Real-Time Map Data Visualizer

FINAL REPORT

Team 21
Dr. Joseph Zambreno
Dr. Joseph Zambreno
Scott Frank, Ami Ikanovic, Isaac Littler, Benjamin Kelly, Zahydee Machado, Elizabeth Nelson, Parth Padmanabhan
sdmay22-21@iastate.edu
https://sdmay22-21.sd.ece.iastate.edu/index.html

Executive Summary

Development Standards & Practices Used

- Microsoft HoloLens 2 Development Standards
- Unity Engine Development Standards
- IEEE 802.11 Standard for Information Technology - Telecommunications and Information Exchange between systems

Summary of Requirements

- Scale and/or rotate the 3D globe at the user's command
- Globe remains stationary in the room independent from the user's location
- 6'x6' unobstructed space (constraint)
- Use of Microsoft's HoloLens 2 (constraint)
- Own API layer to standardize incoming data streams
- Built using Unity Engine
- Accepts three (3) different types of geographical data streams as input
- Maintains stable thirty (30) fps (constraint) *
- One (1) hour of use on the device's onboard battery
- "Plug & Play"/"Pick up and go" ease of usability
- Maintains 720p resolution

Applicable Courses from Iowa State University Curriculum

- COM S 227: Object-Oriented Programming
- COM S 228: Introduction to Data Structures
- COM S 309: Software Development Practices
- COM S 336: Introduction to Computer Graphics
- COM S 437: Computer Game and Media Programming
- SE 317: Introduction to Software Testing
- SE 319: Construction of User Interfaces
- SE 329: Software Project Management
- SE 339: Software Architecture and Design

New Skills/Knowledge acquired not taught in courses

- Flask
- GraphQL
- Unity Game Engine
- Mixed Reality Toolkit
- Augmented reality (AR) development

Table of Contents

List of Figures/Tables/Symbols/Definitions

x.1 Figures

x.2 Tables

x.3 Definitions

Unity - Game development engine, used to create 3D visualizations for the Microsoft HoloLens 2

Flask - Python web framework, used to implement our backend service for data processing

GraphQl - Alternative API architecture to REST, used for our internal data calls from backend to Unity client

Mixed Reality Toolkit - Cross platform toolkit for building AR experience, providing us with components and features to accelerate our development process (AR functionality)

1 Introduction

1.1 Problem Statement

Geographical data is often difficult to interpret and visualize. It can be challenging to portray data in a manner that conveys a sense of global scale to users. Traditional means of displaying large-scale geographical trends exist through mediums that don't promote intuitive comparisons between datasets, presenting a significant barrier for "big picture" analysis.

1.2 Requirements & Constraints

- The application will be able to scale the three-dimensional globe to the user's liking
- The application will be able to rotate the globe to give users a full 360 degree view of all geographic data represented
- The globe will be able remain stationary in the "world" independent from user's location
- This application will require a 6' x 6' unobstructed space (empty 6 x 6 space) (Constraint)
- This application will require the use of a Microsoft HoloLens 2 (Constraint)
- This application will have its own API layer that standardizes incoming data streams (Compatibility layer)
- This application will use the implementation of real time data as well as historical data
- This application will be able to take 3 different types of geographical data streams as input
- This application will be built using the Unity engine
- This application will maintain a stable 30 frames per second (fps) through the HoloLens platform (Constraint) *
- This application will will be able to run for an hour on a single charge of the second generation HoloLens' battery
- This application will have a "pick up and go" ease of use where no technical developer is needed for a user to understand how to run the application
- This application will create a 720p resolution image as specified in HoloLens 2 standards
- The project implementation needs to be completed within 14 weeks (Time constraint)
- This application will need to run smoothly while utilizing the limited HoloLens 2 hardware (Constraint)
- This application will need to run using the Unity engine under the constraint of the system requirements needed for Unity (Constraint)

1.3 Engineering Standards

- Microsoft HoloLens Development Standards
- To meet a threshold of usability and performance for the user
- Unity Engine Development standards
- To ensure ease of development
- IEEE 802.11 Standard for Information Technology - Telecommunications and Information Exchange between systems
- To maintain reliable and stable connection between hardware devices and software application

1.4 Intended Users and Uses

- Benefits and Stakeholders
    - General Users will be able to see a visualization of geographical data and experience a practical application of augmented reality
    - The client will be able to check weather patterns, import custom streams of historical and real time data, and showcase the ability of senior engineering students to faculty
    - Developers will be able to showcase their abilities and develop a fully usable and finished product
- Use Case
    - User will be able to see weather data visualized on a three-dimensional globe
    - Users will be able to scale the size of the three-dimensional globe
    - Users will be able to rotate the globe to see all displayed geographical data
    - Users will be able to move around the space while the globe remains stationary in relation to the space
    - Users will be able to choose, from available data streams, what visualizations appear on the globe
    - Users will be able to select and view historical data visualizations
    - Users will be able to pick up the HoloLens, put it on, and begin to see the application without further technical knowledge
- Benefits visitors and prospective students of Iowa State by showcasing the potential skills they may learn
- Benefits any user wishing to visualize data streams using AR
- Client
    - Check weather patterns
    - Import custom streams of historical and real time data
    - Experience practical applications of augmented reality

2 Planned Design

2.1 Design Context

2.1.1 Broader Context

This 3D data visualization application is being designed as a showcase piece for the Department of Electrical and Computer Engineering at Iowa State. It will provide an educational experience with one of the newest trends in visualization technology and hardware in order to highlight the university's commitment to research and innovation. The application is intended to attract and communicate to prospective students on what their future at Iowa State can look like.

Area	**Description**	**Examples**
Public health, safety, and welfare	This project allows various users to learn and become aware of the circumstances and challenges that populations around the globe are facing.	Depending upon the data called by the program, we can alert users to dangerous weather patterns, virus hotspots, or environmental pollutants.
Global, cultural, and social	The project will reflect the goals and values of the Iowa State University community. It will aim to showcase the potential of the education provided at the Department of Electrical and Computer Engineering and their research goals.	It will showcase the commitment to innovation and research of the ECpE by incorporating new technologies and ideas.
Environmental	Due to the small scale and software-based nature of the project, the environmental impact is minimal and limited to that of manufacturing the single unit HoloLens 2 that will be used to run the project.	This project will require a manufactured HoloLens 2 device to run and energy to recharge it periodically.
Economic	This project might factor into prospective students' decisions about whether to pursue their studies at Iowa State. This could potentially influence the university's enrollment numbers and therefore have an economic impact on it.	Product creates opportunities for the economic advancement of Iowa State University by targeting prospective students.

Table 1 - Design Context

2.1.2 User Needs

- Prospective students need a way to see what sort of projects and education lies ahead if they were to come to Iowa State University and enroll in Electrical / Computer Engineering
- EE / CPRE faculty need a way to showcase their department's achievements and innovative practices to prospective students as well as visitors interested in the department
- Visitors need a way to physically see examples of what goes on in the EE / CPRE department because depending on who they are they could want to create business or educational connections with the department
- Current students need a way to learn about ongoing development of projects to make decisions on research, classes, and areas of study they want to pursue.

2.1.3 Prior Work/Solutions

The majority of visualization systems of global scale are 2D and only compatible with traditional computer monitors or touch screen displays. Those visualizers generally are limited to displaying one type of data such as weather, population, pollution, to serve a specific purpose. There are some 3D visualizers, such as Google Earth, that focus on displaying details of the earth and its geography. The application we found most similar to our application goal is MeteoEarth. This application displays a 3D globe on a 2D display with various weather data types. Through our research we were unable to find a HoloLens application that will do the same real time data visualization that we intend to do.

The following is a Pros/Cons list describing how our target solution will stand against others.

Pros

- "Real" 3D as our solution will use the Microsoft HoloLens 2. This will allow users to see the globe as it is in real life and see geographical phenomena on a proper and global scale.
- Extensive user interactivity involving the ability to move the hololens (display) around a stationary globe
- Scalable globe
- The ability to input any data type following our API standard to visualize on the globe
- Global scale and localized scale of visualization in the same application

Cons

- Using the HoloLens 2 as our primary display creates an expensive initial cost for a user
- The use of the HoloLens 2 only allows one user at a time to see the visualization if a group doesn't have multiple headsets

2.1.4 Technical Complexity

The architecture of the 3D data visualization application has multiple subsystems that connect to create the cohesive whole. Each of these subsystems will involve different scientific, mathematical and engineering principles as part of the development solution.

Backend: Data Collection

- This component will require an understanding on efficient manipulation of large datasets to ensure that an acceptable level of system performance is maintained. It will also require investigation to find reputable data sources that meet the project criteria.

Backend: Custom API Layer

- Because the function of this subsystem is to standardize incoming data into a single source and format, it will require a strong understanding and analysis of scientific weather data and weather patterns. The creation of a custom API layer that accepts the importation of external data will require a lot of efficient and intuitive API design principles

Unity: Application Core Functionality

- This subsystem is a large percentage of the overarching system. The unity application needs to take geological data (coordinate-based data) and correlate that data to a 3d model of the globe. This will require mathematical principles in the field of geometry. It will also require an understanding of computer graphics for converting the individual data points into a cohesive visualization. We will need to create visual meshes that adhere to current standards in weather data visualizations, but also represent it in a 3D manner.

Unity: Application UI/UX

- This subsystem handles the interaction between the user and the program. Good UI/UX design will be required to make the program viable for our intended user base. This will need to involve principles in the areas of accessibility and usability.

2.2 Design Exploration

2.2.1 Design Decisions

1. Display as a HoloLens 2 AR application
2. Use the Unity Game Engine for development
3. Use of OpenWeather API

2.2.2 Ideation

When choosing our platform, we considered the following options:

- VR (Virtual Reality Interface)
- AR (Augmented Reality Interface)
- Sphere Display (Gakken WorldEye)
- 3D Monitor / Projection
- Touch Screen Device/Computer Monitor

We individually did research for APIs, display hardwares, and design softwares. We compiled all the information we found and discussed them. This allowed us all to share our own ideas and learn about more opportunities available to us.

2.2.3 Decision-Making and Trade-Off

We discussed all of our options and listed out our pros and cons for each. We then narrowed our options down to three and presented them to our client. Based on his recommendations, we chose an AR interface: the Microsoft HoloLens 2.

VR ● Pros ○ No physical bounds for visualization ○ Within given budget ● Cons ○ Limited to one user at a time ○ Software heavy/does not take advantage of other skill sets ○ Not providing a physical device **AR (Augmented Reality Interface)** ● Pros ○ No physical bounds for visualization ○ Easier access for the everyday man ● Cons ○ Not as innovative as other options ○ Restricted to device resolution and specifications ○ Software heavy/does not take advantage of other skill sets **Sphere Display (Gakken WorldEye)** ● Pros ○ Unique displays allows for global viewing ○ Includes interesting hardware ● Cons ○ Low Resolution ○ Details/Data hard to display ○ Out of budget	**3D Monitor / Projection** ● Pros ○ Large scale viewing options ○ Multi-user interactable ○ No glasses/headset required ● Cons ○ Limited viewing angles w/ some equipment ○ Not as innovative as other options ○ Software heavy/does not take advantage of other skill sets ○ Out of budget (on the vast majority of solutions) **Touch Screen Device/Computer Monitor** ● Pros ○ Easier than most solutions ○ Within given budget ● Cons ○ Most display are not convex ○ Similar to Google Maps or MeteoEarth ○ Software heavy/does not take advantage of other skill sets

Table 2 - Decision Making

2.3 Proposed Design

Currently, the team is trying to implement a Unity project and use the Microsoft HoloLens emulator. We have been researching how to make calls to our chosen API's. Previously, we got access to a HoloLens device and verified that it meets the expectations for the display. So far, our research and trials have indicated that our design decisions are viable and a good fit for the project.

2.3.1 Design Visual and Description

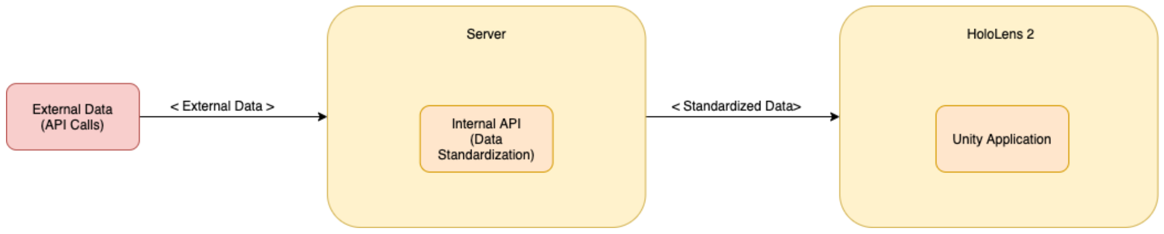


Figure 1 - Flow of Data Diagram

1. External data comes into the server through API calls
2. The server organizes and standardizes the data to our internal API specifications
3. The unity application calls our internal API and receives the standardized data
4. The unity applications translates the data into a 3D visualization
5. The HoloLens 2 displays the visualization for the user

TIMELINE OF EVENTS DIAGRAM (Swim lane)

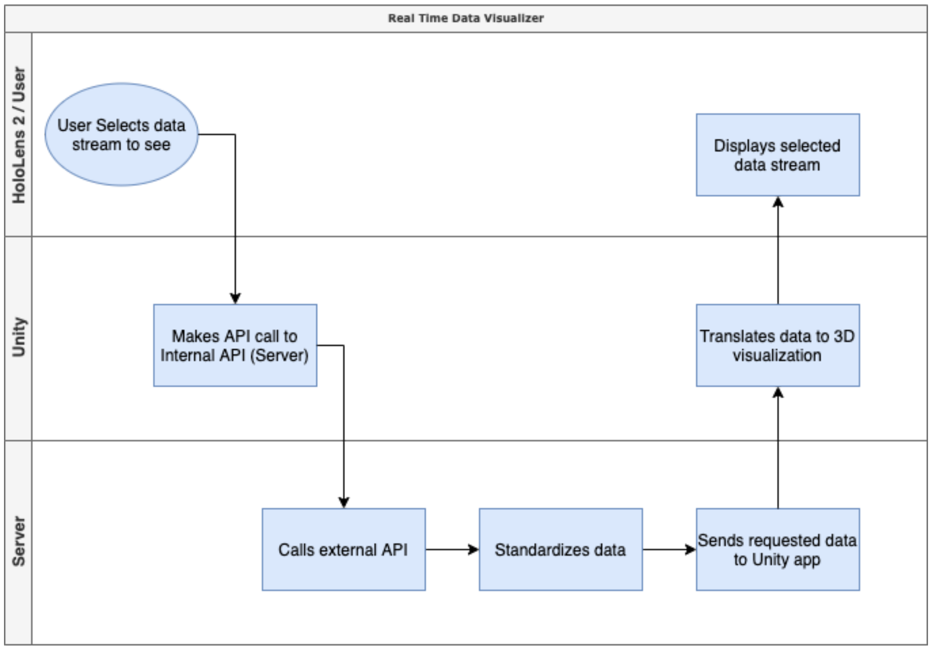

Real Time Data Visualizer
HoloLens 2 / User
Unity
Server
User Selects data stream to see
Displays selected data stream
Makes API call to Internal API (Server)
Translates data to 3D visualization
Calls external API
Standardizes data
Sends requested data to Unity app

Figure 2 - Swimlane Diagram

2.3.2 Functionality

Expected usage sequence:

1. Users are instructed on the project's background, purpose, and what they are about to see and interact with by the person supervising the demo.
2. Users are walked through putting on the HoloLens 2 device and securing it to their head with any necessary adjustments.
3. The program is started by the users and they are left to experiment with it with the supervising person available for any questions or difficulties.

The current design would satisfy the requirements by meeting the client's and potential user's needs.

2.3.3 Areas of Concern and Development

The primary concerns are making sure the product adequately manages to show the intended data in a way that is easy for users to understand and navigate without frustration. This is meant as a demonstration piece for the visualization hardware and techniques, so the quality of the user experience is fundamental to its purpose.

The immediate plans for addressing those concerns are to rely on well-researched UX and UI techniques, do early functional testing with people in the user group, and prioritize performance for a smooth viewing experience.

Questions for the client and faculty advisor include:

1. Approximately how long should it take a user to try out every viewing mode in the application?
2. How much time is a user intended to spend interacting with the application?
3. What is the main takeaway a user should have from the experience?

2.4 Technology Considerations

After deciding to use AR technology, we had to choose between the HoloLens and the HoloLens 2

HoloLens:

1. Already owned by the university (no purchase required, stays within budget)

HoloLens 2:

1. Greater field of view compared to the HoloLens (52° vs 34°)
2. Higher display resolution compared to the HoloLens (2048 x 1080 px vs 1280 x 720 px)
3. Had to be purchased for the purpose of this project (price point beyond budget)

2.5 Design Analysis

Currently, we are developing a prototype consisting of a static 3D visualization of the globe which can be manipulated within the HoloLens 2. We have been successful in creating the globe's sphere and altering its position and size. However, we are still working on grounding the sphere in our reality in some way so it will have a fixed position until the user moves it. The basic premise of our design seems to be highly feasible and leaves room for innovation and creativity.

2.6 Design Plan

Our design process will follow the task schedule outlined above, with the project requirements acting as a guide for design choices (both large and small). Certain design elements will be refined through iterative methods, allowing us to actively gauge their effectiveness.

3 Design Changes

From our proposed design, we had to make a few changes to the scope of the project as we ran into issues during development. We had to implement multithreading into the backend server to allow for multiple data requests in parallel. Caching was also added server-side to help solve the optimization problem. This allows us to store data for up to 10 minutes, being replaced with "newer" data at that time. The HoloLens 2 is rated for 60fps with a fully optimized application. We have been struggling with maintaining that full functionality so we decided to make the trade-off to allow for more on-screen rendering at the cost of fps. In the original design, the backend server would be running on the HoloLens itself, allowing for an "all in one" type application. However, we developed the backend server in Python which isn't compatible with the HoloLens' hardware. ISU's virtual machines provided a suitable replacement option for us though.

3.1 Multi-threading

A major constraint or issue we were facing was the slow rate at which we were getting data from OpenWeather API. When we are trying to get thousands of geolocations and weather values at those locations at one time it takes an unreasonable amount of time to complete these requests. To solve this problem we added multi-threading to the server to make up to 10 requests in parallel. By doing so we could reduce the time to return all requests drastically.

3.2 Caching

Caching was introduced to the server side in the middle of the project to help solve an optimization problem. When making requests by coordinates to OpenWeatherAPI, the service returns all the data it has stored at that location. This includes all the types of data we want to visualize. Well when the user would theoretically switch from one datastream to another we do not want to make all those requests again. By caching all the data returned from calls we can now simply just enter our cache to grab the data we need at this time. This cache is customized to save data for only 10 minutes allowing "new" data to come in at those locations after 10 minutes.

3.3 60 to 30 fps

With the computational constraints of the HoloLens, we chose a framerate target of 30 frames per second. While the headset is capable of running applications at 60 fps, we made the tradeoff of rendering more on-screen items at a lower frame rate (particularly due to the fact that our application doesn't include fast-moving elements).

3.4 ISU virtual machine over a locally-run server

Our backend server can be deployed on a number of platforms. We discussed the option of running the server on a device that externally accompanies the headset (such as a Raspberry Pi). Due to the flexibility and "as needed" operation of virtualized solutions, we chose to run our server on an ISU virtual machine.

3.5 No longer overlaying data for cleaner viewing

Originally, one of our design goals was to allow a user to visualize multiple data streams on the globe at one time. This would allow for easy comparison between the types of data. However, with the ability to add multiple different sources of data, some with overlapping information, the globe would be very busy for the implementation currently in place. We decided to only allow for one data stream to be displayed at a time, but make the ability to switch between data convenient.

3.6 WHO Data Added

One of the goals of our client was to support a variety of types of data inputs. We wanted to use a non real time dataset to showcase the capabilities and potential applications of our platform. The World Health Organizations provides a very large amount of historic data for its member countries, and shows a little bit of what our application can do.

4 Testing

4.1 Unit Testing

Unity Application

Tool: Unity Test Framework

- UI Tests
    - Globe Scaling
        - Pass: Gesture enables and disables zooming mode
        - Fail: Zooming mode can't be enabled or disabled via the UI
    - Globe Rotation
        - Pass: Gesture enables and disables rotation mode
        - Fail: Rotation mode can't be enabled or disabled via the UI
    - Data Overlays
        - Pass: Selecting and deselecting data streams toggles their visibility on the globe
        - Fail: Data streams can't be toggled on and off
    - Application Exit
        - Pass: Pressing on the exit button closes the application
        - Fail: Application can't be exited gracefully from the UI
- Visualization Logic Tests
    - Data Refreshing
        - Pass: Given new data, old overlays are discarded and replaced with newly generated ones
        - Fail: Old overlays are not properly deleted or are not replaced with a correct version
    - Overlay Generation
        - Pass: For each data set, an overlay texture is generated for the globe
        - Fail: Overlays are malformed or missing for any data set
    - Multiple Overlays
        - Pass: Multiple overlays (up to 3) can be overlapped without interfering with each other
        - Fail: Additional overlays obstruct or cover existing ones
- Server Interaction
    - Data Retrieval
        - Pass: Application can make appropriate requests to the server for the data needed by the overlays enabled through the UI
        - Fail: Requests are malformed, unnecessary, or retrieving the wrong data

Data Server

Tool: Postman / Pytest

- External Data Retrieval
    - Data fetching
        - Pass: Application can make appropriate requests to external APIs for the data requested by the Unity client
        - Fail: Requests are malformed, unnecessary, or retrieving the wrong data
    - Query Building
        - Pass: Queries are built with the correct parameters to fetch the necessary data
        - Fail: Queries are missing filters or other necessary information to make the correct request
- Data processing
    - API layer
        - Pass: External data is mutated to fit the visualization client's standard and format
        - Fail: Data is lost or malformed in the mutation

4.2 Interface Testing

Server:

- External API Interface
    - This will be validated and verified using Unit Tests
    - This will be functionally tested using a tool like Postman to make sure requests are compatible with the used external APIs
- Internal API Interface
    - Unit tests will cover the individual API operations
    - Functional tests will be made using a tool like Postman to ensure that the server responds as expected to API calls

Unity Application:

- Server API interface
    - Unit tests will verify that the requests meet the specifications and requirements
    - Functional tests will be made using a tool like Postman to verify that requests are compatible with the server API specification

4.3 Integration Testing

We will test the external API -> HoloLens -> Unity engine path to ensure that our implementation doesn't overwhelm the hardware resources of the HoloLens. The HoloLens framerate, processor/memory/network utilization, and other performance metrics will be viewed and inspected through the Windows Device Portal and the Windows Performance Analyzer. These tools will allow us to ensure that our performance requirements (particularly, those pertaining to framerate and smoothness) are met.

An integration test for the Unity Application will involve verifying that given a mock data stream, the unity application will be able to generate the visualization and respond to various types of UI interactions. This test can be written using the Unity Test Framework in Play Mode.

An integration test for the server will involve verifying that for a request made to its API, it is able to generate the appropriate queries to return a mock stream of external data, mutate it to the appropriate standard, and return the new formatted data to the client.

4.4 System Testing

Functional Requirements :

- The application will be able to scale the three-dimensional globe to the user's liking
    - The Unity UI unit tests and the Unity application integration test will verify this requirement is met.
- The application will be able to rotate the globe to give users a full 360 degree view of all geographical based data
    - The Unity UI unit tests and the Unity application integration test will verify this requirement is met.
- This application will have its own API layer that standardizes incoming data streams (Compatibility layer)
    - The Server unit tests and integration test will verify this requirement is met.
- This application will be able to take 3 different types of geographical data streams as input
    - The Unity UI unit tests and the Unity application integration test will verify this requirement is met.

4.5 Regression Testing

Unit tests and integration tests will automatically run on a test stage of our CI/CD pipeline, so that new additions are proven to work along existing functionality. This will be done with the Gitlab CI/CD pipelines tool and its merged result functionality which will run as if the changes from the source branch have already been merged into the target branch. Merge requests that fail these tests will not be merged into the project. The unit tests and integration tests will be made in accordance with the requirements, so this approach will verify that overall project requirements are still being met with every addition. Additions that don't include any necessary tests for added functionality will also not be merged.

4.6 Acceptance Testing

Extensive manual testing with the HoloLens device will be done by team members and volunteer beta testers to verify that non-functional requirements are being met. The requirements that will be focused on during this testing stage are:

- This application will have a "pick up and go" ease of use where no technical developer is needed for a user to understand how to run the application
- This application will maintain a stable 60 fps through the HoloLens visualization (Constraint)
- This application will need to run smoothly utilizing the limited HoloLens hardware performance (Constraint)

The client will also be asked to interact with the program and give feedback about each feature at different stages of early development and prototyping.

4.8 Results

The tests ensured that our project implementation complied with the project requirements and met the client's needs.

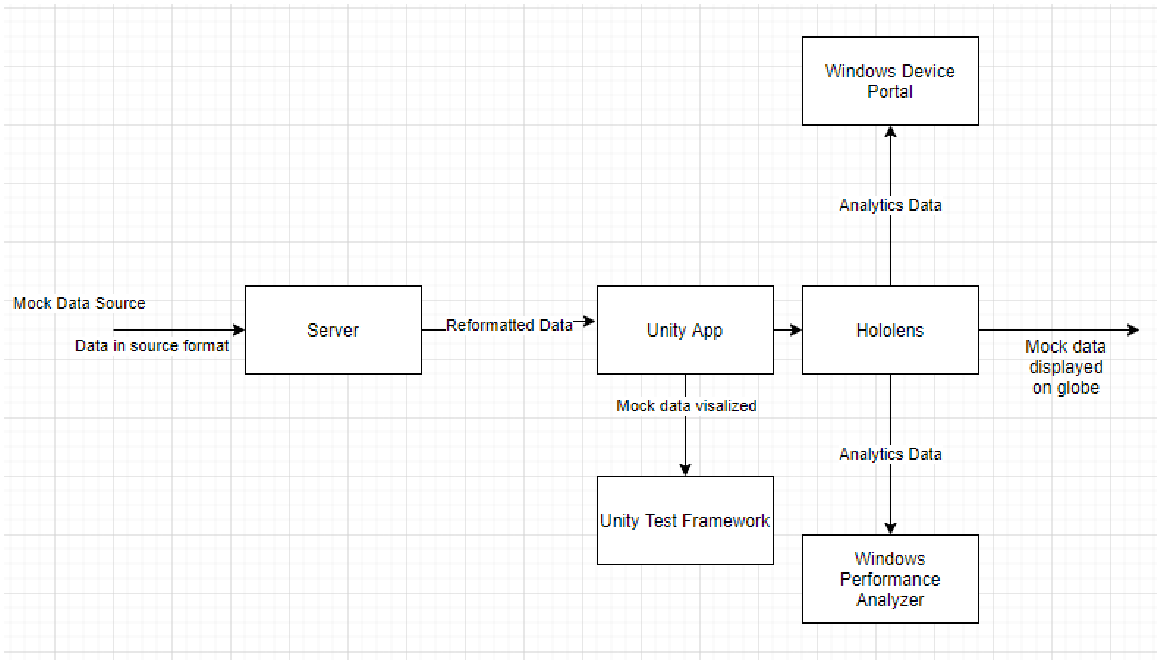


Figure 3 - Test Flow Diagram

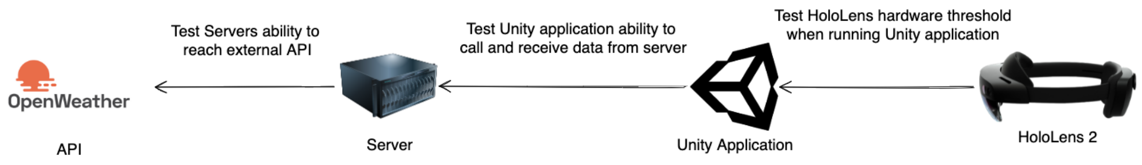

OpenWeather
API
Test Servers ability to reach external API
Server
Test Unity application ability to call and receive data from server
Unity Application
Test HoloLens hardware threshold when running Unity application
HoloLens 2

Figure 4 - High-level Test Flow Diagram

5 Implementation

5.1 Server Implementation

From the beginning of the design phase of our project we have maintained focus on creating a modular design in the backend. Our goal was to develop a base model that works well but also allows for future students and engineers to easily add data streams they want to visualize. To do this we needed to create a backend server that at its core would allow for any type of data to pass through it. Our design philosophy for this was "If a datastream was being added, the front end does not need to see backend code to add visualization for the data stream".

5.1.1 Internal API Implementation

The internal API is the most important part of the server. The API needed to be written in a way that no matter what kind of geographical data stream was being added, it could handle it. To solve this issue we decided to use GraphQL. GraphQL allowed us a flexible and dynamic API that did not have constraints that you would see in a RESTful or SOAP API.

The API was broken down into two main sections, mutations (POST requests) and queries (GET requests).

Mutations:

```
set_channel(channel: Int!, data_set: String!): DataSet!

clear_channel(channel: Int!):  Boolean!
```

Queries:

```
points(viewport: Viewport!, channel: Int!, time: String): [DataPoint!]!

channels: [String]!

datasets: [String]!
```

The 'set_channel' and 'clear_channel' queries are based around the idea of "channels''. Our goal with this design was to minimize the amount of coding and data handling on the frontend. The channels act as a "currently available data streams'' solution. We decided that because on the frontend visualization side we do not want to show more than 3 data streams at one time to avoid bad visualization, we would allow for three available channels of data at one time. Each channel is set to a certain data stream but can be changed out for other types of data with the mutation calls. The 'set_channel' call takes two parameters, channel (Int from 0 - 2) and 'data_set' (String). The 'data_set' is the type of data you want at that channel (openweather_pressure, openweather_temp, WHO life expectancy, ...). The return value of this mutation is a 'Dataset' type. This data type returns information about the datastream being set. The minimum and maximum values, the display_type and the name. These mutations allow the frontend application to set what type of data they want to see.

The 'points' query is the "main" query of the application. This GET request is what returns all the data needed for visualization to the front end. The parameters of the request are a 'viewport' (data type), a ‘channel’ and ‘time’ (not required). The ‘Viewport’ data type is a type made by us to signify what coordinates of the globe to get data for. It takes two coordinates, bottom left and top right to make a rectangle of coordinates and an interval to specify the interval of coordinates wanted by the front end. The return value of this request is an array of ‘Datapoint’ type. The ‘Datapoint’ type consists of four values. Longitude and latitude, value1 and value2. The longitude, latitude coordinates and value1 are required. ‘value2’ was added to allow for types of weather and other data streams that will come with two types of data at a single coordinate. One example of this is ‘openweather_wind’. This data stream has the speed of the wind as one value and the direction in degrees of the wind.

The ‘channels’ and ‘dataset’ queries are information giving queries to the front end. The ‘channels’ query returns what channels are set with what type of datastream. The ‘datasets’ query returns an array of all currently implemented types of datastreams on the backend. These datastreams are located in configuration files on the server application. This type of setup is what allows us to follow our design philosophy of allowing the front end to be written without looking at back end code because we send all the necessary information through the API’s.

5.1.2 OpenWeatherAPI Implementation

The OpenWeatherAPI was our first choice when deciding what kinds of data to visualize. Weather has a long history and many implementations when it comes to visualization. At any time someone with a device connected to the internet can view radars, charts and much more. This is why we chose weather to begin with. It was a good base data stream to create this open modular application. OpenWeatherAPI allowed us to get many different types of weather data in one call and since we were able to get an academic API key we have virtually no limitations on the amount of calls we can make.

After a ‘points’ query is made from the frontend, specifying the ‘Viewport’ and an OpenWeatherAPI data type, the openweather implementation begins by making calls to OpenWeatherAPI. These calls are made for each coordinate in the ‘Viewport’ at the requested interval. Each of these coordinates are called separately from OpenWeatherAPI, giving back a response with all the data types at the coordinate. This is where multithreading comes into play. The application originally made each call one at a time sequentially; with multithreading up to ten calls can be made at once. This decreased our “time it takes to receive data” exponentially. Once all this raw data is collected we cache it on the server. From there this raw data is parsed for specifically what the front end is calling. The cached data can be used again for when the front end changes from one data visualization to another. For example, if ‘openweather_pressure’ is called by the front end to be visualized then within 20 minutes another call for ‘openweather_humidity’ the server will use cached data to fill in the request rather than making new calls to OpenWeatherAPI. This cached data is saved based on coordinate values, so if there coordinates in the new call that are cached in the old call they will be used. Coordinates that do not have any data cached will be requested from OpenWeatherAPI.

5.1.3 WHO Implementation

Integrating the WHO datasets into our backend implementation took a bit of extra work beyond the expected scope, as we needed to implement a reverse geolocator to obtain the country to coordinate mapping required to service data points in our expected format. This was done by generating a lat lon mapping of countries and storing that in a file to be loaded and accessed as an array for quick reverse geolocation. With that implemented all that remains is setting the config to include the specific WHO datasets and making the call to the WHO Athena API.

5.1.4 Modular Config Implementation

One of the priorities of our client was in having the ability to expand upon the implementation easily, especially in the area of supported data sets. To best do this we created a backend python server that on launch imports the dataset resolvers and config files in its directory. To add an additional data set you simply need to write a python class that implements the data point call.

5.2 Unity Implementation

5.2.1 API Requests

The first requirement for the frontend was to be able to make requests to the backend API and fetch data. To do this, we used the built-in UnityWebRequest module. In combination with coroutines, this allowed for making asynchronous POST requests to our GraphQL API and fetching the necessary data. These requests are triggered by UI elements like buttons that pass in a string defining the type of data that should be returned for display. For example, a user clicking on the "Temperature" button will first trigger a mutation request to set the right GraphQL channel for the "openweather_temp" data type. Then, it will make another POST request in the form of a query to fetch temperature data for the entire globe coordinates. When the list of values for each latitude/longitude point is returned from this asynchronous request as a JSON string, it is then parsed into a list of the Data object class. Due to some limitations in Unity's JsonUtility module, the JSON string is being reformatted before it is parsed directly to the object. This process is utilized for other data types like "openweather_humidity", "openweather_pressure" and any others available on the backend. Once the data has been properly parsed, it is ready to be utilized for rendering the visualization.

5.2.2 Particle Data Visualizer

The visualization strategy relies on using particles spawned at coordinates on the globe to represent specific data values. First, there is a prefab object set to be used as the particle (a square plane is being used in this visualization). When the list of Data values is passed into the visualization code (along with a DataInfo object containing the minimum value, maximum value, and name for a specific data type), this is used for checking whether the particles for that specific data set exist and switching the current layout to another if it's a different type than what is currently being displayed. If the particles don't exist yet for that data type, the data is passed into the overlay creation method to spawn them. For each data point, its latitude and longitude values are converted to spherical coordinates. Then, the maximum and minimum values of the data range are used to calculate a normalized value that is used to select a color from that data type's predetermined color gradient. The new particle of the specified prefab is then instantiated and its position is set to the calculated spherical coordinates. Its rotation is set to face the center of the globe model, and its scale is determined by the size of interval between data points. This array of particle instances is kept as a cache for that specific data type and is not regenerated unless it is specified that new data is being fetched. When the user switches between overlays, current instances are deactivated to hide them from the view and ones for the chosen type are either created or reactivated.

5.2.3 3D Earth Model

To build a model of the earth that would have high enough resolution to easily identify land masses while running on the Hololens at a consistent frame rate, a custom earth model had to be made. An attempt was made using Meshes in Unity that would be raised at certain points to look like the earth, however due to the processing power of the Hololens this could not be done as effectively as desired. A model was made using Blender by using a heightmap image of the earth and then raising the land to a desirable height. Textures in Unity allowed for the right colors to be applied, leaving us with the model currently in use in the final release.

5.3 HoloLens Implementation

5.3.1 Hololens Specific Graphics Tools

The Hololens is bleeding edge technology and has its own development kit for Unity, of which multiple versions were released over the course of this project. The Hololens has the ability to render default Unity textures at a cost of its processing power. To increase efficiency, Microsoft released a unique texture component for the Hololens which works as a one-to-one replacement for the Universal Render Pipeline in Unity. The Hololens would also support regular canvas from Unity to build a UI, but the toolkit provided has certain prefabs that would allow the user of the application to gain the complete Hololens experience. Default prefabs are replaced with Microsoft textures that have touch capabilities intended to be used by the Hololens.

6 Closing Material

6.1 Conclusion

Overall, we are proud of our project and all we were able to accomplish. We were able to collect data from an open API and display it onto a globe within the Microsoft HoloLens 2. The data can be diversified from multiple APIs to be displayed in real time. It allows users to more fully immerse themselves in the data, promoting an intuitive understanding. This project has allowed us to challenge ourselves and grow in our understanding of the concepts presented to us during our time here at Iowa State University.

We also acknowledge there is still room for this project to grow. The MRTK is still being developed and released often, with more features and tools being integrated as time goes on. We hope to see this project expand into more APIs in combination with more channels and the ability to overlay data in the scene. However, the foundation we have built is strong and provides a solid base for further development. We are hopeful this project will continue and that we will be able to use the skills we've learned in our adventures after Iowa State.

7 Appendix

7.1 Operational manual

7.1.1 Server Deployment Operations

Summary: This section discusses how to operate the backend server. The following steps will describe how to deploy the server application to the VM machine located in ETG. This section assumes you have the changes made to the code and tested on your local machine and are ready to push it to the production environment.

Link to download code base: https://git.ece.iastate.edu/sd/sdmay22-21.git

7.1.1.1 Prerequisites

- Code base downloaded
- Terminal/Bash/Shell installed on your local machine
- VPN to Iowa State network (if off-campus)
- python3 installed on VM

7.1.1.2 VM information

- OS: Ubuntu 2004 LTS
- Host: sdmay22-21.ece.iastate.edu
- User: vm-user
- Password: wc$sB4W@Mt8T

7.1.1.3 Transfer Backend application to VM

In your Terminal, **cd** to the project directory on your local machine.

```
>> cd sdmay22-21
```

Once you are in the project directory you will do a secure file copy (**scp**) to the VM machines home directory. This will copy over the backend code to the VM.

```
>> scp -r Backend vm-user@sdmay22-21.ece.iastate.edu:~/home
```

Enter in the password to authenticate the full transaction.

7.1.1.4 SSH to VM

Now that the backend code is transferred over to the VM, we are ready to run the application to allow it to make a connection with the frontend application. In your terminal you will **ssh** to the VM hosted by the ETG.

```
>> ssh vm-user@sdmay22-21.ece.iastate.edu
```

You will most likely need to enter in the password to complete the ssh process.

7.1.1.5 Deploying application

Now you can **cd** to the project directory in the VM.

```
>> cd /home/Backend
```

Once you are in the Backend directory you can run the app.py (application) using the **python** command.

```
>> python app.py
```

This will start the app.py file as a python application. The application should now be running on the VM and ready to make connections to outside API's and the front end application.

7.1.1.6 New VM or Machine?

If for some reason you are doing this process on a new VM that was not the original issued one from Spring Semester 2022 or you are trying to run the application on your local machine for the first time. Once you have the Backend folder on either machine. To run you need to install some dependencies that are used in the project.

In your terminal, **cd** to the Backend Directory and complete the following commands:

```
>> sudo apt update
```

```
>> sudo apt install python3-pip
```

```
>> pip install flask
```

```
>> pip install ariadne
```

```
>> pip install requests
```

This will solve the dependency issue you have when trying to run the application for the first time on a machine.

7.1.2 Local Server / API Testing

Summary: This section will describe how to locally run and test the server application along with local API testing.

7.1.2.1 Prerequisites

- Code base downloaded
- Terminal/Bash/Shell installed on your local machine
- python3 installed on local machine or VM

7.1.2.2 Locally deploying the application

Once you have the code base downloaded, **cd** to the Backend directory.

```
>> cd /sdmay22-21/Backend
```

If this is your first time deploying the application you will need to install the dependencies on your machine to run the application. Use the following commands to install the dependencies. The command **sudo** may depend on what OS you are using. This is based on an Ubuntu operating system which you can ssh into (Iowa State's VMs) from any OS.

```
>> sudo apt update

>> sudo apt install python3-pip

>> pip install flask

>> pip install ariadne

>> pip install requests
```

Following the installations of these dependencies you can now run the server application locally.

```
>> python app.py
```

Now to test the internal API system you will need to open a browser to use GraphQL's playground. In your browser connect to your localhost with the following address:

>> http://127.0.0.1:5000/graphql

This is GraphQL's testing playground where you can make API calls to test the server application locally.

7.1.3 Frontend HoloLens 2 Deployment Operations

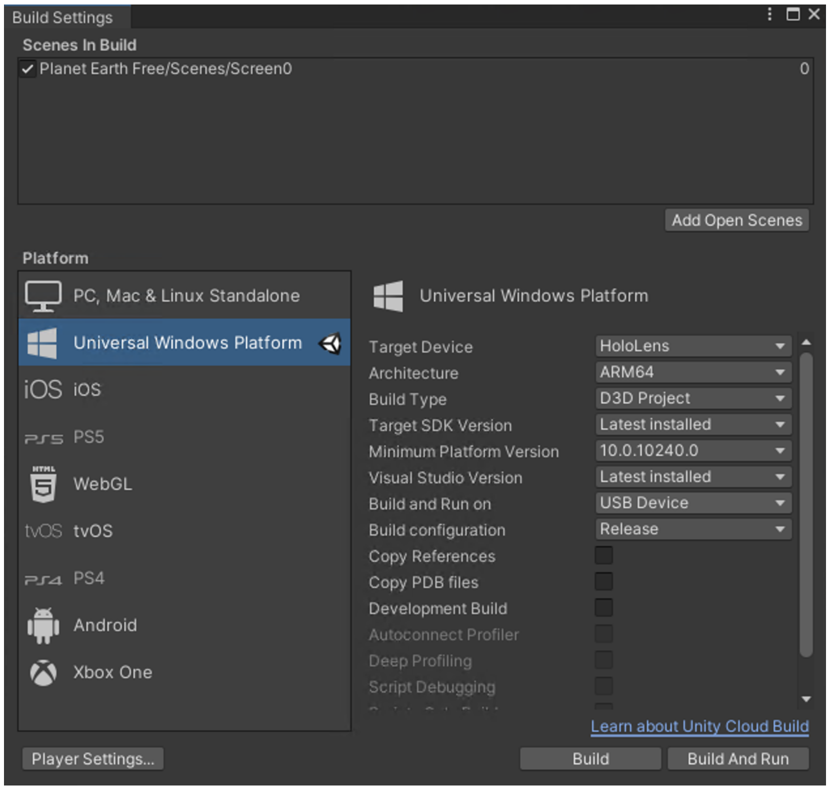
Summary: This section will describe how to get the Unity frontend application setup and deployed onto the HoloLens 2. This assumes that the code is free of compile errors and ready to be deployed and that the HoloLens 2 and PC both have developer mode enabled.

7.1.3.1 Prerequisites

- Unity (version 2020.3.4f1) installed on your local machine
- HoloLens USB-C cable
- Unity project downloaded
- Visual Studio 2019 (16.8 or higher)

7.1.3.2 Open the project in Unity

Use the Unity Hub (comes with the Unity installation) to add your project folder as a Unity project and select the appropriate engine version (2020.3.4f1). Then, open the project from the Hub by clicking on it. This will open up the project in Unity editor. Finally, we will need to export the project from Unity into Visual Studio. To do this, Go to *File -> Build Settings*. Ensure the "Universal Windows Platform" is selected and the build configuration looks like the following:



Planet Earth Free/Scenes/Screen0 0

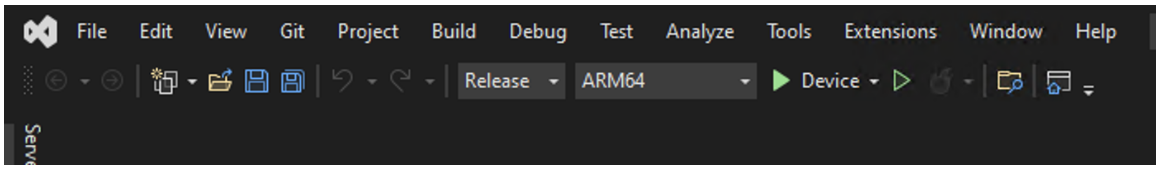
Universal Windows Platform

Target Device	HoloLens
Architecture	ARM64
Build Type	D3D Project
Target SDK Version	Latest installed
Minimum Platform Version	10.0.10240.0
Visual Studio Version	Latest installed
Build and Run on	USB Device
Build configuration	Release
Copy References	☐
Copy PDB files	☐
Development Build	☐
Autoconnect Profiler	☐
Deep Profiling	☐
Script Debugging	☐

Now click "Build". Select a folder to store the build into, and go grab a snack while Unity exports the project. Once Unity finishes, the location of the saved build will open in File Explorer. Navigate to whichever location you chose, and open the "My Project.sln" file.

7.1.3.3 Open the project in Visual Studio

Once Visual Studio loads in the project, ensure the following settings are adjusted at the top of the window. More specifically, "Release" "ARM64" and the play button should say "Device" next to it.



At this point the final step is to ensure that the HoloLens 2 is plugged into the computer you are running Visual Studio on. Go to *Build -> Deploy Solution* and once Visual Studio loads the application onto the device, you are ready to jump into Mixed Reality and load up the application.

7.1.4 Using the Windows Device Portal

The Windows Device Portal was very helpful while developing for this project. It has many tools included that help manage the hololens and debug/optimize the application.

7.1.4.1 Prepare the HoloLens

The HoloLens should already be in Developer mode, but if something has happened, here's the steps:

1. Power on and put on device
2. Use Start Gesture to launch main menu
3. Navigate to the Settings App
4. Select Update and then For Developers on the left
5. Enable Developer Mode
6. Scroll down and enable the Device Portal
7. Click Home on the settings app
8. Select Network & Internet and then Wi-Fi on the left
9. Select Advanced Options
10. Use the "What's my IP address?" voice command and make note of this IP

From a web browser on your PC, go to https://<IP_FROM_PREVIOUS_STEP> Select the advanced, and continue anyway options if prompted.

7.1.4.2 Logging into the Windows Device Portal

If you're prompted for a username/password combo, we have it set to the following credentials:

Username: SeniorDesign

Password: 1234567

7.1.4.2 Useful Tools Relating to This Project

We found the following tools in this device portal to be helpful:

- Mixed Reality Capture
- System Performance
- Apps
- App Crash Dumps
- File Explorer
- Kiosk Mode
- Virtual Input (It's so much easier using a real keyboard vs the holographic one!)

7.2 Future Additions

There are a number of ways we see this project can be continued past the work we have done. Beginning with the backend, we believe converting the backend server from Python to javascript would be an improvement to the current system. This would allow the server to be run locally on the HoloLens 2 rather than on a VM. In addition, increasing the number of API channels available for use in a session would allow for a user to compare more types of data together. We would also encourage the inclusion of more APIs to retrieve data from to increase the different types of data available to users.

As the frontend, the ability to display multiple streams of data would be ideal. This would require an update on the system in which the data is displayed. The ability to change the strength in color of each layer of data or the ability to raise and lower them in proximity to the surface of the sphere would be two options to accomplish this.