PROJECT GLASSHOUSE

L.E.O. Lab Visualization of Ground-Level Sensor Data

CSC 544 (Advanced Data Viz) Term Project Report

Steven Eiselen (seiselen24@email.arizona.edu)

Final Version 12/7/17

Motivation

The Landscape Evolution Laboratory (LEO), is Biosphere II's current largest project. Biosphere's website identifies LEO as "*the world's largest laboratory experiment in interdisciplinary Earth sciences*", states that LEO research will "*advance our understanding of how climate change may impact water resources and ecosystems in arid environments*", and emphasizes how LEO dats will be used to create models on how "*water, soil, plant, and microbes respond to diverse scenarios of climate [and] how water resources and ecosystems in real landscapes may be impacted by ongoing and future climate change*" <source>.

An application of this research connects to one of the main reasons Biosphere II was built: explore how to sustain the needs of human populations living in off-world colonies on Earth's Moon and Planet Mars. With several governments and corporations actively working to realize this dream; alongside a need to sustain Earth's population in the coming decades of climate change and water scarcity (especially in the American Southwest): the development of immersive and interactive visualizations for 'Smart Biomes' equipped with sensor arrays (of which LEO is the world's largest) entails an important endeavor.

Problem Statement - Overview and Mission

From initial conversations with Biosphere II's outreach and science leadership, as well as members of the LEO research team: there exists a need to provide improved visualization, dataset interactivity, and data access/flagging capabilities for the LEO sensor data given the uniqueness of this lab. In particular: the team is interested in a solution for three related 'open viz problems':

1. The capability to view an ‘interactive animation’ of spatiotemporal data across a period of time, such that a user can view LEO data attributes at any ‘frame’ of the animation (i.e. not a ‘pre-baked’ animation output by R and/or MATLAB). In other words: the ability to interactively and visually explore spatiotemporal data produced by the LEO.
2. The capability to maintain basic ‘quality control’ over the sensor network through being able to ‘flag’ parts of the data which are interpreted as being anomalous and potentially ‘bad’.
3. The capability to present immersive, ‘edutaining’ visualizations of LEO data to the public in order to better communicate and present both LEO’s scientific research; and how the landscapes ‘evolve’ over time across the variety of sensor data types supported. This problem ties into a near-future plan to upgrade Biosphere II’s guest experience; which includes plans to feature kiosks at the public tour area overlooking the LEO which would provide interactive visualization of the LEO’s current status as well as its ‘heartbeat’ over longer periods of time.

Therefore, the ‘Mission Statement’ of Project Glasshouse is to provide a ‘first phase’ proof-of-concept interactive visualization of the LEO’s sensor data which could be used for public display on the website and future tour kiosks; as well as initial experiments in utilizing the application for research purposes with the possibility of expanding upon the initial prototype in future iterations.

Understanding LEO’s ‘Physical and Data Space’

Overview:

One of the first and major challenges of implementing a visualization for the LEO was understanding what data it produces, and where this data is collected from. This task encompassed understanding two sides of the same coin: LEO’s ‘physical space’ and ‘data space’. To the former: this meant studying how the LEO is spatially configured, what kinds of sensors exist within it, the types of data recorded by the sensors, and where these sensors are located within the LEO. To the latter: this meant understanding how the data was recorded into the database, how to retrieve data, and how to map data values into a geometrically accurate representation of their corresponding locations in the LEO environment.

LEO Physical Space:

Figure 1 shows LEO’s location within Biosphere II. The lab occupies the former Agriculture module which consists of 3 large interconnected ‘greenhouses’, each of which uniformly rises three times to vertically set-back, half cylinder rooves. LEO correspondingly contains 3 simulated landscapes within these spaces. Each landscape is 30 meters long, 11 meters wide, 1 meter deep, and elevated slightly above ground. As the architecture is laid out in a near East/West orientation, and as the setbacks permit each landscape to rise upwards at an average slope of 10 degrees: the landscapes are named the ‘East Slope’, ‘Center Slope’, and ‘West Slope’ accordingly.

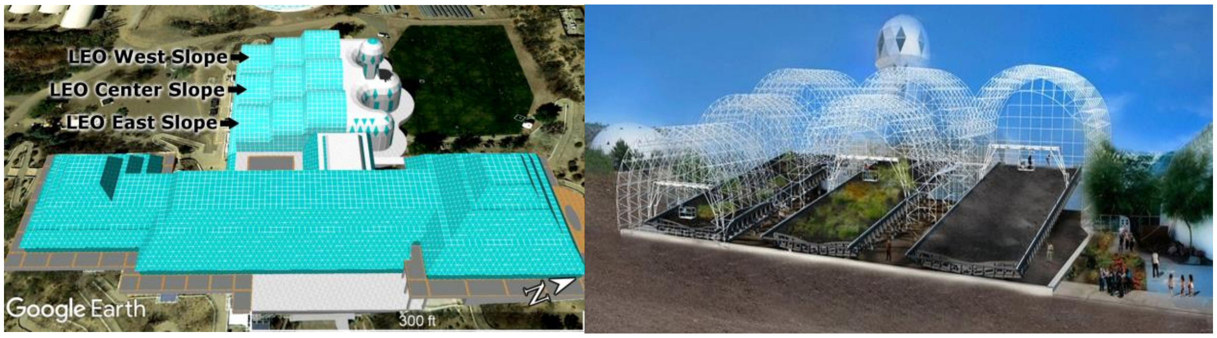

Figure 1 - LEO Location in Biosphere II Complex and Rendering

Furthermore, there are 2 main zones where sensors are located for each landscape. The first are 'masts' hanging down from the ceiling of each sector above their respective landscape. The mast sensors record relative humidity, air temperature, wind flow, and solar radiation. They are not included within the scope of this project. The second zone are within the landscapes themselves. The sensors are laid out in a grid-like pattern at 5 different depth levels from the surface: 5cm, 20cm, 35cm, 50cm, and 85 cm. The landscape i.e. ground sensors record data including temperature, heat flux, water content, water potential, and gas concentration.

Figure 2 below shows the mast and ground sensor instrumentation layouts. Note that the ground sensors are not uniform for each level, causing 'gaps'. This became and remains an implementation issue in terms of interpolating known values per level (along with other heuristics) into 'filling in gaps' between unknown parts of the landscape not covered by sensors to produce the visualization. Also note the amount of sensors. From the LEO Instrumentation wiki page <source>, we observe that there are over 1000 ground sensors per slope. We lastly note the coordinate space for LEO slope sensor locations. Five keys are needed to identify a sensor: Slope ID, 'Up Slope', 'Cross Slope', 'Slope Depth', and 'Sensor Type'. Their domains are as follows:

- Slope ID: [E,C,W] which correspond to the names given to the slopes as aforementioned
- Up Slope: [1,30] Distance in meters 'up the slope from the bottom edge'
- Cross Slope: [-5,5] Distance in meters to the left (negative) or right (positive) of the centerline
- Slope Depth: [1,5] Sensor depth within the slope as aforementioned s.t. $1 \rightarrow 5cm, 5 \rightarrow 85cm$
- Sensor Type: ["5TM Temperature","5TM Water Content"…"MPS2 Temperature"] type of sensor

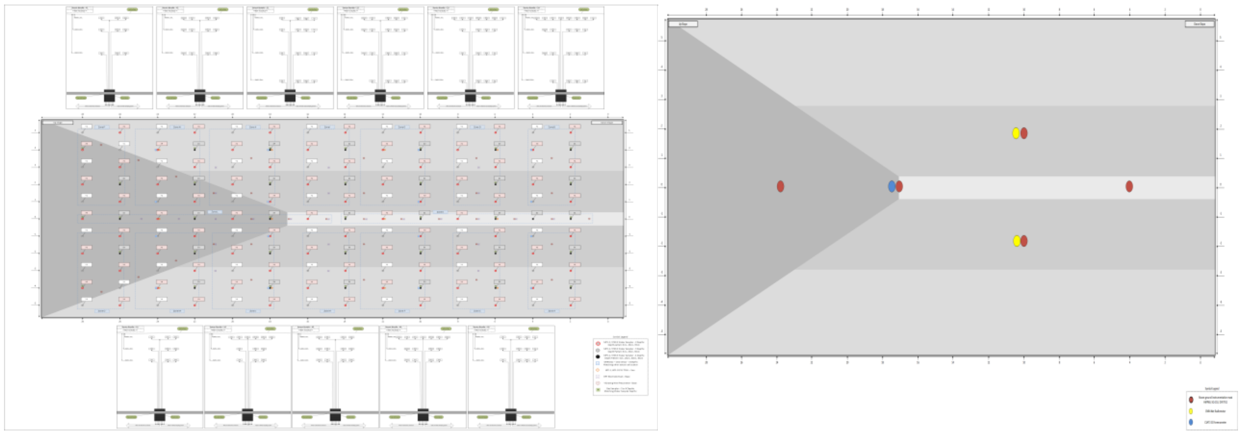
Figure 2 - LEO Ground and Above Ground Instrumentation Layout

LEO Data Space:

The other side of the coin involves representation of the sensor data in the main database or a smaller subset thereof. From a meeting with Biosphere's IT staff member, it was discovered that the LEO utilizes a Relational Database accessed through a SQL interface. While utilization of the database became out of scope for the project, a form system via a user-access login on the Biosphere site provided a means to obtain and work with datasets downloaded from the main database as CSV files. This form allows users to get a dataset encompassing a single sensor value at a single slope at an input time interval. Examples of these can be seen in the sample datasets provided with the term project code. This does present restrictions versus full control over query types, as mentioned at the end of this section.

The attribute names used in the database/CSV datasets utilize the coordinate keys described above, with the important addition of an attribute called 'DateTime' which provides the timestamp of the data in <MM/DD/YYYY HH:MM> format. Furthermore, the CSV files are ordered on the 'Datetime' attribute, making the timestamp effectively a keyframe containing the state of the entire dataset at a certain time. This is great for a spatiotemporal animated visualization. For example, the following chart describes the temperature in degrees Celsius of the 5TM sensor located at the East Slope, 2 meters 'forward' from the bottom edge, 5 meters to the right, at a depth of 5cm, at midnight on 11/14/2017:

DateTime	LEO-E_2_-5_1_5TM
11/14/2017 00:00	23.5

There are other important things of note about the data. Foremost: the value -9999 is assigned to sensors which are offline, which is something that needs to be taken into consideration. Second: the type of data recorded is not mentioned in the attribute name but in a section of text at the top of the CSV file. Note that the sample datasets submitted with the term project have had these headers removed. Lastly: it was discovered that two sets of different data over the same time period could be missing one or more tuples, leading to a mismatch issue when iterating through them. Resolutions to these last two issues are discussed in the 'Future Research and Development' Section.

Limitations of Study

In Terms of Data:

The intricacy of the LEO data was apparent at the beginning of the project and became more noticeable as development began. Supporting visualization of the entirety of the sensor data would encompass a scope way beyond what was possible for a 1.5 month timeframe of a term project. In particular: the number of sensors for each of the three landscapes including ground/slope, mast, and peripheral total slightly over 1850; leading to a total of approximately 5550 for the entire LEO lab. Now consider the scale of this many columns of data for each data frame; at 5 minute intervals per time frame; for a set encompassing days, weeks, even months! Even considering just the slope sensors still amounts to a sizeable proportion of this number. Ergo the decision was made to limit the datatypes to the following:

- 5TM Temperature
- 5TM Water Permittivity
- 5TM Volume Water Content
- MPS-2 Water Potential
- MPS-2 Soil Temperature

In Terms of the Visualization:

As aforementioned, the visualization was restricted to only the slope zone, as visualizing only the slopes is more straightforward than also supporting the masts and peripheral sensors. Also, the quality-control anomaly reporting feature was cancelled in return for supporting isometric contour map visualization; though implementation ideas are discussed in the 'Future Research and Development' Section.

Description of the Deliverable

Initial Version from the Project Proposal: The Deliverable was initially defined as implementing two 'view modes' and two 'view styles' of the aforementioned dataset as follows:

View Modes:

- 'Live View' of the most recent sensor data for public viewing on the Biosphere Website and/or kiosks planned for the public tour; as well as for providing live diagnostics.
- 'Spatiotemporal View' which, when a user inputs a start and end point in time: allows an 'interactive animation' for all sensor readings (and possibly multivariate combinations as well)

View Styles:

- 'Isometric Contour Heatmap' utilizing marching squares to visualize the data in contours. The visualization would contain 3 'columns' representing each LEO landscape slope; of which each column contains 5 'vertically stacked' representations of each level of the landscape slope.
- 'Summary Line Chart' which would show sensors' values across the entire dataset time frame while showing the current time frame's position on the line chart throughout the animation.

Late-Stage Revision and final Deliverable: The actual deliverable shares much in common with the initial one, with two major differences. The first is that getting data directly from the database versus the website login request form was cancelled due to unexpected issues and complications with the core implementation. The second change was cancelling the ability to interactively switch data types 'live' as the visualization was animating due to similar issues. Additional ideas unable to be implemented are discussed in the 'Future Research and Development' Section. The final deliverable is defined as follows:

View Modes:

- 'Two Slopes Compare': Allows the user to visualize two datasets which could have any combination of different slopes and/or data types and/or time periods. Features the 'Summary Line Chart' as was discussed in the proposal.
- 'Three Slopes Live View': Allows the user to visualize datasets representing a uniform data type and time period across all three slopes. This is nearly the same as described in the proposal.

View Styles:

- 'Isometric Chloropleth Grid Heatmap': Effectively takes the first prototype made for the visualization, a top-down grid heatmap – and performs an isometric projection upon it.
- 'Isometric Contour Heatmap': The same representation made with the proposal with an additional feature for 'mouse over' events in the 'Two Slopes' view mode: will highlight the lines on the line chart associated with the cell and its counterpart on the other slope.

Discussion of the Implementation

The best way to provide a written overview of the implementation for both view types and styles is to discuss the execution of code from initialization to interactivity; as this provides the best organization of the work done in terms of both technical flow and chronological ordering. While the 'Project Challenges/Discoveries' and 'Future R&D' sections capture the project's complexity and unfinished ideas, this section discusses what was successfully completed from the experiments and work done throughout the build process.

Phase 1: Pre-Launch User Input

The starting point for the 'Two Slopes' view type involves reading in an input file. *At the time of writing this document, 'Three Slopes' view does not have this functionality yet and is hardcoded to work with the default '5TM Temperature 11/1-11/5' dataset*. First: the file called 'input.js' within the main code directory is used as a kind of command line argument wherein the user must provide the input needed by the application to visualize their data. Specifically: it requires the following for <u>both</u> slope datasets:

- The filename e.g. "LEO_5TM_TEMP_E_11-1_11-5.csv"
- The LEO Slope ID e.g. "E"
- The suffix of the datatype as seen in the CSV file e.g. "5TM" for 5TM sensor data
- A string containing the data type e.g. "5TM Temperature"
- A string containing the data unit e.g. "Degrees Celsius"

The purpose of the filename is self-explanatory. The LEO Slope ID and datatype suffix is needed for getting information from the CSV by attribute id key (see 'LEO Data Space' subsection above for more info). The final two strings are used to display themselves on the user interface. Lastly, the first object definition is identified 'dataInputL' and the second is identified 'dataInputR' as they will be the left side and right side slopes respectively as displayed in the visualization and referenced across the application. *For 3-Slopes: the datasets are identified as "E", "C", "W" for the East, Center, and West slopes and corresponding data respectively.*

Phase 2: Loading in the CSV files / Asynchronous Initializations / Colormaps Pt. 1

The next step for both visualization view types is to call `d3.csv(…)` on each of the filenames. This is tricky, as this operation is asynchronous and needs to fully execute across all input datasets before any code that operates on them can start execution. Therefore: the code encompasses a nested sequence of `d3.csv(…)` calls, leading to a call of 'main' on the successful condition branch of the final `d3.csv(…)` call in the sequence. If any call produces an error, a message prints to console and successive calls are not done. However: if all of the csv data reading operations complete successfully, the datasets are assigned to the 'data' key-value array, the `main()` function is called, and the next phase can begin.

While this occurring, various minor initialization and assignment operations occur on global value definitions, as well as assigning the data type and data unit values from the input in '2 Slopes' mode to their respective HTML UI objects. One significant initialization is of the colormaps.

The initial idea for the colormaps was to use a univariate colormap, in this case: orange to red. However, the LEO Team wanted to use MATLAB's Jet colormap, with which I produced a decent equivalent in d3 which remains commented out in the code for reference. The 'Rainbow Color Map' issue was brought up, and I again produced a decent representation of MATLAB's Parula colormap: which remains the colormap used with the present deliverable.

A side note on colormaps. By the definition of what the '3 Slopes' visualization mode does, there only needs to be one colormap as all three datasets should represent the same data type. However: the '2 Slopes' visualization mode does support two distinct color maps based on Parula as the datatypes are permitted to be different from one another, and perceptual uniformity is no longer a constraint.

Phase 3: Additional setup in 'main' / Utilizing Dataset Length and Extents / Colormaps Pt. 2:

The first few lines within the `main()` function involve determining the lowest number of rows between all datasets. This is the quick-and-dirty handling of unequal dataset sizes. A solution for this has been successfully demonstrated, and its code has been placed alongside the other implementation code for review: but this feature was unable to be implemented by the deadline and has instead been proposed below in the 'Future Research and Development' Section. After this step, four functions are called which update values in the UI that depend on their handlers knowing what the length of the visualization is in terms of total number of in time frames (i.e. the lowest dataset tuple size).

The function `generateExtents()` is then called, which iterates through all tuples of each dataset to provide the global max and min for each dataset. This is the first computationally expensive part of the application, and causes a major delay due to the number of computations it needs to do. The function it uses to local min and max of each tuple will be skipped over for now, as it will be discussed during the visualization initialization step. There is discussion on `generateExtents()`in the 'Future Research and Development' Section about how this initial cost could be utilized to significantly reduce additional lag.

Next, the domain for the '2 Slopes' colormaps (for the left and right slope respectively) are initialized from the left and right datasets' extents. In the case of '3 Slopes', another function produces the extents across all three datasets to maintain perceptual uniformity. The `createColormap()` function is then called, which produces either one or two colormaps based on the visualization type. Finally: the `setupLineChart()` function is called only in '2 Slopes' mode to produce the line chart. An important thing to note is that it skips producing lines for any LEO 'cell' coordinates which either do not have sensors at that location, or whose sensor data is invalid (i.e. -9999).

Phase 4: Organization/Initialization/Animation of Visualization by Style

The final lines of code in main initialize the visualization through calling either the `initContourViz()` xor `initChloroplethViz()` functions (depending on a boolean flag value); then starts a repeating sequence of `updateByVizMode()` function calls via `d3.interval()` to animate the visualization. For both the initialization and update functions of each visualizations style, we notice the same double for loop and switch statement. Each iterates through every slope and level, which can be either 10 or 15 total iterations reflecting the 5 levels per slope and 2/3 slopes based on the visualization mode.

The functions that they call either initialize or update each of the visualizations by the currently selected visualization style. That is: each level of each slope is an independent SVG. The Chloropleth initialization function is similar to one of the CSC 544 assignments with the addition of an isometric transformation, while the update version simply and efficiently updates the data values. The Contour initialization involved becoming familiar with the `d3.contour()` library, and has a much less efficient update which needs to completely redraw the contour map. This is the 3rd major cause for lag within the application.

We finally note that the initialization and update code for the 'Chloropleth' and 'Contour' visualization styles call one of two functions within their `.data()` calls: `generateValueSetValuesOnly()`, or `generateValueSet()`. Each perform the same basic role: produce values for all row/column (length/width) cell coordinates in the dataset at a particular slope, level, and time frame. The only difference between them is what they return, and how they iterate through all LEO row/column coordinate combinations. We discuss this in the next subsection.

Phase 5: Working with the CSV data to create useable Visualization datasets:

The `generateValueSet()` function works with the 'Chloropleth' style and returns an object with a 'Row' and 'Col' attribute in addition to the value which the visualization code uses to place them in their proper coordinate positions in the visualization. The `generateValueSetValuesOnly()` function works with the 'Contour' style (more appropriately: the `d3.contour()` interface), and simply returns an array of values. Note on observation of the code that this version of the function also has a different for loop condition, due to the `d3.contour()` library. This is discussed further in the 'Mirrored Display Issue' subsection of the 'Discussion of Project Challenges and Discoveries' section below.

Despite these differences, both functions call the `getValue()` function. This function encompasses the second cause of lag in the visualization application, due to the computational work that it does to provide an approximate value for cell coordinate passed to it which result in either a -9999 (offline sensor) or null (no sensor at coordinates) value. This function effectively performs a quick-and-dirty nearest neighbor search, spanning out one cell in all directions until finding a neighbor(s). If more than one neighbor is discovered (the common case), then the average between them is taken and assigned to the cell. Improvements for this taks are discussed in the 'Future Research and Development' Section.

Lastly, the `getValue()` function calls yet another utility subroutine that takes in coordinates and returns a legal value, the -9999 value aforementioned, or the null special constant aforementioned. This function is called `keyFromCoords()`, and its parameters deserve to be discussed. They are as follows:

- t = current time frame of the animation
- s = the query's slope coordinate
- l = the query's level coordinate
- r = the query's 'Up Slope' coordinate
- c = the query's 'Down Slope' coordinate

Combined, the `generateValueSet`*`[Only]`*`()`, `getValue()`, and `keyFromCoords()` utiltities access and provide the subset values from the CSV dataset to the caller for use with a d3 `.data()` call to produce the resulting visualization as discussed in ***Phase 5*** immediately above.

Phase 6: Some notes on UX/UI Components not previously discussed:

Visual Elements in both the '3 Slope' and '2 Slope' Visualizations:

- Textbox indicating the currently animated frame / total number of frames
- Textbox indicating the timestamp of the currently animated frame
- Textbox indicating the current visualization mode. See 'CHANGE MODE' button below.
- Information on the currently displaying sensor types and their data units.

Interactive Elements in both the '3 Slope' and '2 Slope' Visualizations:

- STOP Button: Stops the animation, resets the current frame to zero, does NOT autoplay
- PLAY Button: Plays the animation through time of the dataset

- PAUSE Button: freezes the animation. Unlike 'STOP', hitting 'play' will resume animation
- STOP Button: Stops the animation, resets curFrame to zero, does NOT autoplay
- GO TO FRAME Button / Select Frame input box: A user can enter a legal frame number into the input textbox, then click the 'GO TO FRAME' button to jump to that frame. Basic erroneous input has been handled.
- CHANGE MODE: User can switch back and forth from Contour to Chloropleth Visualization Style

Visual and Interactive Elements in '2 Slope' Mode Only:

- The linechart features an animated 'ticker' line at the current frame of animation sequence which will travel along the x axis based on the position of the current frame.
- Y Axes of both datatypes are shown on the left and right side of the line chart correspondingly
- Chloropleth Mode Only: a 'Mouseover' event on a cell will highlight it and its counterpart in the other slope. If the cell has an entry in the linechart, both its line and its other slope counterpart's will also highlight.

Discussion of Project Challenges and Discoveries

Multiple challenges and discoveries were encountered in addition to the ones which have been mentioned earlier in this report. Several of the largest which have not yet been discussed will be covered in this section. Alongside the challenges discussed earlier and the 'Future Research and Development' Section below; this constitutes part of the analytical discussion of the project's results.

The 'Mirror Issue' with Chloropleth and Contour Visualizations.

- ***Chloropleth Visualization:*** As hinted to earlier, there was a major bug discovered wherein the Chloropleth visualization showed the 'Cross Slope' cells in their correct position, but showed the 'Up Slope' cells in reverse order; thus appearing to be horizontally mirrored. This bug was verified by examination of the instrumentation layout diagrams. A simple trick was done to the value of the 'row' parameter value passed to the `getValue()` function to correct this bug.

- ***Contour Visualization:*** In the case of the Contour Visualization, the mirror effect was evident for both the 'Up Slope' and 'Cross Slope' dimensions. This was due to both the same bug as with the Chloropleth visualization, as well as the 'funky' mechanics of the `d3.contour()` library. Trial-And-Error on parametric inputs to the `getValue()` function to corrected this bug.

Dealing with a quickly complex implementation and modularity.

Halfway through the build, the project's scale and complexity began to grow as quickly as the amount of work required on multiple fronts to create the deliverable. Various concerns and considerations were made in taking steps to reduce unnecessary computation costs as best as possible, while providing more modular support, while making sure enough of the pieces fit. Two examples come to mind:

1. There was a plan was to create a 'utils.js' file featuring code shared by the '2 Slopes' and '3 Slopes' visualization types before work on the '2 Slopes' mode began. The decision was made to keep the two parts disjoint and deal with the subsequent code re-use issues; due to potentially significant refactoring work which would have likely sabotaged the '2 Slopes' mode from having ever been implemented. Consequently: there was and remains major code reuse as well as deficiencies in the '3 Slopes' visualization due to time running out on upgrading it.

2. There remains plans to enhance modularity to the point that a single-slope visualization or line chart could be defined and used like a instance object. This was also unable to be realized in the deliverable implementation code.

Dealing with the Data while Realizing its Multidimensional Capabilities.

Handling the -9999 and null value issue for both debugging and value approximation reasons became much more involved than first thought. Furthermore, I began to realize the true 'Big Data' essence of the LEO: Approximations could be determined not only by nearest neighbors in the same slope and level, but also neighbors in the levels above and below the cell (3D), values of the cell forwards and backwards in time (4D), and values from other sensors recording the same data type (5D).

An even crazier idea was in extending these approximations towards being able to 'partially reconstruct' entire missing and/or erroneous timeframe sections of LEO Data. At this point: I resolved that these initiatives could standalone encompass entire theses let alone term project topics; and focused back on the core deliverable.

Discussion of Future Research and Development

Before providing a brief conclusion remark, I wanted to cover some ideas which were unable to be implemented, but are worth mentioning in the event that future iterations of this application are developed. They are organized by topic as follows:

Loading in Data:

- Could embed more info and even a custom header into CSV file to reduce the info needed in 'input.js'. The link <here> shows how to skip over first few lines of a CSV for this purpose.
- Could handle the -9999 error value and null value approximations at initialization and/or during the call of `generateExtents()`
- Could utilize the 'Data Merge' code to match up unequal datasets; as well as "merge" the data from all 3 slopes into a single structure on the 'TimeDate' key for the 3Slopes visualization.
- Alternate Option to current `generateExtents()` function: Could precompute values for 'expected range' extents via earlier independent calculations and place in a file for instant retrieval; while being able to periodically update the values as needed.

Greater Input support:

- Maybe do a splash page where user can choose which app they want [LIVE VIEW | 2 SLOPES]
- Then a simple link redirect to respective app
- Could do this via a form asking user for input info in the splash screen

Future UI/UX Improvements:

- Try to balance out the colormap better than current version
- Update 3Slopes to the progress made with 2Slopes. Examples of 'Things-To-Do' include:
    - Fixing the mirror issues (2Slopes works, so this is a refactor job)
    - Background box over the E/W/C divs to distinguish slopes with text print below
    - Cleaning up the HTML code and better organizing its layout as with 2Slopes
    - Multiple Datatype Support

- Line Chart ‘mouseover’ in Contour Mode (possible big CPU cost and code complexity)
- Forward/backward one frame buttons
- Grid toggle in Contour Mode to biew and interface with LEO cell coordinates
- Create legends on the slope level visualizations showing coordinate values

Concluding Remarks:

Having produced an initial proof of concept for a ‘first phase visualization’ of the LEO, I’ve learned a ton about the lab itself as well as data visualization techniques. Furthermore: I’ve gotten a small glance at the power and capability of not only providing animated, interactive visualization of the LEO, but a taste of implementing powerful, multidimensional approximation algorithms.

It is my hope that either through my continued progress and participation in this visualization project after the term project has been submitted or otherwise: the work done on this project leads to further iterations and improvements upon the ‘LEO Visualization Project’ leading to even better and more immersive interactive visualization capabilities.