United States Geological Survey
Interactive Point Visualization
Final Report

Prepared by: Erin Bailey, Curtis Bilbrey, Alex Farmer, Tim Velgos
Prepared for: Dr. James Dean Palmer
Date Submitted: May 4th, 2015

Project Sponsor: Jay Laura
Faculty Mentor: Dr. James Palmer

Last Updated: May 3rd, 2015
Creation Date: April 22nd , 2015
Version: 1.0

Table of Contents

1. Introduction

From September 2007 to June 2009, the Japanese lunar orbiter Kaguya went into orbit around the moon to collect spectrometer data from its surface. This data was then given to the United States Geological Survey (USGS) for open distribution and scientific use. Until now, USGS has not had an efficient way to access, analyze, and visualize this data.

The Geographical Point Visualization (PointViz) project is a web application designed to view specific data retrieved from Kaguya. An example of a problem this system could solve would be using several different spectrometer attributes to identify iron deposits below the lunar surface. The project is composed of two key components: a database to store the large quantities of data, and a web application that allows users to customize the range of data they wish to see. Users will be able to view the point data in a graphical form, and use visualization techniques, such as a heatmap diagrams, to format the data and visualize connections and correlations that may not otherwise be apparent.

Throughout the development of the projects, there were several issues that had to be addressed. Streamlining the data between the database and the web application was a main issue, which we solve with Django and PyMongo for data transfer. Storing a large quantity of data was also a key concern, and is one of the key reasons that MongoDB was the decided database for this project.

This report will summarize the development processes, as well as the choices that were made to finalize the PointViz product.

2. Process Overview

Before starting the development process for our project, we decided to work in an Agile environment with weekly Scrum meetings with our mentor, Dr. James Palmer. Originally the main focus of our project was to create a database that would contain usable point data and their corresponding attributes. Later, it was decided that we would also build a web application that could connect to the database and retrieve point data to show to users.

The development of the PointViz project began with understanding the data that would be stored in the database. USGS provided a Python program that translated the data from its original .spc format into a series of arrays with attribute data, such as wavelength, radiance, and reflectance data. We then wrote a program to create documents within collections in our MongoDB database to contain the attribute data and connect it to latitude and longitude paired coordinates.

Once the data was successfully stored in the database, we developed the web application using the JavaScript library OpenLayers, as well as Django and PyMongo to send data across the web application and the database.

3. Functional Requirements

3.1.Adjust Settings

These requirements show how the user may adjust the web application's settings.

3.1.1.Zoom-in: The user may zoom in on the current map they are viewing by selecting the button on the left hand side of the image.

3.1.2.Zoom-out: The user may zoom-out of the current map they are viewing by selecting the left hand side of the image.

3.1.3.Panning: The user must have the ability to pan through the image while it is zoomed in on. The user must also be able to hold a click and drag the image in the direction they wish to pan.

3.2.Viewing an Image

These requirements show how the user may interact with the application images.

3.2.1.Select: Certain points on a map of points must be selectable in order to get a more detailed description of what the user is viewing.

3.2.1.1.Associated Attributes: Selecting a point must create a display of all attributes associated with that point as well as a line graph depicting the wavelength associated with that point.

3.2.2.Layer: The user must be able to select a specific layer on a heatmap in order to get a more detailed description of what the user is seeing.

3.2.2.1.Layer Attributes: Selecting a layer must create a display of all attributes associated with that layer.

3.2.3.Specific Hexagon Selection: The user must be able to select a specific hexagon in a bivariate hexbin map to get a more detailed description of what the user is looking at.

3.2.3.1.Attribute Display: Selecting a hexagon must pop out a display of all attributes associated with that hexagon, as well as a line graph depicting the wavelength associated with that hexagon.

3.3.Query Data

These requirements show how the user may query data from the database.

3.3.1.Drop Down Menus: The interface must contain two drop-down menus: the Mineral drop-down menu and the Visualization drop-down menu.

3.3.1.1.Mineral Drop Down Menu: The interface must contain a drop-down menu, where a user can select a mineral to query for. The user must then add that mineral to a list with an "add" button. The user can add as many minerals to this list as they wish.

3.3.1.2.Visualization Drop Down Menu: The interface must contain a second drop-down menu that allows the user to choose what kind of visualization they wish to see, including heat maps and bivariate hexbin maps.

3.3.2.Brush Components: The interface must contain three brush components.

3.3.2.1.Wavelength Brush Component: The interface must contain a range slider for the wavelength, so that the user may define a range of wavelengths that they wish to query.

3.3.2.2.Radiance Brush Component: The interface must contain a range slider for radiance where the user may define a range of radiance that they wish to query.

3.3.2.3.Reflectivity Brush Component: The interface must contain a range slider for reflectivity where the user may define a range of reflectivity that they wish to query.

3.3.3.Brush Ranges: The user must be able to define ranges for however many brushes they wish. For brushes not given ranges by the user, the entire range will be queried.

3.3.4.Brush Warning: If a range is not defined for any of the brushes, warning text will appear underneath the brushes they have selected. For brushes not given defined ranges by the user, the entire range will be queried.

4. Non-Functional Requirements

4.1.Scalability

The following non-functional requirements address PointViz's scalability.

4.1.1.MongoDB: This product must be build with MongoDB so that it may be maintained and expanded once the original developers have finished development.

4.1.2.Python: This product must be built using the Python programming language so that new developers that resume this project will be able to continue development once the original developers have left.

4.1.3.Database Schema: The schema of the database must be built such that similar data from other satellites may be added without requiring the schema to be altered.

4.1.4.Palette Functionality: The product must be built with palette functionality so that a user may add or remove palettes to adjust the query criteria.

4.1.4.1.Example: An example of this could be the reflectivity palette which would only display attributes relating to reflectivity.

4.1.5.Palette Implementation: Palettes should be easy to implement and separate from the rest of the system such that if a new palette needs to be added in the future, the user is not required to edit core functionality.

4.2.Usability

The following non-functional requirements focus on the ease of use for the PointViz project.

4.2.1.Reduction of Visual Clutter

These non-functional requirements involve minimizing data on the user's screen.

4.2.1.1.Filtering: This product must utilize filtering of point data in a way so that a minimal data set is used to represent all point data for a given image.

4.2.1.2.Simple: This must be achieved so the product may be as visually simple as possible.

4.2.1.3.Split: The data will be made visually simple by a filtering process where the data is split into a grid where only subsets of data within each grid component are used to visualize data.

4.2.2.User Interface and Human Factors

The following non-functional requirements focus on user interaction with PointViz.

4.2.2.1.Intended Users: The intended users are researchers and scientists, or others with a scientific background to understand the processed data.

4.2.2.2.User Training: The interface should be intuitive and easy to understand, and no additional resources should be needed beside the help document.

4.2.2.3.Browser Support: The PointViz web application will promise support for IE9, usability with other browsers is not assured.

5. Architecture

The architecture for the PointViz project is split between a Pipe-and-Filter architecture, as well as a Client-Server architecture, as shown in **Fig 5.1** and **Fig 5.2**. The technology stack includes the browser, the OpenLayers JavaScript library, the Django framework, the PyMongo python distribution, as well as MongoDB. This is shown in **Fig 5.3**.

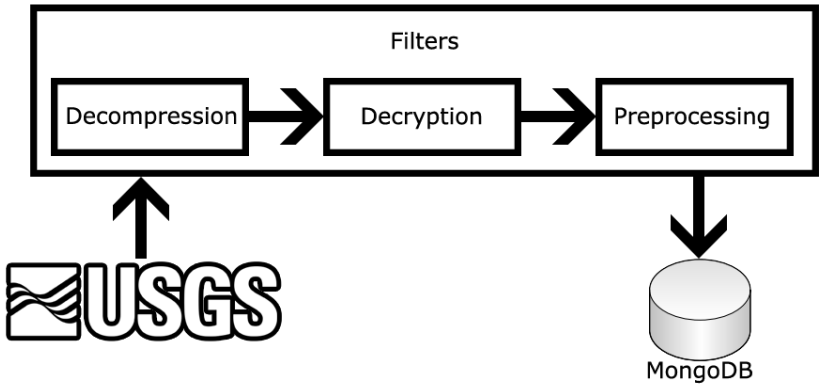

Fig 5.1

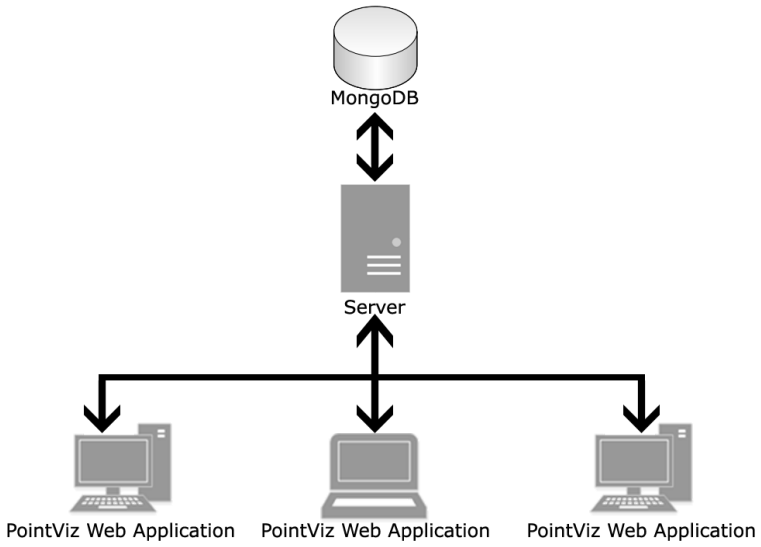

Fig 5.2

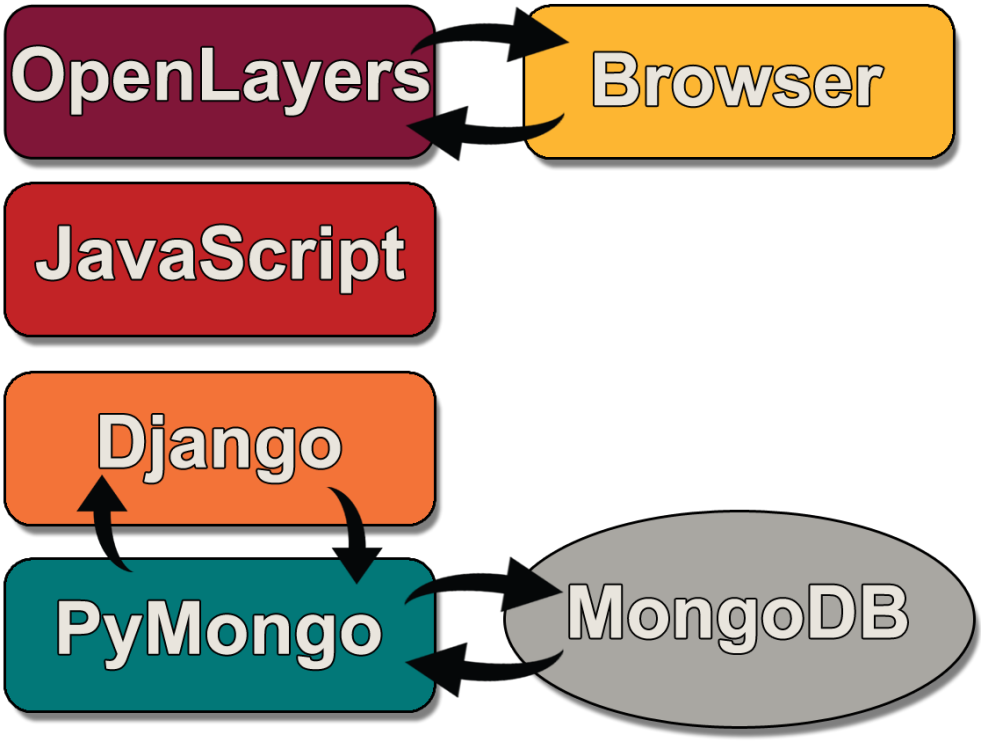

Fig 5.3

As seen in Fig 5.1, the PointViz project has a Pipe-and-Filter architecture concerning the data as it is imported into the database. The large quantities of data are decompressed and decrypted, and then preprocessed into sortable attribute data which is then imported into the database.

The web application shows Client-Server architectural properties through multiple web applications connecting to one server that hosts the database. Each web application, run through a browser, connects to the database and fulfils the Client-server model.

Fig 5.3 shows the technology stack that composes the PointViz project. The MongoDB database uses Django and PyMongo to connect to the web application, where the data is then visualized using the OpenLayers JavaScript library.

The architecture of the PointViz project is almost identical to how it was planned. The only difference would be that originally a MVC architecture was planned, but the end result leaned more towards a combination of Pipe-and-Filter and Client-Server.

6. Testing

The testing of the PointViz application consisted of Usability Testing, Integration Testing, and Unit Testing.

We conducted Usability Testing by giving users of varying technical skills an online survey after using the PointViz application, with questions asking about the ease of use, and providing room for any additional comments. Through this testing we discovered that the zoom feature was difficult for users on laptops without access to mice, and we implemented clickable plus and minus buttons to zoom in that would be more trackpad friendly.

Our Integration Testing focused on three main components: MongoDB, Django, and OpenLayers. To test MongoDB, we checked samples of data that were imported into the database against the values before being entered, to ensure accuracy. We also confirmed that a MongoDB instance was properly created, and if not, an error would be thrown. We then confirmed that each PyMongo query was constructed properly based on query inputs.

To test Django, we confirmed that the python portion of the project properly interacted with the JavaScript/HTML/CSS of the application. These instances were tested separately to evaluate whether they worked independently, and could coexist as independent entities. OpenLayers was then tested to confirm the lat/lon points are displayed on the correct lat/lon points on the map, and that the visualization successfully maps to the points. Finally, the visualization is checked for accessibility and intractability with the user interface.

We also conducted Unit Testing on the PointViz application, by testing our python scripts, the OpenLayers library, and our instance of MongoDB. For more specifics, please see the final testing document.

7. Future Work

USGS has confirmed that they will be using the PointViz application to further research on the data gathered from KSP. The project may be expanded upon to allow additional telescopes to store data within the database so that more detailed analysis can be made possible.

It is our hope that the project becomes a valuable tool for USGS, and that the project may be further refined to best suit USGS for their research.

8. Appendix A: Acronyms, Abbreviations, and Definitions

Term	Definition
JSON	JavaScript Object Notation
Radiance	The flux of radiation emitted per unit solid angle in a given direction by a unit area of a source.
Reflectivity	The property of reflecting light or radiation, especially reflectance as measured independently of the thickness of a material..
Phase, Emission, Incidence Angles	Together these angles describe the position of satellite in relation to the object it's orbiting.
Spectrum	An array of entities, as light waves or particles,ordered in accordance with the magnitudes of common physical property, as wavelength or mass.
USGS	United States Geological Survey
Wavelength	The distance between successive crests of a wave, especially points in a sound wave or electromagnetic wave.
Clean data	Data that has been post-processed through mathematical adjustment functions given by USGS

Dirty data	Data that has had no adjustment, and has been taken directly from the KSP

9. Appendix B: Products and Tools

Software/Tool	Version	Source	Description
MongoDB	2.6.5	http://www.mongodb.org/	A document oriented NoSQL database.
D3	3.4.13	http://d3js.org/	A Javascript library used for manipulating documents based on data.
Python	2.7.8	https://www.python.org/	An object oriented programming language.
JavaScript	1.8.5	N/A	
JSON	N/A	http://www.json.org/	Javascript Object Notation is a lightweight data-interchange format.