Using Open-Source Solutions to Image Analysis with Opticks 4.9.1

By Bryan Goodrich

This project aims to explore Opticks approaches to common image processing tasks to those that are already familiar with image processing tasks as used in ArcGIS 10.0. There are 7 areas that will be discussed that range from importing data and adjusting properties to doing band math and Python scripting. All images are of the same Sacramento area, taken from Landsat TM. Six of these are single-band (1-5 and 7) GeoTIFFs and the other is a single IMG comprised of all six.

Purpose: The aim of this project was to explore the functionality of the Opticks 4.9.1 image analysis software. Incorporated in this project was learning the terminology, the perspectives, and the tools available to accomplish a variety of basic image and data analysis tasks.

The context of this analysis was to follow the same tasks performed in the Geography 342 Remote Sensing course at American River College, using the same available data.

Methods: Outlined for this project were 7 tasks, detailed below.

1. Dating Importing
2. Data Processing
3. Histograms
4. Band Math
5. Image Classification
6. Opticks Wizards
7. Python Scripting

Data importing is generally an easy task to perform, but because there are so many varieties of image types and types of properties that an image can contain, this task can be very involved within Opticks. For instance, you can do basic clipping and set display properties (e.g., set to color display and type, such as SWIR) at the import prompt. You can also select the type of importer desired for the image type being used. However, since this can usually be identified by the file extension, the auto-selected importer option is default.

If one imports directly without setting options, such as using the GeoTIFF single-band Landsat TM images provided in class, this import is straight-forward. The product is an environment window filled with separate windows for each gray scale image that was imported.

However, if one wants to use a multi-band IMG file, they could select the “options” at the import prompt to alter image properties before being brought into the environment. Here, one can select preset color displays, such as True Color, if one sets the color (RGB) channels to the bands in the image. Since Opticks was designed as a hyperspectral image analysis tool, these bands can be set in the “wavelength” tab of the options prompt, either manually or with a wavelength file. For small band images like Landsat TM, this plain text file can easily be set up for a collection of TM images.

Data Processing is often the first step after data is imported into a software environment. This encompasses a number of things, such as setting properties, clipping the image, and making compositions. Since the first two can also be done in the import stage, nothing more will be said about that beyond how to access those properties.

The Opticks environment is similar to the familiar ArcGIS interface the GIS student would be familiar with. There is a display area (the Workspace) and a panel for displaying those windows in a table-of-contents (TOC) manner.

The window element listed in this panel contains a sub-listing for the image and any other properties about the image. For instance, if one performs a K-means classification on one of these images in the workspace, it will put the classification result layer under this workspace window listing. Annotation on a window element will also be listed under its TOC listing.

To view properties of one of these window layers, such as the image, one can simply right-click and select “properties.” The properties for the image layer will give you options similar to those when importing, allowing you to manage statistics and set color displays.

One task that is required is being able to create an image composition: one image with the bands of multiple single-band images. In this class, we had 6 color channels (wavelengths) provided for bands 1 through 5 and 7 of the Landsat TM. Opticks provides a way of combining 3 images into a mosaic, listed under the “geo” menu. This menu can be turned on in the TOC by expanding “toolbars” and selecting “geo” from that listing.

The mosaic prompt shows you the image layers in your workspace windows you can choose from. You simply select those you want to combine and whether to create a new workspace window for the composite or to export it to a file, which can be exported from the window, also. If the first option is chosen, the result is a window element with the three image layers contained therein. This window can then be exported and imported as a single image layer with multiple bands.

Histograms are a typical way of analyzing the statistics of the image. In the ArcGIS environment, this requires opening the properties of a single image layer, making sure statistics are calculated, and then clicking on a button to view the histogram.

Opticks makes this especially simple. With a click of the histograms button in the standard menu, the Histogram Window opens up at the bottom of the environment. There is displayed the histogram for any window in the workspace that is active. To view another one, simply click on another window to activate it. If the layer is multi-band, then tabs under the histogram will appear for each color channel. The histogram stretch will be displayed in the color of that image, also. This is an easy operation to execute in Opticks as compared to ArcGIS.

Band Math poses another import basic process in manipulating and reviewing image data. Opticks contains a band math tool (under “General Algorithms”) that resembles the “raster calculator” in ArcGIS. Unfortunately, it lacks the ability to free-type your expressions, requiring you to use the mouse to click each of the items in the expression.

In any case, the tool functions in much the same way as the raster calculator. The result is a separate window element, named with the expression used in the band math, that contains the result. For instance, one might run an NDVI expression if they know the bands of their red and near-infrared (NIR). However, since this is a common expression, a tool for this transformation already exists in Opticks. If one clicks on the Spectral menu and looks under the Transformations sub-menu, the NDVI tool is listed. It brings up a prompt for the active window element that lists the bands twice, one for the red and the NIR channels. You simply select which band belongs to that channel and submit the operation. The result is the same as if you did this manually using band math.

Image Classification is a basic process for image analysis. Unfortunately, Opticks 4.9.1 only comes with one tool for this out-of-the-box. Under the Spectral menu is a classification sub-menu with the single K-means tool. This common unsupervised classification algorithm is easily implemented. The tool prompt gives a couple of options for thresholds, one for the number of result classes, and a couple check boxes for logistics.

After a few moments, the tool results in an image layer for the active window being classified. The options for this layer allow you to alter colors for the classes generated, for those that were empty (0), and those that could not be classed (-1).

There may be other options for more advanced algorithms through plug-in extensions, such as expectation-maximization (EM) or maximum-likelihood (ML) classifiers, each unsupervised and supervised, respectively. However, these may also be available through Python using the SciPy library, as will be discussed below.

Wizards provide a method to leverage Opticks tools in a non-programmatic way, similar to Model Builder in ArcGIS. The Wizard Builder itself has an intuitive framework to the way Model Builder works, by linking tools together and managing inputs and outputs of the tool.

The prerequisite for using a Wizard is to set up your session. In the sessions menu under Options are a number of directories, one of which is the Wizard path. This path is set to where you store your “tools” after you create them. The tool is a “.wiz” file. When your session beings or you run the “Update Wizard List” under the tools menu, all wizard files in this directory are loaded into your environment, changing the tools you now have available. All the tools available work in this way, internally.

To create a tool, you launch the Wizard Builder. It loads up a three-panel prompt to develop your wizard. On the left-hand side are the tools available that you can drag into the center panel, just like building a model in ArcGIS. On the right-hand side are listed characteristics of the tool you're putting together, such as its inputs and outputs, which are reflected graphically in the center panel. The center panel gives a work flow of your tool just like Model Builder does in ArcGIS.

The key to defining your interface to this tool from the Opticks environment is in how you label your tool in the right panel. The schema is

[menu-name]/wizard-name

The bracket tells what menu to list your tool under. If you used “Spectral” or “General Algorithms,” then your tool displayed by the wizard name you put on the right side of the slash will fall under those already existing menus. However, if you name it something original, then a new menu will be created with your tools that also have that menu name.

The result of this process is to create a “.wiz” file that you store in your Wizard path. Once updated, your tool will be available under the menu it was assigned. The utility of these tools, like in ArcGIS, depend on the user's familiarity with the tools.

Python Scripting is a robust approach to using not only Opticks tools but the wealth of Python available libraries. The standard toolbar includes a button for opening the Python scripting window, just like in ArcGIS. You can load the Opticks module with the appropriate import statement. The nice thing about using Python is that the Opticks library includes methods for accessing (importing) image layers into the Python environment, using methods on those classes of objects to obtain relevant information, and convert these objects into NumPy arrays for further processing with the existing wealth of commands in the NumPy and SciPy libraries.

For example, the following script pulls into Python an image from a single window workspace and displays some of his properties available in its property sheet. From this start, the programmer could use the image element properties to set up and begin running analyses available in the Opticks library or within, say, SciPy.

```
import opticks
v = opticks.View()
n = v.get_name()
r = opticks.RasterElement(n)
di = opticks.DataInfo(r)
di.rows
di.columns
di.bands
di.interleave
```

Discussion: This project did not aim to provide a wealth of output in terms of image analysis results. Instead, the product concentrated on the experience of using open-source image analysis software.

Being new to the remote sensing field, the depth of understanding the scope of what Opticks could provide was limited, which set the scope of this project to what was covered in class. Nevertheless, a variety of basic processes and exploration of advanced tools were tried.

The greatest difficulties were found in two areas: understanding the software work flow and in getting Python to work according to the only available documentation found online.

The first issue was in setting up the Opticks *session*. As mentioned when discussing wizards, you can set up your paths to your custom wizards, input and output default locations, and more. Many of these features are saved in your session file that defines your work flow, similar to setting up the default scratch workspace and such for a map document. I was going to see what advances in the software were made in Opticks 4.10, but it does not support older session files, so that was scraped.

Python proved a challenging task because the documentation is nearly impossible to find. The example scripts the Opticks team provided on their website did not even work, using commands that may have been from an earlier version of the python module. In any case, I explored the scripts in the directory where they are called from within Opticks. This allowed me to search for commands that were similar to the ones used in the scripting examples. With a little bit of backwards engineering, I was able to get a slight grasp on things.

Another avenue that may be helpful in averting problems with the opticks module is to use the help feature on their functions. They developed their module to Python standards, so they often come with descriptions of how to use the methods or what methods exist for a given class. If using an Integrated Development Environment (IDE) like Spyder, this sort of information may be present as you code.

Conclusions: Opticks provides a powerful framework for image analysis that is entirely free. As an open-source solution, it is also part of a community of experts always looking to improve the software. With utilities like their Python module and Wizard Builder, not to mention the extensive amount of plug-ins available that were not explored here, Opticks can serve the needs of any analyst. As hopefully demonstrated here, the learning curve for venturing into this solution is not too steep.