**NASA DEVELOP National Program**



Langley Research Center

*Summer 2015*

CALIPSO Cross-Cutting

Interfacing CALISPO Data through a Graphical User Interface

 **Technical Report**

Rough Draft – June 25, 2015

Grant Mercer (Project Lead)

Nathan Qian

Jeff Ely, NASA DEVELOP, National Program (Science Advisor)

Dr. Kenton Ross, NASA DEVELOP National Program (Science Advisor)

Previous Contributors:

Jordan Vaa

Courtney Duquette

Ashna Aggarwal

# I. Abstract

[Placeholder - do not put anything here until the final draft submission. The abstract in the project summary is where the working draft of the abstract should “live”]

**Keywords**

CALIPSO, CALIOP, Lidar, Atmospheric Aerosols

# II. Introduction

Background Information:

Earth’s climate is a complex system that involves many variables and factors. One such factor are aerosols. Aerosols can come from anthropogenic (pollution or biomass burning) or natural (dust, sea salt, and volcanic) sources (Omar et al. 2009). Once in the atmosphere, aerosols can affect cloud formation and cloud radiative properties. Additionally, aerosols can suppress and enhance precipitation, simultaneously affecting the water cycle as well as radiative heating (Winker et al, 2010). Besides environmental impacts, aerosols can negatively affect human health in heavily populated regions (Winker et al, 2003). Unlike greenhouse gasses, aerosols are a major source of uncertainty in climate models due to their variable distribution and properties (Winker et al, 2003). Because of these effects, Earth scientists track and identify aerosols to analyze the aerosols’ impact on human health and Earth’s climate

In April 2006, the Cloud-Aerosol LiDAR Infrared Pathfinder Satellite Observation (CALIPSO) was launched to provide measurements of clouds and aerosols and their interactions and roles in the climate system (Winker et al, 2010). CALIPSO carries three instruments for aerosol measurement. The primary instrument is the Cloud-Aerosol LiDAR with Orthogonal Polarization (CALIOP), a new-nadir viewing two-wavelength polarization-sensitive LiDAR (Winker et al, 2009). The remaining two are passive sensors: a wide field camera and an infrared imaging radiometer (Winker et al, 2009). CALIPSO can determine the aerosol’s type through an algorithm by utilizing backscatter and volume depolarization ratio measurements (Omar et al. 2009). The algorithm takes altitude, integrated attenuated backscatter, location, surface type, and volume depolarization ratio as parameters (Omar et al 2009). With the addition of CALIPSO, researchers have access to more aerosol data and can better analyze aerosols’ effect on climate.

Prior to the spring of 2015, the CALIPSO science team used an IDL program written in 2007 to display CALIPSO data. Since 2007, the program has been maintained and updated with improvements. However, since IDL is a proprietary language, users outside of NASA cannot easily use not customize the CALISPO visualization tool. In 2010, a Python program, called CloudSat and CALIPSO plotting tool (ccplot), was written to mimic the IDL program to display CALIPSO data and was used by the CALIPSO science team. Unlike IDL, Python is an open source language, which allows users to more easily use and change programs written in Python. Despite the conversion to Python, ccplot proved to be obtrusive as CALIPSO data is displayed as an unadaptable image and can only be manipulated through a command line. In the spring of 2015, a team at NASA DEVELOP started development of a new CALIPSO visualization tool that will address the issues of the previous CALIPSO visualization programs. The spring term produced a working piece of software that displayed CALIPSO data, however, the team were unable to finish implementing full user interaction with CALIPSO data. For this term, our team intends to finish where the spring term left off.

Project Objectives:

For the summer term, our team plans to finish implementing missing features left over from the spring term’s project and release the CALIPSO visualization tool. The new features added will allow users to draw shapes that mask specific aerosols and share these shapes through a database. Specifically, a user can trace a shape or multiple shapes in the backscattered or depolarized plots from CALIPSO data that represents a specific aerosol. The user can then tag the shapes with relevant data, such type of aerosol, the aerosol’s source, or the aerosol’s composition. When the user exports the shapes to the database, the shapes are saved in a JSON file with additional metadata, such as the time of creation and the corresponding CALIPSO data product file. When the shapes are saved in the database, the user can retrieve them using queries based on the CALIPSO data product or user generated tags. The code base left over from the spring term already displays L1 and L2 data products from CALIPSO. However, our team will make additional improvements, such as bug fixes and code documentation, to make the CALIPSO visualization tool more reliable and user friendly. With these additions, researchers can better collaborate on aerosol tracking and identification.

Study Area:

The scope of this project is global, as the CALIPSO visualization tool can read any CALIPSO data.

Study Period:

The timeframe of this project ranges from 2006, when CALIPSO was launched, to the present day.

National Application:

Our end product will help with the health and air quality application area. The CALIPSO visualization tool will aid researchers in identifying, tracking, and documenting aerosols and their effect on the environment and air quality.

Project Partners:

Our end user is the CALIPSO science team. Our main point of contact with our users are Dr. Charles Trepte and Dr. Amber Soja, who provided guidelines on the CALIPSO visualization tools features and use cases. The shape drawing and database functionality of the new CALIPSO visualization tool will help the CALIPSO science better track and identify aerosols and let them more easily share aerosol research with each other.

# III. Methodology

Data Acquisition:

The CALIPSO visualization tool reads L1 and L2 CALIPSO data products. The data products are saved in Hierarchical Data Format (.hdf), a specific file type used for storing large scientific data. CALIPSO data can be acquired from the Atmospheric Science Data Center (ASDC).

Data Processing:

When the user launches the CALIPSO visualization tool, the program displays a blank screen, awaiting further user instruction. The user can either import CALIPSO HDF files to create new shape objects, or import preexisting shape objects from the database to examine or edit. If the user loads CALIPSO HDF data, the user is prompted by a file dialog that explores the local file directory to retrieve the requested HDF file. After the selection, the user can select which plot to display (backscattered, depolarization ratio, or vertical feature mask). If user selected the incorrect plot or needs to switch, the user can easily select another plot to display. The plot is then shown, and the user can draw shapes specific features of the plot or load shapes from a previous session. Whenever the user moves the cursor, the program displays and updates the coordinates to help with precision plotting. After a shape is drawn, it can be clicked and dragged to relocate it within the plot. If the user makes a mistake with the shape or no longer needs it, the shape can be erased. If the screen becomes too crowded with shapes, the user can temporarily hide the shapes to help focus on the plot. Besides the drawing functionality, the user has access to some navigation features. The user can zoom in on specific areas of the plot and can pan the plot to center a specific subsection of the plot. Lastly, the user can undo and redo navigation operations. If the user wants to clear navigation history and all drawings on the plot, the user can easily reset the plot with the press of a button. When the user is finished drawing, the user can save the shapes, which is stored locally as a JavaScript Object Notation (JSON) file or export them directly to the database. While the shapes are being saved, metadata such as the HDF file name and time of creation is included in the JSON file.

When the user loads shapes from the database, the user is prompted by a dialog displaying all the shapes in the database. The user can filter the shapes using queries or user-generated tags. After selecting which shapes to use, the user can then download the shapes for personal use with access to the same functionality mentioned before.

To support all the features in the CALIPSO visualization tool, the program followed a strict object-oriented design. Several classes were created that handled the internal workings of the program to handle the various use cases that user can potentially create. Figure 1 shows the structure and layout of the CALIPSO visualization tool.

Figure . Needs a caption and should be cited in text

The main class of the program is the PolygonList. It maintains and aggregates a list of PolygonDrawer objects for each of the plots (backscattered, depolarized ratio, and vertical feature mask). Additionally, PolygonList has another list that keeps track of the current plot and utilizes aliasing to update the list as additional objects are drawn onto the plot. The last item of the list is a blank PolygonDrawer that will update when a user draws a shape. Whenever the user draws changes a shape (deleting, recoloring, moving, etc.), PolygonList will determine which shape the user requested and make the necessary change. In addition to PolygonDrawer, PolygonList has a PolygonWriter and a PolygonReader and signals DatabaseManager. PolygonWriter, PolygonReader, and DatabaseManager are used in PolygonList to load and save shapes from JSON files and the database.

Within the PolygonList, the PolygonDrawer class stores the internal attributes of each shape. Every shape a user draws is an instance of a PolygonDrawer. PolygonDrawer saves attributes such as the coordinates of the shape’s vertices and the shape’s color. PolygonDrawer then calls the canvas to draw the appropriate shape by passing in the attributes. After the shape is drawn onto the canvas, PolygonDrawer continues to maintain the internal variables. When the user prompts the program to save the shapes, PolygonList will request PolygonDrawer’s internal variables to pass onto DatabaseManager or PolygonWriter for saving. When the user loads shapes into the program, PolygonList will read in the necessary information from PolygonReader and DatabaseManager and PolygonDrawer will save it and draw corresponding shape.

For file input and output (I/O), PolygonReader and PolygonWriter handle JSON reading and writing for the program. When the user loads a JSON file, PolygonReader will open the requested JSON file and feed its information to PolygonList, who will create PolygonDrawers for each loaded shape. PolygonWriter operates in the opposite manner. When the user saves the shapes into a JSON file, PolygonList will take the internal attributes of all PolygonDrawers and feed them into PolygonWriter. PolygonWriter will then open or create the requested JSON file and save the shapes. If name of the JSON save file already exists, it will overwrite the existing data. Both of the classes have I/O exception handling for common I/O errors, such as if the requested file does not exist, if the requested file is restricted, or if the requested file is either read or write only.

Like the two I/O classes, the DatabaseManager class handles shapes from input and output sources. The DatabaseManager has a dbPolygon, which holds the information represented in PolygonDrawer for DatabaseManager to convert into a SQLite entry or vice-versa. The DatabaseManager makes extensive use of SQLAlchemy, which encapsulates SQLite queries and management by providing an object oriented interface consistent with the rest of the class structure. When the user saves shapes to the database, the PolygonList class will pass the relevant information over to DatabaseManager for saving. Likewise, when the user loads shapes from the database, DatabaseManager will pass the shapes’ information to PolygonList, who will then save the data for local use. The DatabaseManager saves all entries onto a database where any researcher can access to download or upload their own shape files.

Unlike the previously mentioned classes, the Calipso directly interfaces with the user. The Calipso class displays the graphical user interface (GUI). The GUI displays CALIPSO data, the different CALIPSO data plots, and the shapes the user has drawn or loaded. The Calipso class has a PolygonList which maintains all the shapes drawn and interfaces with local and database I/O for saving and loading shapes. Here, the user can draw shapes, zoom in the plot, pan around the plot, and saving and loading shapes. Additionally, the Calipso class interfaces with other auxiliary classes not displayed in Figure 1. These ancillary classes read the HDF data, draws the appropriate plot onto the screen, and populates the screen with menus, buttons, and toolbars for the user to use. Since the Calipso class is responsible for displaying the graphics of the program to the user, it is also responsible for booting the program upon launch.

The entire program was written in Python 2.7. The CALIPSO visualization tool relies on several Python libraries and dependencies. The GUI utilizes the Tkinter library to draw shapes and display interface objects on the screen. For line calculations in shape drawing, the program used numpy methods. Database management relies on functions supplied by SQLAlchemy. The program imports ccplot and matplotlib to read CALIPSO HDF files and plot the data onto the screen. During development, the program was written using the Anaconda Python distribution, as it natively includes Tkinter and numpy. However, the standard Python distribution can be used so long as the required dependencies are installed.

[TODO: add installation process]

# IV. Results & Discussion

Insert images, graphs, maps, charts, etc. here. Choose the most important results to highlight here. No word cap, but two to six pages is a good range.

Things to discuss:

* Analysis of Results: What can you tell from your graphs, images, etc.? What does this mean for your project?
* Errors & Uncertainty: What factors could you not account for, what things didn’t work out like you expected they would, etc.
* Future Work: If this project was to be selected for another term, what would be the focus? What other areas would be of interest?

# V. Conclusions

Final conclusions. Word count: 200-600 (~a page).

# VI. Acknowledgments

Insert here. Keep to a concise paragraph or bullets of names. End with the following sentence.

This material is based upon work supported by NASA through contract NNL11AA00B and cooperative agreement NNX14AB60A.

# VII. References

Omar, Ali H., David M. Winker, Mark A. Vaughan, Yongxiang Hu, Charles R. Trepte, Richard A. Ferrare, Kam-Pui Lee, Chris A. Hostetler, Chieko Kittaka, Raymond R. Rogers, Ralph E. Kuehn, and Zhaoyan Liu. "The CALIPSO Automated Aerosol Classification and Lidar Ratio Selection Algorithm." *Journal of Atmospheric and Oceanic Technology J. Atmos. Oceanic Technol.* 26.10 (2009): 1994-2014. *Web of Science*. Web. 17 June 2015.

Winker, D. M., J. Pelon, J. A. Coakley, S. A. Ackerman, R. J. Charlson, P. R. Colarco, P. Flamant, Q. Fu, R. M. Hoff, C. Kittaka, T. L. Kubar, H. Le Treut, M. P. Mccormick, G. Mégie, L. Poole, K. Powell, C. Trepte, M. A. Vaughan, and B. A. Wielicki. "The CALIPSO Mission: A Global 3D View of Aerosols and Clouds." *Bulletin of the American Meteorological Society Bull. Amer. Meteor. Soc.* 91.9 (2010): 1211- 229. *Web of Science*. Web. 17 June 2015.

Winker, David M., Jacques Pelon, and Patrick McCormick. “The CALIPSO mission: Spaceborne Lidar for Observation of Aerosols and Clouds.” *SPIE Proceedings* 4893 (2003): 1-11. *SPIE*. 24 Mar. 2003. Web. 17 June 2015. <http://proceedings.spiedigitallibrary.org/>.

Winker, David M., Mark A. Vaughan, Ali Omar, Yongxiang Hu, Kathleen A. Powell, Zhaoyan Liu, William H. Hunt, and Stuart A. Young. "Overview of the CALIPSO Mission and CALIOP Data Processing Algorithms.” *Journal of Atmospheric and Oceanic Technology J. Atmos. Oceanic Technol.* 26.11 (2009): 2310-323. *Web of Science*. Web. 17 June 2015.

# VIII. Content Innovation

In preparation for DEVELOP’s coming microjournal, please select two content innovation features to support your paper. For each item, please list the name of the feature, and include the tool itself if possible (eg. glossary terms and definitions). If the tool does not work in Microsoft Word (eg. Interactive MATLAB Figure Viewer), please list the file name and upload the related file to the microjournal folder on the DEVELOP Exchange. If you choose to use Inline Supplementary Material, please also include where the material should appear in the text.

**Some options include:**

AudioSlides

Database Linking Tool

Data Profile

Executable Papers

Featured Author Videos

Featured Multimedia for this Article (video and podcast options)

Glossary Viewer

Inline Supplementary Material (figures, tables, computer code)

Interactive Map Viewer

Interactive MATLAB Figure Viewer

Interactive Plot Viewer

Nomenclature Viewer

# IV. Appendices

Insert here