A SANDBOX ENVIRONMENT FOR THE COMMUNITY SENSOR MODEL STANDARD. T. M. Hare, J. R. Laura, I. R. Humphrey, T. J. Wilson, M. A. Hahn, M. R. Shepherd and S. C. Sides, U.S. Geological Survey, Astrogeology Science Center, Flagstaff, AZ, 86001, (thare@usgs.gov).

Introduction: The heart of any digital photogrammetric system is the software used to spatially locate camera data accurately to the ground, known as a camera or sensor model. The U.S. Geological Survey's Astrogeology Science Center (ASC) has begun to develop and test different instrument camera models using the Community Sensor Model (CSM) standard [1]. Here we present ongoing work ASC is undertaking to provide a programming sandbox environment for employing the CSM standard. We define a sandbox as a testing environment that allows programmer experimentation outside a given production environment and to help demonstrate and teach available capabilities to others.

Background: A camera sensor model can be defined as a mathematical description of the relationship between the three-dimensional object (e.g. target's surface) and the associated two-dimensional image plane. As described in [2], the quantities needed to define a sensor model can be divided in two broad categories: interior and exterior. The interior parameters are intrinsic to the sensor design and calibration and typically include focal length, location of the principal point, and lens distortions. For more complicated instruments, the interior parameters may also include wavelength dependencies, gain and pixel summing settings, and (for pushbroom sensors) the timing of line exposures and time delay integration (TDI) settings. The exterior parameters describe the location and orientation of the sensor with respect to the target's reference coordinate system. For planetary applications, this information is typically stored in the form of SPICE (Spacecraft, Planetary ephemeris, Instrument, C-Matrix, and Event) kernels and delivered by the Navigation and Ancillary Information Facility (NAIF, [3]).

The CSM Standard: The Community Sensor Model (CSM) Working Group was established by the U.S. defense and intelligence community with the goal of standardizing camera models for various remote sensor types [4]. The CSM standard, now at version 3.0.2, provides a well-defined application program interface (API) for multiple types of sensors and has been widely adopted by Earth remote sensing software systems.

It is worth noting that the CSM API does not make the creation of a camera sensor model technically any easier besides serving as a solid framework for others, but once written, it more easily allows interoperability between different photogrammetric applications. And the API has been designed and continuously tested by industry experts and thus the planetary domain will benefit from this more than decade-long effort by the CSM Working Group.

This year our ASC programming team has implemented the MESSENGER Mercury Dual Imaging System (MDIS) CSM framing camera model for both the narrow angle and wide angle cameras (NAC and WAC respectively) [5]. This code is available for testing and download at the ASC GitHub site: https://github.com/USGS-Astrogeology/CSM-CameraModel

Sandbox: For our sandbox testing environment the programming team chose to use the Jupyter Notebook and the Python programming language. The Jupyter Notebook is an open web application that allows you to share code, equations, visualizations, and any associated documentation (http://jupyter.org/). While Jupyter notebooks now support several programming languages, the environment was born from IPython [6] and thus Python is still the most robust language supported. Fortunately, we also continue to see support for Python grow within the planetary community [7]. Using Python also gives us a single high-level interpretive language to access SPICE (SpiceyPy; https://github.com/AndrewAnnex/SpiceyPy) and support for planetary images using the Geospatial Data Access Library (GDAL; http://www.gdal.org). Jupyter notebooks were heavily used to prototype, in Python, the algorithms for C++ implementation in the CSM.

The last step to complete our sandbox testing environment required the CSM to properly work within a Jupyter Notebook environment. To do this, it was necessary to wrap the CSM C++ source code in Python. This was done using Cython (http://cython.org/) and allows for Python to directly call the underlying C or C++ functions. While still in testing, this code is available at the ASC GitHub site https://github.com/USGS-Astrogeology/CSM-CyCSM and should allow for almost any CSM to be functionally wrapped in Python.

For static examples of our sandbox Jupyter notebooks, please visit the ASC GitHub site: https://github.com/USGS-Astrogeology/CSM-SET (under the folder notebooks) and see Figure 1.

Conclusion: The CSM standard is the first step in realizing a camera model standard that can be used and shared across NASA's and international planetary missions. Using the CSM within an interactive Jupyter Notebook, we find an ideal exploratory environment for sensor model development, data analysis, validation, portability, and finally the capability of demonstrating results to collaborators.

Acknowledgments: This effort has been supported by NASA's Planetary Spatial Data Infrastructure (PSDI) interagency agreement.

References: [1] Hare and Kirk, 2017, LPSC XLVIII, abs #1111. [2] National Geospatial-Intelligence Agency, (2011), Frame Sensor Model Metadata Profile Supporting Precise Geopositioning, NGA.SIG.0002_2.1. [3] Acton, C.H. (1996), Ancillary Data Services of NASA's Navigation and Ancillary Information Facility, Planetary and Space Science, Vol. 44, No. 1, 65-70. [4] Community Sensor Model Working Group, (2010), Community Sensor Model Technical Requirements Document, v. 3.0, NGA.STND.0017_3. [5] Hawkins, S.E., Boldt, J.D., Darlington, E.H. et al. Space Sci Rev., 2007, 131: 247. doi:10.1007/s11214-007-9266-3. [6] Fernando Pérez, Brian E. Granger, IPython: A System for Interactive Scientific Computing, Computing in Science and Engineering, vol. 9, no. 3, pp. 21-29, May/June 2007, doi:10.1109/MCSE.2007.53. URL: http://ipython.org. [7] Laura, J. R., et al., 2015, LPSC XLVI, abs #2208.

USGS-Astrogeology / CSM-SET

Unwatch 4 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs Settings

Branch: master CSM-SET / notebooks / Creating a Camera Model Object.ipynb Find file Copy path

jlaura Updates to the notebooks d5bb3b1 11 days ago

1 contributor

674 lines (673 sloc) 443 KB Raw Blame History

Creating a Camera Model

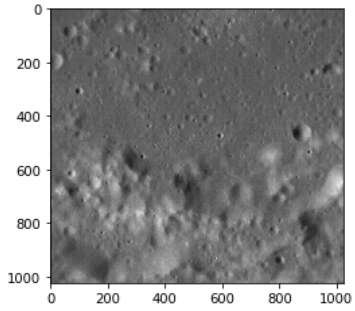
Here we load the previously created MDIS NAC ISD file and create a working camera model. This notebook loads the ISD from file to keep the overall length short. It is equally possible to simply create a camera model directly from the image and kernel data (with the SPICE calls directly populating a CSM ISD object).

In [1]:
```
import json
import gdal
import pyproj
import numpy as np

# CSM Imports
from cycsm.isd import Isd
import usgscam as cam
```

In [12]:
```
d = gdal.Open('../tests/data/EN1007907102M.cub')
arr = d.GetRasterBand(1).ReadAsArray()
imshow(arr, cmap='gray')
```

Out[12]: <matplotlib.image.AxesImage at 0x7fe3c7798080>



In [13]:
```
res = np.empty((arr.shape[0] * arr.shape[1], 4))
res[:,0] = arr.ravel()
```

In [14]:
```
size = model.imagesize
c = 0
for i in range(int(size[0])):
    for j in range(int(size[1])):
        x, y, z = model.imageToGround(i,j,0)
```

Figure 1. Shows our CSM sandbox within a Jupyter Notebook. Step 1 shows required Python libraries to be imported. Skipping to step 12 to shows the initial image read for the MESSENGER image. Step 14 shows a simple test for every pixel using the critical routine *imageToGround* within the MDIS CSM camera model.