HiPS to Py

- Organization: OpenAstronomy
- Suborganization: Astropy

Student information

Personal details

- **Name**: Adeel Ahmad
- **Email**: adeelahmadadl1995@gmail.com
- **GitHub handle**: adl1995
- **IRC (#OpenAstronomy)**: adl1995
- **Timezone**: UTC +05:00
- **Blog**: http://adl1995.github.io/
- **Blog RSS:** https://adl1995.github.io/feeds/all.rss.xml

Academic details

- **University**: National University of Computer and Emerging Sciences, Islamabad
- **Degree**: Computer Science
- **Graduation year**: 2018

Background

In my previous semester, I enrolled myself in Digital Image Processing course. During the tenure of this course I implemented filters (Gaussian, Sobel, Prewitt, Laplacian), edge detectors (Canny, Marr Hildreth), morphological operators (Convex hulling, Erosion, Dilation), object detectors (Generalized Hough transform, simple convolution), and seam carvers. These were implemented purely in Python, making no use of external libraries other than `NumPy` and `Matplotlib`. This attenuated my interest in Computer Vision and I have been an active researcher in the domain since then. I have showcased these project on my GitHub profile. The most interesting algorithm I implemented was Generalized Hough Tranfrom. I was amazed as to how something as simple as an equation of line could lead to detection of objects in images.

Although the 'HiPS to Py' project involves Computer Vision concepts at a brief level mostly related with affine transformation, it would still benefit from my knowledge of the domain. Apart from this, I have also been a PHP web developer for almost two years, and I have worked on multitude of projects ranging for SaaS to e-commerce websites. I work remotely on Upwork and Fiverr. My clients are either IT organizations or independent contractors. I have also worked on web automation and data scraping using `Selenium` and `BeautifulSoup`. These skills will be useful for fetching the tiles and metadata through HTTP requests and making them asynchronous. The tiles would also need to be stored on in-memory or on-disk cache. Errors and timeout events also need to be handled. The skills that I acquire are through self learning and consistency. Currently, I am building an application using Google Application Engine.
I started collaborating with Open Source organizations about a month ago, and my experience has been excellent so far. The mentors have been very kind, welcoming and have provided me with assistance along the way. Given my background in Computer Vision and being good at problem solving, I firmly believe this prestigious organization and its users will benefit by the HiPS client for Python. It would also provide me with a working ground knowledge on Astronomy.
During the tenure of this project, I hope to improve my skills with tools like `Jupyter`, `Pytest`, `Sphinx`, and `Git`. It will also provide me with a professional level overview on how to write applications, make plans, organize projects, working remotely with other developers, and design a good API.

Project Details

Mentors

- Christoph Deil
- Thomas Boch

Abstract

> Design and create a Python client for Hierarchical Progressive Surveys (HiPS). The library extend a Low and High level API for exploring or creating WCS / HEALPix images. Currently, there are clients built using HiPS, such as Aladin and Aladin Lite, but they are written in Java and JavaScript, respectively. The goal of this project is that for a given World Coordinate System and image size (nx, ny), create a NumPy array containing the image by fetching tiles and projecting them onto an image array.

Detailed description

Hierarchical progressive surveys (HiPS) utilizes the HEALPix framework for mapping a sphere (in our case, part of a sky) and compiles / transforms it into tiles and pixels. For this project, a HiPS Python client is to be implemented which will enable users to create WCS and HEALPix images. Functionality for exploring HiPS information / metadata will also be available.
The goal of this project would be to create a new HiPS client under `hipspy/hips`. The **main dependencies** for this software include:

- Python >= 3.5
- NumPy
- Astropy
- Healpy

Some other **possible dependencies** for tile drawing include:

- Reproject
- SciPy

The HEALPix framework

This project will involve the HEALPix 'Hierarchical Equal Area isoLatitude Pixelization of a sphere' framework for discretizing high resolution data. This framework has implementations in many languages and extends a data structure (with a library) for each language. The main features provided by this software are:

- Pixel manipulation
- Spherical Harmonics Transforms
- Visualization
- Input / Output (supports FITS files)

In a nutshell, the pixelization procedure subdivides a spherical sphere in which each pixel is equidistant from the origin - meaning it covers the same surface area. This produces a HEALPix grid. The framework provides two numbering schemes for pixels, namely **RING scheme** and **NESTED scheme**. It provides three coordinate systems, namely **Celestial** (HiPS default), **Galactic**, and **Ecliptic**.
Some alternative tools to HEALPix are Hierarchical Triangular Mesh (HTM) and Tessellated Octahedral Adaptive Subdivision Transform (TOAST).
HiPS uses the Celestial coordinate system by default. Also, HiPS does not support the Ecliptic coordinate system.

Working of HiPS

The multi-resolution representation of original images provides the basis for visualizing data in a progressive way as the pixels that are required for a given view can be accessed through pre-computed HEALPix maps, and the nested pixel numbering scheme provides a simple hierarchical indexing system that encodes pixel inheritance across different orders.

HiPS scheme groups pixels in different tiles. The general relationship between tiles and pixels is that a tile with n^{tile} pixels along each side forms a HEALPix mesh of order k^{tile}. Tiles store map information from HEALPix. These tiles are presented as square arrays and it is possible to store them in different file formats. The files are organized in different directories. Here, tiles are used as files and tile orders are used for grouping data in directories - all following a naming convention. For more information on the method of storing files, this document can be viewed, written by Pierre Fernique.

HiPS images & catalogues

HiPS generates tile images for tile orders. When mosaicking / stitching images, the angular resolution is taken into account. Next important thing to consider is whether to emphasize on display quality or photometric accuracy, which depends on our use case. Image encoding can be done either in **FITS**, **PNG**, or **JPG** file format. For most cases it is enough to only generate FITS and PNG files. The lowest order pixel values correspond to a large area of the sky.

Drawing HiPS tiles

To draw HiPS tiles, affine transformation is used. For displaying a HiPS tile, first the best order is chosen to fit the display. Properties such as maximum order and coordinate system (either ICRS or Galactic) are read from a file. The four corners of an image are mapped onto the display using affine transformation.

Approach

For a world coordinate system and an image with dimensions (x * y), I will create a `NumPy` array with the image information by fetching tiles and projecting them onto this image array. Some lower-level functionality like fetching tiles will also be available.
The repository where the code will reside is already set up (https://github.com/hipspy/hips). I will create pull requests for the code I write, which will be verified by the mentors. The project structure will also be set up by the mentors.

Benefits

HiPS addresses the challenge of big data in astronomy. It describes data in a multi-resolution manner, so it effectively creates an ease in its access. It can also be utilized for data sharing and interoperability. HiPS provides a means for conserving scientific details of astronomical data. MOC maps generated using HiPS facilitate the comparison of sky region between different data sets, and can be used to establish regions of intersection between multiple surveys. It also helps to avoid complex queries on the database.

Current progress

The repository for the current work done lies at GitHub (https://github.com/adl1995/HIPS-to-Py). This includes my notes from the HiPS paper, and a package I authored named `hips-tools`. Apart from this it also includes numerous test scripts I created.
Firstly, there's a Python script which fetches HiPS tiles and displays them using `Matplotlib`. The calculation of Norder, Npixels, and Nside was done using `Healpy`. The retrieved image was then bytes decoded using the `BytesIO` library.
For retrieving a HiPS tile in JPG format, there's a script named `save-healpix-fits.py`. This fetches a tile and writes it as a FITS map. I achieved this using Astropy's `astropy.writeto` method. A summarization of all the scripts is listed below:

- `fetch-tiles-threaded.py`: fetches HiPS tiles with and without threading and displays the response time for each.
- `save-healpix-fits.py`: fetches a HiPS image tile and saves it using Astropy.
- `build-wcs.py`: creates a WCS object and draws markers on a HiPS image tile after loading it.
- `hips-demo.py`: fetches a HiPS image tile (PNG, JPG, FITS), and displays it using Matplotlib, stores it locally, and then displays it from files.
- `astropy-basic.py`: an exmaple script showing the basic functionality of Astropy's utility functions.

After going through all the conversions and successfully writing a `FITS` map, I then loaded it using `astropy.io.fits.open` method.
As multiple tiles have to be fetched for time efficiency, concurrency has to be achieved. So, I wrote a script utilizing Python's threading library. The elapsed time was calculated using the `time` library. For fetching the tiles `urllib`, `grequests`, `aiohttp`, and `asyncio` were used. A comparison of their response times are listed here:
https://github.com/adl1995/HIPS-to-Py/blob/master/notes/response-times.md

The pros of `grequests` is that it takes less time when large number of requests have to be sent. But `urllib` (with threading) gives a better response time when requests are numerous.
Using `aiohttp` with `asyncio` seems to be the best option. Its response time is almost 50% less than `grequests`.
Apart from this, I downloaded the `Aladin` desktop application and examined its various functionalities. It offers a tool for enabling HEALPix grid on the currently displayed image. This helped me better understand how to the grid gets divided at different zoom levels, and the relation between order and pixel numbers.
Also, I have familiarized myself with how to format code, write test cases and documentation using the regions package.

Communication

After a brief discussion with the mentors, the decision is to do weekly hangouts and then collaborate daily via GitHub and a daily summary for the next day. Communication will be through Skype, GitHub or emails.

Method of publishing code

Keeping in view the above mentioned communication structure, at least one, usually several small pull requests per week (ideally, getting work merged by the end of the week) will be made to the `hipspy/hips` repository.

Deliverables

The work will revolve around writing a high and low level API. The high level API will provide functionalities such as creating `WCS` and `HEALPix` images. The current API document is available at https://github.com/hipspy/hips/blob/master/notes/API-proposal.md. Lower level functionality includes implementing in-memory and on-disk cache, in addition to basic input / output of HiPS tiles.
Another major part of the project involves writing test cases, docstrings, and high-level documentation. The `Sphinx` documentation generator will be used for this purpose.
The current algorithm for drawing tiles lies here: https://docs.google.com/document/d/1ooKTeaosHv6Bnovwfk2LOpvxi_PHLGgacz0uLnMtB38/edit?usp=sharing. For now the discussion is around two algorithms, one focusing on speed and the other on precision.
All the functionalities will be extended through classes and focus will be on writing clean code, code organisation, tests, and documentation. The code will be submitted regularly through pull requests.

Development environment

My current development environment includes `pip`, `Jupyter`, `IPython`, `PyCharm`, and `Sublime`.

Timeline

Time Period	Plan
May 04- May 30 **(Community Bonding Period)**	• Get familiarised with test cases using assertions etc. in `Pytest`. • Familiarise myself with docstring / documentation standards using `Sphinx`. • Read codebase of existing packages like `regions` and other affiliated packages. • Continue experimenting and prototyping to implement tile drawing as specified here.
	Part 1 starts
May 31 - June 06 (1 week)	• Add functionality to fetch HiPS tiles and compute tile list and corners for a given WCS. • Compute HiPS order corresponding to the requested image size / resolution. • Implement functions for coordinate transforms. • Write test cases and documentation.
June 07 - June 13 (1 week)	• Work on fetching a HiPS tile. • Add docstrings and test cases. • Implement functions for coordinate transformation.
June 14 - June 21 (1 week)	• Add method `make_image_hpx` for creating a HEALPix image. • Write documentation and unit tests. • Open pull requests.
June 22 - June 28 (1 week)	• Write functionality for getting HiPS list and HiPS information. • Add a Jupyter widget for previewing survey information (optional).
	Part 1 completed **Part 2 starts**

June 29 - July 05 (1 week)	● Add in-memory and on-disk cache using the `HipsTileCache` class. ● Write `HipsTileMeta` for representing tile metadata. ● Add utility functions to download HiPS data from server to local storage (optional). ● Write test cases and documentation.
July 06 - July 12 (1 week)	● Add Input / Output methods for multiple file formats. ● Write test cases and documentation.
July 13 - July 19 (1 week)	● Write `HipsTileFetcher` for fetching list of tiles and put them in cache. ● Write test cases and documentation.
July 20 - July 26 (1 week)	● Work on related classes. ● Write test cases and documentation.
July 27 - August 03 (1 week)	● Add tile drawing using affine transformation. ● Functionality for drawing HiPS tiles using affine transformation. ● Write test cases and documentation.
August 04 - August 11 (1 week)	● Fix problems related with tiles distortion using the deformations reduction algorithm. ● Write `make_image_wcs` method for fetching HiPS images, and load them in a `NumPy` array. ● Update documentation.
August 12 - August 17 (1 week)	● Add optimizations, write test cases. ● Implement the more precise tile drawing method using interpolation i.e. by finding tile

	and pixel corresponding to a given HEALPix index.
August 18 - August 21 (**~** 1 week)	● Write high level documentation. ● Add optimizations, write test cases. ● Add support for HiPS catalogues (to be discussed). ● Add `async` and drawing ability in parallel using multiprocessing (optional). ● Link the implemented part with GU interface (optional).
	Part 2 completed
August 21, 2017 - August 29, 2017 **(Students Submit Code and Evaluations)**	● Clean up code. ● Improve documentation. ● Add test cases. ● Code refactoring (if required). ● Resolve merge conflicts (if any).
August 29, 2017 - September 05, 2017	Mentors submit final student evaluations.
September 06, 2017	Results Announced.

Availability

My final exams end on May 20th. So, I will have ample time for the community bonding period. After that, I will be free during the whole summer i.e. almost three months. I do not have any other commitments, so I can focus all my attention to GSoC.