IMPROVED ACCESS TO KAGUYA HYPERSPECTRAL IMAGE DATA.

Z. Haugaard[1], D. Ohn[1], C. Philabaum[1], K. Rodriguez[12], M. Shepherd[12], J. Laura[2] and T. Hare[2], [1]Northern Arizona University School of Informatics, Computing, and Cyber Systems, [2]USGS Astrogeology Science Center. Corresponding Author: jlaura@usgs.gov

Introduction: From September 2007 to June 2009, the Japanese Aerospace Exploration Agency's (JAXA) "KAGUYA (SELENE)" carrying the Spectral Profiler (SP) sensor collected hyperspectral observations of the Moon's surface. The instrument was used to capture reflectance spectra from the visible and near infrared regions with broad spectral coverage and at high spatial and spectral resolution[1]. The collected dataset is made up of some 68 million individual spatially aware points with associated metadata[2]. This data can be used many science applications including understanding surface properties.

JAXA maintains the KAGUYA data archive, a critical tool for SP data access. The access mechanisms follow the familiar planetary paradigm of textual search followed by a listing of available data products. Unfortunately, data access using this mechanism suffers from three major issues. First, results are served such that products are aggregated by image. This aggregation is critical when providing context from the associated Terrain Camera (TC) image, but can be viewed as redundant because the individual point observations are of critical interest within the SP data set and, when served as an Open Geospatial Consortium (OGC) compliant layer, any context imagery can be utilized (assuming high quality registration and availability as an OGC compliant service). Second, the existing access mechanisms do not yet support map based spatial search requiring the user to define textual Regions of Interest (ROIs). Finally, the results are offered as paginated, individual downloads (aggregated by image), making the collection of data over larger spatial extents problematic. In order to support the ability to perform exploratory spatial data analysis with the SP data, we have sought to begin addressing these concerns.

We seek to address the previously identified issues by creating a multilayer (web stack) application for the delivery, visualization, and exploratory analysis of the KAGUYA SP dataset.

Architecture: The Kaguya Spectral Profiler Explorer (KASPER) software system relies upon a three-layer architecture. The three components are: (1) the frontend, written as a javascript based application that runs in the user's browser, (2); a middleware layer, which is an application server providing services to the frontend; and (3) the backend, composed of a containerized, OGC compliant, Geoserver instance backed by a containerized MongoDB database sharded across a cluster of nodes.

Every piece of this architecture is built upon standard technologies, such as Leaflet, Geoserver and MongoDB; and protocols and OGC compliant formats such as Web Map Service (WMS), Web Feature Service (WFS), and GeoJSON. This makes our solution highly reproducible and applicable to other datasets.

Backend: The backend is comprised of two components: A sharded MongoDB database containing the KAGUYA SP dataset, and a Geoserver instance which provides spatial visualizations of the dataset.

MongoDB was chosen as our database system for reasons of flexibility and scalability. In comparison to a traditional relational database system such as MySQL or PostgreSQL, MongoDB is highly malleable due to its unstructured nature, a feature that has allowed for the easy modification of the data schema during development. As the dataset is complete and data does not need to be added or updated, traditionally desirable traits such as fast writes are not necessary; the only requirement of our database system is that it supports multiple concurrent reads.

Geoserver, an OGC compliant, java-based geospatial server which generates and serves visualizations of the KAGUYA dataset. It allows for the creation of map tiles from geospatial queries to the database, which can be consumed by a mapping application. It publishes data in two forms: image data by way of a Web Mapping Service (WMS), and vector data by way of a Web Feature Service (WFS). As Geoserver receives requests from the middleware, it can dynamically create a new map layer by passing the query to the database and serving it to the client. Using this technique we maintain performance when working in a big data environment.

Middleware: The middleware layer of the KASPER software system has two primary functions: (1) it serves the static frontend application, and (2) it provides a set of Representational state transfer (REST) API services. The RESTful API services allow for the frontend to make resource requests through the middleware layer. These include queries that can result in GeoJSON responses for point data and hyperspectral reflectance data or calls to MongoDB and Geoserver to generate, load, and serve a new raster map layer.

The server is built using the asynchronous javascript runtime Node.js and the lightweight web framework Express. These technologies integrate naturally with MongoDB. Interaction between the middleware application server and Geoserver is handled by a custom Python script, which can generate new map tiles based on a user defined query. Node.js facilitates this by way of the standard library child_process, which allows for Node to spawn child processes on a per request basis to generate new map layers and to pass them back to the client.

Frontend: The frontend javascript application is responsible for displaying geospatial data via an interactive web map interface. The map interface has been developed using the open-source javascript mapping library Leaflet, and visualizations of hyperspectral data are rendered as HTML5 charts with the tool Chart.js. Both technologies were chosen based upon their accessibility, flexibility, and integration with the OGC compliant formats previously discussed.

The Leaflet mapping interface allows for the natural exploration of the dataset as one would any standard web map: users can pan across the map, zoom in and out, set coordinates manually through simple input fields, and click on individual observations to view associated data. Leaflet can also ingest and display new layers dynamically as the user interacts with additional control interfaces to send queries along such criteria as location, incidence angle, and emission angle.

When an observation is selected by the user via the Leaflet interface, specific hyperspectral data can be displayed. The spot data (whether reflectance, radiance, or a derived spectra) can then be rendered as a line graph using the Chart.js visualization library, as shown in FIG 1. Chart.js allows for these visualizations to be rendered on the fly, and offers interactivity such as hovering to display the exact reflectance value at any given wavelength.

As the user interacts with the frontend javascript application, data is requested and displayed asynchronously. This allows the frontend application to maintain a high level of interactivity despite the large amounts of data it provides access to.

Conclusion: The KASPER software system, as outlined above, is a powerful tool that allows for intuitive access to the KAGUYA SP dataset through an web based user interface that rests upon standards compliant technologies (e.g., Leaflet, Node.js, OGC compliant geoserver, MongoDB). KASPER is able to deliver the SP data quickly and provide an interface for the exploratory analysis of the dataset thanks to its novel architecture, which streams large amounts of data through the system dynamically and across multiple formats.

The KASPER system can be installed with relative ease, as its components are all highly portable. The backend database and Geoserver instance exist in a software container (Docker).. The middleware software server can be installed on any standard operating system compatible with the Node.js javascript runtime, and all dependencies may be installed using the Node Package Manager.

References: [1] Matsunaga, et al. (1999) *J. Remote Sens. Jpn*. 490-507. [2] Yamamoto, et al. (2011). *IEEE Trans. Geosci. and Rem. Sens.* 4660-4676.