
Web Based Visualization Tool for Climate Data Using Python

Hannah Aizenman * and Michael Grossberg

Glasslab, City College of New York, New York, New York

David Jones and Nick Barnes

Climate Code Foundation, Sheffield, United Kingdom

Jason Smerdon and Kevin Anchukaitis

Lamont-Doherty Earth Observatory, Columbia University, New York, New York

Julien Emile Geay

Department of Earth Sciences, University of South California, Los Angeles, California

ABSTRACT

Many scientists need to provide public visualization and exploration tools for the datasets they create. For instance, the Common Climate Project's (CCP) mission of building a website for sharing climate data required a tool to visualize that data. Unfortunately, although there are web sites that have custom built solutions, there are no widely available simple, Open Source tools for deploying a web application to facilitate exploring time series and spatial components of climate data online. So, as part of a Google Summer of Code Project, we developed an Open Source web based embeddable visualization application that extracts data from NetCDF files and then displays it in a variety of user specified ways.

CCPviz has a highly modular architecture to maintain its flexibility because it was designed to handle many different types of data (paleoclimate, proxy, reanalysis, etc), and meant to be embedded in various websites. Data extraction, processing, and visualization are handled separately in an underlying library that relies on NumPy, SciPy, and Matplotlib. The server side code, which is written using the Pyramid web framework, then provides a RESTful interface to this library so that each URL can map to an image and so that the user has flexibility in how the images are generated. The tool also provides a minimal HTML/JavaScript GUI interface so that scientists can embed the viewer in their websites, but even the HTML, CSS, and JavaScript are heavily separated so that the interface can easily be styled to fit within a larger website. The software was developed using standard Open Source software tools on hosted Internet servers, making new code easily and frequently available for review and release. Python was used because the prevalence of Open Source libraries and frameworks for scientific computing, data visualization, and web development greatly simplified integrating the various layers of the project. Python was also used because the diversity of Python software facilitates the incorporation of new features or tools in the same family as CCPviz.

A demonstration of this tool is available at <http://134.74.146.36/hannah/CCPviz.html>

1. Introduction

Large datasets of climate and weather data are being produced from multiple sources. With cheaper and increased computer power, and cheaper disk storage, it is now possible to more easily create long model simulations at higher resolutions. Increasingly high resolution satellite data and reanalysis projects which blend both

model prediction and measurements create readily available large uniformly sampled spatio-temporal resolution datasets. Measured data from networks of ground stations and proxy data from trees, glaciers, oceans, corals, fossils, and historical records are also becoming more accessible, yielding sometimes irregularly sampled data that provides a host of information on climate variability.

The great size of many of these data sets mean that just providing a download link does not really make the data accessible to the public or even other scientists. It is important for the scientists who produce the data to have a way to make it explorable. Providing a visualization tool through a web browser is one easy way to do this. Several sites have developed custom tools for serving their data. In some cases you can upload your data and then access a web-based visualization tool on the site, but there are few solutions if you want to serve and embed your data into your web site with a visualization tool.

Our motivating example came from the need for the Common Climate Project's mission to extend their web site with a tool for sharing and visualizing climate data. A requirement was that the tool should be easily embedded into an existing web site, should be based on Open Source technology, and be available under a BSD style type license. It should be able to map a physical variables like pressure or temperature, spatially, along with a color bar. It also should be possible to easily create time series plots for particular locations. Whether the data is viewed temporally or spatially, the domain of interest should be selectable by the client. In addition, the tool should read a broadly used climate data format standard NetCDF. No such technology was available.

As a Google Summer of Code project, we have developed a web application with a highly modular architecture. It was designed to be flexible enough to handle many different types of data (paleoclimate, proxy, reanalysis, etc). Data extraction, processing, and visualization are handled on the server using the Python scientific and visualization stack of NumPy, SciPy, and Matplotlib and written as a library. This library is exposed as a RESTful interface using the Python Pyramid web framework. Thus one of the great advantages of using Python is ease of software maintenance since only one programming language is needed from the point data is read from disk through calculation to the point the server produces a response.

The client side technology is a mixture of HTML, CSS, and JavaScript so that scientists can embed the viewer in their websites. The HTML, CSS, and JavaScript were modularized so that the interface can easily be adapted to fit within a larger website. The web framework can be run with a web server like Apache, or a reverse proxy like Nginx and we present the data through a RESTful API using the JSON standard for serialization. As a result, the server technology can be integrated into a site using PHP, Java or any other server technology. Nevertheless, the Pyramid web framework is a major Python web framework and can also be used to build a more extensive tool.

In this paper we will describe the motivation, design and process that was used to develop this software. The software was developed as part of a Google Summer of Code project. The code has been made available on Google

Code and a demo is running at <http://134.74.146.36/hannah/CCPviz.html>.

2. Prior Work

While there are many tools for visualizing climate datasets, most of these are either desktop applications or websites that provide tools for their locally hosted data. Some of these websites aggregate datasets from many institutions, such as Incorporated Research Institutions for Seismology Data Management Center (2011) and EPIC, which was created by Denbo et al. (2006). Other institutions provide a tool to explore data they created, such as the Earth System Research Laboratory (2011), but since all these tools are site specific, a researcher would either have to use the datasets on the website or submit their datasets for uploading (assuming their dataset meets the submission criteria for an aggregation site).

For data on a local machine, a scientist also has the option of using one of the many desktop applications available for visualizing data. Tableau Software (2011) makes tools that can create beautiful graphs and serve these visualizations, but it is closed source commercial software which limits the ability to extend and share. The CDAT suite of libraries, developed by the PCMDI Software Development Group (2007), GrADS, Institute of Global Environment and Society: Center for Ocean-Land-Atmosphere Studies (2003-2011), and Ferret, Hankin et al. (1996), are Open Source and optimized for climate data sets and provide libraries and other tools for visualization. They are not embeddable web apps and thus do not represent a complete web based solution.

Live Access Server (LAS), created by Hankin et al. (2001), and DChart, also created by Sirott (2006), are two Open Source tools that serve data using openDAP and provide simple visualization tools. DChart serves local data and provides a web interface, but lacks customizable visualizations and interfaces. In many ways Live Access Server extends DChart, allowing the user to fill in missing metadata, and customize visualizations and the user interface through XML configuration files, and allowing the user to request visualizations using XML messages. LAS is closest in goal to our tool with some important technical distinguishing features. LAS is based on Java and XML, while the tool we develop is based on Python and JSON. XML is somewhat heavier and slower to parse, which is one factor in it being displaced by JSON as a data serialization format. In addition Python is displacing Java for web development due to its greater ease of use, readability and compactness. Finally LAS does not provide an open-source BSD license, meaning it cannot be embedded as widely as our tool.

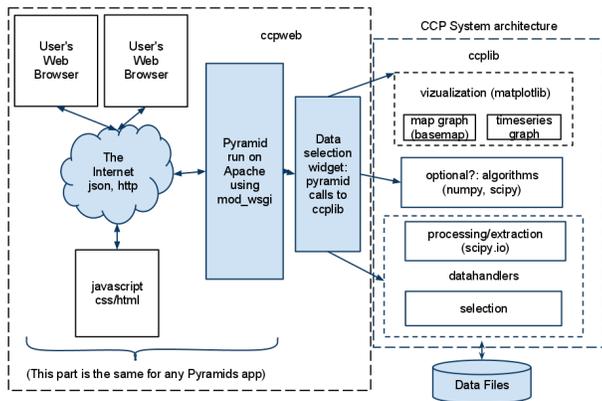


FIG. 1. Diagram of the architecture for the Common Climate Visualization Web Application (CCPviz). The figure illustrates the three distinct sections of the tool: the data extraction and visualization handled by the stand-alone ccplib library, the web framework layer that translates user input into ccplib function calls, and the JavaScript/HTML UI layer that sits in the users browser.

3. Software Architecture

To maintain a flexibility, CCPviz has a component based architecture. As seen in figure 1, the tool has three main pieces: a library for working with the data, a web framework layer for transforming web based user input into library function calls, and a user interface that can be embedded into any website. A component based architecture was chosen so that any layer could be swapped out with a different implementation as needed.

a. Shared Library: ccplib

The backend library that handles all the data processing server side is called ccplib. The guiding principal in creating this library was to use a highly modular design so that individual functions can be changed without having to rewrite everything. This library was also extensively tested using the Python unit-test library since testing was integral to our goal of building a robust application. We stressed the importance of unit tests because it greatly reduced debugging time by separating individual functionality, thereby highlighting what errors were occurring.

(i) Data Extraction

SciPy (Jones et al. (2001)) provides an input/output module that our module uses to pull the data out of NetCDF files because the SciPy library provides many tools for analysing scientific data, but support for NetCDF4 using netcdf4-Python, written by Whitaker (2008), is planned for the next version of the tool. The excellent PyNIO li-

brary developed by Computational And Information Systems Laboratory at the National Center for Atmospheric Research (2011) was not used because we could not find a way to install the library using a Python package management system, such as PIP (PIP development team (2011)). We placed a lot of importance on having a scriptable install because we wanted deployment to require as few steps as possible. Translating between times and indices in the file is handled using the coards-0.2.2 library written by Almeida (2007). All the extraction functions are grouped into the datahandlers module where each file type gets its own file and the module returns a data object containing the attributes of the data (which are extracted from the NetCDF files) and methods for extracting temporal and spatial slices of the data.

(ii) Data Processing

This object is then passed to the algorithms module, which is really just a placeholder that processes the data object—for example by extracting the data and masking missing values—so that the data can be pushed into some algorithm provided by NumPy, SciPy, or any other library the user chooses to incorporate. The algorithms module exists to support locally customized data processing actions through the data object.

(iii) Visualization Module

The graphs are created using the visualization library, which relies on Matplotlib (Hunter (2007)) for generating the figures. The library contains a parent graph object that handles boilerplate such as rendering a figure and labeling it, and then uses children objects to create specific types of graphs, such as spatial and temporal graphs. The Basemap Matplotlib Toolkit, also written by Whitaker (2011), is used to generate all the spatial graphs because it handles numerous map-projections and is integrated with Matplotlib. The visualization module breaks out the different types of graphs so that support for other types, such as skew or 3d scatter can be added on without having to modify much of the existing code.

b. Pyramid Based Web Service

The layer of the application that turns CCPviz into web service was written using the Pyramid Python Web Framework (Pyramid Development Team (2010)). Pyramid was chosen because it is can be used to write fairly lightweight applications. The Pyramid layer takes URLs and parses them into arguments for some of the functions in ccplib and then returns an image based on that URL. This ensures that computed images are addressable resources with their own unique URL. Besides being part of RESTful best practices, this makes it easier to cache, decreasing server load. The URL mapping is based on the Pyramid traversal

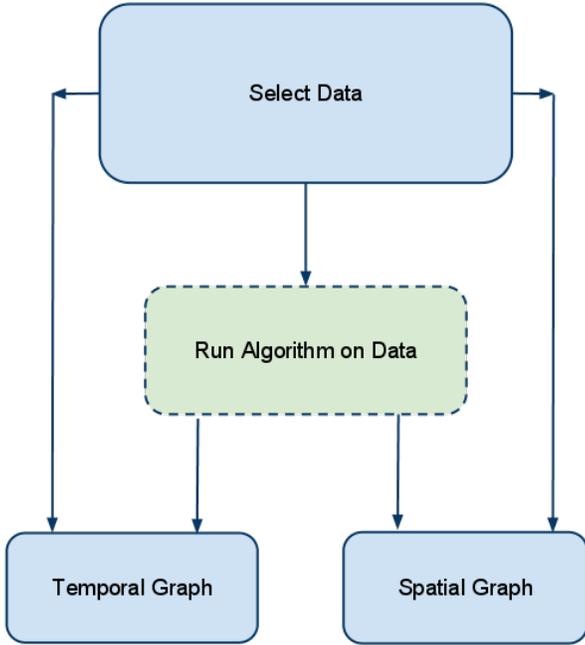


FIG. 2. Diagram of CCPviz’s traversal architecture

model wherein every site can be modeled as tree. As seen in figure 2, the dataset is the root of the tree, then the next node is either an algorithm or a graph type, and the last node is always a graph. New datasets are added using INI style configuration files, which at a minimum need to know the location of the data on a local or network drive (support for remote data is planned) but can also contain various settings for visualizing the data. The remaining URL defines what the graph looks like and is parsed using regular expressions into arguments that are passed into `ccplib`.

c. Web Client

The user interface is essentially independent of the Pyramid and `ccplib` layers of the web application and was designed to be embedded inside a web page as seen in figure 3. The user interface is written in HTML and JavaScript and works by constructing URLs that Pyramid parses from the input fields. That URL returns a graph which is then displayed on the page. The user interface was designed to float on top of the other two layers so that the interface could be designed based on the audience. This also means that an organization could create multiple interfaces for the same datasets, for example a simple one with rough controls for their educational website and a complicated one with as many options as the dataset allows for their research site. The visualization was done in Matplotlib rather than one of the JavaScript visualization libraries—such as `d3` (Stanford Visualization Group (2011)), `protovis` (Stanford Vi-

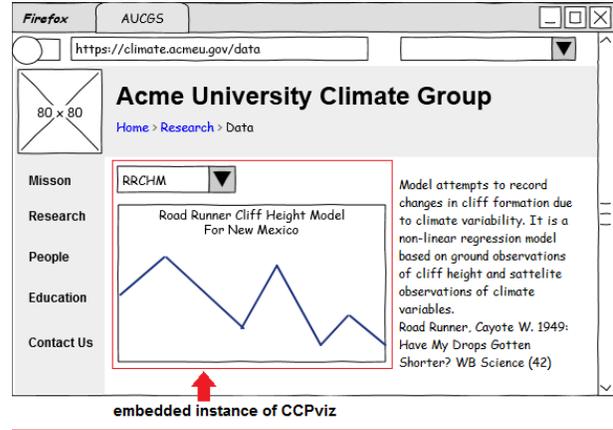


FIG. 3. This wireframe diagram of the GUI illustrates how it embeds into a website without disrupting the content on the site.

sualization Group (2010)), `dygraph` (Vanderkam (2008)), and `infovis` (SenchaLabs (2011))—because the size of the datasets meant that it was much more efficient to do the visualizations server side. However, the RESTful API gives the user the freedom to write an interface that creates the visualizations client side with little extra work.

4. Development Process

Because CCPviz was developed as part of Google Summer of Code, it was designed following the principals of Open Source software development. Its design, functionality, and implementation was discussed on a public mailing-list for projects run by the Climate Code Foundation (CCF) and now has its own list. The code is hosted on Google project hosting, which provides mercurial access to the code and a public bug and issue tracker. The code was written following CCF standards so that there would be some consistency to it and to keep it readable and understandable. The CCF mentors also encouraged code review, which the code greatly benefited from.

5. Application: Temperature Models for the Common Climate Project

CCPviz was created for the Common Climate Project, a group whose mission is to foster transparency in the climate reconstruction research community through the use of open-source tools. CCP plans to facilitate the sharing of climate research through use of a website where data and analysis code can be uploaded and viewed, and therefore needed a tool people could use to explore any data uploaded to the site. So, CCPviz was designed with a focus on climate data, and was therefore tested using GISTEMP, described in Hansen et al. (2010), and NCAR

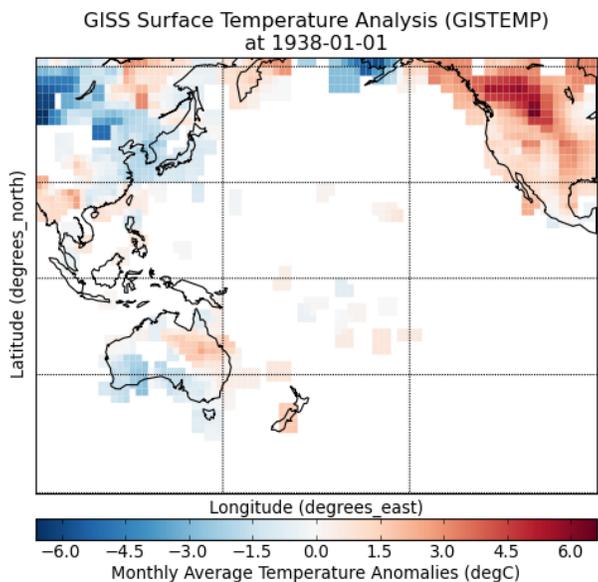


FIG. 4. This shows an example of the kind of map that can be created with CPPviz tool. Note that any date, collection of dates, or region on earth can be plotted.

CCSM1.4, described in Ammann et al. (2007). Since they were both global temperature models, they were similar enough to provide a good comparison. The differences in the datasets also led to a good understanding of the limitations of CCPviz, especially in what kinds of timeseries and grids it can handle. We decided to limit support to gridded NetCDF data that follows the COARDS conventions defined in Cooperative Ocean/Atmosphere Research Data Service (1995) for the first version of the CCPviz because many climate datasets are gridded, COARDS compliant, and in NetCDF4, but it was designed so that adding support for more file types was possible.

a. Global Temperature Maps

Climate datasets are not all recorded using the same grids and projections can change the interpretation of visual data, since some projections will diminish the poles and others will stretch them. Because of this, CCPviz handles multiple projections and changes the projection based on what data is being requested, for example a region map may need a different projection from a global map, and multiple coordinate specifications because of the inconsistency in how coordinates are encoded. Coloring can also strongly affect interpretation, so the visualization module switches between using an opposing color scheme for data that has a mean of zero, as seen in figure 4, and a gradient scheme of one color for most other types of data, though all the user can set which colormaps they would like to use

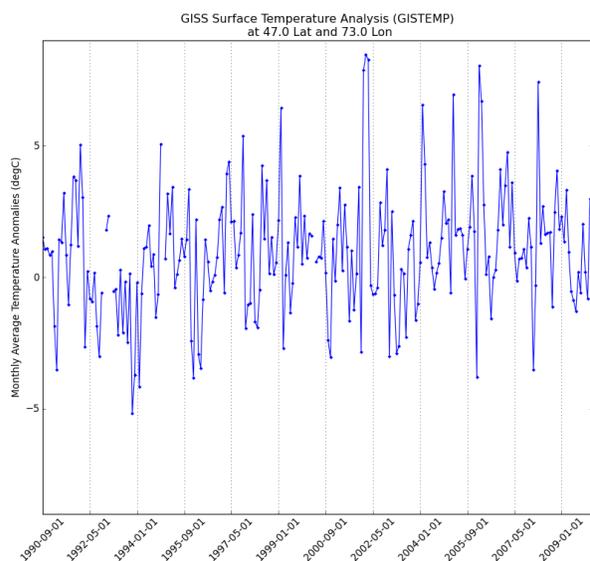


FIG. 5. This shows an example of the kind of time series that can be created with CPPviz tool. Note that any point on earth and any variable can be plotted as specified by the user.

based on what Matplotlib offers.

b. Time-Series Plots

Climate datasets can span anywhere from days to centuries, at resolutions anywhere from minutes to decades, so CCPviz can plot various time resolutions and adjust to changes in resolution based on zooming. The visualization module does not try to fill in missing data, as seen in 5 and instead just leaves out that time, because a major goal of the project was to build a tool that shows the data as it is on disk or only processed in ways the user specified.

c. Data Explorer

As seen in figure 6, the user interface provides the user with fields so that they can select specific times, places, or ranges of both to look out. All the fields are populated using Ajax requests for content served using Pyramid, so the lists of datasets and algorithms are created based on information supplied by whoever deployed the application. To simplify the task of choosing points, the text fields are auto-complete search boxes populated based on the values encoded in the file meta data. For example, the time search box is filled with values, translated from COARDS to a more readable format, from the time field in the NetCDF file. There is also a text box containing the valid ranges because users requested information on what values they could enter.

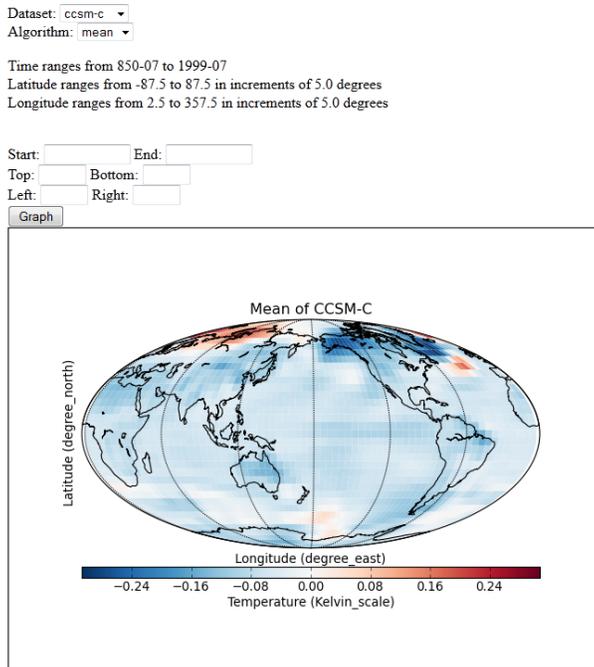


FIG. 6. Screenshot of the web interface.

6. Conclusion

We have created an web embeddable tool that provides a basic but complete solution for scientists who want to make climate data accessible through a web site. Our web application provides the data as both a region selectable map and time series at any point. It is built with Open Source standard web technology and licensed in a flexible way for maximum re-use. Because it can be embedded in many ways, the tool we developed can be extended, for example, to provide one interface for scientific collaborators, and—by changing some of the JavaScript and styling—a very different one for an educational website for children.

In the future we hope to expand our tool so that it can be used to share a wider variety of climate data with the public. In order to fulfill this goal, we would like to add support more file types and sparsely and irregularly gridded data. We would also like to further develop the currently bare-bones client interface with richer HTML5 and JavaScript components. CCPviz is free to download at <https://code.google.com/p/ccp-viz-toolkit/>.

Acknowledgments.

This project was jointly mentored by the Common Climate Project and the Climate Code Foundation (<http://climatecode.org/>) and was created for the Common Climate Project as a part of for Google Summer of Code 2011. It is also somewhat based on code developed at

Glasslab at City College of New York.

REFERENCES

- Almeida, R. D., 2007: coards 0.2.2. <http://pypi.python.org/pypi/coards>.
- Ammann, C., F. Joos, D. Schimel, B. Otto-Bliesner, and R. Tomas, 2007: Solar influence on climate during the past millennium: Results from transient simulations with the near climate system model. *Proc Nat Acad Sci USA*, (104), 3713–3718, doi:10.1073/pnas.0605064.103.
- Computational And Information Systems Laboratory at the National Center for Atmospheric Research, 2011: Pyngl and pynio. <http://www.pyngl.ucar.edu/>.
- Cooperative Ocean/Atmosphere Research Data Service, 1995: Conventions for the standardization of netcdf files. <http://ferret.pmml.noaa.gov/Ferret/documentation/coards-netcdf-conventions>.
- Denbo, D. W., N. N. Soreide, K. McHugh, J. Osbourne, J. Sirott, P. Sorvik, and W. Zhu, 2006: Epic web browser. <http://www.epic.noaa.gov/epic/ewb/index.html>.
- Earth System Research Laboratory, P., 2011: Psd interactive plotting and analysis pages. <http://www.esrl.noaa.gov/psd/cgi-bin/data/getpage.pl>.
- Hankin, S., J. Callahan, and J. Sirott, 2001: The live access server and dods: Web visualization and data fusion for distributed holdings.
- Hankin, S., D. E. Harrison, J. Osborne, J. Davidson, and K. O'Brien, 1996: A strategy and a tool, ferret, for closely integrated visualization and analysis. *Journal of Visualization and Computer Animation*, 149–157.
- Hansen, J., R. Ruedy, M. Sato, and K. Lo, 2010: Global surface temperature change. *Rev. Geophys*, 48, RG4004, doi:10.1029/2010RG000345.
- Hunter, J. D., 2007: Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9 (3), 90–95.
- Incorporated Research Institutions for Seismology Data Management Center, 2011: Incorporated research institutions for seismology. <http://www.iris.edu>.
- Institute of Global Environment and Society: Center for Ocean-Land-Atmosphere Studies, 2003-2011: Grid analysis and display system (grads). <http://www.iges.org/grads/>.

- Jones, E., T. Oliphant, P. Peterson, et al., 2001: SciPy: Open source scientific tools for Python. <http://www.scipy.org/>.
- PCMDI Software Development Group, 2007: Climate data analysis tools. <http://www2-pcmdi.llnl.gov/cdat>.
- PIP development team, 2011: PIP installs python. <http://www.pip-installer.org/en/latest/index.html>.
- Pyramid Development Team, 2010: The pyramid web application development framework. <http://www.pylonsproject.org/>.
- SenchaLabs, 2011: JavaScript InfoVis Toolkit. <http://thejit.org/>.
- Sirott, J., 2006: Dchart: A remote scripting web application for in-situ opendap data. *22nd International Conference on Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology*.
- Stanford Visualization Group, 2010: Protoviz. <http://mbostock.github.com/protoviz/>.
- Stanford Visualization Group, 2011: Data-driven documents. <http://mbostock.github.com/d3/>.
- Tableau Software, 2011: Tableau software. <http://www.tableausoftware.com/>.
- Vanderkam, D., 2008: dygraphs javascript visualization library. <http://dygraphs.com/>.
- Whitaker, J., 2008: netcdf4 0.9. <http://pypi.python.org/pypi/netCDF4/0.9>.
- Whitaker, J., 2011: Basemap matplotlib toolkit 1.0.2. <http://matplotlib.github.com/basemap/>.