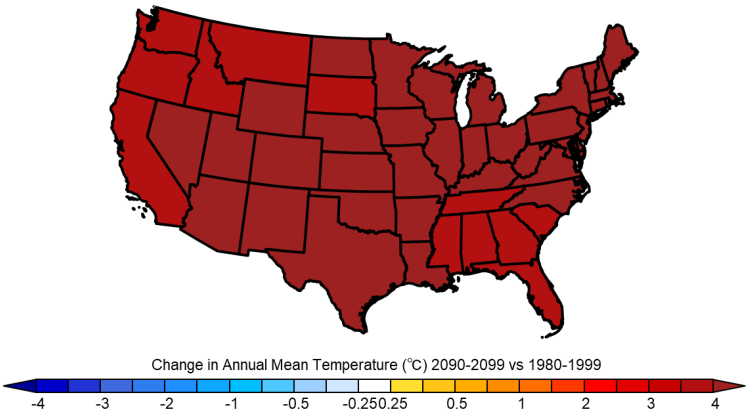
States (U.S.) Geographic Survey and the Lawrence Livermore National Laboratory in 2012. They allow users to view predicted changes in parameters such as temperature, soil moisture, and precipitation. These changes are displayed on color maps, with reds indicating significant increases and blues indicating significant decreases (Alder et al., 2013).

Both the GCCV and RCCV tools allow users to compare values over one or more decades. For instance, users can compare the difference in predicted mean temperatures from the 1980s to the 2020s (Figure 4a) against the difference in predicted mean temperatures from the 1980s to the 2090s (Figure 4b). The RCCV also allows users to view predicted changes for specific regions in the U.S. down to the county level. However, as can be seen in Figure 4, trying to analyze complicated differences in means can quickly become confusing to the reader with this kind of 2D visualization.

All 2D data visualization maps discussed have strengths and weaknesses. It is valuable for users to see the changes in the data over time, as with the GCCV and RCCV, but it may be difficult for users to grasp climate patterns when only one parameter is displayed in only two dimensions. Conversely, bivariate color graphs are useful to see aridity, relative humidity, and temperature, but make it difficult to grasp the changes when no time context is given.



(a)



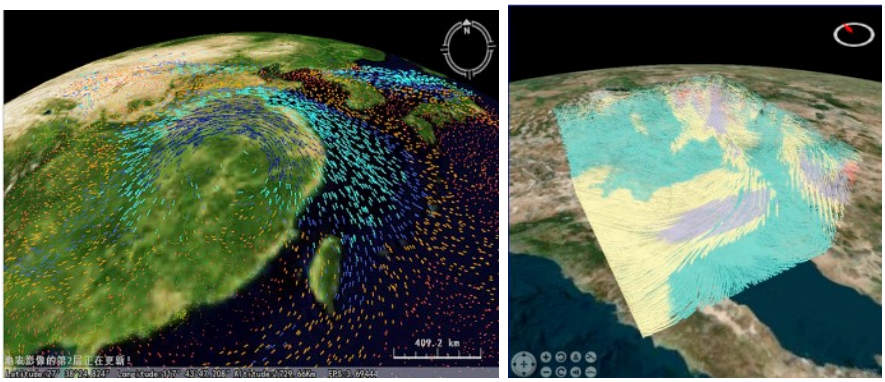
(b)

Figure 4: The RCCV (Alder et al., 2013) visualizes predicted temperature changes over two time spans: (a) A comparison between the 2020s (predicted) and the 1980s. (b) A comparison between the 2090s (predicted) and the 1980s.

.

2.1.2 *Three-Dimensional Renderings*

Increases in computational capabilities have made it possible to visualize large sets of climate data by using online simulations and virtual globes, such as Google Earth and World Wind, which is an open-source application programming interface (API) created by National Aeronautics and Space Administration (NASA). These interfaces support using vectors to show the paths of particles, and volume rendering to add textures and colors to various sections of 3D-space. Furthermore, these renderings may change in real time, allowing users to observe the fluctuations of climate phenomena interactively. For instance, Liu et al. (2015) used vector simulation to determine wind speeds and directions in areas affected by a cyclone (Figure 5a). Similar applications have been developed by Zhang et al. (2016) to visualize wind currents and dust storms in real-time (Figure 5b). Their findings suggest that World Wind may be helpful for conceptualizing datasets.



(a) (b)

Figure 5: (a) Vector simulation of a cyclone in World Wind (Liu et al., 2015). (b) Wind currents simulated in World Wind using volume rendering (Zhang et al., 2016). Wind currents depend not only on latitude and longitude, but also on elevation.

These three-dimensional APIs can also be used to render data not related to wind patterns or vectors. For instance, Du et al. (2015) developed an API to display CO2 fluxes over the oceans (Figure 6). Flux levels were displayed using colors, ranging from blue (downward flux), to red (upward flux), with elevation indicating the carbon content of a region of the ocean.

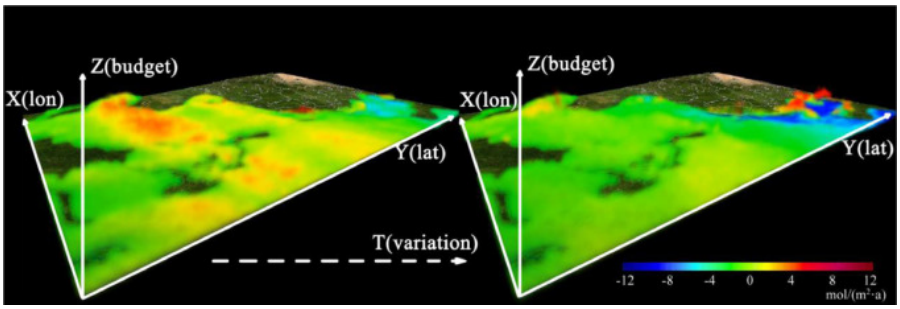


Figure 6: Simulation of CO_2 flux levels of the ocean, with positive values indicating an upwards flux (CO_2 leaving ocean) and negative values indicating a downwards flux (CO_2 entering ocean) (Du et al., 2015).

In the visualization of Figure 6, the color scheme allows the user to see increased acidification (negative flux) over much of the ocean surface surveyed, which also increases in severity over time (Du et al., 2015). Combined with the height field attribute and the World Wind API, this visualization can display carbon levels and fluxes across the entire globe over time. This provides a viable, perhaps more intuitive, alternative to 2D charts, which are rarely configured to display changes over time and typically require contour lines to display multiple variables (Potter et al., 2009).

Real-time globe environments, such as World Wind, illustrate complex time series more realistically than do 2D maps. However, like 2D maps, these globes have only been used to visu-

alize the effects of a single variable on the system, rather than multivariate interactions. This means that users can draw only incomplete conclusions from these visualization methods. For instance, while the methodology of Du et al. (2015) and his colleagues can display CO_2 flux, it is unclear what the effects of this would be on atmospheric or oceanic conditions. Likewise, while the methodology of Liu et al. (2015) effectively displays the path of a cyclone, it is difficult to gather details of the storm, such as the cyclone's effects on the upper ocean or changes in the cyclone's intensity, from their method of visualization. This presents a gap in literature, which could possibly be resolved by introducing a visualization method that displays multiple variables and their interactions with their surroundings. Such a method could also have an interactive component allowing users to focus on areas of interest, where different variables may appear to have a correlation, and this could be included as an analytical feature in its interface.

There is also a significant gap in literature in that no method of data visualization has been tested for its effectiveness in conveying the effects of climate data to technical and non-technical audiences, which means that improvements to these visualization methods are merely based on speculation and a general sense of inadequacy. It would be useful to test this new interactive method, as well as the older methods, to determine which visualization method is the most effective for communicating these climate data to interested audiences.

2.1.3 *Storing and Processing Data*

When designing a data visualization platform to reveal trends and interactions between datasets, it is also important to consider how the sets will be stored and processed. Some issues that arise from processing large datasets include interpreting multiple file types, processing large quantities of data, and rendering the corresponding graphics on a user's display (Zhang et al., 2016). Researchers have addressed these problems by developing improved database management techniques, compressing data when possible, and outsourcing computations (Zhang et al., 2016).

Climate data are recorded all over the world from a large variety of sources and in heterogeneous formats. Idreos et al. (2015) highlight the need for systems built for data exploration, where users may not be familiar with the details of how a certain dataset is stored, but wish to query the system for data in an exploratory manner. This can be accomplished with middleware, a layer between the user interface and the database that improves the efficiency of searching for interactions between and within datasets. According to Idreos et al. (2015), predictive analytics that search for interesting correlations, and data caching that store commonly retrieved data, can streamline this process. An exploratory system such as this could therefore help users visualize and examine data from multiple unfamiliar sources more easily.

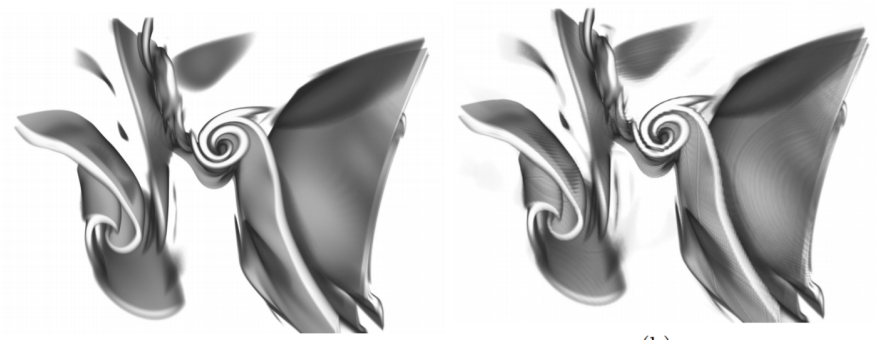
Users may also wish to have a more comprehensive visualization by compiling many datasets from separate sources. How-

ever, these could be stored in different formats, making them difficult to compare. To address this issue, Sun et al. (2012) developed a PHP program called KML Generator to extract data fields from database sources so that a single file could be used to generate the final visualization, prior to rendering. Employing similar strategies to compile requested data sources into a single format would save time when accessing data during the visualization rendering process.

Another component of animated data visualizations is the efficiency with which frames are generated. Data scheduling tasks must be established to ensure that visualizations can be generated in time for the user to view them. A technique developed by Du et al. (2015) allows for external data to be read asynchronously, so that entire datasets do not need to be loaded at once. This technique employs a node-based strategy where frames are simultaneously generated and displayed so that the next frame is prepared as the current one plays. In this method, only two graphics processing unit (GPU) buffers are in use at any given time. Therefore, the loading of data does not interfere with the process of rendering images (Du et al., 2015). When testing this model, Du et al. (2015) concluded that the resulting frame rate is not significantly affected by dataset size, demonstrating that this is an efficient method for generating and displaying animated visualizations in real time.

No matter how efficient the rendering process may be, generating images from data still takes time and requires significant computational power. Two approaches to reduce computation times are to simplify the data or to outsource process-

ing power. One method for simplifying data was developed by researchers from the National Center for Atmospheric Research (NCAR) during the development of their Visualization and Analysis Platform for Ocean, Atmosphere, and Solar Researchers (VAPOR) (Norton and Clyne, 2012). In order to render large data on normal desktop computers, VAPOR utilizes progressive data access, which sacrifices accuracy to speed up computations (Norton and Clyne, 2012). They found that for some datasets, especially those used for volume rendering, the progressive data access approximation is adequate for visualization purposes, as evidenced by the negligible difference between the original data in Figure 7a and the approximation in Figure 7b.



(a) (b)

Figure 7: Close-up of VAPOR's volume rendering of a region reveals little difference between (a) the original, uncompressed data and (b) the compressed data (Norton and Clyne, 2012).

In cases where data fidelity is very important, rather than relying on users to have sufficient resources, it is beneficial to outsource major computations to the cloud and simply serve clients the final product. Cloud platforms such as Amazon Web Services (AWS) can be used to gain access to remote GPU clus-

ters that share resources to quickly accomplish tasks. This system, developed by Zhang et al. (2016), allows for customizability of visualization algorithms and takes advantage of AWS's scalability features so that the platform can include more remote GPUs if more power is needed. Employing a system like this would allow climate data visualization platforms to operate independently of client resources so that rendering images can be done in minimal time Zhang et al. (2016).

2.2 VIRTUAL REALITY

Previously discussed methods of climate data visualization suffer from a lack of interactivity and immersion. These methods can display variation in few variables at a time, but it is difficult to gauge the combined effects on the surrounding environment, the oceans, and the atmosphere due to changes in the variables. Current methods also make it difficult to visualize the interaction between various climate variables, limiting the predictions they can make and the trends they can observe. A solution to this lack of interactivity is to use VR. This would allow users to visualize atmospheric and oceanic effects, and hone in and analyze specific areas of interest for trends and correlations in a multivariate visualization. Literature suggests that VR already has analogous technical applications in the military, medicine, and engineering, so it is plausible that the benefits of VR could be applied to climate data visualization as well (Bellini et al., 2016; Desai et al., 2014; Mathur, 2015; Winoto et al., 2016).

The concept of VR has existed since the mid-1990's. Over the past three decades, the available technology has grown im-

mensely, yet many challenges still remain for prospective developers. Even today there are no standardized tools to use or procedures to follow when working with VR (Ray, 2015). Beginning in the 1990's, developers attempted to create virtual environments with various VR toolkits (Ray, 2015). Unfortunately, researchers had numerous issues working with these toolkits. Existing toolkits are rarely reused when developing virtual environments, as they are often device- and use-specific. As a result, developers will often create their own toolkit from scratch (Ray, 2015). However, time is an issue when developing a new toolkit, because creation often takes years, as seen by the large gap between the creation of CAVELib, an early VR toolkit, and those from the toolkit wave of the early 2000's (Ray, 2015).

Ray (2015) blames the lack of publications about VR studies as the root cause of having no standard toolkit or procedure. He suggests that researchers publish their findings, so that a standard can be developed over time. Ray also provides guidelines to create such a standard. A toolkit should work by default, be unintrusive, and be easy to use. Working by default entails having interoperability between different pieces of hardware. In addition, the architecture should consist of a modular design, looking to augment instead of replace existing work.

2.2.1 *Modern-Day Applications*

Virtual reality technology has a variety of applications ranging from home entertainment to medicine and engineering. For instance, in the video game industry, consumers are eager to have a more immersive experience that puts them right in the

center of all the action (Bellini et al., 2016). Additionally, the United States military currently utilizes advanced simulation to provide soldiers with combat and flight training (Bellini et al., 2016). VR is also commonly used by engineers for computer assisted design (CAD) and has become a popular method for prototyping new designs, and viewing them from different perspectives (Bellini et al., 2016). In the medical field, VR medical training substituted traditional surgical training (Mathur, 2015).

Just as VR technology presents new possibilities for engineers, doctors, and the military to visualize their designs better, it also presents new opportunities for scientific visualizations. Our research team aims to develop new data visualization methods that utilize VR's immersive capabilities. These methods will allow scientists to analyze the data and present their results and findings.

2.2.2 *Current State and Limitations of Virtual Reality*

In order to better understand the potential of visualizing climate data with VR, it is important to evaluate the device that we use for its strengths and weaknesses. Currently, the most popular and widely used VR device family is the Oculus Rift by Oculus. We use the Oculus Rift DK2 model to develop our data visualization environment. It has a display resolution of 960 pixels by 1080 pixels, 100°field of view, and a refresh rate of up to 75 Hz, making the Oculus Rift a potentially immersive experience for the users (Desai et al., 2014). The Oculus Rift has a gyroscope to measure angular velocity, an accelerometer

to measure acceleration, and a magnetometer which transfers data at 1000 Hz to measure direction, making the headset responsive to head movements and improving the overall user experience (Desai et al., 2014). Other capabilities include head and positional tracking, which can be used as controls for the visualization tool (Oculus, cited 2019). This emphasis on immersion is an important consideration in the selection of Oculus as the team's visualization platform.

Despite the Oculus Rift's many strengths, it shares many of the same drawbacks with other VR devices. One weakness is the screen door effect, which results in empty black spaces between pixels on the screen (LaValle et al., 2014). This effect can distract users from the life-like visuals. Another limitation of Oculus Rift is ghosting, the trailing image left behind when a moving object moves quicker than the pixels can refresh (Desai et al., 2014). This ghosting produces a blurring effect and decreases the apparent resolution of the images. This means that objects on the screen have a maximum speed at which they can move without producing the ghosting effect. Another disadvantage of the Oculus Rift is that some subjects in VR studies have reported motion sickness. This was so pertinent of an issue that a question on motion sickness appeared on a survey to participants involved in a study using VR to help autistic children learn words (Winoto et al., 2016).

2.2.3 *Future of Virtual Reality*

A Goldman Sachs research report predicts that the development and growth of VR technology will be comparable to the

growth of personal computers (PCs), smartphones, and tablets (Bellini et al., 2016). Following the trend that PCs have taken over the past three decades, VR will benefit from the economy of scale, driving the prices of VR products down. Bellini et al. (2016) also describe the VR platform as a new potential computing platform that offers a new level of interaction with technology, just as the tablet introduced the concept of touchscreen interaction. In that sense, VR can be seen as an extension to existing computing technologies. The huge potential that VR technology presents has not gone unnoticed by tech companies as over $3.5 billion of investment have been poured into VR and augmented reality (AR) technologies in just the past two years (Bellini et al., 2016). Given the massive potential for growth in VR, the team believes that VR is the best platform on which to develop its new and modern climate data visualization tool.

2.3 HUMAN FACTORS

Emotional appeal and user perceptions of models and visualizations are becoming increasingly important, especially for communicating data to technical audiences outside of the primary field of study and to the general public. Studies on emotional aspects of data visualization suggest that human perception plays an important role in how audiences understand data (Grinstein and Levkowitz, 2013).

Datasets tend to be influential when presented emotionally, as they make the data more relatable to users (Herring et al., 2017). Two important factors that appeal to emotions are spatial and temporal proximity. This refers to how close the data are to

the user in space and time, respectively. For example, a user living in the District of Columbia (D.C.) would have high spatial proximity to a dataset collected in D.C., but if the dataset were from 1856, they would have low temporal proximity. Studies have found that data representing the near past or near future are far more impactful to the user than data that are too far into the future or too far into the past. Also, people find data visualizations that are changing with time more interesting than multiple snapshots. This is known as temporal fluidity, which can help enhance temporal proximity (Kostelnick, 2016). Through the use of VR, the users' spatial and temporal proximity could be heightened much further by bringing them closer to the time and location of the data presented.

Another technique that helps to appeal to the emotions of the user is making the data visualization method more user-friendly and interactive. One simple technique is the manipulation of color. Color creates visual stimuli that physiologically, aesthetically, and culturally arouse the user's emotion (Elliot and Maier, 2014). Colors have been proven to enhance both user engagement and excitement when used in data models. Since data are so content specific, colors become far more important (Elliot and Maier, 2014). For example, a user may want to use colors to add emphasis on a specific aspect of the data to evoke an emotional or physical reaction.

Visualization techniques such as dense pixel displays and iconic displays improve visual designs for climate data. Dense pixel displays use single pixels to represent each data value with color, which allows the user to see mass data (Keim, 2002).

This allows the users to see detailed information on local correlations, dependencies, and hot spots and compare data trends (Keim, 2002). Iconic displays allow the user to see data more clearly and can vary depending on the data being shown. Oftentimes, combining aesthetically pleasing visuals with other techniques can further enhance the user experience. An example of a technique that can be combined with visual aspects is the use of haptic icons (HIs). These are brief signals conveying an object's state, function, or content which are combined with haptic feedback, which the user receives through touch. This allows the user to use hand gestures to interact with a system (MacLean and Enriquez, 2003). Utilizing haptic techniques in conjunction with aesthetically pleasing visual design may improve emotional appeal and understanding of the data more than using any one of the techniques by itself.

2.4 CONCLUSION

While current methods of climate data visualization and analysis are workable, they struggle to visualize multiple variables, as well as with allowing interactivity with the datasets (Alder et al., 2013; Liu et al., 2015; Potter et al., 2009; Wickham et al., 2012; Zhang et al., 2016). This prevents researchers and other interested individuals from grasping the full effects of variations in the data, since these interactions are what allow researchers to understand climate phenomena in the first place. For instance, global temperature distributions are connected to wind patterns in the atmosphere and the surface temperature of the ocean, which are in turn connected to global precipita-

tion patterns, but current methods do not allow users to see this sort of relationship (Alder et al., 2013; Liu et al., 2015; Potter et al., 2009; Wickham et al., 2012; Zhang et al., 2016). Current 2D maps are almost completely restricted to univariate or bivariate data visualization, while 3D globe interfaces have not yet been implemented to allow users this sort of control or interactivity (Teuling et al., 2011; Zhang et al., 2016).

A potential means of improving this lack of interactivity between the user and the data displayed is to integrate VR technology with these methods. VR has already been used in medical, military, engineering and educational applications in order to give users a better understanding of important tasks (Bellini et al., 2016; Desai et al., 2014; Mathur, 2015; Winoto et al., 2016). In the area of climate data visualization, VR could allow users to visualize multiple datasets simultaneously, focus on trends of interest, and depict the effects of changing variables on the oceans and atmosphere. Furthermore, psychological studies suggest that spatial proximity and suitable color schemes help users better perceive the significance of data; VR would incorporate both of those features. Together, these features would create an experience in which climate researchers and other interested individuals can view how large sets of climate data interact with each other to produce many of the oceanic and atmospheric patterns observed today. In the following chapters, we describe the details of our development and evaluation of such a visualization tool.

Part II

OUR RESEARCH

3

METHODS

3.1 HARDWARE, SOFTWARE, AND ARCHITECTURE

3.1.1 *Oculus Rift Virtual Reality Headset*

Visualizing climate data within a VR framework allows for enhanced interactivity between the user and the data. This provides users with the control to choose which parts of datasets to focus on in order to observe important relationships. Moreover, VR allows users to be fully immersed in an environment representing their data. Being fully immersed in a 3D virtual environment provides a user with true depth perception, allowing for a more intuitive understanding of 3D data. Data visualizations in VR have the unique capability to be more interactive because gestural and other novel interfaces are possible.

We use the Oculus Rift as the hardware platform to develop our climate data visualization tool. The Oculus Rift has an accelerometer, gyroscope, and magnetometer, from which head orientation (yaw, pitch, roll) can be inferred. In addition, the Oculus Rift comes with an infrared camera, which tracks the position of an array of infrared micro-LEDs on the headset, allowing developers to track head position (x, y, z) of the user (Desai et al., 2014). The Oculus Rift provides handheld Oculus Touch Controllers to interface with the virtual environment. However, we have instead opted to work with a gesture based control system called the Leap Motion Controller. This provides

greater opportunity to fully immerse users in the virtual environment, using hand gestures to control the visualization directly. The Leap Motion Controller is described in more detail in the following section.

3.1.2 *Leap Motion Controller*

The Leap Motion Controller is a combination hardware and software package that tracks multiple joints in the user's hands. The physical device consists of two cameras and three infrared LEDs, and is attached at the front of the Oculus headset as shown in Figures 8 and 9.

Figure 8: Front view of Leap Motion Controller attached to the Oculus Rift (Hunt, 2016).

Cameras with wide-angle lenses record frames of grayscale images at a high field-of-view at the near-infrared spectrum. Based on a live feed of data from these cameras, the software component of the controller, called the Leap Motion Service, attempts to reconstruct a 3D representation of the scene. Following this reconstruction, the service extracts tracking information in order to pinpoint the location of key joints in the user's hands. It then projects that positioning information into the virtual environment to generate virtual hands that follow

Figure 9: Example of a user operating the Oculus Rift with the Leap Motion Controller attached in front.

the user's hands. Statistical filtering algorithms attempt to predict and maintain real time location of the hand based on movements of these joint locations, creating a fluid real-time connection between the user's real hand and the virtual hand in the virtual environment.

3.1.3 *Software*

Our software primarily uses Python and Unreal Engine (version 4.19). We integrate Python scripts into our pipeline for data loading and preprocessing. Network Common Data Form (NetCDF) files are loaded using the NetCDF module for Python (`netCDF4-python`). Other notable modules include `numpy`, which is used for large-scale matrix operations, and `matplotlib`, which is used to store and apply the generated colormaps.

Unreal Engine features a level editor (Figure 10), which acts as a canvas for various objects to be placed throughout the vir-

tual environment, and is used for the core display capabilities. Development in Unreal Engine is done in either C++ or Unreal Engine's proprietary graphical user interface tool called Blueprints (Figure 11). We use both to build our visualization.

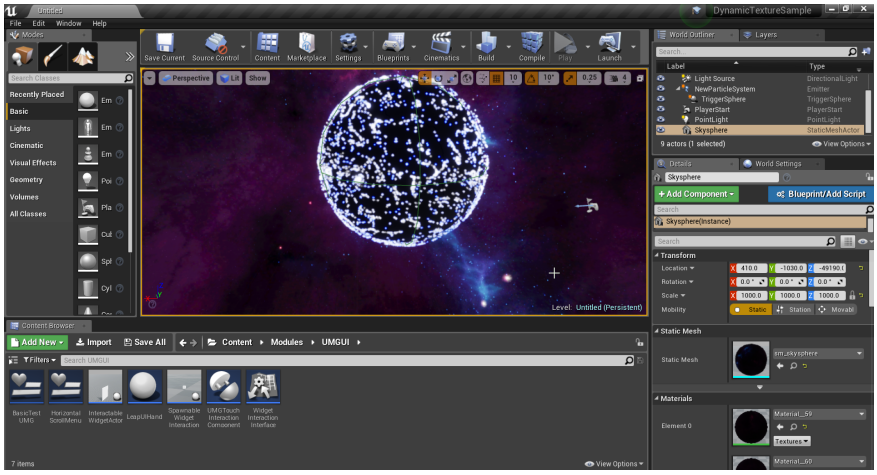


Figure 10: Screenshot of the Unreal Engine editor.

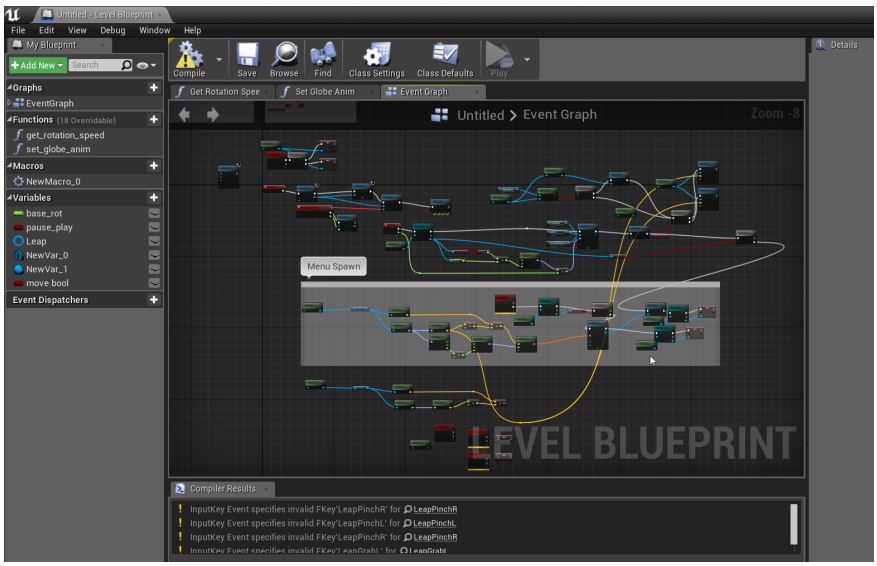


Figure 11: Screenshot of Unreal Engine's Blueprint mode.

Our C++ program drives the visualization. For the global visualization, this code calls our Python preprocessing script for data input. For topography data in the local visualization, the Large Scale 3D Terrain Generator (L3DT) tool is used to trans-

form raw topography data into a format that Unreal Engine can read and visualize. L3DT is also used to fill in missing values in the topography data. After inputting the data, the C++ program generates a set of textures from the input dataset. Finally, it controls the animation by updating the texture displayed at each tick. We use C++ at this step to call our Python script externally; Blueprints does not allow us to do this. Moreover, C++ is a more flexible environment than Blueprints is when working with traditional data structures.

Meanwhile, the Blueprints is the backbone of our user interface. The Blueprints features a graphical user interface (GUI) with various function blocks. One can connect the inputs and outputs of various blocks to develop code in a flowchart-like environment. An example of a Blueprint is shown in Figure 11. The graphical nature of Blueprints accelerates our own user interface development, especially with its drag and drop functionality. This allows us to use existing assets from Unreal Engine to organize different user interface elements (buttons, sliders, etc.) together and have a specific action tied to each element. For this step, Blueprints is a more intuitive and modular solution than C++.

3.1.4 *Data Formats*

Our software interacts with a variety of different file formats during runtime. These files store information such as different climate data or texture data. A brief description of the data formats we work with are summarized below in Table 1.

Table 1: List of data file formats with their extensions and a description of each.

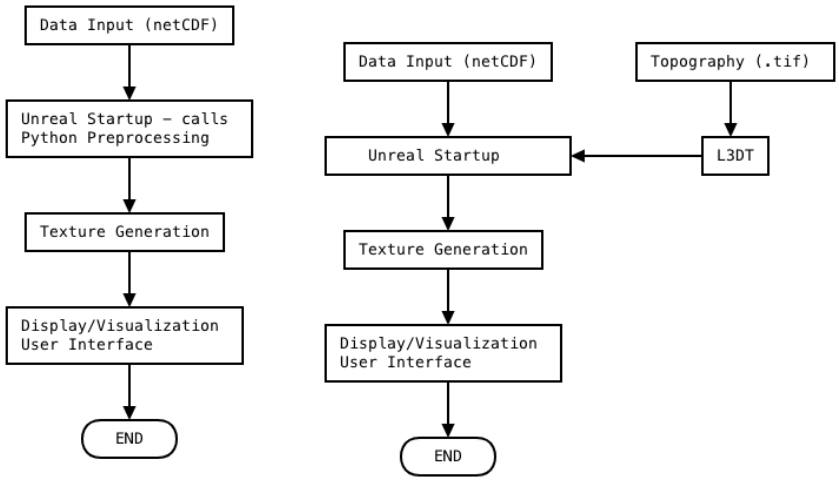
NAME	EXTENSION	DESCRIPTION
NetCDF or Network Common Data Form	.nc4 or .nc	A common standard file format for gridded, self-describing data, frequently used for climate data
Tagged Image File Format	.tif	A file format used for storing raster graphics
Fluid Grid ASCII	.fga	A specialized text file specifying the values of vectors in 3D space
Binary File	.bin	A generic file type that stores binary data
R 16	.r16	A format which stores height data that Unreal Engine can read

Climate data are also commonly stored in file formats such as GRIdded Binary (GRIB), Hierarchical Data Format (HDF), or Georeferenced Raster Imagery, but this project focuses on the NetCDF file format for our proof of concept.

3.1.5 *Architecture*

Our software currently has two display options: (1) a combination of magnitude and direction of wind data at a global scale, and (2) a combination of sea surface temperature and topography data at a local scale around the Chesapeake Bay. The pipeline that carries the input dataset to the visualization is similar in either case, but there exist minor differencess in how data are loaded into Unreal Engine. An overview of the general software architecture design of both our global and local visualizations is summarized in Figures 12a and 12b, respectively. We provide a brief summary of each step of the pipeline in this

section, while the following sections will delve into greater details of each part of our architecture. After the visualization is generated in the Unreal Engine virtual environment, it is then transmitted to the Oculus Rift headset.



(a) Global Pipeline (b) Local Pipeline

Figure 12: Software architecture pipelines for our (a) global visualization and (b) local visualization.

For the global visualization, we first preprocess an input set of climate data into a form that Unreal Engine can visualize. The preprocessing is broken into two steps: Python preprocessing and Unreal Engine initialization.

First, Unreal Engine initializes objects in the virtual environment on startup. In the C++ initialization code for our globe or plane objects, we insert a call to a Python application which generates a file of the colors to display for each point in each timestep of the data.

Since Python generates the files necessary for display, Unreal Engine has no access to the raw data or their attributes. Therefore, we must pass information to Unreal Engine so that it can notify the user of the displayed data, such as variable names,

units, time ranges, etc. This necessitates the creation of a metadata file which contains this information. We explain how each of these files is generated and displayed in the next section. For the local visualization, the NetCDF dataset can be fed directly into Unreal Engine for texture generation. However, the topography data must be processed through L3DT first.

Next, using the processed input data, Unreal Engine generates a texture which is applied to an actor, a sphere for the global visualization and a plane for the local visualization. At each clock this texture updates, thereby animating the visualization. The animation restarts after the final timestep. Concurrently, a user may use their hands to interface with the visualization through the Leap Motion Controller.

3.2 DATA PREPROCESSING IN PYTHON

3.2.1 *Introduction*

Preprocessing of the original data is applied in Python prior to reading the data into Unreal Engine. The visualization engine does not require knowledge of the specifics of the data, making the pipeline more efficient and modular. To initialize the pipeline, we input a source data file, containing weather or climate data, and a predefined colormap. However, if a colormap is not provided then a list of colors describing the intended gradient should be input. Moreover, if the data file contains one or more vector fields, then the radius and granularity of the globe must also be given. Finally, start and end times, as well as specific timestamps, can be input, but are not required.

The preprocessing is broken down into multiple stages. In stage 1, we generate a metadata file which contains basic information about the data being displayed. Such information may include real-world names of the variables, units, time ranges, and the dimensions exactly as given in the source data file. This information can then be displayed in the virtual environment to inform the user about the data they are analyzing.

In stage 2, the source data are mapped according to a predefined colormap, if supplied, and then written to a temporary binary file. The user can also specify colors to use in the colormap gradient and generate a colormap automatically according to a *perceptually-uniform color space*, for which differences between colors closely follow those of the human eye. We use this approach as an alternative to the traditional method of linearly interpolating between the digitally-represented values 0-255, and describe the motivation behind this choice more in the next section.

In stage 3, we convert vectorized data to the Fluid-Grid ASCII (.fga) format, which can be used by Unreal Engine to display particle fields to visualize data such as wind vectors. This contains a list of the individual 3D vectors of the vector field that are to be displayed. Stages 2 and 3 are described in more detail in the following sections.

3.2.2 *Stage 2 of the Data Preprocessing*

Stage 2 involves specifying the colors used for displaying the source data in Unreal Engine. Traditionally, all colors displayed on digital screens are represented using three values: red, green,

and blue (RGB). Each value falls in the range 0-255. This is also known as an 8-bit color representation, as each value is represented using 8 bits. All values being zero equate to black, while all values being 255 equate to white.

For linear interpolation between two colors in this space, one would increment in equal steps between each RGB value between each color. For example, incrementing in five steps between blue (0, 0, 255) and red (255, 0, 0) results in the following values: (0,0,255), (51,0,204), (102,0,153), (153,0,102), (204,0,51), (255,0,0)

However, even though these values are equidistant numerically, they are not equidistant in how the human eye perceives color. A *perceptually-uniform color space* attempts to correct this discrepancy by creating a space where the linear distance between two points in the space is proportional to the perceived distance. Figure 13 compares the digitally-uniform color gradi-

(a) Digitally-uniform

(b) Perceptually-uniform

Figure 13: Comparison of a blue to red color band that is (a) digitally- vs. (b) perceptually-uniform.

ent to the perceptually-uniform color gradient of blue to red. In Figure 13a, the blue and red sections of the digitally uniform color gradient are abnormally stretched out towards the center, with very little relative mixing taking place. Compared to Fig-

ure 13b, the digital band also appears slightly darker. These two traits are the main differences between a digitally-uniform and a perceptually-uniform color space. The perceptually-uniform color space in Figure 13b follows the Hunter Lab space developed by the International Commission on Illumination (CIE) during the late 1940s (Optical Society of America, Inc., 1948).

To determine the perceptually-uniform RGB space, the Hunter Lab space uses a parameter L, which signifies a lightness value, and two values a and b, which roughly equate to translations along red and green for a, and yellow and blue for b. Since this space is perceptually uniform, linearly interpolating between points in this space translates to the same visual difference in the colors.

3.2.2.1 *Conversion from sRGB to Hunter*

The RGB present in display screens is referred to as the *standard RGB (sRGB)* space. To convert from sRGB to Hunter and back, an intermediate space known as the *XYZ* space is used, because this space was used to develop all other color spaces.

The XYZ space was developed by CIE in the 1930s to produce a link between color wavelengths and the perceived colors of a human eye (Smith and Guild, 1931). For this reason, XYZ values for a certain color are the same regardless of the device being used to display them (QuickTutorial, cited 2018). Before we find XYZ values for colors, we must define an illuminant, or a reference "white" point, from which all other values are

determined. Illuminant D65 is widely used as a reference white point. The XYZ values for this illuminant are as follows:

$$\begin{aligned} X^* &= 95.047 \\ Y^* &= 100.000 \\ Z^* &= 108.883 \end{aligned} \tag{1}$$

We will use these values in our conversions.

A *gamma correction* must be applied to the values to convert the sRGB to a linear space before a linear transformation can be used to convert to XYZ. The gamma function is as follows:

$$\gamma(u) = \begin{cases} u/12.92 & u < 0.04045 \\ \left(\frac{u+0.055}{1.055}\right)^{2.4} & \text{otherwise} \end{cases} \tag{2}$$

where u is each individual RGB value, scaled to be between 0 and 1. After gamma correction, calculating the XYZ values is a simple matrix multiplication (IEC, 1999).

$$\begin{bmatrix} X_{D65} \\ Y_{D65} \\ Z_{D65} \end{bmatrix} = \begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix} \begin{bmatrix} R_{linear} \\ G_{linear} \\ B_{linear} \end{bmatrix} \tag{3}$$

XYZ values produced through this transformation are scaled such that $Y^* = 1$. However, the real reference value of Y^* is 100,

as shown in (1). Therefore, we multiply $(X_{D65}, Y_{D65}, Z_{D65})$ by 100 before converting to Hunter. The conversion rules are:

$$\begin{aligned} L &= 100\sqrt{\frac{Y}{Y^*}} \\ a &= K_a\left(\frac{\frac{X}{X^*} - \frac{Y}{Y^*}}{\sqrt{\frac{Y}{Y^*}}}\right) \\ b &= K_b\left(\frac{\frac{Y}{Y^*} - \frac{Z}{Z^*}}{\sqrt{\frac{Y}{Y^*}}}\right) \end{aligned} \tag{4}$$

where,

$$\begin{aligned} K_a &\approx \frac{175}{198.04}(X^* + Y^*) \\ K_b &\approx \frac{70}{218.11}(Y^* + Z^*) \end{aligned} \tag{5}$$

3.2.2.2 *Conversion from Hunter to sRGB*

To perform the reverse conversion, we apply the inverse of the operations listed in the preceding section in reverse order. Rearranging the formula for L in (4) gives us $Y = Y^*\left(\frac{L}{100}\right)^2$. We can use this equality with the other equations in (4) to solve for X and Z as follows:

$$\begin{aligned} X &= X^*\left(\frac{a\sqrt{\frac{Y}{Y^*}}}{K_a} + \frac{Y}{Y^*}\right) \\ Z &= Z^*\left(\frac{Y}{Y^*} - \frac{b\sqrt{\frac{Y}{Y^*}}}{K_b}\right) \end{aligned} \tag{6}$$

To transform to the linear RGB space, we first divide the XYZ values by 100 to force $Y^* = 1$. Then we multiply each XYZ point by the inverse of the matrix in (3):

$$\begin{bmatrix} R_{linear} \\ G_{linear} \\ B_{linear} \end{bmatrix} = \begin{bmatrix} 3.2406 & -1.5372 & -0.4986 \\ -0.9689 & 1.8758 & 0.0415 \\ 0.0557 & -0.2040 & 1.0570 \end{bmatrix} \begin{bmatrix} X_{D65} \\ Y_{D65} \\ Z_{D65} \end{bmatrix} \tag{7}$$

Finally, the gamma corrections are applied on the resulting values to convert to the sRGB space using the inverse of the gamma function in (2), with a slightly different cutoff point.

$$\gamma^{-1}(u) = \begin{cases} 12.92u & u < 0.0031308 \\ 1.055u^{1/2.4} - 0.055 & \text{otherwise} \end{cases} \tag{8}$$

The function outputs values mostly between 0 and 1. However, some values are outside this range and must be clipped appropriately. At this point, multiplying by 255 and rounding to the nearest integer will give us sRGB values in the range 0-255.

With respect to the user inputs, each of the provided RGB colors is first converted to Hunter. Then linear interpolation is performed between these points to get a total of 256 points from the left color to the right color. These 256 points constitute the colormap that is applied to the original data at all timesteps.

For an example of how these maps are stored, please refer to Appendix A.1 and A.2.

3.2.3 *Stage 3 of the Data Preprocessing*

Stage 3 focuses on the display of 3D vector fields. We assume the vector fields are mapped onto a sphere, with a given radius

r. This sphere is housed in a cube of size $2r$, while the distance between points is the step resolution μ. These two values are set by the user before creation of the vector field. The source data file provides the u and v components of the vector field at each latitude ϕ and longitude λ on a defined grid. In our visualizations, the point $(r, 0, 0)$ in Cartesian coordinates is designated by $(0°N, 0°E)$. The preprocessing of this data prior to display is broken up into two parts: (1) calculating the appropriately rotated 3D vectors for each known point, and (2) interpolating between adjacent points on the surface.

3.2.3.1 *Rotating our Known Vectors*

Initially, the u and v values of the wind vectors in the data are defined relative to the surface of the sphere. We need to transform them so that they are defined relative to the absolute position in 3D space. Each of the resultant vectors will lie at the specified latitude and longitude coordinates on a plane tangent to the sphere. The transformations involve a set of rotations utilizing the latitude and longitude of our vector. To begin, we will "flatten" the vectors so that they each have initial direction $(u, v, 0)$.

Initially, $(0°N, 0°E)$ maps to $(0, -r, 0)$ on the sphere. To get $(0°N, 0°E)$ to be at the point $(r, 0, 0)$ in our absolute 3D space,

we first rotate the vector $\frac{\pi}{2}$ radians (90°) around the z-axis (signs follow the right-hand rule) using the rotation matrix:

$$R_1 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{9}$$

Next, we apply a rotation so that the vector is at the specified latitude level. However, 0°N, i. e. the equator, lies on the xy-plane. Therefore, to get the correct rotation amount, we subtract the latitude value from $\frac{\pi}{2}$. The rotation is about the y-axis, and so the rotation matrix is,

$$R_2 = \begin{bmatrix} \cos\left(\frac{\pi}{2} - \phi\right) & 0 & \sin\left(\frac{\pi}{2} - \phi\right) \\ 0 & 1 & 0 \\ -\sin\left(\frac{\pi}{2} - \phi\right) & 0 & \cos\left(\frac{\pi}{2} - \phi\right) \end{bmatrix} = \begin{bmatrix} \sin\phi & 0 & \cos\phi \\ 0 & 1 & 0 \\ -\cos\phi & 0 & \sin\phi \end{bmatrix} \tag{10}$$

Finally, for longitude orientation, we rotate around the z-axis once more. There are no corrections here, because the units are °E, which is oriented in the positive rotation direction:

$$R_3 = \begin{bmatrix} \cos\lambda & -\sin\lambda & 0 \\ \sin\lambda & \cos\lambda & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{11}$$

Multiplying through these matrices $R_3R_2R_1$ with our initial vector $(u, v, 0)$ gives:

$$R_3R_2R_1 \begin{bmatrix} u \\ v \\ 0 \end{bmatrix} = \begin{bmatrix} -u\sin\lambda - v\cos\lambda\sin\phi \\ u\cos\lambda - v\sin\lambda\sin\phi \\ v\cos\phi \end{bmatrix} \tag{12}$$

The final step is calculating the new position of this vector. This is a simple conversion from spherical coordinates to Cartesian coordinates using our radius, latitude, and longitude. The correction to our latitude value is also included:

$$\begin{aligned} x = r\sin\left(\frac{\pi}{2} - \phi\right)\cos\lambda &= r\cos\phi\cos\lambda \\ y = r\sin\left(\frac{\pi}{2} - \phi\right)\sin\lambda &= r\cos\phi\sin\lambda \\ z = r\cos\left(\frac{\pi}{2} - \phi\right) &= r\sin\phi \end{aligned} \tag{13}$$

We now have the rotations and positions of our input vectors to be displayed in 3D space. The next step is to interpolate our known values among all of the points with unknown values that lie on the sphere with designated resolution, μ.

3.2.3.2 *Interpolating Through Points on the Sphere*

The grid coordinates x, y, and z in our Unreal Engine display space follow discrete radius levels $(-r, -r+\mu, -r+2\mu, \cdots, r-2\mu, r-\mu, r)$. Any point whose distance from the origin falls in the range $[r, r+\mu]$ will be deemed "close enough" to be interpolated from the larger grid.

We assume that the known values in the original data follows a rectangular mesh. In this case, each point that is assigned an interpolated value uses the four closest points at the appro-

priate radius that have known values. These four points will always form a quadrilateral enclosing our target point. Excluding special cases, each target point will have two latitude and two longitude levels bounding the point. The latitude and longitude values of our target point are calculated using standard Cartesian to spherical coordinates formulas, with the added correction to the elevation value. We linearly interpolate along both the horizontal and vertical direction; the latitudes and longitudes stay constant along each direction, respectively. Fig-

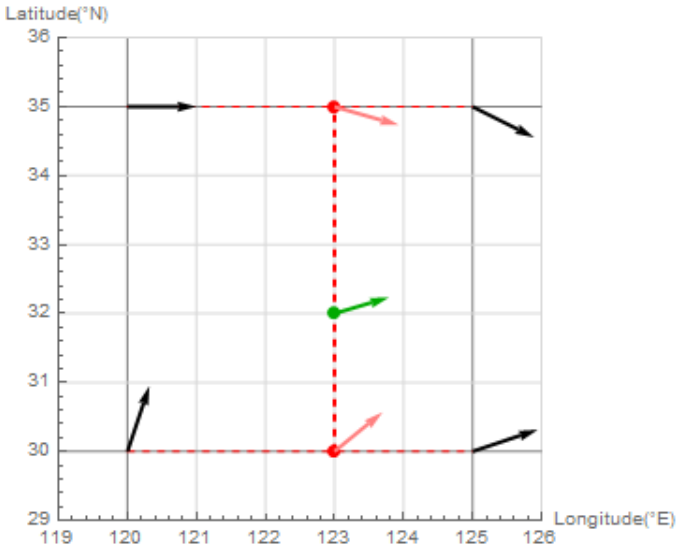


Figure 14: The black vectors are the known points, and the green point is our target point. We interpolate on the two sides above and below the target point. We interpolate vertically to calculate the proper vector for our target point.

ure 14 shows an example interpolation where our target point is at (123°E, 32°N), and we have known values at points at (120°E, 30°N), (125°E, 30°N), (120°E, 35°N), and (125°E, 35°N), as shown by the black vectors. We first interpolate in the longitudinal (East/West) direction between the two points at each of the latitude levels to get two new pink vectors at (123°E, 30°N)

and (123°E, 35°N). Then, we interpolate again between these two vectors to get our green target vector at (123°E, 32°N).

A special case is when the calculated latitude value is either greater than or less than the maximum or minimum values of the known latitude range. In this case, our four points will all come from a single latitude level and four separate longitudinal levels. Afterwards, interpolation is done in a similar manner. Since this is a linear interpolation of four vectors in 2D space, it is possible that the interpolated vector is not perfectly tangential to the sphere.

Our initial Unreal Engine display space was a box with side length $2r$ and centered at the origin. All vectors in this space are initially set to the $\mathbf{0}$ vector. The above analysis only sets nonzero vectors at points that are close to the surface of the sphere. In the case where a particle passes through other points with a $\mathbf{0}$ vector, no additional force will be applied to the particle, and it will keep moving in its original direction. This can have the visual effect of particles leaving the sphere, obscuring the actual vector field and hampering the user experience.

For this reason, we also apply an inward gravity field to all points with a $\mathbf{0}$ vector to force the particles back toward the sphere. An outward gravity field is also applied to all points that are within the sphere radius. To further ensure that the eight points along the three poles are not interacting with the space outside the box, we pad the vectors with an extra layer of gravity. This results in a box side length of $2r + 2$, while the sphere radius is still r.

Finally, recall that we assumed that the input data followed a rectangular mesh. For non-rectangular meshes, such as triangular or hexagonal meshes, a similar process would be followed, taking the closest points from each interpolated point. For example, a triangular mesh would take the closest 3 points.

For code snippets detailing how the vector rotations and interpolations are performed, please see Appendices A.3 and A.4.

3.3 DATA PROCESSING IN UNREAL ENGINE

3.3.1 *Global Visualization*

After the source data are preprocessed, they are then read into Unreal Engine. Objects in the Unreal Engine virtual environment are classes called Actors, which have properties that can be edited through source code. Each Actor can have a mesh applied to it which dictates its appearance and how it interacts with light in the virtual environment. Our virtual globe is a sphere Actor with a mesh applied to it which displays an animating texture using the perceptually uniform color grid calculated during preprocessing. When the mesh is generated as the environment is loaded, we use the Python to C API to set the arguments to the script and call it from within Unreal Engine. We then read the information from the newly created metadata and colormap files to create an array containing the RGB color value for each point on the grid at each timestep. Each time Unreal Engine refreshes the display, the texture is updated to display colors from the next timestep, wrapping around to the first time step if the end has been reached. We apply this programatically created mesh to our sphere Actor, and the texture

is warped to match this shape. We also apply a premade mesh on top of the colored texture mesh to display the boundaries of continents.

In order to display the movement of wind vectors on a global scale, we use the Unreal Engine object called Particle System. Using Particle System, new particles can be spawned along the surface of the sphere, but these particles remain stationary unless placed under the influence of a Vector Field object. We import the .fga format vector data generated during preprocessing directly into Unreal Engine, which converts it into a Vector Field object. The particles spawned on the surface of the sphere then move in the direction of the vectors specified by the .fga file. The tightness parameter of the vector field allows the "momentum" of particles to change. We set this tightness parameter as high as possible so that any particle will immediately change direction if it moves into a gridded area controlled with a different vector. This ensures that particles always move tangent to our globe.

3.3.2 *Local Visualization*

3.3.2.1 *Motivation*

In order to allow users to study local phenomena in more detail, we create a prototype local visualization as a proof of concept for how a regional display would look and feel. Moreover, since many available datasets are region-specific, this visualization increases the possible types of datasets users can load into the tool. With this local visualization, we increase the functionality of the tool and give users a sense of its further potential. The

Chesapeake Bay is one of the most interesting ecosystem in Maryland and is a frequent topic of study at the University of Maryland (UMD). For this reason, we focus on visualizing Chesapeake Bay water temperature over a period of 18 days.

3.3.2.2 *Raw Climate Data*

We use the National Oceanic and Atmospheric Administration (NOAA) Coast Watch East Coast Node, which includes regional climate data for many regions along the U.S. east coast. We also use the daily Advanced Very High Resolution Radiometer (AVHRR) dataset, which contains retrieved surface water temperature observations. The dataset has 1 km horizontal resolution and can be downloaded in the NetCDF data format for any desired day starting from 2009. In addition to the raw data files, the database also allows users to download 2D images (in .png format) that NOAA automatically generates for each dataset. This allow us to validate our own visualizations of the dataset. We examine the period from August 17, 2017 to September 6, 2017, when Hurricane Harvey hit the Chesapeake Bay.

To automate downloading the data, we call a Python script, which is included in Appendix A.5. Once the datasets are downloaded, we load the data using the `netcdf4` C++ library into the Unreal Engine program memory. During the initialization stage of the local visualization, the program iterates through each of the 18 data files separately and saves them into a 3D array that is indexed by latitude, longitude, and time.

Next, we animate the the visualization by periodically updating a 3D pixel array that is mapped onto the display. This pixel array is indexed by x, y and RGB. In other words, we need to specify the RGB values for each pixel. In the update routine, we process the data in the next time frame by converting each raw temperature value into RGB values, where red corresponds to higher temperatures and blue corresponds to lower temperatures. We save these values into the pixel array. At the end of the update routine, we display this updated color pattern stored in the pixel array.

There are two main objects in the local visualization. The first is a static mesh plane, which floats in the environment and is the main physical object displayed. By itself, the static mesh plane has no color or texture. However, we apply the dynamic texture material, the second main object, to the static mesh plane to give it color. The update routine mentioned above updates this dynamic texture material periodically to give an appearance of animation over time.

3.3.2.3 *Topography Data*

Another aspect of the local visualization is the topography data. Topography data are obtained from websites, such as OpenTopography, in a .tif format. For our local example, we choose to download topography data for the exact same area as our climate data in order to smooth the process later.

The L3DT tool is used to transform the topography data to heightmaps in .r16 format so that Unreal Engine can quickly read and visualize the data. L3DT is also used to fill in un-

known areas that are missing data with elevation level zero for our example. In general, L3DT offers two options for filling missing values, either by interpolating based on neighboring data or by filling in with a default value such as zero.

In order for Unreal Engine to read the .r16 files, several steps must first be taken. Enabling the world composition is necessary for Unreal Engine to read and stitch the files together. After this, the files are loaded into the level as tile landscapes. Each individual tile is now a level and by loading all of the levels at once, the full terrain is visible and ready to be integrated with the climate data.

3.3.2.4 *Unreal Engine Integration*

After the climate data and the topography data are loaded into Unreal Engine, they are overlaid into one coherent visualization. Lining up these visualizations so that the positioning is geographically accurate and the appearance is a seamless transition from one data source to the next is a difficult task. We first used data corresponding to the same region on Earth so the latitude and longitude boundaries of both datasets match exactly. Next, we manually adjusted the aspect ratio and size of the static mesh actor (the plane for the climate data) to match those of the topography. Finally, the climate data plane is translated so that it sits slightly above sea level on the terrain. The resizing and scaling of the static mesh actor is tuned manually using Unreal Blueprints. Since the topography data and water temperature data are loaded into the environment through in-

dependent processes, we have two separate objects which are overlaid on top of each other.

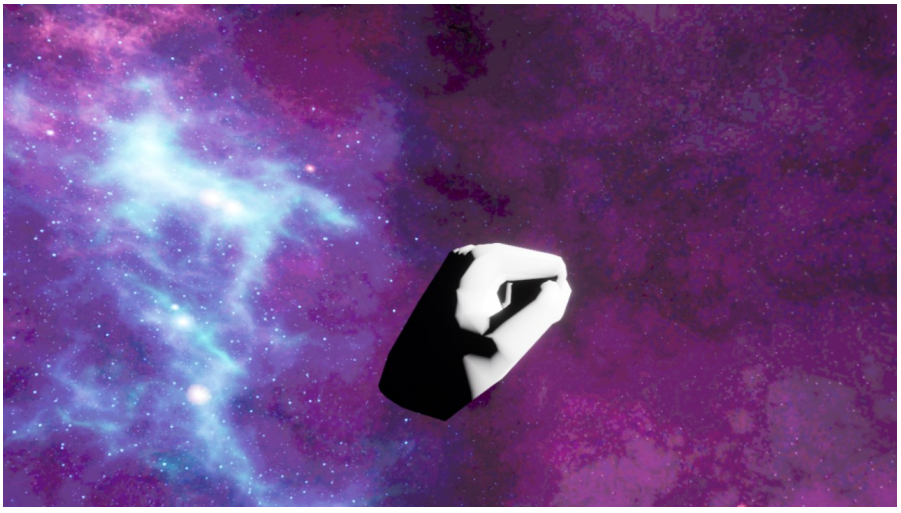
3.4 DEVELOPMENT OF THE USER INTERFACE

Our user interface combines the Leap Motion Controller with Unreal Engine Blueprints for optimal user interactivity. A plugin by Kaniewski (2018) interprets information from the Leap Motion Service and provides it to Unreal Engine. This allows us to project a user's hands into the virtual environment. Additionally, the plugin allows hand joints to be tracked at a high refresh rate. Thus, a user may control aspects of the visualization with their hands when a built-in gesture is recognized. Moreover, accessing fingertip location is essential in constructing an interactive menu.

Kaniewski's plugin supports four built-in gestures: left pinch, left fist, right pinch, and right fist. Examples of the left pinch and left fist gestures are shown in Figures 15a and 15b, respectively. In Blueprints, recognition of one of these gestures is an Event, and certain actions may follow. For example, in the global visualization, when a user pinches his or her right hand, the interactive 3D menu appears in front of the user. To ensure the menu spawns facing the user, we use the user's current position and forward vector, which is a unit vector that points in the direction the user is looking in the world (Epic Games, 2018). We place the menu a set distance along the user's forward vector, then orient the menu to face the user. Refer to Appendix A.6 to view the corresponding Blueprint.

The menu contains traditional user interface elements, such as sliders and buttons, which are tied to specific actions when triggered. To interact with the menu, we track the position of the user's fingertip. When a user's fingertip intercepts one of the menu items, a certain action is called. For example, in the global visualization users can control the rotation speed of the globe by moving a slider. The plugin also allows developers to track various other joints in a user's hands. One could use this information to expand upon the four built-in gestures.

(a) Left pinch



(b) Left fist

Figure 15: Examples of built-in Leap Motion Controller gestures, provided by Kaniewski's plugin (Kaniewski, 2018), in a virtual environment.

3.5 FOCUS GROUP FORMAT

In order to gain feedback on our prototype for further evaluation, we organized a focus group composed of professors from the Department of Atmospheric and Oceanic Sciences (AOSC) and students from both AOSC and the broader community at UMD. We received an Exempt status from the Institutional Review Board (IRB) to collect survey data from participants anonymously after they use our visualization tool. Each participant signed a consent form assuring their anonymity and agreeing to the minimal risks enumerated on the form in Appendix B.1.

We recruited our participants by reaching out to professors, as well as PhD, Master's, and undergraduate students studying the climate with an interest in data visualization. After receiving responses to our initial request, we reached out a second time requesting our participants to select a 20-minute time slot to meet with us at the focus group location to try our prototype. The sign-up form is shown in Appendix B.2.

The survey questions given to participants gauged the intuitiveness and usefulness of our visualization prototype from the perspective of a climate researcher. The full survey is listed in Appendix B.3, but in general, questions focused on expected usage, ease of interpretation, and comparison to traditional 2D visualizations.

4

RESULTS

4.1 GLOBAL VISUALIZATION

Currently, our global visualization can transform a NetCDF file into the components necessary to generate a 3D environment (Section 3.2.1). A user will provide the following to the program: the dataset file to visualize, the color map to display, the start and stop timesteps, and whether the data should be visualized as vectorized data. After the necessary files are generated, the visualization launches.

Once the global visualization has been launched, the user sees the globe with all of the continental land masses outlined. The data are displayed as colors in the perceptually-uniform color space discussed in Section 3.2.2, and the texture is animated through all of the time steps present in the data.

As an example of the capabilities of our prototype, Figure 16 shows a still screenshot of a visualization of the highest hourly temperature throughout March of 2011. The data were retrieved from the Climate Forecast System Reanalysis (CFSR) by the National Centers for Environmental Prediction (NCEP) at NOAA, which is a high-resolution global dataset containing time series data for the atmosphere, ocean, land, sea surface, and ice (Saha et al., 2010). As the globe rotates, users can compare the temperature in different areas of the world. It is easy to see that areas toward the equator are generally warmer. As the

texture on the globe animates, the temperatures increase and decrease cyclically, denoting days and nights. Users can also see that large land masses generally retain more heat throughout the day, shown by the red color of east Africa and India in Figure 16.

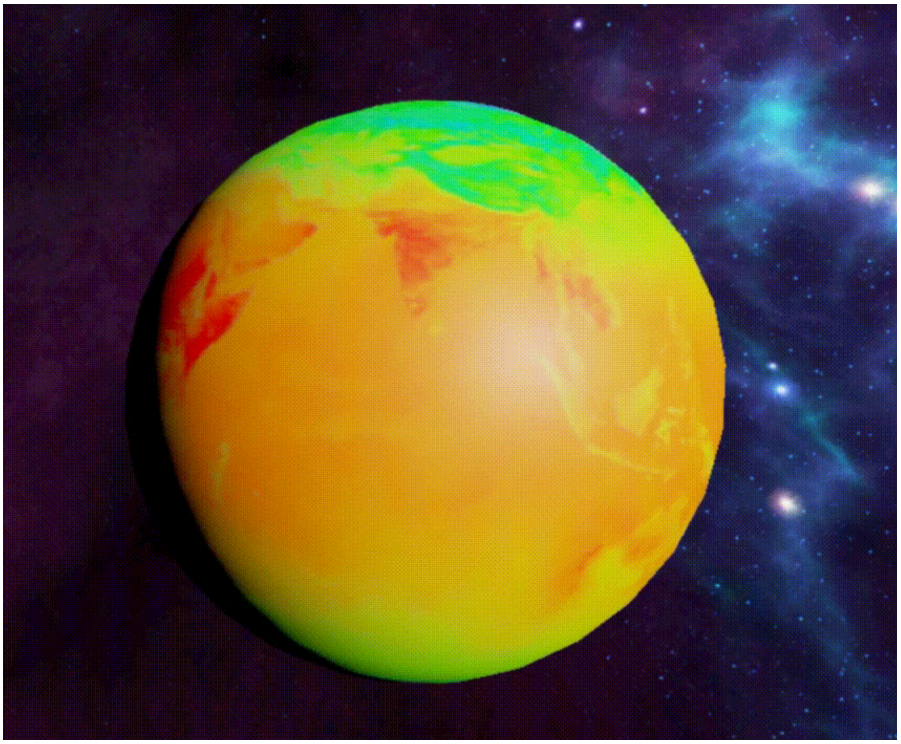


Figure 16: This screenshot shows a high resolution dataset of maximum hourly surface temperature.

If the data passed to the program are vectorized, then a particle field is also displayed above the colors as shown in Figure 17, where individual particles move around the globe according to defined vectors at the first time step in the data. This allows the user to view the intensity, direction, and movement of vectorized data. Users can easily see areas of fast-moving currents and areas of rotation. Figure 17 shows a static image of the final visualization depicting wind data at midnight, September 1st, 2009 over the west Pacific Ocean. These data were also re-

trieved from NOAA's CFSR (Saha et al., 2010). Rotating storms can be seen forming over Indonesia and southeast of Japan; in real time, the particles in these areas appear to move faster than those over land masses since the intensity in these storms is stronger.

;

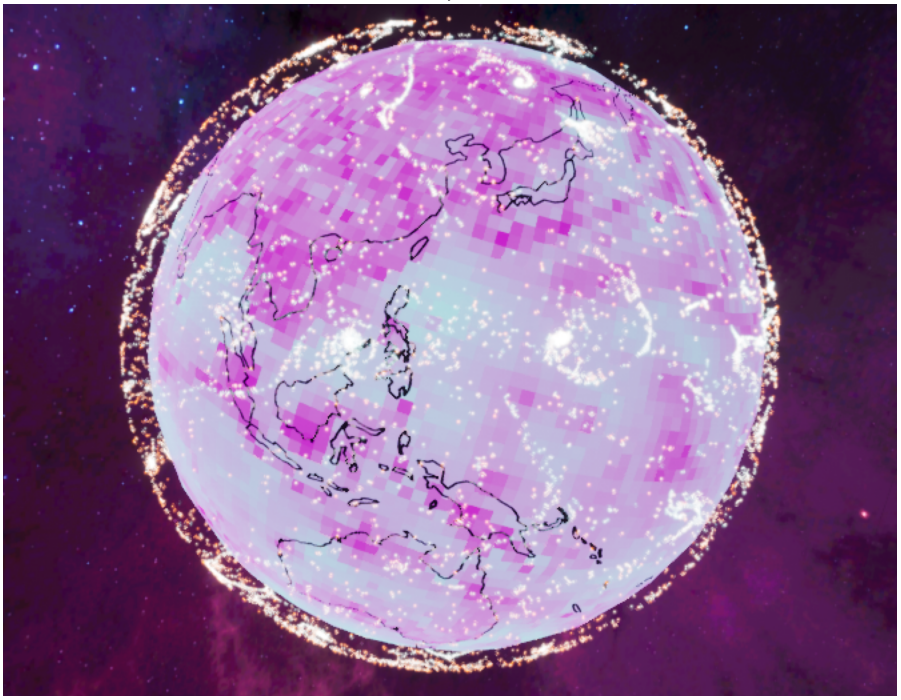


Figure 17: In this screenshot, white dots represent the particles that will move around the sphere. The colors on the globe represent the magnitudes of the data, colorized according to a perceptually-uniform color scheme.

There are several ways to interact with the global visualization. By making a pinching motion with their right hand, a user can open the interactive menu shown in Figure 18. The user can then use their fingertips to interact with the menu to adjust the display. Changing the location of the slider allows the user to adjust the globe's rotation speed. A user can then view the data on all sides of the globe as it rotates. By selecting a check box, a user can also pause the animation of the underlying colored

texture to view data at a specific timestamp of interest. These functions allow users to control which areas and times of the data they are most interested in analyzing, or if they would prefer to view an animated visualization of the entire dataset.

Figure 18: This screenshot shows an interactive menu for the global visualization. With their fingertip, users may move the slider to adjust rotation speed, and click the check boxes to toggle rotation or animation of the globe.

4.2 LOCAL VISUALIZATION

After launching the local visualization, the user sees a close-up view of the Chesapeake Bay region. In this example (Figure 19), the local visualization displays topography data and water temperature data simultaneously. However, one can visualize any other climate dataset, assuming it has the same longitude and latitude coordinates as the topography dataset. In Figure 19, the water temperature data are displayed with an RGB color scale on the texture, which is animated through all of the time steps present in the data. In the color scale, warm temperatures are represented with red hues while cold temperatures are represented with blue hues. Water temperature data can be seen on

the flat areas, which correspond to regions of water. The user can also view the topography data over the land. The topography is shown with an earthy color to give a more realistic representation.

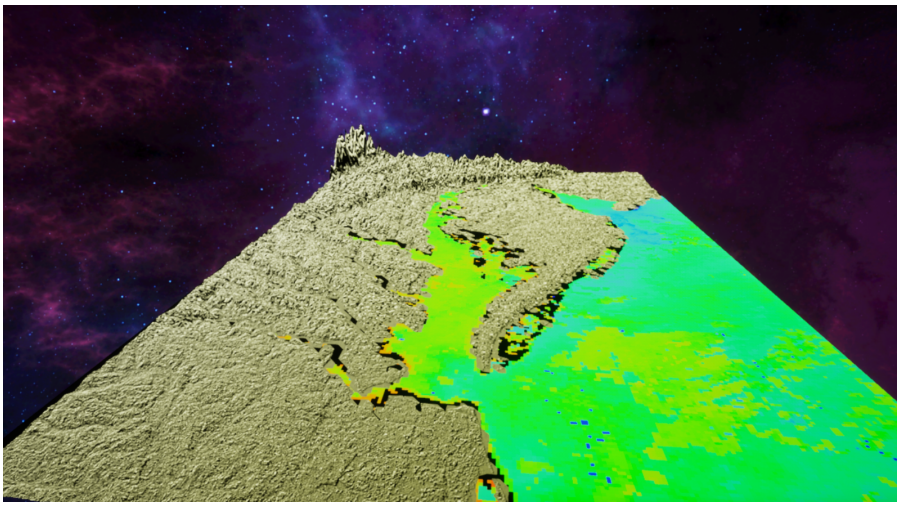


Figure 19: A still screenshot of the Chesapeake Bay local visualization of surface water temperature and topography over the land.

A user may interact with the visualization through three gestures provided by the Leap Motion Controller. By making a pinching motion with their left hand, a user may pause the underlying animation to look at specific timestamps of the data. Figure 20 shows a user's hands above the visualization.

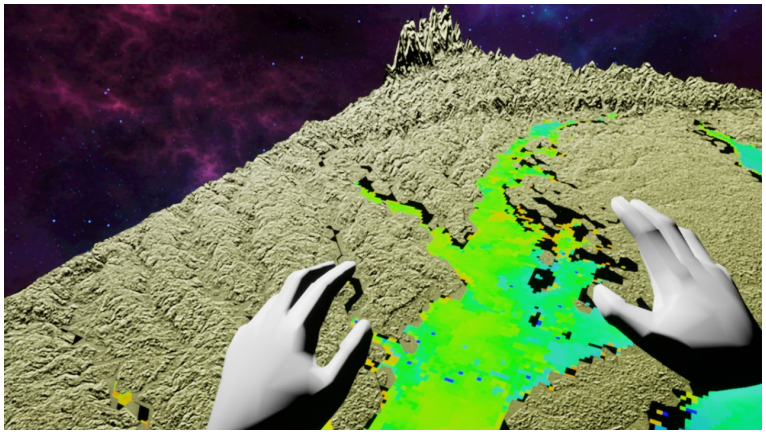


Figure 20: The user's hands are projected into the Chesapeake Bay visualization via the Leap Motion Controller.

A user may also zoom into and out of the Chesapeake Bay by closing their left hand and right hand into a fist, respectively. Figure 21 shows a user standing on the Chesapeake Bay. This view is a result of successive zooms.

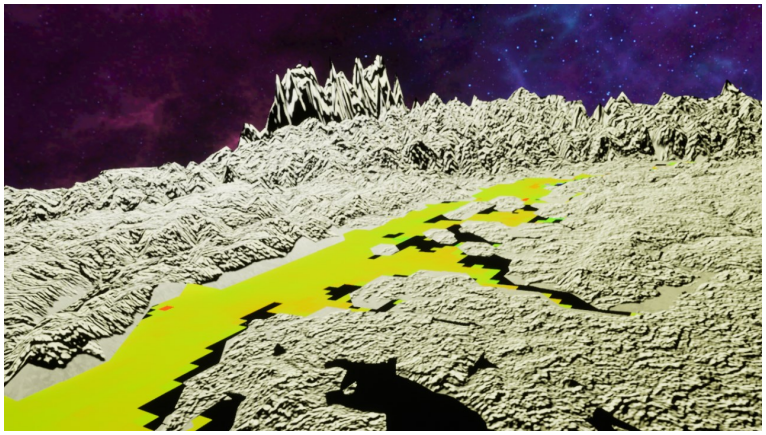


Figure 21: A zoomed in view of the Chesapeake Bay visualization after several zoom actions

4.3 FOCUS GROUP RESULTS

For our focus group, we had a total of eleven participants. Of these eleven participants, eight were undergraduate students and three were professors or post doctoral researchers in the AOSC department at UMD. Overall, most participants had a positive view of our prototype and a good overall experience testing the device (Figure 22).

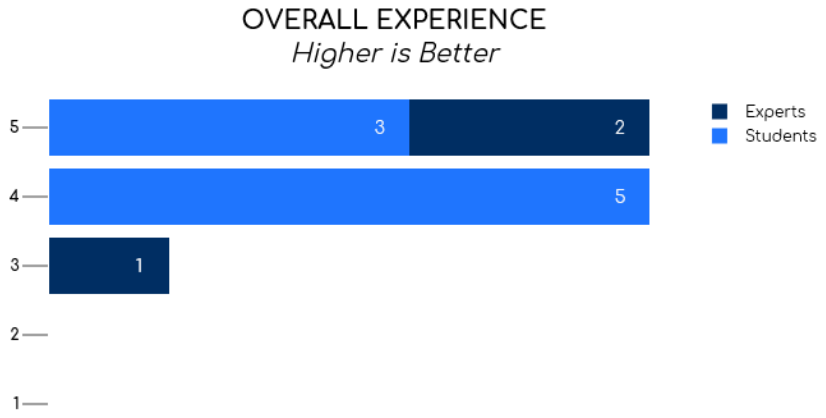


Figure 22: Ratings of the overall experience of using the product, from start to finish. Higher ratings indicate a better experience.

Many participants found that wearing the Oculus headset was comfortable and using the Leap Motion Controller hand gesture controls was intuitive (Figures 23 and 24). However, one participant did have issues wearing the Oculus headset while simultaneously wearing their glasses, but we believe this is a minor problem as many other participants were comfortable wearing the Oculus with their glasses on.

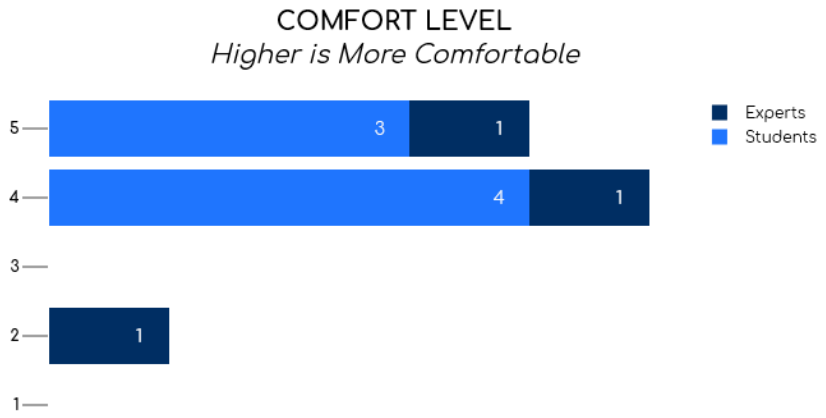


Figure 23: Ratings of how comfortable our participants were when using the device. Higher ratings indicate more comfort levels.

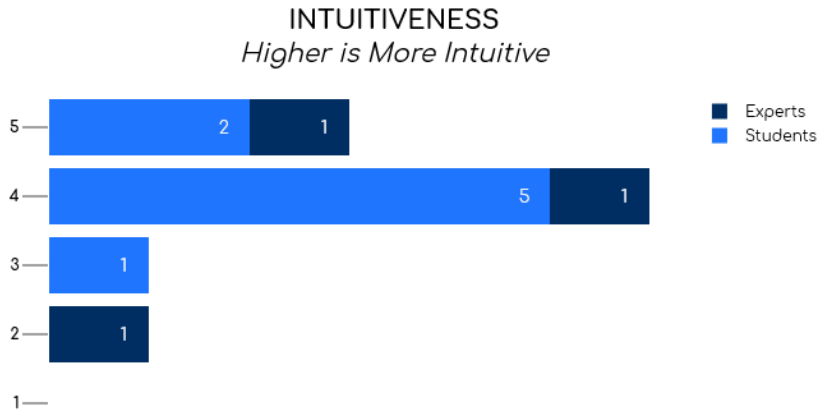


Figure 24: Shows how our participants rated the intuitiveness of the prototype. High numbers indicate higher intuitiveness levels.

There were some issues with the interactions due to how sensitive the Leap Motion Controller is to small hand movements. This issue is particularly apparent with menu interactions as several participants had problems clicking the menu buttons. We found that the best approach to teaching others how to use the device was to demonstrate the gestures before they put on the headset. The issue with the ease of interaction can be seen in Figure 25 where the normal distribution indicates the issues users had with interacting with the prototype. This, coupled with the fact that the majority of users view the ease of interaction as an important feature, indicates that a lot of development time will need to go into the user interface to ensure that interactions are improved in the future (Figure 26).

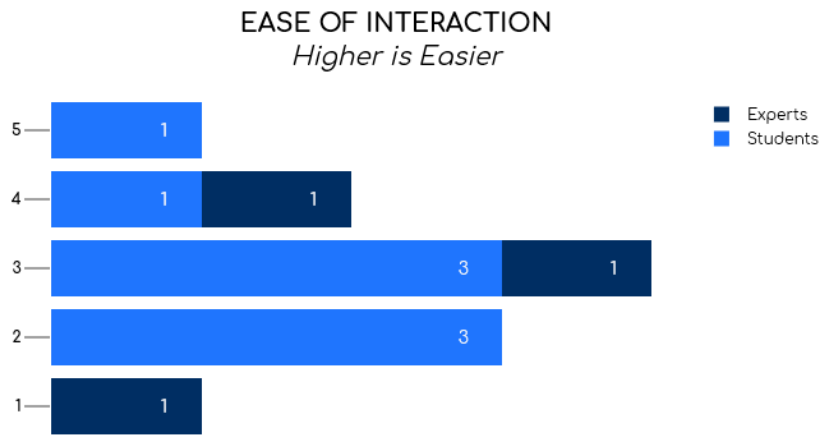


Figure 25: Ratings of how easy it was to interact with the device. Higher ratings indicated easier levels of interactivity.

For our local visualization, many people thought the visualization was good but wanted to see more information. However, they had a relatively easy time navigating this part of the visu-

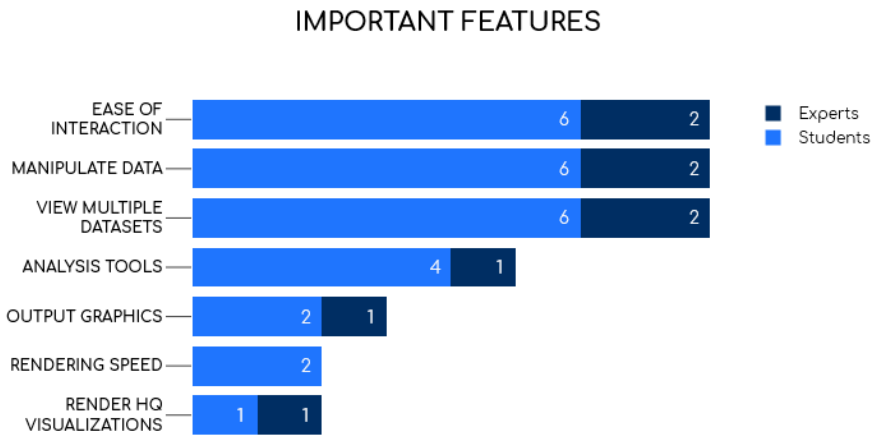


Figure 26: A list of the features that our participants considered most important in a climate visualization tool. Multiple selections were allowed.

alization and found the controls to be relatively intuitive. The topography was seen as interesting but could be more relevant given varied datasets over the topography, such as rainfall and temperature. In general, participants wanted several features such as zooming, a legend showing timestamps, color scale, and data type information, and multiple related datasets such as salinity and algae. One interesting suggestion was to overlay this type of visualization with geographic economic data to see if there are any interesting trends and correlations.

For our global visualization, our participants were impressed with the wind particle simulation and the patterns it displayed. Controls for this visualization were slightly more challenging for some participants but most adapted to it after some time. Participants had a difficult time seeing the continent outlines as they tended to blend in with the color scheme of the globe data. Participants also desired a legend to show pertinent infor-

mation, the ability to change color schemes on the fly, and the ability to zoom in and out. In addition, participants noted that seeing how the wind data change over time would be incredibly useful. Overall, many participants saw a high amount of potential, with continued development, in using VR for weather and climate visualizations (Figure 27).

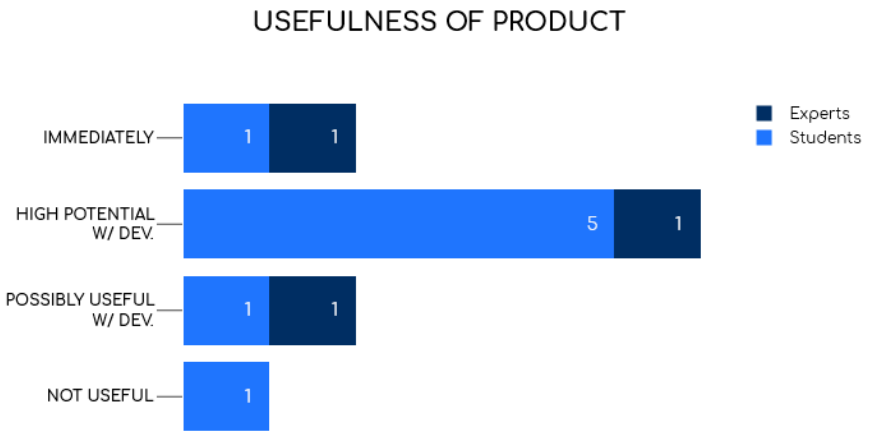


Figure 27: Depicts how our participants rated the usefulness of the prototype in the present and the future.

For additional graphs depicting our results for questions not covered in this section, please see Appendix B.4.

5

DISCUSSION

Throughout this project, we explored novel methods of data visualization using virtual reality to determine if greater immersion and interactivity would benefit a user when drawing meaningful conclusions from data. While our proof of concept showed promise for the extent that VR can improve user experiences in the field of data visualization, there is still further research and experimentation to be done to fully realize the potential of this type of software. We will discuss some of the shortcomings and potential improvements to our system.

5.1 GLOBAL VISUALIZATION

The global visualization displayed a new approach to color with the use of perceptually-uniform color spaces. These spaces allow users to have a more intuitive sense of the differences between values, especially if values are closer to the extremes. The particle simulation of vectorized data also offers a realistic 3D approach to the movement of particles. With these enhanced visuals, researchers can immediately glean insights as opposed to 2D static images.

In our focus groups, our participants mainly said that the wind visualization itself was very well-done. However, they also notified us that some parts of the visualization were inaccurate. For example, apparent cyclone behavior seen in the northern hemisphere was swirling in the clockwise direction,

when it should be rotating be counterclockwise due to the Coriolis effect (and vice versa for the southern hemisphere).

After further review, we also noticed that particle movement in other locations across the globe did not correspond properly to a 2D map. We also noticed that some .fga files presented differently in Unreal Engine versus other physics engines like Blender. Further investigation entailed researching how Unreal Engine processes .fga files, and as of this writing, no clear answer has been documented. Therefore, in the future, this problem needs to be addressed in full.

One goal we had at the beginning of our project was to display multivariate data at many different altitudes to observe interactions between atmospheric and oceanic datasets. Due to technical and time limitations, we were only able to visualize atmospheric data at one altitude with our vectorized data. It would be useful in further research to determine how viewing volumetric layer rendering of 3D compares to viewing data slices in 2D, where possible data saturation may occur.

5.2 LOCAL VISUALIZATION

Our local visualization combined both climate and topography data in a localized region. The 3D nature of virtual reality helps one to understand the depth of the region coupled with the climate data. There are two main future improvements to this visualization. First, we would like to visualize more datasets. Based on our focus group, we found that what users would like to able to see multiple related datasets from the same region. There are several ways that this can be implemented. The

first method is to allow the user to toggles between the datasets. The second method is to visualize all the datasets in the environment simultaneously. While more complex to implement, this method leverages full potential of a VR tool by visualizing datasets ways that other formats would not be able to. We imagine that this could be implemented in the future by adding another plane mesh for each additional dataset.

The second future improvement for the local visualization is streamlining the process of creating the visualization itself. In the future, we would like to automate the process of both loading the topography data and lining up the topography data with the climate data. Currently, both are done manually in L3DT and Unreal respectively. Automating these two processes will save a significant amount time by allowing many types of local visualizations to be generated in real time.

5.3 USER INTERFACE

Our user interface combined the Leap Motion Controller with Unreal Engine Blueprints for optimal user interactivity. This interface allowed users to interact with the visualization directly with their hands. In the local visualization, users could stop and start the underlying animation to observe specific timestamps of interest within the dataset by simply pinching (Figure 15a) with their left hand. Meanwhile, in the global visualization, users could use their fingertips to interact with a menu to control the rotation and animation of the 3D globe.

In our focus groups, most participants found the gestural control to be intuitive. However, there were some issues with

interacting with the 3D menu, due to the sensitivity of the Leap Motion Controller. These users suggested further ways to interact with the data, such as zooming into and out of regions of interest, and changing the visualization's color scheme on the fly.

After the focus group, we devised a way for users to zoom into and out of the local visualization by closing their left hand and right hand into a fist, respectively. These built-in gestures (Kaniewski, 2018), however, may not be the most intuitive way to zoom. Thus, one avenue of future work entails developing custom gestures by using the positional tracking data of bones and joints provided by the Leap Motion Service. To zoom, one should perform a gesture similar to the zoom gesture popularized by smartphones.

A second avenue of future work involves improving the 3D menu of the global visualization. As suggested in the focus group, users should be able to change the color scheme of the visualization; the menu should facilitate this. Additionally, users should be able to switch the dataset being visualized through the menu. When optimized, the user interface should function unnoticed - it cannot act as a hindrance or barrier for users to overcome when visualizing and analyzing data.

5.4 FUTURE DIRECTIONS

Currently, our project primarily focuses on simply loading, visualizing, and interacting with the available climate data in the Unreal Engine environment. As mentioned previously, there is room for improvement when it comes to each of these core

components of our visualization system. Some additional improvements we can make to our visualizations as a whole is to make more information about the data available to the user. This would entail adding basic information such as the name of the dataset, legends for what the color scheme means, timestamp for the data values, etc. Furthermore, users expressed an interest in being able to directly see the values associated with the color map, as well as summary data such as mean values. All of this information can be added to our visualization for each dataset in order to improve the usefulness of our current visualization.

The results of our focus group also show that viewing a 3D visualization is a useful way to present users with a spatial interpretation of data. This could be especially applicable in situations where data can be seen moving in multiple dimensions, such as inside thunderstorms. We recommend that any continued development of our tool should focus on extending the current functionality to encompass a broader set of inputs to allow for layering of related datasets or datasets with information at multiple altitudes.

In addition to improving the existing components, another tangible improvement we can make is to add other useful components to our system, namely analytic tools. While simply being able to visualize data in a 3D environment can already enhance understanding of the data, being able to perform more advanced analysis techniques on the data would add further value to what users can get out of our visualization system. For example, it would be interesting to calculate the correlation be-

tween different variables being displayed and highlight level of correlation in the visualization.

The system could be adapted in the future for either of two different audiences: general users or climate researchers. The user interfaces and functionality developed could vary to serve these two purposes. While climate researchers may prefer flexibility in the visualization including analytic tools to suit their needs, someone using the system to show the impacts of a changing climate to a less experienced user may prefer a simpler user interface with a visualization that will be more impactful to the viewer. The immersive nature of virtual reality allows for this project to be tailored to both audiences, allowing for both greater interactivity and analysis in three dimensions and the ability to create informative and stirring visualizations. The current state of our projects serves as a launchpad for future teams to achieve this vision.

Part III

APPENDIX

A

CODE LISTINGS

For additional code snippets not described here, as well as the rest of our codebase, please clone our Github repositories. The first contains color transformations, while the second houses the development in the Unreal environment. Links:

1. `https://github.com/teamdiva2019/ColorProduction`
2. `https://github.com/teamdiva2019/ProjectDIVA`

A.1 COLORMAP DATA STRUCTURE

One colormap from `matplotlib` is the *plasma* colormap.

Figure 28: The plasma colormap

In our environment, this colormap is represented by a text file of floating point numbers between 0 and 1, denoting the fraction of red, green, and blue respectively, in 256 equal steps, as shown in Listing 1.

Listing 1: The first 10 lines of `plasma.txt`. The rest of the file is omitted for brevity's sake.

```
0.050383 0.029803 0.527975
0.063536 0.028426 0.533124
0.075353 0.027206 0.538007
0.086222 0.026125 0.542658
0.096379 0.025165 0.547103
0.105980 0.024309 0.551368
0.115124 0.023556 0.555468
0.123903 0.022878 0.559423
0.132381 0.022258 0.563250
0.140603 0.021687 0.566959
...
```

A.2 COLORMAP GENERATION

In Listing 2 below, `RGBToHunter` and `hunterToRGB` are functions that transform between the Hunter Lab space and the sRGB space (Section 3.2.2.1 and Section 3.2.2.2). We convert the gradient colors to Hunter, establish 256 equal steps among them, bulk transform back to RGB, and finally write to a text file.

Listing 2: The code to generate a colormap given a list of equally-spaced colors to denote the gradient

```
def makeColorMap(mapName, gradientStops):
    for i in range(len(gradientStops)):
        gradientStops[i] = RGBToHunter(gradientStops[i])
    totalCols = 256
    colsPerStop = int((totalCols - len(gradientStops)) / (
        len(gradientStops) - 1) + 1)
    gradient = np.zeros((colsPerStop * (len(gradientStops) -
         1) + 1, 3))
    for i in range(len(gradientStops) - 1):
        redRange = np.linspace(gradientStops[i, 0],
            gradientStops[i + 1, 0],
                               colsPerStop, endpoint=False)[
                                   np.newaxis, :]
        greenRange = np.linspace(gradientStops[i, 1],
            gradientStops[i + 1, 1],
                                 colsPerStop, endpoint=False
                                     )[np.newaxis, :]
        blueRange = np.linspace(gradientStops[i, 2],
            gradientStops[i + 1, 2],
                                colsPerStop, endpoint=False)
                                    [np.newaxis, :]
        # Concatenate the ranges to get a range of points
        fullRange = np.concatenate((redRange, greenRange,
            blueRange), axis=0).T
        # And add to the upper gradient array
        gradient[i * colsPerStop:(i + 1) * colsPerStop] =
            fullRange
    # Add the final color
    gradient[-1] = gradientStops[-1]
    # Convert to RGB
    finalMap = hunterToRGB(gradient)
    # Save the final map
    direct = os.path.join(os.path.dirname(os.path.realpath(
        __file__)), '..//ColorMaps')
    np.savetxt(os.path.join(direct, mapName + '.txt'),
        finalMap, fmt='%1.10f')
```

A.3 VECTOR ROTATIONS

In Listing 3 below, `vects3D` is an m x n array, where m and n are the number of latitude and longitude points we have, respectively. Each element is a triplet of values denoting the vector in Cartesian coordinates. Meanwhile, `exactVectLocs` has the same size, but holds the exact loactions of these vectors in 3D space. Additionally, `latPoints` and `lonPoints` hold arrays of our latitude and longitude points, respectively. Finally, please note that the `numpy` package is being used in this code snippet.

Listing 3: The code snippet where we rotate flat 2D vectors into 3D

```
for i, lat in enumerate(latPoints, 0):
    for j, lon in enumerate(lonPoints, 0):
        # Find the rotation matrix...refer to section.
        rotMatrix = np.dot(
            [
                [np.cos(lon), -np.sin(lon), 0],
                [np.sin(lon), np.cos(lon), 0],
                [0, 0, 1]
            ],
            # pi / 2 - lat gives you phi
            [
                [np.cos(np.pi / 2 - lat), 0, np.sin(np.pi /
                    2 - lat)],
                [0, 1, 0],
                [-np.sin(np.pi / 2 - lat), 0, np.cos(np.pi /
                     2 - lat)]
            ]
        )
        # Before we rotate, we need to convert to a
        # "top-down" view, so the +x-axis will point down
        # and the +y-axis will point right.
        # (x,y,0) ==> (-y,x,0)
        np.dot(rotMatrix, np.array([[0, -1, 0], [1, 0, 0],
            [0, 0, 1]]), out=rotMatrix)
        # Apply the entire rotation matrix
        vects3D[i, j] = np.dot(rotMatrix, vects3D[i, j])

        ### Calculate the 2d point.
        ### Basically spherical coordinates
        exactVectLocs[i, j] = radius * np.array([
            np.sin(np.pi / 2 - lat) * np.cos(lon),
            np.sin(np.pi / 2 - lat) * np.sin(lon),
            np.cos(np.pi / 2 - lat)
        ])
print('Transform complete!')
```

A.4 VECTOR INTERPOLATION

In Listing 4 below, `radius` and `resStep` hold our box radius in the Unreal space and the distance between points respectively. Meanwhile, `padWidth` is an integer describing how many units to pad our display by. At this stage, `fgaVectors` is a 3D array, where each element is a triplet of values denoting the vector at that location. The physical location of these vectors is calculated using the values described above. Finally, `windscale` is the amount we scale our resultant vectors by in the final visualization.

Note the presence of two functions `cart2sph` and `findBoxPointsAndWeights`. The former converts Cartesian coordinates into spherical. The latter finds the corresponding latitude and longitude indices in `latPoints` and `lonPoints` of the bounding box around a target vector to interpolate. Details of these two functions have been left out for brevity's sake. Please see our `ColorProduction` repository above for further details.

Listing 4: This snippet shows how we calculate the interpolated vectors between defined vectors.

```
# From our padding, value space goes from radius
# to radius. But our index space now starts at
# the padWidth, and goes the length of the value space.
valueSpace = np.arange(-radius, radius + 1e-10, resStep)
indexSpace = np.arange(padWidth, padWidth + len(valueSpace))
valueProduct = product(valueSpace, valueSpace, valueSpace)
indexProduct = product(indexSpace, indexSpace, indexSpace)
for point, indices in zip(valueProduct, indexProduct):
    x, y, z = point
    xi, yi, zi = indices
    distFromOrigin = np.linalg.norm([x,y,z])
    # If the distance from the point to
    # the origin is "close enough", then we
    # interpolate on this point. Here, "close enough"
    # means above the sphere and less than resStep away.
    if 0 <= distFromOrigin - radius < resStep:
        # Find the spherical coordinates of this point.
        # The method returns the latitude and longitude
        # in the correct ranges.
        rho, lat, lon = cart2sph(x, y, z)

        # Use numpy's searchsorted function to find
        # the closest locations for latitude and longitude.
        # searchsorted returns a right associated index.
        # The reason for the mod is that if the value is
        # off the deep end, then searchsorted returns
        # the length of the array, which is invalid index.
```

```
        # So mod the length the array to turn it 0, and
        # the left index (which should wrap around), works
30      # as intended.

        boxLocs, weightLat, weightLon =
            findBoxPointsAndWeights(lat, lon, latPoints,
            lonPoints)

        # Point 4 is assumed to be diagonally opposite
35      # Point 1, and Point 2 is assumed to be
        # horizontally opposite Point 1.
        # Point 1 is the lower left point.
        # First we need to find the two vectors directly
        # above and below (left and right also works) our
40      # target location.
        interAbove = (1-weightLon) * vects3D[boxLocs[0]] +
            weightLon * vects3D[boxLocs[1]]
        interBelow = (1-weightLon) * vects3D[boxLocs[2]] +
            weightLon * vects3D[boxLocs[3]]
        # Now we interpolate vertically
        # between these two points
45      # If something is 20% of the distance from point
        # A to point B, then it's 80% of point A and
        # 20% of point B.

        interedVec = (1-weightLat) * interBelow + weightLat
            * interAbove
50
        # We're done with the interpolation with this
        # vector, so now using the index values all
        # the way above, assign to fgaVectors...
        # scaling as necessary.
55      fgaVectors[xi, yi, zi] = interedVec * windScale
```

A.5 DOWNLOAD LOCAL DATASET SCRIPT

In Listing 5 below, the Python script downloads surface water temperature datasets from the NOAA Coast Watch East Coast Node. The `START_DAY` and `END_DAY` parameters can be adjusted to specify the desired time range to be downloaded.

Listing 5: The Python script that downloads all the desired datasets from the NOAA Database for the local visualization

```
import requests

START_DAY = 229
END_DAY = 246

DOWNLOAD_PATH = "./raw_data/"
DOWNLOAD_LINK = "https://eastcoast.coastwatch.noaa.gov/data/
    avhrr/sst/daily/cd/"
DOWNLOAD_FILE_NAME1 = "AVHCW_2017"
DOWNLOAD_FILE_NAME2 = "_DAILY_MULTISAT_SSTMASKED_CD_1KM.nc4"
SAVE_FILE_NAME = "chesapeake_temp_"

for i in range(START_DAY, END_DAY+1):
   res = requests.get(DOWNLOAD_LINK + DOWNLOAD_FILE_NAME1 +
       str(i) + DOWNLOAD_FILE_NAME2)
   with open(DOWNLOAD_PATH + SAVE_FILE_NAME + str(i) + ".nc4
       ", "wb") as fout:
      fout.write(res.content)
      print("Downloaded day #" + str(i))
```

A.6 MENU SPAWN BLUEPRINT

Figure 29 outlines the Blueprint which ensures that the menu spawns facing the user. As described in Section 3.4, we take advantage of the user's forward vector. We spawn the menu 50 units along the forward vector, then rotate the menu by 180 degrees about the Z axis to face the user.

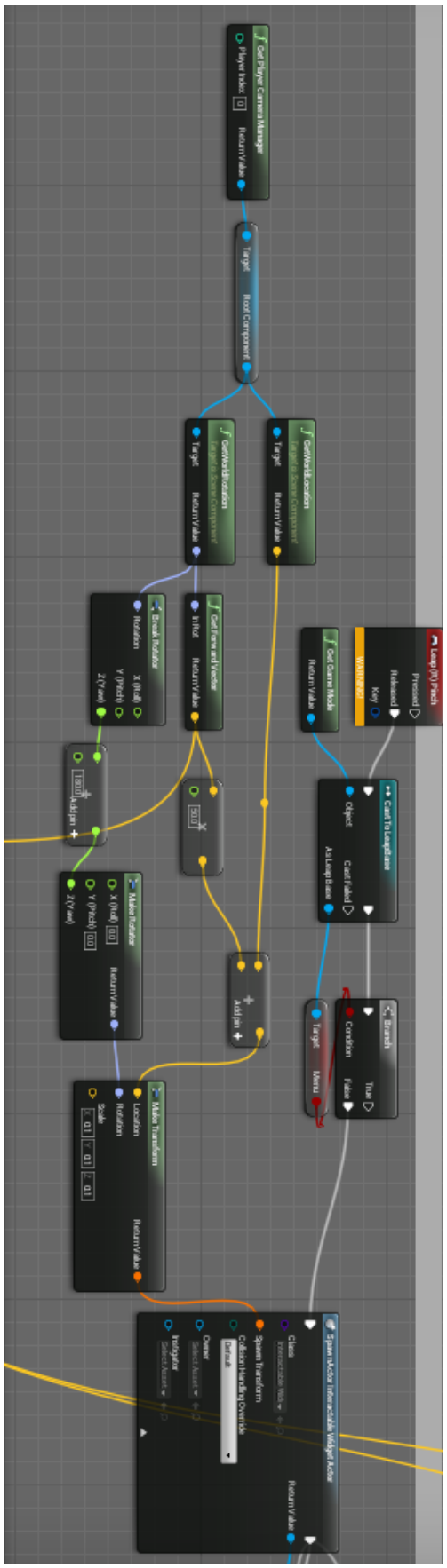


Figure 29: A screenshot of the Blueprint code which ensures the menu spawns facing the user.

B

FOCUS GROUP FORMS

B.1 CONSENT FORM

Initials: _____ *Date:* _____

Institutional Review Board
1204 Marie Mount Hall ▪ 7814 Regents Drive ▪ College Park, MD 20742 ▪ 301-405-4212 ▪ irb@umd.edu

CONSENT TO PARTICIPATE

Project Title	*Gemstone Team DIVA (Data Imaging and Visualization Analysis)*
Purpose of the Study	*This research is being conducted by **Gemstone Team DIVA** under **Dr. Stephen Penny** at the University of Maryland, College Park. We are inviting you to participate in this research project because you are interested in atmospheric science or climatology. The purpose of this research project is to gauge your response to climate data visualizations presented in an interactive, virtual reality format as opposed to traditional 2-D visualization methods.*
Procedures	*The procedures involve wearing an Oculus Rift virtual reality headset to view and interact with climate data visualizations.*
Potential Risks and Discomforts	*There may be some risks from participating in this research study. Some users have reported eye strain, headaches, or dizziness after wearing virtual reality headsets. You may stop participating at any time if you feel uncomfortable or dizzy.*
Potential Benefits	*There are no direct benefits from participating in this research. We hope that, in the future, other people might benefit from this study through improved understanding of the utility of virtual reality for use in data visualization.*
Confidentiality	*Any potential loss of confidentiality will be minimized by recording your survey or interview responses without connection to your identity. All participant identities will be stored in a document which can only be accessed by team members performing this study.* *If we write a report or article about this research project, your identity will be protected to the maximum extent possible. Your information may be shared with representatives of the University of Maryland, College Park or governmental authorities if you or someone else is in danger or if we are required to do so by law.*
Medical Treatment	*The University of Maryland does not provide any medical, hospitalization or other insurance for participants in this research study, nor will the University of Maryland provide any medical treatment or compensation for any injury sustained as a result of participation in this research study, except as required by law.*
Right to Withdraw and Questions	*Your participation in this research is completely voluntary. You may choose not to take part at all. If you decide to participate in this research, you may stop participating at any time. If you decide not*

Page 1 of 2 IRBNet Package: *XXXXXXX-XX*

Figure 30: First page of our consent form.

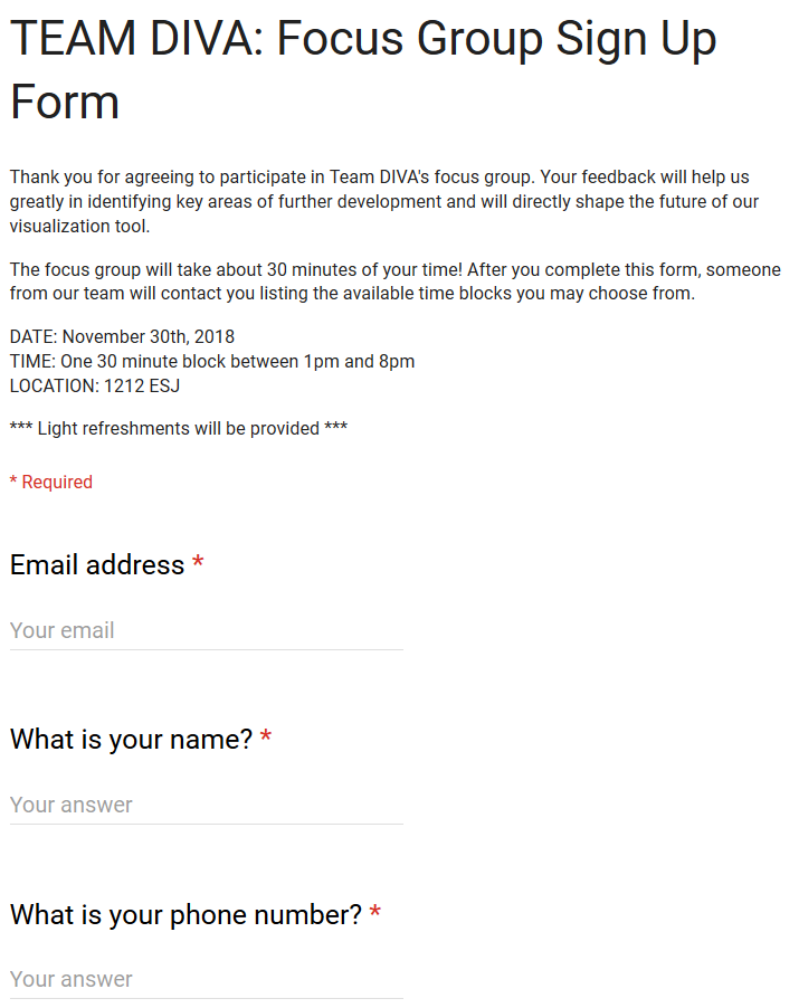
Initials: _____ *Date:* _____

	to participate in this study or if you stop participating at any time, you will not be penalized or lose any benefits to which you otherwise qualify. *If you decide to stop taking part in the study, if you have questions, concerns, or complaints, or if you need to report an injury related to the research, please contact the investigator:* ***Stephen Penny*** ***Room 2431, Atlantic Building*** ***University of Maryland, College Park*** ***pennysg@umd.edu*** ***301-405-8022***	
Participant Rights	*If you have questions about your rights as a research participant or wish to report a research-related injury, please contact:* University of Maryland College Park Institutional Review Board Office 1204 Marie Mount Hall College Park, Maryland, 20742 E-mail: irb@umd.edu Telephone: 301-405-0678 *This research has been reviewed according to the University of Maryland, College Park IRB procedures for research involving human subjects.*	
Statement of Consent	*Your signature indicates that you are at least 18 years of age; you have read this consent form or have had it read to you; your questions have been answered to your satisfaction and you voluntarily agree to participate in this research study. You will receive a copy of this signed consent form.* *If you agree to participate, please sign your name below.*	
Signature and Date	**NAME OF PARTICIPANT** **[Please Print]**	
	SIGNATURE OF PARTICIPANT	
	DATE	

IRBNet Package: *XXXXXXX-XX*

Figure 31: Second page of our consent form.

B.2 FOCUS GROUP INTEREST FORM



TEAM DIVA: Focus Group Sign Up Form

Thank you for agreeing to participate in Team DIVA's focus group. Your feedback will help us greatly in identifying key areas of further development and will directly shape the future of our visualization tool.

The focus group will take about 30 minutes of your time! After you complete this form, someone from our team will contact you listing the available time blocks you may choose from.

DATE: November 30th, 2018
TIME: One 30 minute block between 1pm and 8pm
LOCATION: 1212 ESJ

*** Light refreshments will be provided ***

* Required

Email address *

Your email

What is your name? *

Your answer

What is your phone number? *

Your answer

Figure 32: First page of our focus group sign-up form.

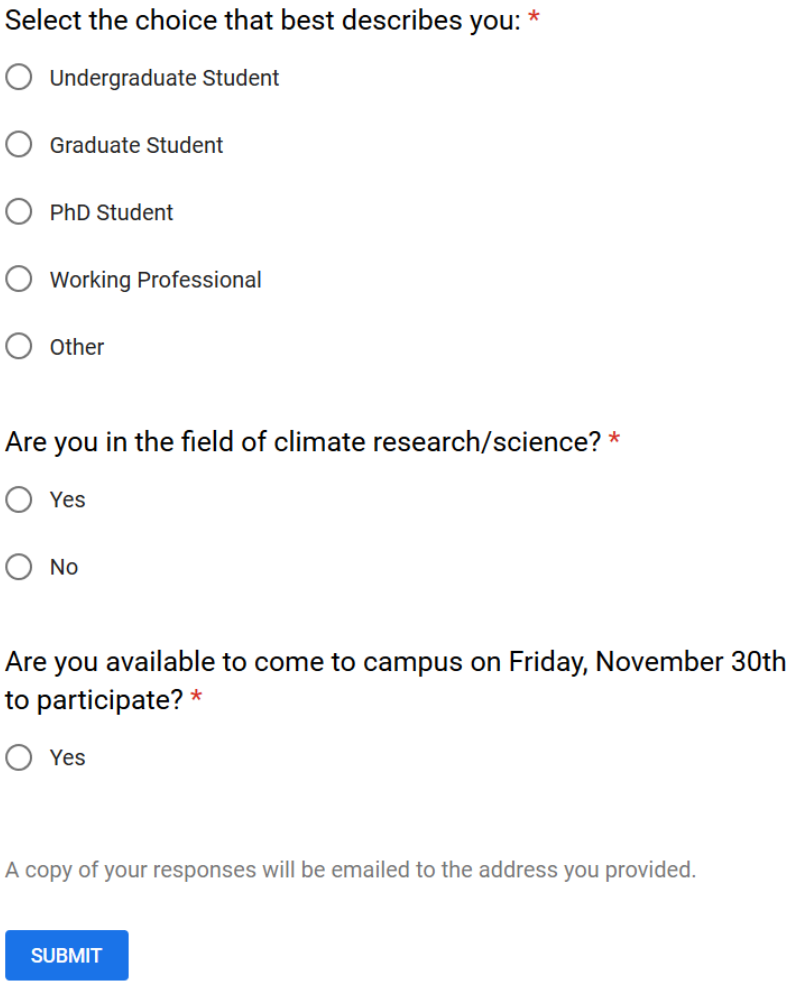


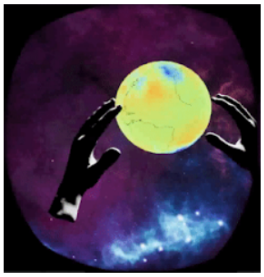
Figure 33: Second page of our focus group sign-up form.

B.3 FOCUS GROUP SURVEY FORM

Team DIVA Fall 2018 Prototype Survey

Based on your experience interacting with the prototype, please answer all questions honestly and provide any related feedback or comments. This software is still in development and your comments will allow us to determine how it can be improved. This survey is anonymous and will not be connected to your identity in any way.

* Required



How useful do you believe this visualization tool might be for your own research? *

- ◯ This tool would be useful to me right now
- ◯ High potential to be useful with further development
- ◯ Possibly useful with further development
- ◯ Not likely to be useful to me

How intuitive was the 3D visualization in comparison to traditional 2D methods? *

	1	2	3	4	5	
Less intuitive	◯	◯	◯	◯	◯	More intuitive

How easy was it to interact with the demo using the hand tracking interface? *

	1	2	3	4	5	
Difficult to use	◯	◯	◯	◯	◯	Easy to use

How would you rate your overall comfort while using the tool? Consider eye strain, motion sickness/dizziness, or other unpleasant effects *

	1	2	3	4	5	
I was very uncomfortable	◯	◯	◯	◯	◯	I felt fine

Figure 34: First page of our focus group survey.

Was it easy to interpret the data being visualized? *

	1	2	3	4	5	
It is unclear what I was seeing	○	○	○	○	○	Very intuitive

Please rate your overall experience *

	1	2	3	4	5	
Poor	○	○	○	○	○	Amazing

What types of data would you want to use this tool to visualize? *

- [ ] Model output fields such as: Temperature, Wind, Snowfall, Humidity
- [ ] Relationships between two variables
- [ ] Relationships among more than two variables
- [ ] Raw observations of geophysical quanitites
- [ ] Time series data
- [ ] Empirical Orthogonal Functions (EOFs) and teleconnections
- [ ] Combined data from atmosphere, ocean, land, etc.
- [ ] Other:

What features are most important to you in an Earth science visualization tool? (Pick your top 3) *

- [ ] Ease of interaction
- [ ] Ability to view multiple datasets at once for comparison
- [ ] Flexible color palette options
- [ ] Analysis tools
- [ ] Movie replay capabilities to visualize 4-dimensional dynamics
- [ ] Ability to render high resolution visualization in specific areas
- [ ] Ability to render lower resolution data displaying global trends
- [ ] Ability to output publishable graphics
- [ ] Speed of rendering
- [ ] Speed of accessing and loading datasets
- [ ] Ability to manipulate data and visualization on the fly
- [ ] Other:

Figure 35: Second page of our focus group survey.

What was your favorite aspect of the demo? *

Your answer

What was your least favorite aspect? *

Your answer

What other types of datasets do you think would benefit from a 3D visualization and analysis of this kind? *

Your answer

How could this tool be improved to be of more use in your own research? What features would you like to see added? *

Your answer

Do you currently use any software tools to analyze Earth science data for analysis, visualization, or for generating publication-quality plots? What are your favorite and least favorite tools and why? *

Your answer

Additional Comments

Your answer

SUBMIT

Figure 36: Third page of our focus group survey.

B.4 FOCUS GROUP GRAPHS

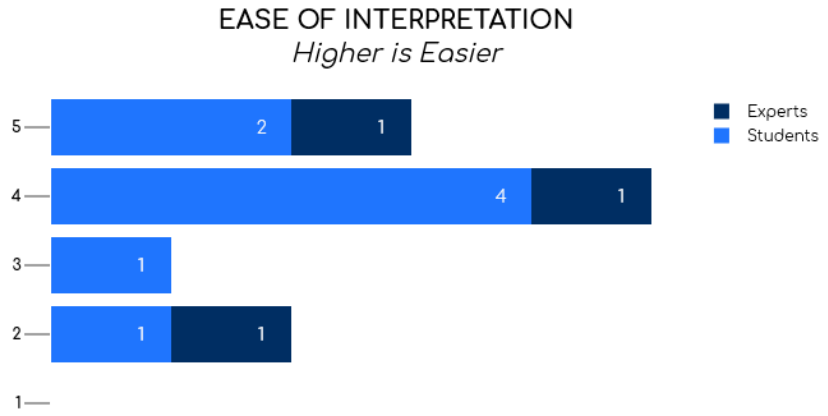


Figure 37: Ratings of how easy it was for our participants to interpret the data being displayed. Higher ratings indicate easier levels of interpretability.

Figure 38: A list of some of the data types our participants were interested in displaying using our product. Multiple selections were allowed.

BIBLIOGRAPHY

Alder, J., S. Hostetler, and D. Williams, 2013: An interactive web application for visualizing climate data. *Eos, Trans. Amer. Geophys. Union*, **94**, 197–198, doi:10.1002/2013e0220001.

Allen, M., and Coauthors, 2018: Summary for Policymakers. *Global warming of 1.5°C. An IPCC Special Report on the impacts of global warming of 1.5°C above pre-industrial levels and related global greenhouse gas emission pathways, in the context of strengthening the global response to the threat of climate change, sustainable development, and efforts to eradicate poverty*, V. Masson-Delmotte, P. Zhai, H.-O. Pörtner, D. Roberts, J. Skea, P. R. Shukla, A. Pirani, W. Moufouma-Okia, C. Péan, R. Pidcock, S. Connors, J. B. R. Matthews, Y. Chen, X. Zhou, M. I. Gomis, E. Lonnoy, T. Maycock, M. Tignor, and T. Waterfield, Eds., IPCC, 3–26.

Bellini, H., W. Chen, M. Sugiyama, M. Shin, S. Alam, and D. Takayama, 2016: Virtual & and Augmented Reality: Understanding the Race for the Next Computing Platform. Profiles in Innovation Tech. Rep., The Goldman Sachs Group, Inc., 64 pp. [Excerpt available online at https://www.goldmansachs.com/insights/pages/technology-driving-innovation-folder/virtual-and-augmented-reality/report.pdf.].

Desai, P. R., P. N. Desai, K. D. Ajmera, and K. Mehta, 2014: A Review Paper on Oculus Rift-A Virtual Reality Headset. *International Journal of Engineering Trends and Technology*, **13**, 175–179, doi:10.14445/22315381/IJETT-V13P237.

Du, Z., L. Fang, Y. Bai, F. Zhang, and R. Liu, 2015: Spatio-temporal visualization of air–sea CO2 flux and carbon budget using volume rendering. *Computers & Geosciences*, **77**, 77–86, doi:10.1016/j.cageo.2015.01.004.

Elliot, A. J., and M. A. Maier, 2014: Color Psychology: Effects of Perceiving Color on Psychological Functioning in Humans. *Annual Review of Psychology*, **65 (1)**, 95–120, doi:10.1146/annurev-psych-010213-115035.

Epic Games, 2018: Get Actor Forward Vector. Epic Games. [Available online at http://api.unrealengine.com/INT/BlueprintAPI/Utilities/Transformation/GetActorForwardVector/].

Filonik, D., and D. Baur, 2009: Measuring Aesthetics for Information Visualization. *2009 13th Int. Conf. Information Visualization*, Barcelona, Spain, Institute of Electrical and Electronics Engineers, 579–584, doi:10.1109/IV.2009.94.

Grinstein, G., and H. Levkowitz, 2013: *Perceptual Issues in Visualization*. Springer Science & Business Media, 165 pp.

Herring, J., M. S. VanDyke, R. G. Cummins, and F. Melton, 2017: Communicating local climate risks online through an interactive data visualization. *Environmental Communication*, **11 (1)**, 90–105, doi:10.1080/17524032.2016.1176946.

Hunt, C., 2016: How to use Leap Motion with your Oculus Rift. Windows Central, [Available online at https://www.windowscentral.com/how-use-leap-motion-your-oculus-rift].

Idreos, S., O. Papaemmanouil, and S. Chaudhuri, 2015: Overview of Data Exploration Techniques. *Proc. of the 2015 ACM SIGMOD Int. Conf. on Management of Data*, Melbourne, Victoria, Australia, ACM, 277–281, doi:10.1145/2723372.2731084.

IEC, 1999: Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB. IEC Stan. 61966-2-1:1999, IEC, 51 pp.

Kaniewski, J., 2018: LeapUnreal Plugin v2.17.1 for UE4.19. Epic Games. [Available online at https://github.com/leapmotion/LeapUnreal/releases/tag/v2.17.1].

Keim, D. A., 2002: Information visualization and visual data mining. *IEEE Trans. on Visualization and Computer Graphics*, **8 (1)**, 1–8, doi:10.1109/2945.981847.

Kostelnick, C., 2016: The Re-Emergence of Emotional Appeals in Interactive Data Visualization. *Technical Communication*, **63 (2)**, 116–135.

LaValle, S. M., A. Yershova, M. Katsev, and M. Antonov, 2014: Head tracking for the Oculus Rift. *2014 IEEE Int. Conf. on Robotics and Automation*, Hong Kong, China, Institute of Electrical and Electronics Engineers, 187–194, doi:10.1109/ICRA.2014.6906608.

Liu, P., J. Gong, and M. Yu, 2015: Visualizing and analyzing dynamic meteorological data with virtual globes: A case study

of tropical cyclones. *Environmental Modelling & Software*, **64**, 80–93, doi:10.1016/j.envsoft.2014.11.014.

MacLean, K., and M. Enriquez, 2003: Perceptual Design of Haptic Icons. *EuroHaptics 2003*, Dublin, UK, F. Newell and S. O'Modhrain and I. Oakley, 351–363.

Mathur, A. S., 2015: Low cost virtual reality for medical training. *2015 IEEE Virtual Reality*, Institute of Electrical and Electronics Engineers, 345–346, doi:10.1109/vr.2015.7223437.

Norton, A., and J. Clyne, 2012: The VAPOR Visualization Application. *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*, E. W. Bethel, H. Childs, and C. Hansen, Eds., Chapman and Hall/CRC, 415–428, doi:10.1201/b12985-25.

Oculus, cited 2019: Developers. [Available online at https://developer.oculus.com/].

Optical Society of America, Inc., 1948: Minutes of the Thirty-First Meeting of the Board of Directors of the Optical Society of America, Incorporated. *JOSA*, **38**, 651, doi:10.1364/JOSA.38.000651.

Potter, K., A. Wilson, P.-T. Bremer, D. Williams, C. Doutriaux, V. Pascucci, and C. Johhson, 2009: Visualization of uncertainty and ensemble data: Exploration of climate modeling and weather forecast data with integrated ViSUS-CDAT systems. *Journal of Physics: Conference Series*, **180**, 012 089, doi:10.1088/1742-6596/180/1/012089.

QuickTutorial, cited 2018: Tristimulus Value of Color: Device Independent Color Representation. [Available online at https://www.youtube.com/watch?v=d74gJxUdsq8].

Ray, A., 2015: Reflections on the state of developing virtual environments. *Int. J. of Virtual Reality*, **15 (2)**, 18–29.

Saha, S., S. Moorthi, H. Pan, X. Wu, J. Wang, and Coauthors, 2010: The ncep climate forecast system reanalysis. *Bulletin of the American Meteorological Society*, **91**, 1015–1057, doi:doi:10.1175/2010BAMS3001.1.

Smith, T., and J. Guild, 1931: The C.I.E colorimetric standards and their use. *Trans. of the Optical Soc.*, **33**, 73–134, doi:10.1088/1475-4878/33/3/301.

Sun, X., S. Shen, G. G. Leptoukh, P. Wang, L. Di, and M. Lu, 2012: Development of a Web-based visualization platform for

climate research using Google Earth. *Computers & Geosciences*, **47**, 160–168, doi:10.1016/j.cageo.2011.09.010.

Teuling, A. J., R. Stöckli, and S. I. Seneviratne, 2011: Bivariate colour maps for visualizing climate data. *Int. J. Climatol.*, **31 (9)**, 1408–1412, doi:10.1002/joc.2153.

Wickham, H., H. Hofmann, C. Wickham, and Cook, 2012: Glyph-maps for visually exploring temporal patterns in climate data and models. *Environmetrics*, **23 (5)**, 382–393, doi:10.1002/env.2152.

Winoto, P., C. N. Xu, and A. A. Zhu, 2016: "Look to Remove": A Virtual Reality Application on Word Learning for Chinese Children with Autism. *Universal Access in Human-Computer Interaction. Users and Context Diversity*, M. Antona, and C. Stephanidis, Eds., Lecture Notes in Computer Science, Vol. 9739, Springer, 257–264.

Zhang, T., J. Li, Q. Liu, and Q. Huang, 2016: A cloud-enabled remote visualization tool for time-varying climate data analytics. *Environmental Modelling & Software*, **75**, 513–518, doi:10.1016/j.envsoft.2015.10.033.