Visualization of Climate and Earth System Modeling

DESIGN DOCUMENT

Team: May1701
Client: Professor Chaoqun (Crystal) Lu
Advisor: Professor Johnny Wong

Team Members/Roles:
Kellen Johnson: Team Communication Leader
Anish Kunduru: Team Leader
Eli Devine: Webmaster
Julio Salinas: Concept Holder

Team Email: may1701@iastate.edu
Team Website: http://may1701.sd.ece.iastate.edu/

Revised: 12/2/16 v 2.0

Contents

1 Introduction

1.1 PROJECT STATEMENT

Dr. Chaoqun (Crystal) Lu, a Professor at Iowa State, has created a way to model and project various compound levels upon Earth. To display this data, our client does calculations using an algorithm (that is unknown to us) and uses another system to map these calculations onto a multi-colored image stored in a .tiff file (See appendix 8.3). What our client would like to do is take these calculations and projections and move them into an easily query able 3D space which will allow end-users to dynamically select what they would like to view. The end result is to reduce these massive datasets into a compact, easily viewable, and highly interactive 3D space.

1.2 PURPOSE

Gas compounds and various elements introduced to Earth affect our daily lives. From crops to real-estate to emission law-making, various stakeholders need a way to view historical compound readings and project future readings for decision making. Instead of creating these calculations and sending hundreds or thousands of images for individual viewing, we would like to create a website that will allow these decision makers to choose what they would like to see and from when they would like to see it. Additionally, we would like to encourage more user interaction by moving from 2D pictures to interactive 3D maps.

1.3 GOALS

We would like to enable the client to take her work and transpose that work onto a revolutionary 3D space using ArcGIS for Server. By doing this, the client will be able to demonstrate her calculated findings to various policy-makers, and educate interested parties about the findings of her research. We plan to do this by creating a product which will allow the client to simply upload ASCII files (see Appendix 8.1) to a server. This ASCII file will then be parsed into raw CSV (comma separated values) data (see Appendix 8.2) and dynamically set into various 3D layers that will be hosted on her server as a map service provided by ArcGIS for Server. End-users will be able to dynamically pull any map service they wish to view our client's projections and historical data on their own web browser. Our primary mission is to provide a platform to perform these functions in a timely manner. Additional requirements are: be consistent with all uploads and be sufficiently modular and maintainable such that any future work done with this product should not require significant changes to core (code) functionality.

2 Deliverables

- Server: The Server that will serve as our host for ArcGIS for Server, and will store all data that the client chooses to upload. Additionally, it will host the website as an additional deliverable. It is possible that the web server will be a different server from our ArcGIS RESTful (REpresentation State Transfer) server.
- Parser: The program which will parse all files uploaded by our client and transform her ASCII files into CSV files to be used as the datasets that are used for creation of maps and layers. In later prototypes, the parser will also be responsible for converting the CSV files into tables, and finally saving the results in a database. Ultimately, the parser will also manage server-side scripting that will happen via ArcPy (automated Python scripting).
- Automated Python Scripting: Due to the fact that ArcGIS can be automated with Python scripting, all CSV files that are outputted by the Parser will have to be read using a Python script which will create the basemap (the foundation of a map service), append and create the layers (represent the points in the CSV files on a MapImageLayer), and publish to our ArcGIS Server (create the map service).
- Website: The final website which will allow for end-users to view various stored data that has been published by the Parser interaction with ArcGIS for Server. This will be provided by ArcGIS for JavaScript 4.1 as an interactive 3D map. Users will be able to select individual months and years (potentially days in the future) and display content that will be dynamically pulled from our ArcGIS Server. All basemap information will be pulled from the Esri CDN (content delivery network) at no cost to our client.

3 Design

3.1 SYSTEM SPECIFICATIONS

- Our client will submit her data as a single ASCII format (see appendix 8.1). The data will be formatted in a way such that the plotted data of several maps can be represented on a matrix, with each compound making up similar sparsity on the matrix. In other words, different molecules will have similar densities (technically, 1 minus density – for the mathematically inclined) on the ASCII chart. Additionally, the filename for the ASCII file will follow a standardized naming convention that represents the compound and relevant timeline intervals.
- The website will be accessible by the client and any end-user, provided they are connected to the Iowa State VPN. Making the website available outside of the ISU VPN would be trivial on our end, but would require the proper vetting/red-tape from IT services.
- We assume that the client will entirely run this product off a university owned and provided server, and has no plans to make money off of the product (as agreed to in the ArcGIS Server Agreement). This is important primarily because we are using ArcGIS as an educational license, which is provided to us for no cost as long as it is solely utilized for educational and informational use.

3.1.1 Non-functional

- The parsing of uploaded ASCII data should take less than 15 seconds, provided the dataset is smaller than 50,000 points.
- The product shall check the server for new ASCII datasets every 15 minutes; meaning datasets should be viewable by all users within approximately 15 minutes of being uploaded by the client.
- The viewing (rendering) of selected data shall be generated in less than 30 seconds; meaning the viewable data sets will be loaded by the product in less than 30 seconds.
- Once selected for creation, a dataset should be automatically created (from a CSV file) into a map service (using our automated Python scripting) in less than 10 minutes. Note that this timing is purely subjective to the speed of the server that we are given and is hard to quantify without real-world experience. Furthermore, the ArcGIS API calls cannot be modified and are not parallelizable, so multiple ASCII files uploaded in short time will simply need to be queued. As such, this requirement is flexible as will likely be changed in a later revision of this document.
- The end-user should be able to choose the color scheme to display the various datasets in.
- When values are toggled, the values should be appropriately sized to the point that they will not cascade, or billboard, toward the user. The user should be able to differentiate between various points through color and/or size.
- The system will be able to seamlessly support 8 simultaneous users. Rationale: If more than 8 users attempt to use the service at the same time, global page load times will increase up to 30 seconds (as stated earlier). After load times exceed 30 seconds, users will begin to get error messages. This requirement is largely based upon the CPU power the University is able to allocate to our server.
- 2 users will be able to simultaneously browse the same map (same compound for the same month). Rationale: If more than 2 users attempt to view the same map at the same time, page load times for those users will increase up to 30 seconds. After load times exceed 30 seconds, those users will begin to get error messages. This requirement is largely based upon the CPU power the University is able to allocate to our server.
- Web server usage will take priority over the processing of newly imported compounds. In other words, if 8 users are trying to browse maps, then processing of any additional ASCII files will slow down until web usage diminishes. This ultimately means that performance targets for the processing of newly added data (as stated in earlier requirements) is contingent upon the server not being utilized at the time.

3.1.2 Functional

- The product shall allow the client to upload raw ASCII data to the server.
- The product shall parse all uploaded data from an ASCII file into a format readable by ArcGIS.
- The product shall automate the steps traditionally taken to create a map-service (typically done via ArcGIS Desktop) by utilizing Python to take the CSV files and create a map, layers, and finally, a publish a map service.
- The product shall allow end-users to select the compound on the relevant timeline they would like to view. These datasets will be ordered by year and month. If the client would

like to add more intervals or sorting methods for data, modifications must not require more than 8 staff hours from a coding standpoint.

- The product shall allow users to queue multiple datasets at the same time; meaning the end-user will be able to view multiple compounds at the same time (Example: CH_4 and CO_2 concurrently).
- The product shall be viewable from an Internet browser when accessed using the Iowa State VPN (stipulations for this are mentioned earlier).

3.2 PROPOSED DESIGN/METHOD

The core idea is to publish and display our data in a dynamic and easily automatable way. After speaking with the geographic information system (GIS) domain experts in Durham Hall, we determined that the best way to map our client's data would be through ArcGIS for Server. We decided that Esri's product would be a good option as they are the industry leader in GIS mapping services, and any future work done to the system should be easier for domain experts. Additionally, it will come at no cost to the client (assuming she only uses it for research purposes). Furthermore, ArcGIS for Server will allow us to do all of this data management, data manipulation, and object creation entirely on an Iowa State server, where we have total control. After creating our map services, we will be able to use the ArcGIS for Server RESTful API to load any hosted map service (3). Our current system parses ASCII into a CSV file which is then parsed into an Excel workbook that is readable by the GIS software. From here, the data has to converted to an ArcGIS table that can be imported into a geodatabase. At this point, ArcPy is utilized to create the appropriate layers on a template basemap and publish the map as an active map service on our ArcGIS server(5). Finally, our server-side Java scripts will create a template JavaScript-based webpage for our newly created map service. This can be easily automated as our map services will be created based upon a standardized naming convention.

3.3 DESIGN ANALYSIS

3.3.1 ArcGIS Online

ArcGIS Online is an online platform designed to easily create and share maps, scenes, apps, and layers created for ArcGIS for Web. The benefit of such a system is that it could easily extend ArcGIS for Desktop (something our client is familiar with) and would require little configuration or extra effort by the client. Ultimately, this idea was scrapped due to the exorbitant cost it would place on the University's budget (in the thousands). While an ArcGIS for Server license is traditionally quite expensive, we determined that we could obtain a license for free due to the educational nature of this project.

3.3.1.1 Parser 1

After collaborating with the Client about the layout of the ASCII files, a parser was written to parse her ASCII tables (see appendix 8.1) into a CSV file (see appendix 8.2) which could be hosted on the server and dynamically loaded as 2D points on a 3D map. The Parser was written in Java, and reads through the ASCII files one line at a time, calculating points from the origin provided in the ASCII file. For North America, the origin point is (latitude, longitude … 7, -169). All values which represent "no data" were intentionally left out to save bandwidth and processing time. The "origin" of these tables serves as the "lower left hand corner" of the .tiff images in which the client has been representing her data (see appendix 8.3).

3.3.1.2 Implementation of Modeling Without ArcGIS for Server

3.3.1.2.1 Using a CSV Layer via ArcGIS for JavaScript

After parsing all of the Data into CSV files, we attempted to dynamically load these files onto a 3D map using the ArcGIS for JavaScript Web API. We would quickly determine that loading such a large number of plot points is unacceptably slow (on the order of minutes for slower laptops). Additionally, the CSV layer type that we used only allowed ~17,000 points to be loaded this way before discontinuing the plotting of points (see Appendices 8.5 & 8.6). While we were correctly reading the client's data (can be inferred by comparing shape of colored .tiff (Appendix 8.2) to shape of output from Appendices 8.5 & 8.6), this idea was scrapped due to performance and technical constraints. Due to this unsuccessful implementation, we decided to look towards ArcGIS for Server for answers.

3.3.1.2.2 Using ArcGIS Online

After the unsuccessful attempt at using client-based functionality, we shifted towards modeling using ArcGIS Online (http://www.gis.iastate.edu/), a service that we were granted access to by the University. We attempted to create various layers, basemaps, data points, and tiles which could be published and hosted on the Iowa State ArcGIS server. After much time spent and no progress made, what we found out that we unable to create maps/layers dynamically using the ArcGIS Online and that ArcGIS Online is very expensive in terms of hosting. As a whole, our team was completely unsuccessful in creating, publishing, and hosting these objects. Had we been successful, we may have unintentionally cost the University thousands in ArcGIS publishing credits. In addition to that, hosting costs would have increased exponentially for the client due to the fact that every object we would have created would likely have been several gigabytes, and ArcGIS online has a payment feature which bills per hour for every 10 megabytes stored. After learning this, we continued communication with Josh Obrecht and Robin McNeely (ArcGIS Analysts at Iowa State). After a quick meeting with Robin, we were suggested to use ArcGIS for Server (which seems is not widely discussed online in the Esri's ArcGIS APIs). This product should suit all of our needs, and completely remove the expensive publishing and hosting with goes along with ArcGIS Online.

3.3.2 ArcGIS Server

3.3.2.1 Parser 2

As we analyzed the performance of our original Java-based parser, we realized that it took a while to parse the large ASCII data sets that were provided by our client. In an attempt to speed this up, we began work on a faster parser that works by eschewing the programmer-friendly Scanner calls and String object creation for byte-level parsing. The core idea here is to avoid using high-level function calls and stick to the most basic format supported by the computer. Additionally, we made sure to request a server with plenty of RAM so that we could buffer the entire ASCII file into RAM in one swoop and dump the entire file to the disk after total parsing of the data. While the new parser is difficult to programmatically read and understand (nature of opting for speed over modularity), our preliminary testing is that it is orders of magnitude (roughly 10 times faster) since it eliminates internal Java looping and cache misses of unbuffered disk calls.

3.3.2.2 Implementation of Automated Scripting

After our introduction to ArcGIS for Server, we put in a request for a Windows Server which would allow for us to host our ArcGIS for Server service. After the IT department set up the server, we were able to successfully move forward with the installation of ArcGIS for Server. As this server and software were recently acquired (November 2nd), extensive implementation has yet to take place. Our current prototype takes the parsed files that we have created, models them into our desired layer format (manually), and then hosts them as a map service using ArcGIS for Server. We then have to manually import the relevant map service link into our HTTP template that can load the map service onto a map using the ArcGIS REST API (via ArcGIS for JavaScript). Our ultimate goal is to automate this entire process using a combination of custom server-side Java and Python code and calls to ArcPy.

In short, ArcGIS for Server comes with a Python module named ArcPy which we will be able to use to automate the creation of an ArcGIS map service. We will then use ArcGIS for JavaScript to take the hosted map service and plot the points represented by the CSV file on a base-map provided by the Esri CDN. This map will then be given a layer which will take all of our CSV input and "plot" the points on said map for each zoom level on the map. A small example of our intended final result can be seen at: https://developers.arcgis.com/javascript/latest/sample-code/sandbox/sandbox.html?sample=get-started-layers (6) or at our website. This section will be updated with implementation in the next version of the Design Document, but a workflow of this process can be seen in Appendix 8.7.

4 Testing/Development

4.1 INTERFACE SPECIFICATIONS

4.1.1 Google Earth API

The original system we intended to use for modeling the data. This was the software suggested by the client and her technical adviser. While this would have been a viable option, the API and complete service of Google Earth has become deprecated and will be shut down at the end of 2016 (1).

4.1.3 ArcGIS for JavaScript 4.1

The option chosen as an alternative to Google Earth based on the recommendations of our adviser Jonny Wong and our client. Because Iowa State has an ArcGIS license and Esri is an industry leader in GIS software, we felt confident that should we come across any issue, it would be easy to get help. Furthermore, any modifications that would need to be made would be made easier for future developers. This platform (further explained in section 4.2.1) will serve as the "3D space" for our project, but all processing and hosting will be done by ArcGIS for Server and custom Python Scripting.

4.1.4 ArcGIS Server

The current system (further explained in 4.2.2) that has been chosen to create the modeling of the product. This will serve as the backbone of the project, and will allow for our hosting, publishing, and automation of ArcGIS.

4.1.5 Server Specifications

All map service implementation will be done on our Iowa State Server (on which ArcGIS Server is installed). These are the *minimum* hardware requirements for this server:

- 8 GB RAM
- 4 logical cores
- Disk space for each map service that needs to be hosted (yet to be determined).

These are the software requirements for the server:

- Windows Server R2 Update
- If the allocated server is in a virtual environment, it must be virtualized using Microsoft Hyper-V or VMware vSphere 5.0, 5.1, 5.5, or 6.

4.2 SOFTWARE

4.2.1 ArcGIS for JavaScript

ArcGIS for JavaScript (4.1) is an easily usable and free API which can be referenced from JavaScript source code (4). For our purposes, this API allows end users to visually load and interact with data represented in a map service through RESTful calls on our ArcGIS server.

4.2.2 ArcGIS for Server

ArcGIS for Server is a standalone software package that serves as an instance of an ArcGIS host. This software will allow us to dynamically create map services to represent the client's unique modeling data (2). This data will be modeled on the 3D space by the client, through ArcGIS for JavaScript, as mentioned above. This part of our project is currently under active development.

4.2.3 ArcGIS for Desktop

Standalone software that allows users to model and create maps. These maps can then be displayed as a local model or can, optionally, be published to a map server.

4.2 PROCESS

4.2.1 ArcGIS for JavaScript

4.2.1.1 Parser 1

All ASCII files are provided by the client in a single format. We make sure to test our implementation completely on accuracy (making sure that all values are at least looked at) and speed (how quickly it takes to get from start to finish). The results of this testing can be seen in Section 5.1.1.

4.2.1.2 Adding Data-Types

Since data-types are all called through the ArcGIS JavaScript API, we simply tested to make sure that all data that was being parsed by our program was being posted onto the same point (latitude, longitude) that is desired. Therefore, any data-type added was simply tested for accuracy of location. The results are explained further in section 5.1.2.1.

4.2.1.3 Loading CSV Files via ArcGIS for JavaScript

After parsing all of the data and choosing which data-type to render as, we tested the loading in terms of speed (how long the rendering takes) and accuracy (making sure all points get posted). The results are explained further in section 5.1.2.1.

4.2.1.4 Publishing Layers using ArcGIS for Server

Since layers are publishable objects, we were going to test the publishing of layers in terms of accuracy (correctness of data) and load time. As we have yet to find any online examples visualizing data on a global scale, we cannot project an exact time. Since tile loading happens on a granular level, we feel confident that initial page load will take less than 10 seconds, while more detailed tiles will load dynamically based on the end user's Internet connection (and our server's backbone).

4.2.2 ArcGIS for Server

4.2.2.1 Parser 2

The second implementation of the Parser is being tested in the same way as the first. Time will be crucial as the quicker this is done, the quicker the file will be available for being transformed into a hostable map object. As well as optimal speed, the Parser must also have 100% accuracy in transforming the ASCII files into a CSV format. Results of testing Parser 1 and Parser 2 can be seen in section 5.2.1.

4.2.2.2 Implementation of Automated Scripting

We have not implemented this section yet, therefore we will have to fill this out once completed. However, once implemented, we will be testing for time and accuracy. Time will be tested in 2 phases. Phase 1 will be how long it takes the automated Python Script to run from start to finish. Phase 2 will be how long it takes from the end of the automated script to publish a successful map service for deployment on ArcGIS for Server (how long until it shows up on the server). Accuracy will be tested in relation to the .tiff files. If the shape of the layering matches that of the .tiff file, then we will know that our implementation is correct.

5 Results of Successful Implementation/Testing

5.1 ArcGIS for JavaScript

5.1.1 Parser 1

The first version of the parser runs in roughly 18 seconds on a 6GB RAM, i5 Processor HP laptop. While the speed is not necessarily where we would like it to be at this time, it completes with 100% accuracy and exports all files into CSV format (see Appendix 8.2). Another implementation of this parser is currently in progress which has a completion time of roughly 2 seconds. This is explained further in section 5.2.1.

5.1.2 Implementation

5.1.2.1 Loading directly from a CSV file

After successfully parsing all of the data into CSV files, we attempted to dynamically load these files onto a 3D base-map. We would quickly determine that loading such a large number of plot points is unacceptably slow (on the order of minutes for slower laptops). Additionally, the layer type that we used only allowed ~17,000 points to be loaded this way before discontinuing the plotting of points (see Appendices 8.5 & 8.6). While we were correctly reading the client's data (can be inferred by comparing shape of colored .tiff (Appendix 8.2) to shape of output from Appendices 8.5 & 8.6), this idea was scrapped due to performance and technical constraints. After this unsuccessful attempt, we decided to seek further help from Robin McNeely and Josh Obrecht.

5.2 ArcGIS for Server

5.2.1 Parser 2

A timing analysis was done to check the performance of Parser 1 vs Parser 2 in 10 runs. What we found was that Parser 2 goes from start to finish in almost 1/10th of the time of Parser 1. The only current downside to Parser 2 is that the output is not entirely correct, so this will need to be fixed as soon as possible before we completely commit to this parser. The full analysis can be seen in Appendix 8.4.

5.2.2 Implementation

Has not been fully implemented for testing. Will update when applicable.

5.3 ArcGIS for Desktop

5.3.1 Map-Service Creation

Our current prototype takes the parsed files that we have created, models them into our desired layer format (manually), and then hosts them as a map service using ArcGIS for Server. We then have to manually import the relevant map service link into our HTTP template that can load the map service onto a map using the ArcGIS REST API (via ArcGIS for JavaScript). Our ultimate goal is to automate this entire process using a combination of custom server-side Java and Python code and calls to ArcPy.

6 Conclusions

As previously stated, we would like to provide the client with a platform that will allow her to display climate change on a 3D space. Our end goal is to parse the data into a format understandable by ArcGIS for Server and visually model it using ArcGIS for JavaScript. This JavaScript API will be called from a webpage where interested parties can dynamically load and interact with these hosted map servers. Doing this will not only assist the client in her research, but will assist potential stakeholders who are trying to understand the impact of human decisions on the environment.

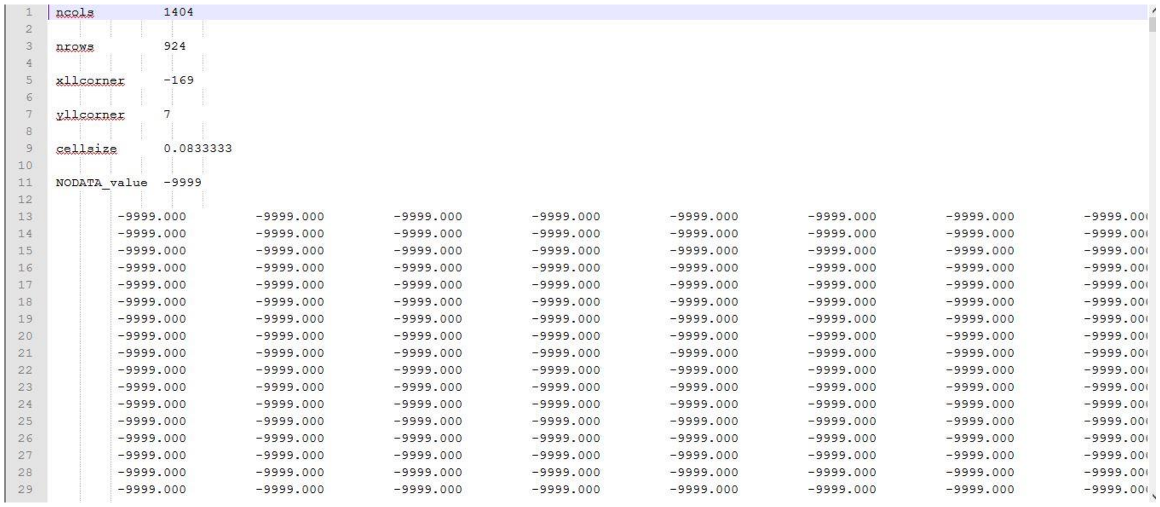
After learning about the deprecation of Google Earth and the limitation of ArcGIS Online, we have deduced the following: Google Earth was not plausible from the start due to the fact it will be entirely shutdown at the end of the year. ArcGIS Online (while effective) cannot meet our needs due to the cost. Our analysis has led us to the conclusion that ArcGIS for Server will be the best course of action for this task. By hosting and publishing our own map services, we will be able to inexpensively load, modify, and display the client's data. Additionally, the use of an ArcGIS server will ensure that visuals are precomputed and represented as cached image tiles that can later be called via the ArcGIS for JavaScript API. Our hope is that these RESTful calls will result in significantly reduced runtimes compared to our initial attempt of loading and rendering the data client-side, all at once. As a team, we are excited to move forward using ArcGIS for Server.

7 References

(1) "Google Earth API Developer's Guide | Google Earth API (Deprecated) | Google Developers." Google Developers. Google, n.d. Web. 04 Nov. 2016.
(2) "ArcGIS for Server | Features." Esri. Esri, n.d. Web. 04 Nov. 2016.
(3) "ArcGIS Server REST API." Resources.arcgis.com. Esri, n.d. Web. 04 Nov. 2016.
(4) "ArcGIS API for JavaScript | ArcGIS for Developers." ArcGIS for Developers. Esri, n.d. Web. 04 Nov. 2016.
(5) "Getting Started with Arcpy.mapping Tutorial." ArcGIS for Desktop. Esri, n.d. Web. 04 Nov. 2016.
(6) Esri. "ArcGIS API for JavaScript Sandbox." Esri, 19 Aug. 2016. Web. 04 Nov. 2016.

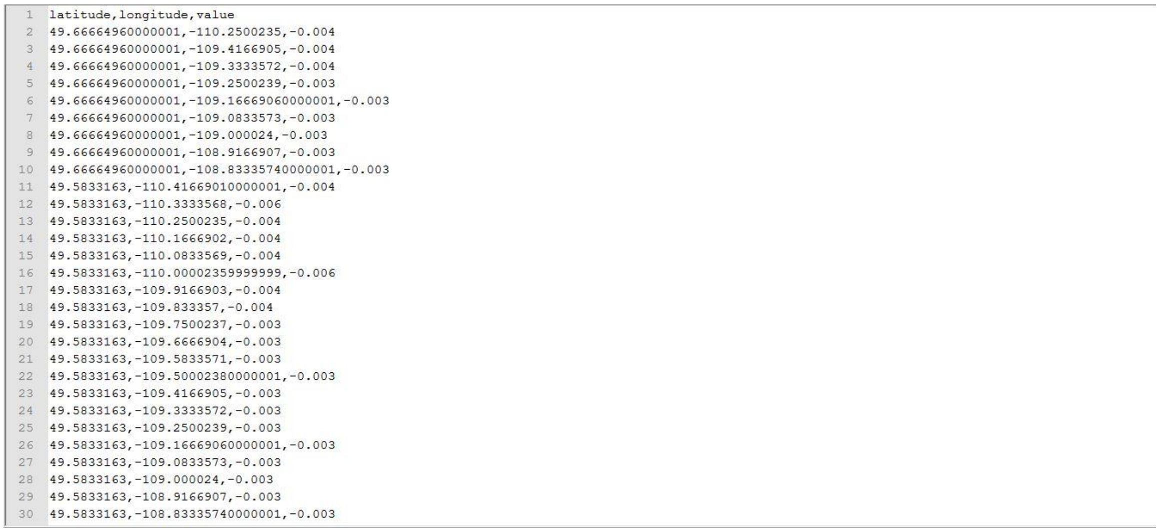
8 Appendices

8.1 – The ASCII Tables provided as raw data



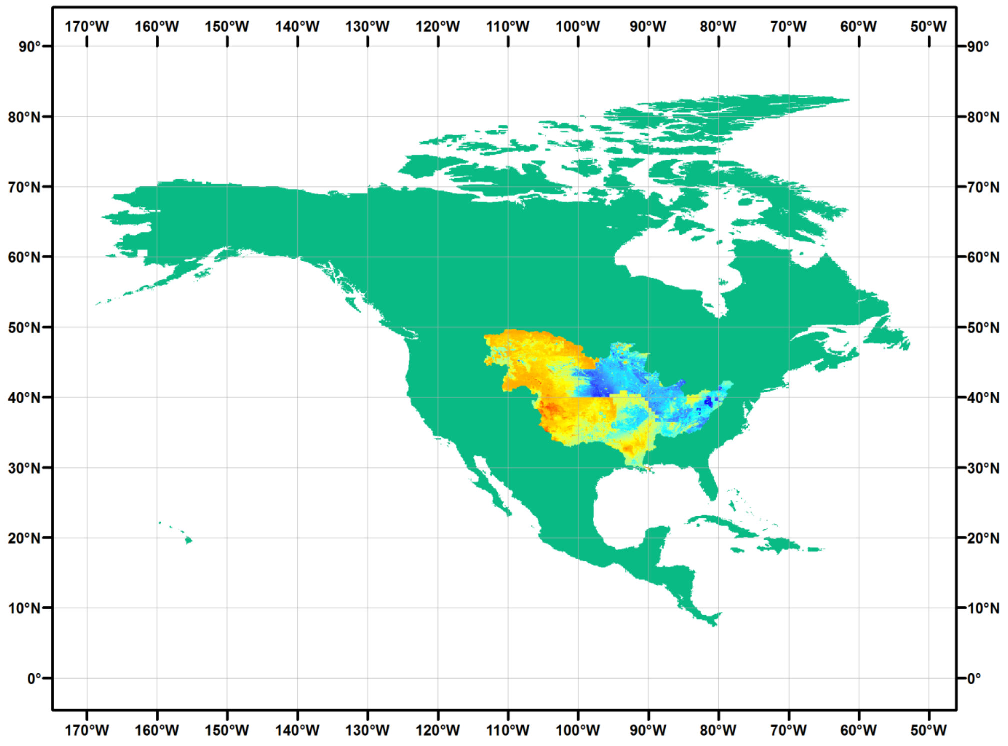
```
ncols         1404

nrows         924

xllcorner     -169

yllcorner     7

cellsize      0.0833333

NODATA_value  -9999

        -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.00
        -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.00
        -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.00
        -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.00
        -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.00
        -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.00
        -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.00
        -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.00
        -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.00
        -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.00
        -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.00
        -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.00
        -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.00
        -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.00
        -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.00
        -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.00
        -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.000       -9999.00
```

8.2 – Output (CSV) of First Implementation of Parser

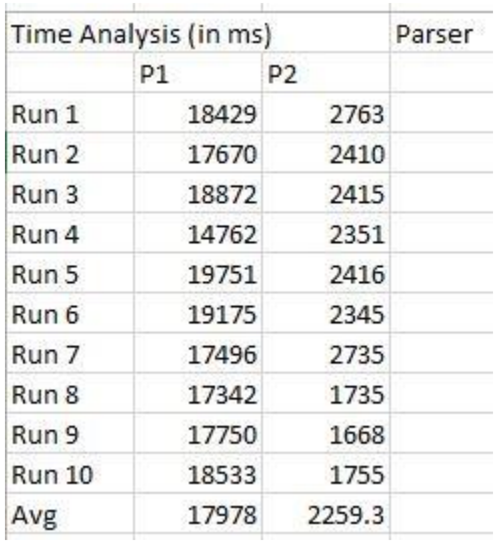


```
latitude,longitude,value
49.66664960000001,-110.2500235,-0.004
49.66664960000001,-109.4166905,-0.004
49.66664960000001,-109.3333572,-0.004
49.66664960000001,-109.2500239,-0.003
49.66664960000001,-109.16669060000001,-0.003
49.66664960000001,-109.0833573,-0.003
49.66664960000001,-109.000024,-0.003
49.66664960000001,-108.9166907,-0.003
49.66664960000001,-108.83335740000001,-0.003
49.5833163,-110.41669010000001,-0.004
49.5833163,-110.3333568,-0.006
49.5833163,-110.2500235,-0.004
49.5833163,-110.1666902,-0.004
49.5833163,-110.0833569,-0.004
49.5833163,-110.00002359999999,-0.006
49.5833163,-109.9166903,-0.004
49.5833163,-109.833357,-0.004
49.5833163,-109.7500237,-0.003
49.5833163,-109.6666904,-0.003
49.5833163,-109.5833571,-0.003
49.5833163,-109.50002380000001,-0.003
49.5833163,-109.4166905,-0.003
49.5833163,-109.3333572,-0.003
49.5833163,-109.2500239,-0.003
49.5833163,-109.16669060000001,-0.003
49.5833163,-109.0833573,-0.003
49.5833163,-109.000024,-0.003
49.5833163,-108.9166907,-0.003
49.5833163,-108.83335740000001,-0.003
```

8.3 – The original mapping method of the client

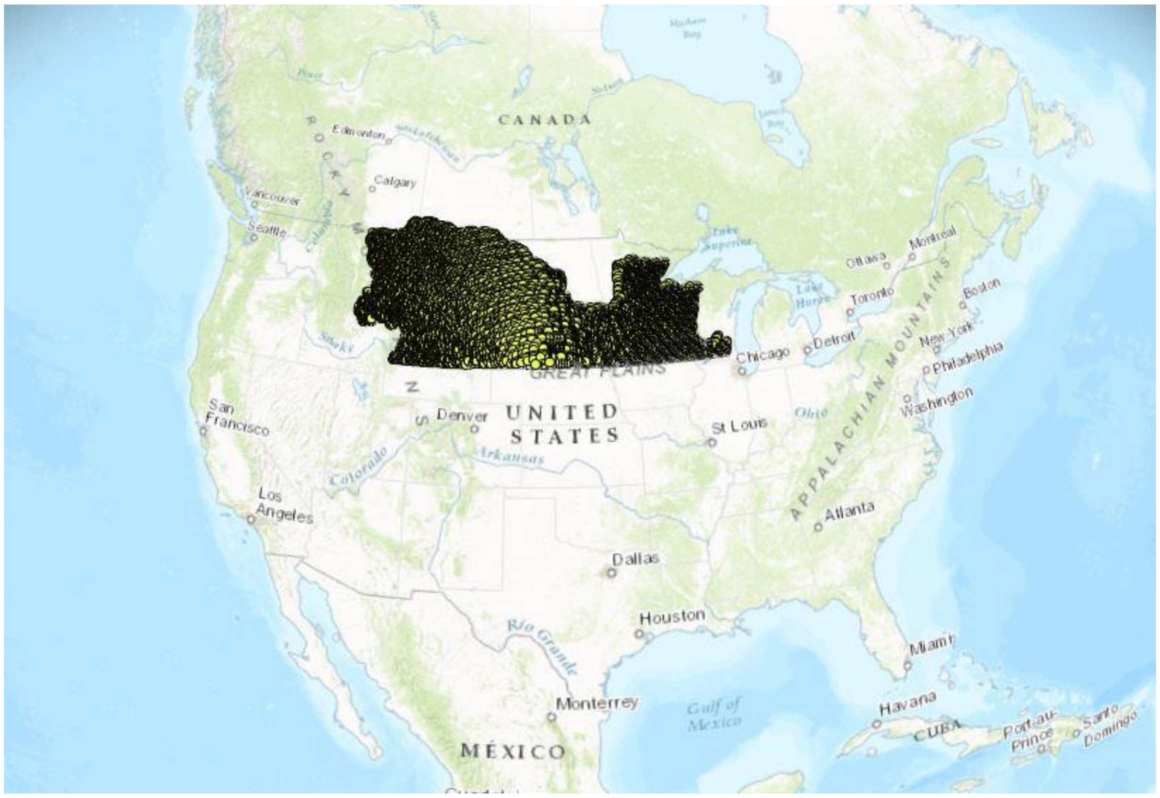


8.4 – Time Analysis of Parsers

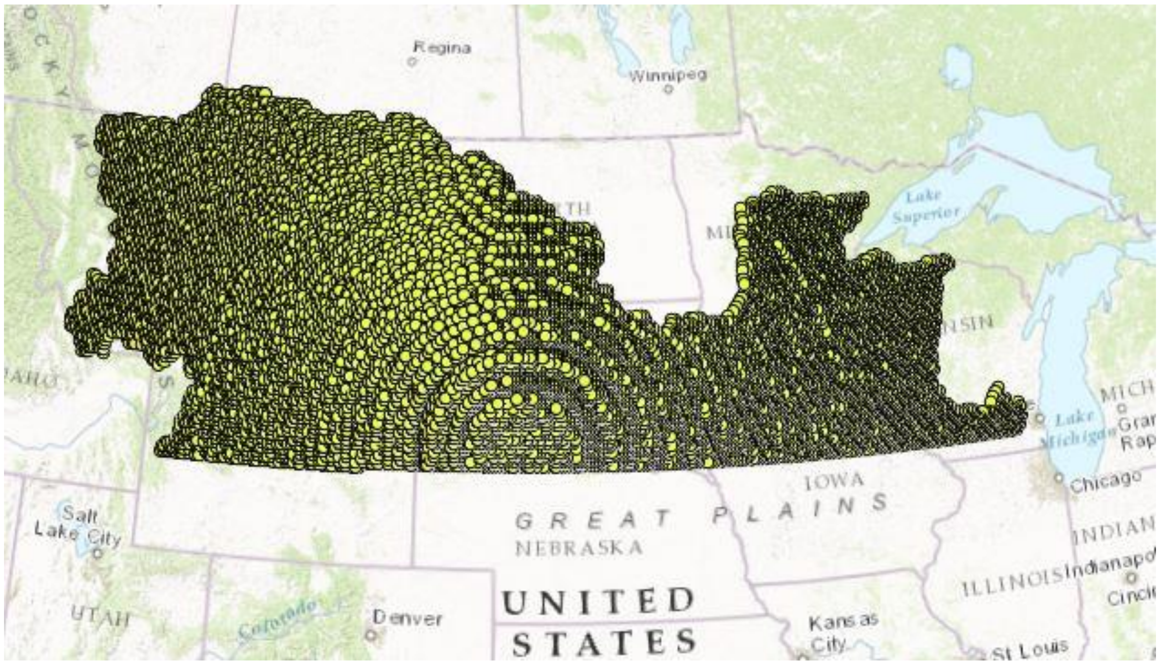


Time Analysis (in ms)			Parser
	P1	P2	
Run 1	18429	2763	
Run 2	17670	2410	
Run 3	18872	2415	
Run 4	14762	2351	
Run 5	19751	2416	
Run 6	19175	2345	
Run 7	17496	2735	
Run 8	17342	1735	
Run 9	17750	1668	
Run 10	18533	1755	
Avg	17978	2259.3	

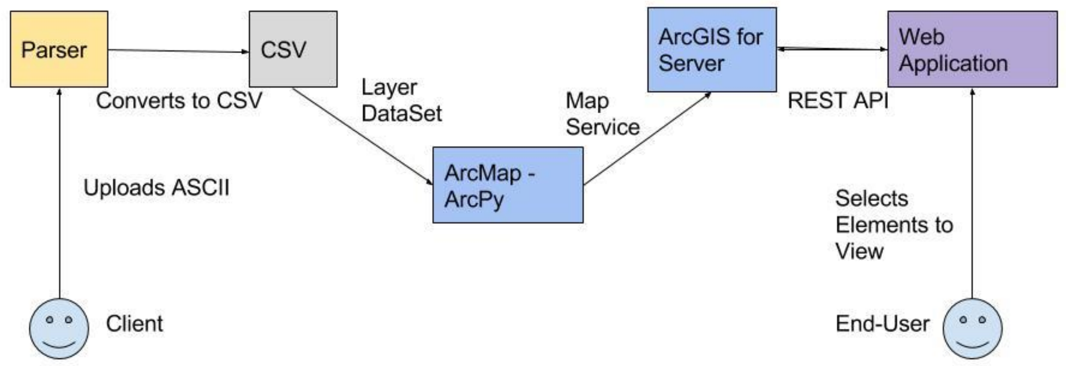
8.5 – Attempted CSV modeling using only ArcGIS Online

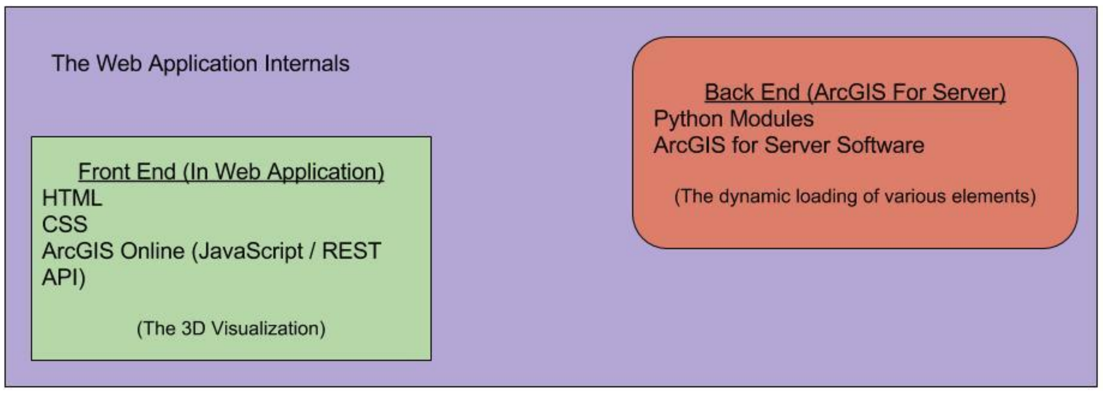


8.6 – Attempted CSV modeling using only ArcGIS Online (Zoom-In)

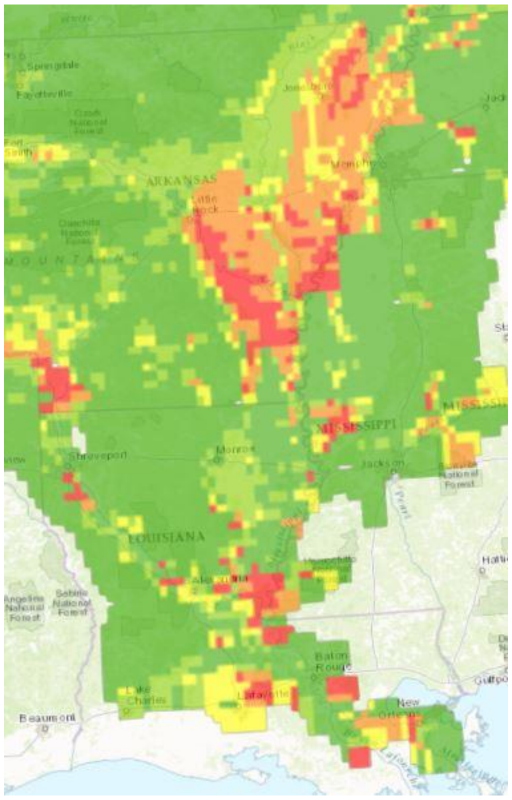


8.7 – Workflow / Block Diagram




The Web Application Internals
Front End (In Web Application)
HTML
CSS
ArcGIS Online (JavaScript / REST API)
(The 3D Visualization)
Back End (ArcGIS For Server)
Python Modules
ArcGIS for Server Software
(The dynamic loading of various elements)

8.8 – Prototype


Springdale
Fayetteville
Ozark National Forest
Fort Smith
ARKANSAS
Little Rock
Ouachita National Forest
Jonesboro
Memphis
MISSISSIPPI
Jackson
Shreveport
Monroe
LOUISIANA
Alexandria
Baton Rouge
Lafayette
Lake Charles
Beaumont
New Orleans
Gulfport
Angelina National Forest
Sabine National Forest