

Vizic: A Jupyter-based Interactive Visualization Tool for Astronomical Catalogs

Weixiang Yu^{a,b,*}, Matias Carrasco Kind^{b,c}, Robert J. Brunner^{c,b}

^a*Department of Physics, University of Illinois at Urbana-Champaign, Urbana IL, USA*

^b*National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, Urbana IL, USA*

^c*Department of Astronomy, University of Illinois at Urbana-Champaign, Urbana IL, USA*

Abstract

The ever-growing datasets in observational astronomy have challenged scientists in many aspects, including an efficient and interactive data exploration and visualization. Many tools have been developed to confront this challenge. However, they usually focus on displaying the actual images or focus on visualizing patterns within catalogs in a predefined way. In this paper we introduce *Vizic*, a Python visualization library that builds the connection between images and catalogs through an interactive map of the sky region. *Vizic* visualizes catalog data over a custom background canvas using the shape, size and orientation of each object in the catalog. The displayed objects in the map are highly interactive and customizable comparing to those in the observation images. These objects can be filtered by or colored by their property values, such as redshift and magnitude. They also can be sub-selected using a lasso-like tool for further analysis using standard Python functions and everything is done from inside a Jupyter notebook. Furthermore, *Vizic* allows custom overlays to be appended dynamically on top of the sky map. We have initially implemented several overlays, namely, Voronoi, Delaunay, Minimum Spanning Tree and HEALPix grid layer, which are helpful for visualizing large-scale structure. All these overlays can be generated, added or removed interactively with just one line of code. The catalog data is stored in a non-relational database, and the interfaces have been developed in JavaScript and Python to work within Jupyter Notebook, which allows to create customizable widgets, user generated scripts to analyze and plot the data selected/displayed in the interactive map. This unique design makes *Vizic* a very powerful and flexible interactive analysis tool. *Vizic* can be adopted in variety of exercises, for example, data inspection, clustering analysis, galaxy alignment studies, outlier identification or just large scale visualizations.

Keywords: Jupyter, Python, Visualization, catalogs, large-scale structure of universe - methods: numerical

1 INTRODUCTION

For the past decades, conducting large-area sky surveys became one of the most powerful approaches to carry out observations within the astronomy community in order to understand the Universe. The Sloan Digital Sky Survey (SDSS) (York et al., 2000) set the foundations for such surveys followed by others, such as DES and Pan-STARRS (The Dark Energy Survey Collaboration, 2005; Kaiser et al., 2002). Modern telescopes and world-class supercomputing facilities produced exceptionally good images and exceptionally precise measurements of astrophysical sources, challenging astronomers to face unprecedentedly massive datasets. Moreover, future large sky, for example, LSST and Euclid (Tyson, 2002; LSST Science Collaboration et al., 2009; Laureijs et al., 2012), will obtain data of orders of magnitude larger than what we have today. These massive datasets place a new challenge on astronomers in the light of data exploration and visualization.

Various groups and individuals have tried to confront this challenge by developing new astronomical applications that are capable of displaying and visualizing large datasets. Notable examples include *Aladin Lite* (Boch and Fernique, 2014), *VisiOmatic* (Bertin et al., 2015), *ASCOT* and *Toyz* (Moolekamp

and Mamajek, 2015), to mention just a few. The majority of these softwares are web-based applications that focus on the visualization of high-resolution images with additional functionalities of catalog over-plotting or image analysis. Meanwhile, *GLUE* (Beaumont et al., 2015), a desktop application that specializes in visualizing selection propagations between different plots of the same dataset, has became a popular tool for exploring hidden patterns within catalogs and images. Unfortunately all these mentioned tools, while being very powerful, have some drawbacks when it comes to the study of large-scale structure and flexible customization. Image-focused applications are limited by the fundamental characters of photometric observations, whereas images produced by sky surveys are fixed in terms of compositions, in other words, the sources are immovable. In this regard, catalogs are more flexible considering astronomical sources in a catalog can be easily sorted and filtered. However, tools that are more friendly with tabular data are usually less good at interactive visualizations of geospatial data. They either render data points into simple scatter plots or summarize data distribution into pixels like with *VaeX* (Breddels, 2016), which lacks rich interactions with individual object while providing a very powerful way to visualize distributions of extremely large catalogs. The regarding limitations of existing softwares in addition to emerging new technologies for

*wyu16@illinois.edu

building interactive web tools have generated a need to improve existing tools and develop new ones.

Thanks to the increasingly powerful web technologies (e.g., HTML5 (W3C, 2014), CSS (W3C, 2011a) and JavaScript (International, 2009)) and the improved data bandwidth, nowadays much more complex tasks can be carried out in a browser. Along with the great availability of graphic design JavaScript libraries, web browser has became a fertile ground for visualization projects. In addition, thanks to projects like Jupyter (Ragan-Kelley et al., 2014) we now have the powerful combination of a web-based application that can run anywhere with the flexibility of a Python scripting environment where scientist can customize plots, algorithms and workflows only restricted by the limitations of the programming language.

Considering astronomers are usually experienced scripting programmers (but less so in web development) and the fact that Python is one of the most common languages used today, we have created *Vizic*, a Jupyter-based interactive visualization tool, which is a Python package designed to work with the Jupyter Notebook App. This tool (formally IPython (Pérez and Granger, 2007) Notebook) is a server-client application that allows creating and running Jupyter notebooks in which Python code can be executed interactively in executable cells. Unlike other existing tools, *Vizic* tries to make the connection between a source catalog and its images by drawing astronomical sources on an interactive map, where we can clearly visualize the spatial distribution of the observed objects, and at the same time keeping full control over the compositions, through objects filtering, color mapping, over plotting layers, among other useful features to visualize patterns on the data in a much more efficient manner. Since Jupyter Notebook App is a server-client system, *Vizic* can be used for accessing data archives at remote locations as well as working with catalogs stored on a local machine. Regarding the rising popularity of Jupyter Notebook App within the science community, we believe in providing an interactive interface without limiting the scripting ability to extend and to use this package, which will enable a faster scientific discovery.

The rest of the article is organized as follows. In Section 2, we start with a brief introduction to *Vizic* and highlight some of its important features. In Section 3, we discuss the technical details regarding the architecture and dependencies of *Vizic*. In Section 4, we present an example application of *Vizic* in visualizing large-scale structure. Section 5 provides the installation instruction, and Section 6 assesses the performance of *Vizic* with various loads. Lastly, in Section 7, we gives the conclusions, and describes future plans and possible extensions.

2 VIZIC

Vizic is a Python and Javascript library, which provides the API to display and interact with sky maps created from catalogs within Jupyter notebooks. At the front-end, the interactive sky maps are generated using *Leaflet*¹ (Agafonkin., 2016), a

popular JavaScript library for building interactive map applications on the web, and rendered through the widget framework provided by *ipywidgets*² (Project Jupyter Contributors, 2016), which is a library of interactive HTML widgets for Jupyter notebooks and IPython kernels. *Vizic* works side by side with MongoDB³ (MongoDB, 2016), therefore an active MongoDB instance, either running in the background or in a remote location (see Figure 1 for a workflow diagram), is required when *Vizic* is used. MongoDB is a non SQL, document-oriented database, which is built for scalability and performance. Since the data is stored in documents instead of multiple tables with complex relations, MongoDB database can spread the data over different servers and nodes. In the case of storing simple astronomical catalogs, MongoDB is particularly suitable. MongoDB also provides a geospatial index for making extremely efficient location-based queries. The rational to use MongoDB is discussed in more detail in Section 3.4.

The interactive sky maps can be easily created from catalogs stored in pandas (McKinney, 2010) DataFrames or catalogs that are previously ingested into the database by providing the collection names. Data stored in other formats, for example, *Astropy* (The Astropy Collaboration et al., 2013) Table object, can also be fed into *Vizic* after a simple conversion to pandas DataFrame. A *pandas* DataFrame is a Python object that handles arrays and tabular data very efficiently.⁴ When a DataFrame is provided, *Vizic* will format the data to match the mapping mechanism used, which is described in more detail in Section 3.2. In the sky maps, astronomical objects are drawn using their shapes, sizes and rotation angles determined by source catalog extractor softwares such as *SExtractor* (Bertin and Arnouts, 1996) to cite an example. If relevant shape information is not available, the objects will be drawn as filled circles with appropriate radius provided in the catalog. If neither the shape data nor the size data is provided, the objects will be drawn as filled circles with a fixed radius of 0.5 pixel at the minimum zoom level, however alternative sizes can be specified as an argument, and scaled up as the map is zoomed in.

There are two main categories of widgets provided by *Vizic*, namely, map widgets and control widgets. Map widgets are responsible for creating and managing sky maps and overlays. The core map widgets include *AstroMap* and *GridLayer*. The *AstroMap* widget is the container for all map layers, and the *GridLayer* widget is the tile-based layer for displaying the astronomical sources or simply the catalog layer. Additional custom overlay widgets are also provided for aid in visualizing large-scale structure (see Section 2.3). The control widgets are the main interfaces that take advantage of the extensive interactivity of the displayed objects (see Section 2.2). For example, through control widgets, we can filter the displayed objects by their property values and apply custom colormaps to these sources. These control widgets do not possess mutual reliance, therefore they can be used independently from each other. Such a modularized design brings a much cleaner

²<https://ipywidgets.readthedocs.io/en/latest>

³<https://www.mongodb.com/>

⁴<http://pandas.pydata.org>

¹<http://leafletjs.com/>

graphical user interface (GUI) comparing to other tools. Users can create their own application and interface by selecting only the needed components and interact with the data and the maps from within the Jupyter notebooks. After all, *Vizic* provides the foundation for a complete and customizable analysis, which is complementary to the standard scripting analysis workflows.

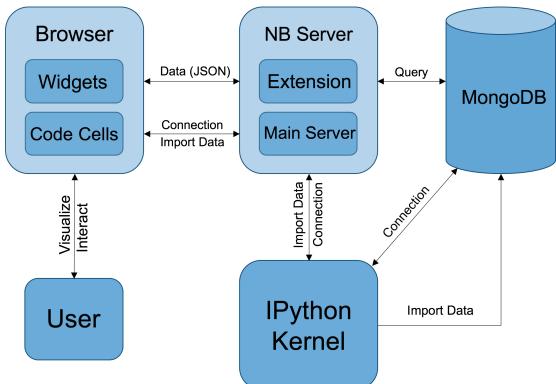


Figure 1: A diagram showing the workflow of *Vizic*. “NB Server” refers to Jupyter Notebook server.

2.1 Sky Maps

Before any sky maps can be created, the connection to a running MongoDB instance has to be established. The connection could be initiated by creating a `Connection` class object with a URL that specifies the address of the MongoDB instance and a port number. In the background, the IPython kernel creates a MongoDB client using the provided information and assigns the client to an attribute of the `Connection` object, which is required for creating `GridLayer` widgets. After the connection is successfully established, the `AstroMap` widget and the `GridLayer` widget instances can be created by initializing the corresponding Python object. To display catalogs in the notebook cells, we need to add the `GridLayer` widget to the `AstroMap` widget using the `add_layer` function. Then we can zoom and pan the map interactively as with a Google Map⁵. The transformation of the celestial coordinates is accomplished by a customized coordinate reference system (CRS) JavaScript library that determines the projection scale based on the extent of the map measured in degrees. In this manner, we can use RA and DEC information from the sources directly.

When creating catalog layers from `pandas` DataFrames, the default columns used for reading the locations are the standard RA and DEC. The default columns used for reading the shape and orientation of each object are A_IMAGE, B_IMAGE, THETA_IMAGE, which are respectively the semi-major axis, semi-minor axis and the rotation angle used in the Scalable Vector Graphics (SVG) (W3C, 2011b) ellipse adopted to represent that object. This process is repeated for every object in

⁵<https://www.google.com/maps>

the catalog. Different column names can also be specified in the arguments for reading this information.

2.2 Interaction

Beyond the basic panning and zooming, further interactions with the sky map are carried out in three directions: the ability to easily and interactively query the catalog from the MongoDB database, the ability to filter the objects in the map based on their properties, and the ability to apply different colormaps to the displayed objects. *Vizic* utilizes advanced web technologies, such as HTML5 and D3.js⁶ (Bostock, 2016), as well as the framework provided by ipywidget⁷ to make such interactivity possible.

2.2.1 Data Query

Vizic provides two methods to interactively query the catalog data. The first method is carried out through the selection tool and the query button widget. The selection tool allows users to make lasso-like selections on a sky map, where the geospatial coordinates of the selection bound are automatically stored in the `AstroMap` object. When the query button is clicked, *Vizic* sends a request to the database, asking for all of the objects that resides within the selection bounding box. The MongoDB engine retrieves the requested objects using the geospatial index and returns them to the IPython kernel where *Vizic* then reads the returned data using a `pandas` DataFrame and assign that DataFrame to the `select_data` attribute of the corresponding `GridLayer` object.

The second method is to directly click on the displayed astronomical sources. Since these objects are drawn using Scalable Vector Graphics (SVG) elements instead of a HTML canvas, each object can individually responds to mouse events. When such a SVG element catches a “click” event, it queries the database for the particular object it represents. The returned data is first parsed into a `pandas` Series object and then assigned to the `object_catalog` attribute of the `GridLayer` object. To continuously observe the data returned while clicking through different objects, a user can create a `PopupVis` widget for the rendered `GridLayer` widget. The `PopupVis` widget displays the data in a HTML table and constantly checks for updates from the `object_catalog` attribute.

The selection tool mentioned and the `PopupVis` widget can be found in Figure 2, which shows a customized GUI built using the widgets provided by *Vizic*. Once data is retrieved from the MongoDB, it can be further processed from other notebook cells, providing a continuous analysis framework.

2.2.2 Data Filter

Vizic also allows the users to filter displayed objects by their properties. In Jupyter notebooks, the filtering is controlled by a

⁶<https://d3js.org>

⁷<https://github.com/ipython/ipywidgets>

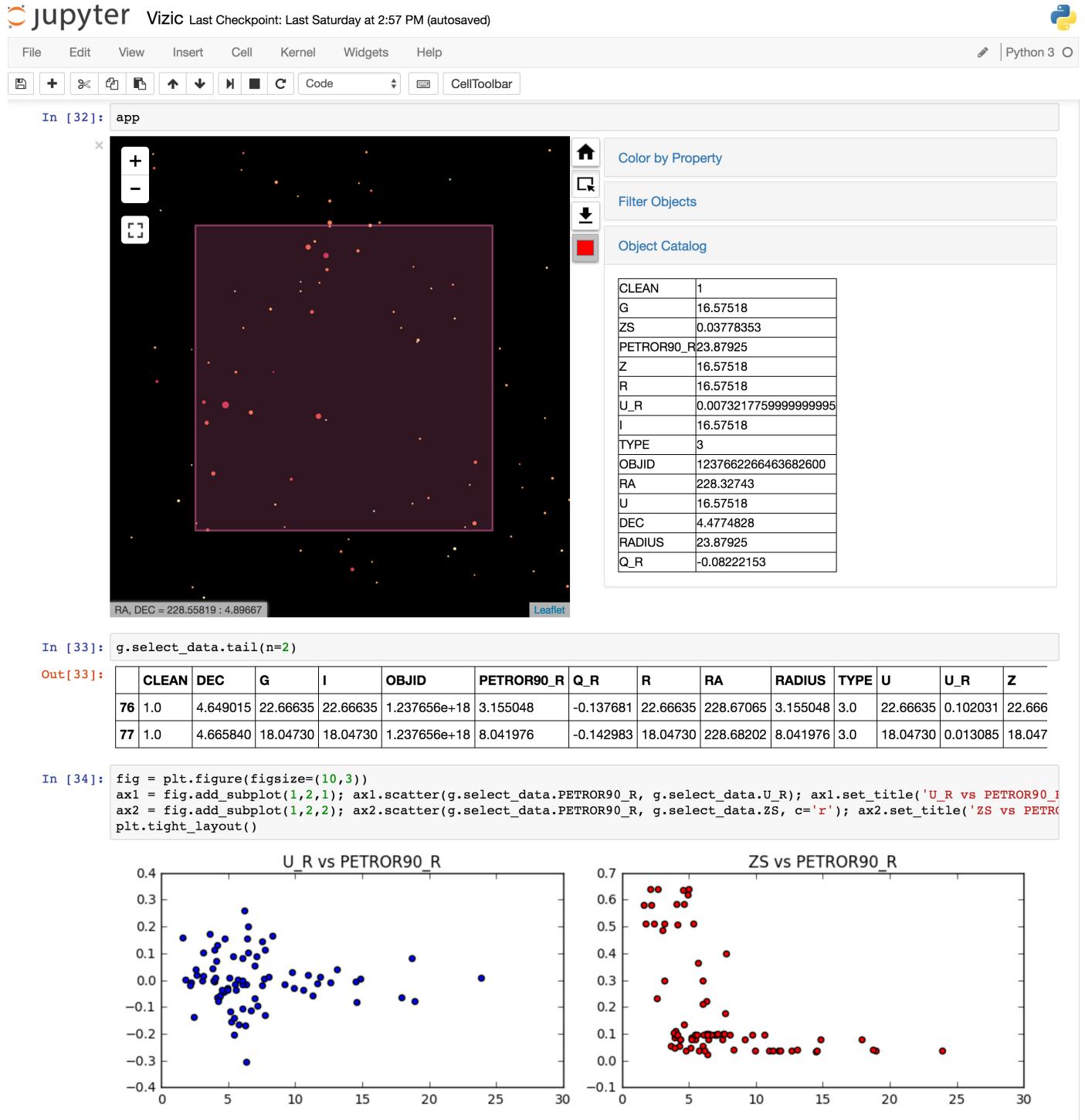


Figure 2: A GUI created from widgets provided by *Vizic*, where the dashboard is completely customizable. The pink rectangle on the sky map is the selection bounding box. The table below the “Object Catalog” toolbar displays the catalog entry for a clicked object. The middle section shows the data returned by the selection tool in a *pandas* DataFrame for direct manipulation through the notebook. Two plots at the bottom of this figure are generated using the data returned by the selection tool. The objects on the sky map are colored by their magnitude in *I* band. And the size of these objects are scaled up by a factor of 3 to better visualize the effect of color mapping.

range slider bar. The slider bar and another dropdown menu are wrapped into a single control widget, *FilterWidget*. The dropdown menu in this control widget is responsible for switching the property field used to filter the objects. If a new

property field is selected from the dropdown menu, the slider bar will update itself with the new maximum value and minimum value for that particular field. The property fields used to filter the object as well as the maximum and minimum val-

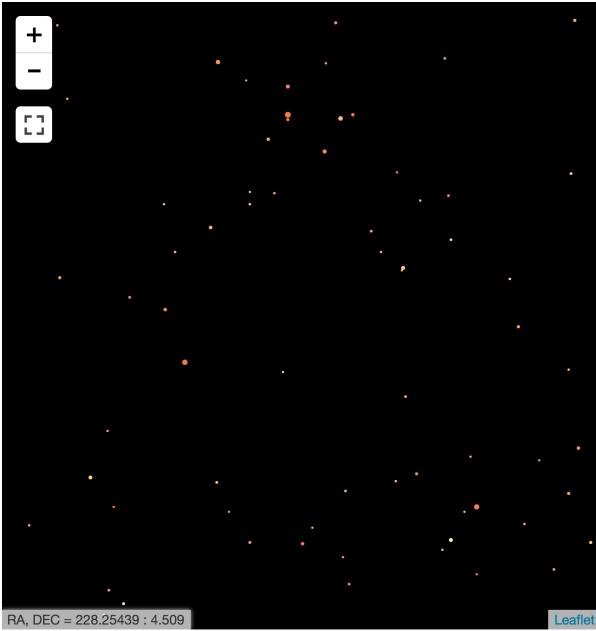


Figure 3: The same area as shown in Figure 2, but only with object that has an I magnitude (Doi et al., 2010) between 17.955 and 27.249.

ues for each field are automatically determined and stored into the meta document⁸ in the catalog collection during the data ingestion process.

Every time the selected range on the slider bar changes, the `GridLayer` object validates the new range, updates the record and pushes the change to the front-end. As soon as the front-end widget receive the updates, it uses D3.js to select all objects that are outside the specified range and hides them. If a new property field is selected, *Vizic* resets the filtering parameters and revalidates each object.

The objects filtering feature can be beneficial in many scenarios. For instance, we can easily determine outliers in the catalog by observing changes on the sky map while moving the slider bar from one side to another. If the redshifts are provided, we can also visually inspect how cosmic structure evolves by shifting the selected range on the slide bar. In Figure 3, we shows the same area of the sky displayed by Figure 2, but only displaying the objects with an I band magnitude between 17.955 and 27.249. Figure 4 is a screenshot of the `FilterWidget` used.

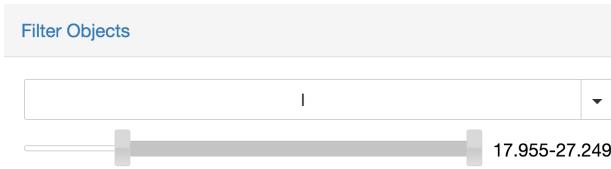


Figure 4: A screenshot showing the FilterWidget

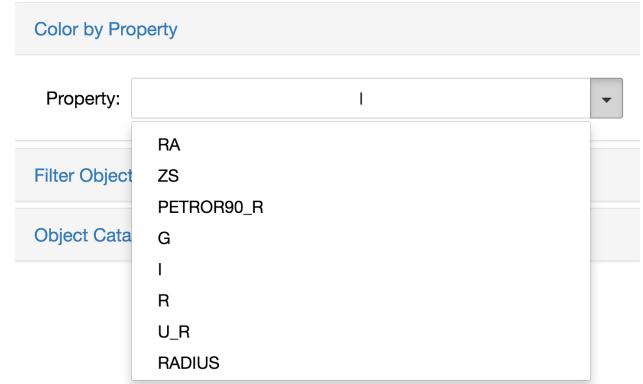


Figure 5: A dropdown menu example showing properties that can be used for applying colormaps

2.2.3 Color Maps

Color mapping is another powerful feature offered by *Vizic* for visual inspections of a given dataset. The default colormap used by *Vizic* is the spectral scheme. Any of the property field available in the filtering widget could be selected for color mapping. The `CFDropdown` control widget is a dropdown menu (see Figure 5) that serves as an interface for switching the property field used for color mapping. When the custom color mapping mode is enabled, the front-end JavaScript maps the value of the selected property from each object in the catalog to a range of $[0, 1]$ in a continuous and linear scale. Using the scaled value, *Vizic* selects the matching color from the colormap and assigns the color value to the SVG element representing the particular object. Beside the spectral scheme, *Vizic* provides a variety of additional colormaps defined in D3.js⁹. The user can change the colormap applied using the control widget, `ColorMap`. As a new colormap is chosen, the colors for the SVG elements will be reassigned based on the new color space. However, if the property field being used is changed, the scaled value of each object is subject to recalculation.

2.3 Custom Overlay

Vizic allows custom overlays to be appended on top of the tile-based layer. *Vizic* provides four pre-defined overlay layers, a Voronoi diagram (Voronoi, 1908) layer, a Delaunay triangulation (Delaunay, 1934) layer, a minimum spanning tree (MST) (Boruvka, 1926; Kruskal, 1956) layer and a HEALPix¹⁰ (Górski et al., 2005) grid layer. HEALPix stands for **Hierarchical Equal Area isoLatitude Pixelation**. These overlays are implemented through four different layer classes. New layer widget can be added and removed from the map using the `add_layer` function and the `remove_layer` function of the `AstroMap` class respectively. Once the overlay is added, it can scale and adapt itself to follow the interactive movements on the map. The rest of this subsection describes each overlay individually and

⁸A document with ‘`_id`’ equals ‘`meta`’

⁹<https://github.com/d3/d3-scale-chromatic>

¹⁰<http://healpix.sourceforge.net>

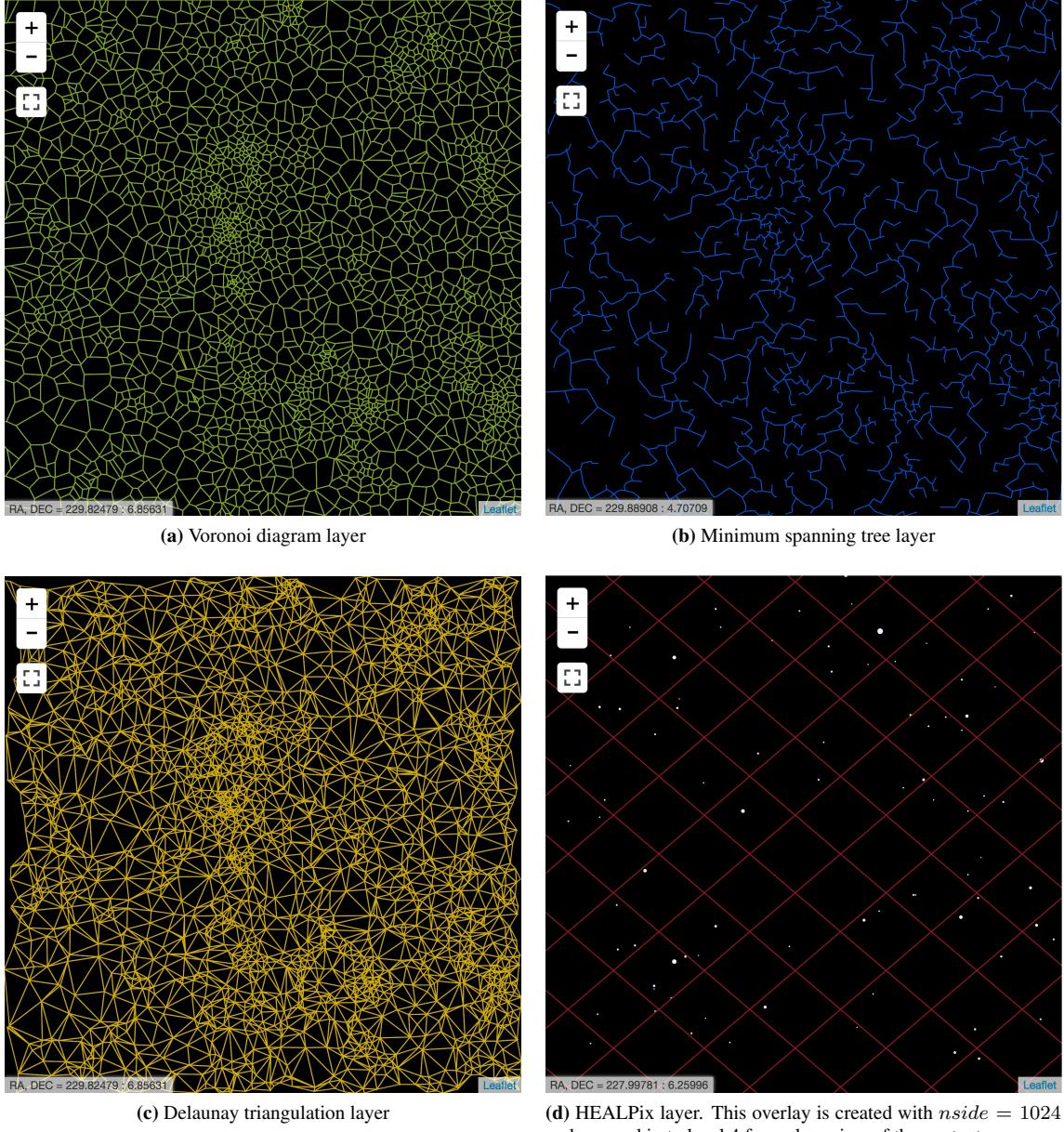


Figure 6: Custom overlays appended on the same area of the sky, they can be over plotted on top of one another.

briefly discusses its construction procedure.

2.3.1 Voronoi Diagram Layer

A Voronoi diagram for n given points (sites) in a plane divides up the plane into n regions (Voronoi cells), such that each region contains only one site and for every point inside that region the associated site is always the closest site. Each Voronoi cell is also a convex polygon. The shared side between any two polygons is the perpendicular bisector of the line segment connecting the two corresponding sites. In astronomy, Voronoi diagram is commonly used in the study of cosmic structure (e.g., Icke and van de Weygaert, 1987; Ramella et al., 2001; van de

Weygaert, 2007). *Vizic* generates the Voronoi diagram by treating astronomical sources as sites and computing the diagram based on their locations in the sky.

The Voronoi diagram overlay for a particular sky map can be created by initializing a `VoronoiLayer` object with the corresponding `GridLayer` object as the first argument. Inside the `VoronoiLayer` initialization function, we first query the database for all objects on the given tiled catalog layer and convert their celestial coordinates into pixel coordinates on the screen following the projection mechanism described in Section 2.1. Then we compute the diagram for these objects and render the diagram by drawing the Voronoi cells using SVG polygons, which allows another level of customization such as color, width, etc.

2.3.2 Delaunay Triangulation Layer

We have also implemented a Delaunay triangulation overlay layer. The Delaunay triangulation for a set of points in a plane is a triangulation such that none of the points lies inside the circumcircles of the triangles in this triangulation. A Delaunay triangulation can also be seen as the dual graph of the Voronoi diagram for the same set of points. The Delaunay triangulation layer can be created using `DelaunayLayer` class. Similar to how we construct the Voronoi diagram overlay, we first project the given objects in a catalog onto the screen and calculate the Delaunay triangulation from the positions of these objects, finally we use SVG path elements to connect the vertices of each determined triangle.

2.3.3 Minimum Spanning Tree Layer

In graph theory, a spanning tree of a connected and undirected graph is a tree that contains all the nodes in that graph. If each edge in a given connected graph is assigned a weight, a MST of that graph is a spanning tree such that the total weight for all edges in this tree is the least. In this overlay, we use astronomical objects as the nodes and assign each edge a weight equals to its length measured in degrees.

The minimum spanning tree is computed using a combination of packages in the SciPy Stack (van der Walt et al., 2011; McKinney, 2010; Jones et al., 2001–; Pedregosa et al., 2011). We first use `sklearn.neighbors` (Pedregosa et al., 2011) to calculate the distance between each object and its nearest 20 (default value) neighbors and then determine the MST from all possible trees that can be created using the given edges (Jones et al., 2001–; Kruskal, 1956). The result is a compressed sparse matrix where the entry is zero if no edge exists between the two objects indexed by the row number and column number respectively. By connecting the objects referred by each non-zero entries, we obtain the MST.

The minimum spanning tree overlay can be created by initiating a `MstLayer` object with the `GridLayer` object as the first argument. Beyond visualizing the MST for a given set of objects on the sky map, we can also cut off the branches to select different structures (e.g., Barrow et al., 1985; Bhavsar and Ling, 1988). The user can prune the tree using the function `MstLayer.cut(rmax, nmin)` with r_{max} being the allowed maximum weight for saved edges and n_{min} being the minimum number of edges that a saved sub branch must include. As a result, only structures in a single branch with edges smaller than r_{max} are kept. Figure 7 shows a trimmed MST with a maximum weight of 0.055 degree or approximately 0.66 Mpc/h at a redshift (z) of 0.2 for groups with a minimum of eight members.

2.3.4 HEALPix Grid Layer

HEALPix is a method of pixelation that divides a spherical surface into quadrilaterals of equal area. Each quadrilateral, in this case, is considered as a pixel. The lowest resolution of a HEALPix

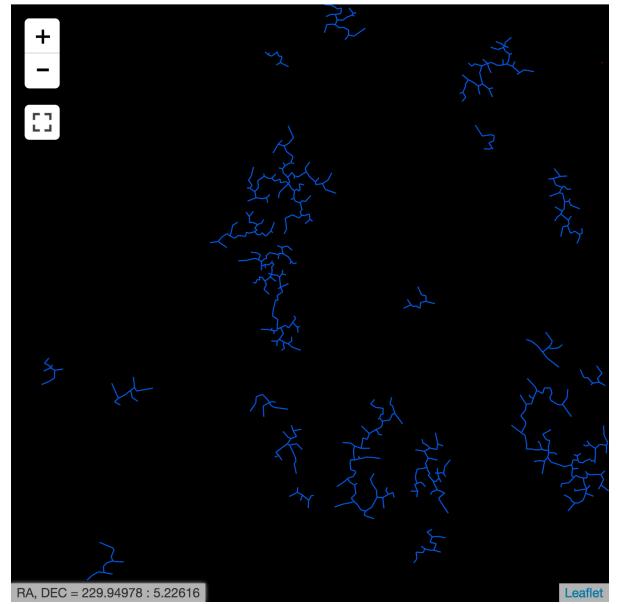


Figure 7: The tirmmed MST with $r_{max} = 0.055^\circ$ (0.66 Mpc/h at $z=0.2$) and $n_{min} = 8$

grid consist of 12 pixels and each higher resolution increase the total number pixels by four. In our implementation, we use `healpy`¹¹ to create the HEALPix grid, then project and connect the vertices of each pixel to draw the grid. A demonstration of an appended HEALPix layer is shown in Figure 6d.

To display a HEALPix grid, a `HealpixLayer` object needs to be created first, with the `GridLayer` object of the target catalog layer as the first argument. The resolution of the HEALPix grid can be specified by giving a new value to the keyword argument, `nside`, where the default is 1024. Like other overlay layers, a HEALPix grid overlay can be added or removed from the sky map using the `add_layer` or `remove_layer` function. The advantage of using HEALPix grid is that we can then easily compute densities or other quantities per HEALPix pixel, which will improve the interaction with the datasets further allowing extra types of density based analysis.

3 ARCHITECTURE & DEPENDENCIES

3.1 Extension Bundle

Technically speaking, *Vizic* is an extension bundle to the Jupyter Notebook App. It comes with a notebook extension and a server extension. The notebook extension is written in both Python and JavaScript. The Python code mainly consists of the APIs to create interactive widgets and a set of functions to interact with them. The JavaScript section is the front-end interface that actually constructs the widgets in the browser, and also handles the interactions on the widgets by either sending state changing message to the kernel or responding directly at the front-end. The Python code that runs in the kernel and its JavaScript counterpart that runs in the browser communicate with each other

¹¹<https://github.com/healpy/healpy>

through the “comms” messaging API provided by the Jupyter Notebook.

Beyond the excellent platform offered by Jupyter Notebook, *Vizic* also takes advantage of *ipywidgets* and *ipyleaflet*¹². *ipywidgets* offers a widget-model-view framework for building widgets within Jupyter notebooks. All widgets in *Vizic* are built based on the widget-model-view structure. The widget-model-view structure uses IPython *traitlets*¹³ to keep attributes or states synchronized at both front and back end and to trigger further actions according to state changes, which makes it possible to complete tasks both interactively using widgets and from the command line within the notebook cells. IPython *traitlets* is a framework that lets Python classes have attributes with type checking, dynamically calculated default values, and on change callbacks. That way the user can build custom widgets to control the state of the sky maps and layers or to run and trigger additional scripts based on state changes. Such flexibility makes *Vizic* even more powerful and extensible.

In addition, *ipyleaflet* is a notebook extension package that integrates the *Leaflet* library into the Jupyter Notebook framework environment.

The server extension is written also in Python using *Tornado*¹⁴. This extension connects the kernel and the map widgets to the MongoDB database. Whenever the front-end sky map asks for new data, the request will go through the server extension and return with the query results. To enhance the performance, we used *Motor*¹⁵, an asynchronous Python driver for MongoDB, to access the database from the server. As a result, multiple tiles on a map can be loaded concurrently, and multiple maps visualizing different catalogs can be displayed in one single notebook without a heavy performance cost.

3.2 Slippy Map

The main concept that stands behind the interactive maps created by *Vizic* is the “slippy map” implementation. The “slippy map” or tiled web map is the standard method of serving large maps on the web. A well-known example adopting this approach is Google Maps. For a better illustration of the “slippy map” mechanism, we have constructed a 3-D diagram of its pyramid-like data structure using a Hubble image as an example (note that *Vizic* does not display images), and with the diagram shown in Figure 8. The images at different zoom levels are identical in terms of content but with different resolutions. The higher the zoom level, the higher the resolution of the image. At each zoom level, the image is divided into many small square tiles, each with a size of 256×256 pixel. The number of tiles at zoom level n is be determined by

$$N_{tile} = 2^n \times 2^n$$

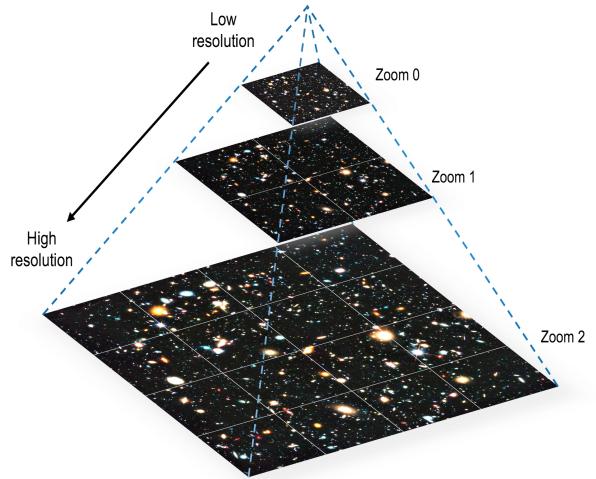


Figure 8: 3D diagram explaining the pyramid like structure of the tiles system. Image Credit: NASA, ESA, H. Teplitz and M. Rafelski (IPAC/Caltech), A. Koekemoer (STScI), R. Windhorst (Arizona State University), and Z. Levay (STScI)

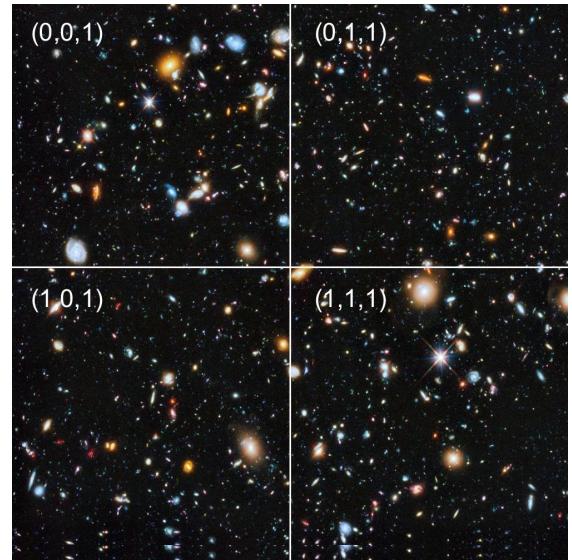


Figure 9: A figure showing the coordinate system used to locate the tiles. Image Credit: NASA, ESA, H. Teplitz and M. Rafelski (IPAC/Caltech), A. Koekemoer (STScI), R. Windhorst (Arizona State University), and Z. Levay (STScI)

Since the map window on our monitor is fixed, the number of small tiles that the window can display is relatively stable. The JavaScript engine at the front-end only needs to request the tiles that can be displayed and disregard the rest for efficiency. The initial idea behind this algorithm is that for an image that is much larger than the size of map window measured pixels, a big portion of this image is hidden at high zoom levels, which makes it very inefficient to load the entire image to the client side. However, at lower zoom levels, many pixels are compressed into a single pixel on the display, therefore we are not able to perceive the information offered by the full-

¹²<https://github.com/ellisonbg/ipyleaflet>

¹³<https://traitlets.readthedocs.io/en/stable/>

¹⁴<http://www.tornadoweb.org/>

¹⁵<https://motor.readthedocs.io>

resolution image. The “slippy map” implementation avoids the heavy loads by offering a pyramid-like multi-resolution structure, which in fact has been proved to be very efficient. In order to correctly locate and request the tiles for *Vizic*, each tile is also assigned an ID following the pattern (x, y, z) , where x and y are row number and column number respectively and z is the zoom level (see Figure 9).

Traditionally, in an interactive map, each tile is a PNG image, we have made a step forward by vectorizing these tiles before display. We store each object in the catalog as a document in the MongoDB database and create the visual representations on the fly. Before importing the catalog into the database, we split the data into $2^n * 2^n$ groups following a similar grid as shown in Figure 9, where n is the maximum zoom level with a default value of 8, which can be changed at the catalog ingestion process. Each group represents a tile at the maximum zoom level and objects in that group is assigned the tile ID of the group. When a lower zoom level tile is requested, the ID for the requested tile is translated into a set of tile IDs at the maximum zoom level. This way we can further reduce the size and complexity of the data stored in the database. The projection functions are shown below:

$$\begin{aligned} x_{max} &= xc * 2^{z_{max}-z} - 1 \\ x_{min} &= (xc + 1) * 2^{z_{max}-z} \\ y_{max} &= yc * 2^{z_{max}-z} - 1 \\ y_{min} &= (yc + 1) * 2^{z_{max}-z} \end{aligned}$$

$x_{max}, x_{min}, y_{max}, y_{min}$ are the maximum and minimum values of x and y in the tile ID respectively. z_{max} is the maximum zoom level and z is the zoom level of the requested tile. At lower zoom levels, we reduce the load by leaving out objects that are too small to see (i.e., not resolved by the screen).

In particular, objects that have a semi-minor axis (or radius) smaller than one third of a pixel after being projected onto the screen will not be included in the data query. Here the number one third of a pixel is chosen because it provides extra tolerance for conversion offsets than one half of a pixel. A shape that is smaller than one pixel either has a very low opacity or becomes invisible on the screen, so it is unnecessary to spend resource on those objects. By projecting tile coordinates and leaving out invisible objects, we have successfully emulated the “slippy map” implementation in a vectorized way for catalog data.

The “slippy map” implementation is ideal for both displaying static images and displaying vectorized data, however the current scope of *Vizic* is not to display static images. Even though, *Vizic* can be extended to display both images and catalogs in the same manner.

3.3 Custom Overlays

As described in Section 2.3, *Vizic* allows custom overlays to be appended on top of the visualized catalogs. We make this possible by extending the `Layer` class in *Leaflet* and providing specific drawing constructions based on the type of geometric shape contained in the overlay. Since it is unpractical to

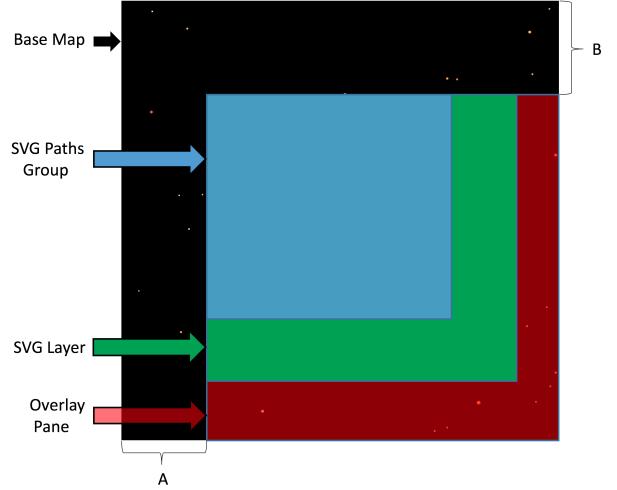


Figure 10: A diagram displays the layer structure of the mapping system. A is the offset in longitude direction. B is the offset in latitude direction.

redraw the overlay using canvas element every time the viewport changes (e.g., zoom and pan), we instead create the overlay with a collection of SVG elements. SVG elements are vector-based, therefore we only need to create them once and resize them as the map size changes. Because the *Leaflet* map is a multi-layered system (see Figure 10), to accommodate the offsets generated by rescaling of the overlay when zoom level changes, we put these SVG elements into a group element, and scale and shift them together. The offsets are determined as follows:

$$\begin{aligned} A &= lon_{base} - lon_{op} \\ B &= lat_{base} - lat_{op} \end{aligned}$$

A and B are the offsets in the longitude and latitude directions, (lon_{base}, lat_{base}) is the coordinate for the base map and (lon_{op}, lat_{op}) is the coordinate for the overlay pane. In *Leaflet*, panes are used to control the ordering of layers on the map. Overlay pane is the default pane for all the vector overlay layers.

3.4 MongoDB

Vizic uses MongoDB databases to store and serve the catalogs. MongoDB is a document-oriented database, which stores rows from a traditional RDBMS database into documents to enable faster query and better scalability. Considering astronomical catalogs are simply collections of observed objects without complex relations between each columns when it comes to visualizing them, and since *Vizic* only asks for objects and their properties from a catalog using their positions in the sky, MongoDB is a better choice comparing to a traditional RDBMS database.

Vizic stores the data for each object in a document, each collection of documents is a catalog and each database can contain many collections. Every document is indexed by its tile

ID and the size of the object measured in arcsecond to enhance query performance. *Vizic* also takes advantage of MongoDB’s internal geospatial index to efficiently retrieve data from the selection tool.

Another important feature of MongoDB is that it allows flexible data structures within a document. In other words, different documents within one collection can have different fields, which is extremely useful for visualizing a composite catalog where the objects come from multiple surveys and each survey provide their own measurements. Using a relational database, such tasks would be more difficult to accomplish.

4 EXAMPLE APPLICATION

In this section, we present an example usage of *Vizic* in scientific research. We first create an interactive map using a subset of the spectroscopic catalog from SDSS Data Release 13 (DR13) (SDSS Collaboration et al., 2016; Gunn et al., 1998). This subset covers the area from 227° to 230° in right ascension and from 4° to 7° in declination with a total of nine square degrees. This sample catalog contains only the objects from the $[0, 0.2]$ redshift bin (Smee et al., 2013). For demonstration purpose, we choose not to provide the shape and size information when creating the sky map.

Then we look for galaxy clusters and voids on the sky map and mark them with circle layers using different colors. For the sake of this example, the positions and sizes are selected by eye. The approximated centers of the clusters and voids are determined by the mouse position tracker located on the bottom left corner of the map (See Figure 11). Next, we apply the MST overlay onto the sky map. By pruning the tree with different combination of r_{max} and n_{min} , we can confirm our previous structure selections with the tree groups that are kept. We can observe that for a more strict cut in r_{max} , i.e. reducing maximum linkage length between two points, as shown in Figure 11a, compact structures selected by the MST are correlated to those circles shown in green (i.e., clusters), while for a more relaxed cut, in Figure 11b, the structures shown follow the underlying filament structure around voids (white circles) much better.

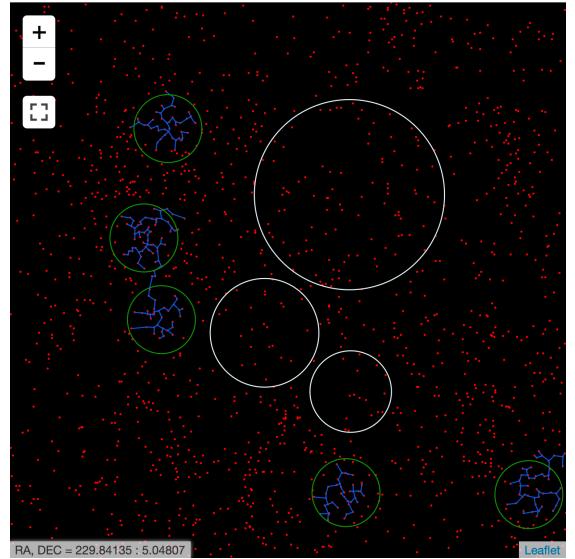
Additionally, other layers can be over-plotted, data can be selected for further inspection, objects can be filtered, colored and scaled to aid with visualization and analysis. Multiple maps, multiple cuts and multiple datasets can be used interactively with other scripting procedures and functions on the data from other cells in one single Jupyter notebook, increasing the potential and effectiveness of *Vizic*.

5 DISTRIBUTION & INSTALLATION

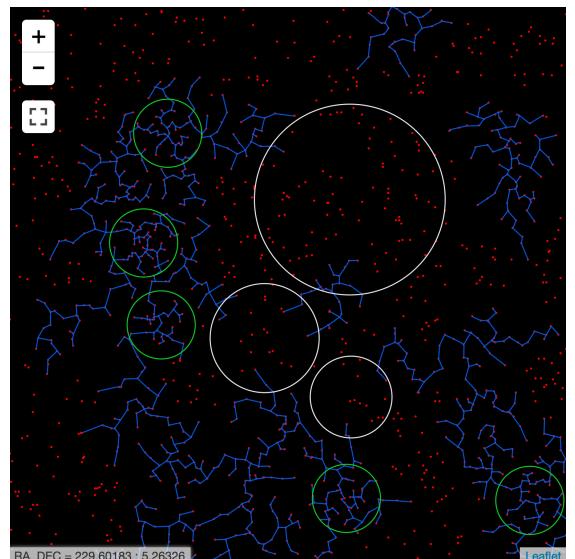
Vizic is registered with Python Package Index¹⁶, therefore it can be installed with *pip* for Python 3. It can also be installed from source code by downloading the GitHub repository¹⁷. *Vizic* re-

¹⁶<https://pypi.python.org/pypi/vizic>

¹⁷<https://github.com/ywx649999311/vizic>



(a) The trimmed MST with $r_{max} = 0.05^{\circ}$ (0.6 Mpc/h at $z=0.2$) and $n_{min} = 30$



(b) The trimmed MST with $r_{max} = 0.08^{\circ}$ ((0.957 Mpc/h at $z=0.2$)) and $n_{min} = 20$

Figure 11: Two figures showing different combination of r_{max} and n_{min} to prune the full MST to identify structures. The green circles mark the galaxy clusters while the white circles mark the voids.

quires a running MongoDB instance, the installation and setup instructions can be found from the official MongoDB website¹⁸. More detailed instructions on the installation of *Vizic* and its required dependencies as well as tutorials, API documentation and examples can be found in the official *Vizic* documentation¹⁹.

Instead of installing the package in a local machine and keeping a MongoDB instance running, the user can also build a Docker²⁰ container from the Dockerfile included in the GitHub

¹⁸<https://docs.mongodb.com/manual/>

¹⁹<http://www.wx-yu.com/vizic/index.html>

²⁰<https://www.docker.com>

repository. Both the MongoDB database and the Jupyter notebook server run inside the container with only designated ports exposed to the host for easy deployment. An example dataset and notebooks are included as part of the *Vizic* distribution on GitHub as well.

6 PERFORMANCE & DISCUSSION

Visualizing large datasets through a browser has always been a difficult task. Unlike some of the existing tools, the goal of *Vizic* is not to display as many objects as possible at once, but instead to reduce the number of displayed sources without losing important information using the “slippy map” implementation described before.

We set up several tests to examine the overall performance of *Vizic* regarding data ingestion and map interaction and deployment. All tests were performed on a Macbook Pro running OS X El Capitan equipped with a 2.5GHz Intel Core i7 (i7-4870HQ) processor, 16Gb of RAM and a SSD. Both MongoDB instance and the Jupyter App server were running locally. Unless specifically stated, *Vizic* was tested using the browser Chrome.

Figure 12 shows the time in seconds for the data ingestion as a function of catalog size (number of objects), and all catalogs ingested in this test contain fifteen columns. The linear baseline shown in this figure is a trendline for the first three data points on this plot. We notice that the actual performance curve starts to deviate from the linear baseline when the catalog size exceeds 3 million. We presume such diverging behavior as a result of computational resources being exhausted.

The second test performed, as shown in Figure 13, profiles the timings for map loading triggered when applying interactively different zoom levels. To focus on the effect of catalog sizes on the tiles loading performance, we created the test datasets by continuously and repeatedly throwing a collection of objects onto a $15^\circ \times 15^\circ$ area until the desired number of objects is reached, hence increasing the density of the objects each time. This collection of objects were selected from a same $15^\circ \times 15^\circ$ area in SDSS Data Release 13.

The tile loading time is a direct indication of the responsiveness of the map when zoom level changes interactively. In our test, we fixed the map window to 512 by 512 pixels and aligned the center of map with the center of the window. Then, four tiles were loaded simultaneously each time the zoom level changes. We used the Chrome DevTool to record the loading time for each tile and take the average as the final result. Accordingly, if more tiles are loaded at the same time, for example, in the full screen mode, the loading performance might differ but follow a similar trend seen in Figure 13.

Another critical measurement is the smoothness of the panning motion on the map. Without triggering new tile loadings, the responsiveness of the panning motion is proportional to the number of SVG elements added to the Document Object Model (DOM) tree, the larger the DOM tree the longer it takes to pan the map. In the tests performed above, the tile loadings were usually slower than the panning motion and we didn’t experi-

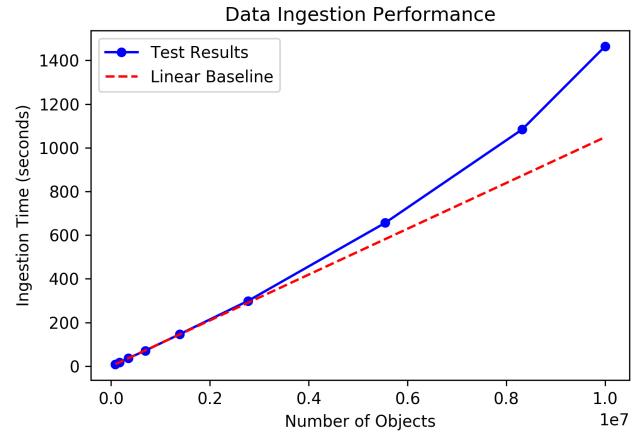


Figure 12: Data ingestion time as a function of catalog size. The linear baseline was created using the first three data points on this plot, showing the diverging character of the performance curve after 3 million objects.

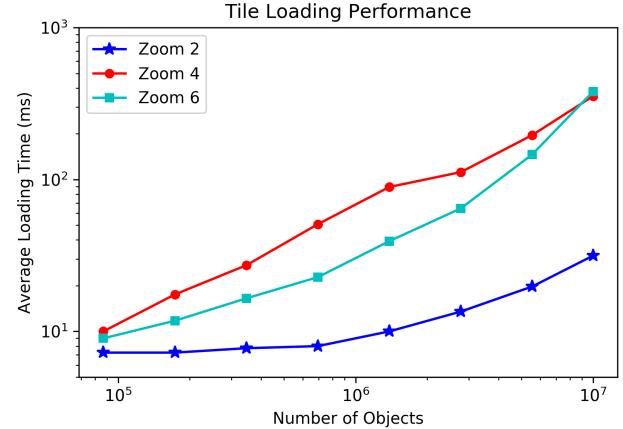


Figure 13: Display response and loading time as a function of catalog size for different zoom levels.

ence any significant lags, where the maximum number of objects loaded in one tile was around fifteen hundred. Nonetheless, higher latencies were observed in the full screen mode, as a direct result of increased number of SVG elements added to the DOM tree.

As shown in the performance tests, the largest catalog being visualized on the mentioned machine include 10 million rows and 15 columns. Due to the constraints of the available hardware, we were not able to further examine the limit of *Vizic*. However, we believe *Vizic* will perform well with any dataset that has less than 50 million objects, on a more powerful machine. In the future release, we expect to make *Vizic* capable of handling catalogs on the order of 10^8 and above. Currently if a user would like to work with a dataset on that order, we suggest the user to divide the entire dataset into several subareas and then create a map for each subarea. In a case that displaying the whole dataset and observing the big picture is most critical, the users can turn to tools like *VaeX*, but sacrificing the ability to further interact with the data through object filtering and

coloring, as well as the application of custom overlays.

At this stage, the pre-defined custom overlays are not suitable for use on a single large dataset (typically over 100K data points being displayed at a time). However, these overlays could also be made to work by first dividing the region covered by the large catalog into several small areas and then creating a map for each smaller area. If the interesting objects happen to be near the edges of these small maps, the user can use the provided selection tool to query the catalog from the large map and create another new map from it. In the future release of *Vizic*, a performance improvement related to the custom overlay is expected.

7 CONCLUSIONS & FUTURE WORK

In this paper, we present *Vizic*, a Python visualization package, which is designed to work with the Jupyter Notebook App. To offer an easy and complete analysis environment, *Vizic* utilizes the best part of both images and catalogs by visualizing astronomical sources on an interactive sky map and provides powerful interactions with the Jupyter notebook cells for further interactive scripting analysis with the data.

While resembling the spatial distributions of the sources on the original images, the interactive sky maps created using *Vizic* also allow us to filter or color displayed astronomical objects by their property values. The lasso-like selection tool provides us the ability to interactively select interesting objects from the sky map and retrieve their catalog from the linked MongoDB database. Alongside the returned DataFrame containing the catalog for the selected objects, we can further explore and analyze the hidden patterns among the data using various plotting packages available for Python and Jupyter, like *matplotlib*, *scikit-learn* and many others. In addition, *Vizic* offers us the option to over-plot custom layers on the base map. Four pre-defined custom overlays are included in this package. The Voronoi diagram overlay, the minimum spanning tree overlay and the Delaunay triangulation overlay provide an easy and fast way to visualize cosmological structures. The HEALPix grid overlay is a convenient tool to interactively inspect the density field generated by the catalog or to complement with other astronomical data. *Vizic* supports multiple datasets and multiple layers for a full exploration between different catalogs.

We also demonstrate an immediate application of *Vizic* by identifying galaxy clusters, voids and filaments. By utilizing the tree pruning feature provided by the minimum spanning tree overlay, we can match the saved tree branches with previously selected structures to visualize and detect these structures. The result shows that *Vizic* is a powerful and friendly tool for visualizing large-scale structure. However, its usage is not limited to just galaxy catalogs, any form of catalog data can be easily handled by *Vizic*, such as star and galaxy clusters, star forming regions in a galaxy, etc.

In general, *Vizic* provides a new way to efficiently visualize and interact with astronomical catalogs. Nevertheless, these features provided by *Vizic* can also be applied to data from other scientific fields.

Multiple improvements over the current version has already been planned. A major improvement is to further increase the scalability of this tool. While keep increasing the number of objects that *Vizic* can efficiently visualize (with a maximum loading time of 0.5 second for each zoom level), we also would like to match the responsiveness of the custom overlays to that of the tiled base layer. We are currently testing new method of building these overlay layers using Web Components. With Web Components we can isolate each overlay from the outside world, so that the CSS style of the elements inside the overlay component is not affected by style changes originated from outside the overlay. Such feature can dramatically reduce the amount of time consumed by style recalculations when the map shifts locations.

Another improvement is integrating *Vizic* with other interactive plotting libraries and making it even easier for astronomers to explore the catalogs by taking advantage of the scripting power of Python, in a way that multiple tools are connected and the change in one is reflected in the other one, providing a complete user analysis experience in a Jupyter notebook.

ACKNOWLEDGEMENTS

RJB acknowledges support from the National Science Foundation Grant No. AST-1313415.

Some of the results in this paper have been derived using the HEALPix (Górski et al., 2005) package.

Funding for the Sloan Digital Sky Survey IV has been provided by the Alfred P. Sloan Foundation, the U.S. Department of Energy Office of Science, and the Participating Institutions. SDSS-IV acknowledges support and resources from the Center for High-Performance Computing at the University of Utah. The SDSS web site is www.sdss.org.

SDSS-IV is managed by the Astrophysical Research Consortium for the Participating Institutions of the SDSS Collaboration including the Brazilian Participation Group, the Carnegie Institution for Science, Carnegie Mellon University, the Chilean Participation Group, the French Participation Group, Harvard-Smithsonian Center for Astrophysics, Instituto de Astrofísica de Canarias, The Johns Hopkins University, Kavli Institute for the Physics and Mathematics of the Universe (IPMU) / University of Tokyo, Lawrence Berkeley National Laboratory, Leibniz Institut für Astrophysik Potsdam (AIP), Max-Planck-Institut für Astronomie (MPIA Heidelberg), Max-Planck-Institut für Astrophysik (MPA Garching), Max-Planck-Institut für Extraterrestrische Physik (MPE), National Astronomical Observatories of China, New Mexico State University, New York University, University of Notre Dame, Observatório Nacional / MCTI, The Ohio State University, Pennsylvania State University, Shanghai Astronomical Observatory, United Kingdom Participation Group, Universidad Nacional Autónoma de México, University of Arizona, University of Colorado Boulder, University of Oxford, University of Portsmouth, University of Utah, University of Virginia, University of Washington, University of Wisconsin, Vanderbilt University, and Yale University.

REFERENCES

- Agafonkin, V., 2016. Leaflet - a JavaScript library for interactive maps. URL: <http://leafletjs.com/>.
- Barrow, J.D., Bhavsar, S.P., Sonoda, D.H., 1985. Minimal spanning trees, filaments and galaxy clustering. *MNRAS* 216, 17–35. doi:10.1093/mnras/216.1.17.
- Beaumont, C., Goodman, A., Greenfield, P., 2015. Hackable User Interfaces In Astronomy with Glue, in: Taylor, A.R., Rosolowsky, E. (Eds.), Astronomical Data Analysis Software and Systems XXIV (ADASS XXIV), p. 101.
- Bertin, E., Arnouts, S., 1996. SExtractor: Software for source extraction. *Astronomy and Astrophysics, Supplement* 117, 393–404. doi:10.1051/aas:1996164.
- Bertin, E., Pillay, R., Marmo, C., 2015. Web-based visualization of very large scientific astronomy imagery. *Astronomy and Computing* 10, 43–53. doi:10.1016/j.ascom.2014.12.006, arXiv:1403.6025.
- Bhavsar, S.P., Ling, E.N., 1988. Are the filaments real? *Astrophysical Journal, Letters* 331, L63–L68. doi:10.1086/185236.
- Boch, T., Fernique, P., 2014. Aladin Lite: Embed your Sky in the Browser, in: Manset, N., Forshay, P. (Eds.), Astronomical Data Analysis Software and Systems XXIII, p. 277.
- Borůvka, O., 1926. O jistém problému minimálním. *Práce Moravské přírodovědecké společnosti, sv. III, spis 3*, 37–58.
- Bostock, M., 2016. D3: Data-Driven Documents. URL: <https://d3js.org/>.
- Breddels, M.A., 2016. Interactive (statistical) visualisation and exploration of a billion objects with Vex. Proceeding IAU Symp. 325 Astroinformaticsroceeding IAU Symp. 325 Astroinformatics URL: <http://arxiv.org/abs/1612.04183>, arXiv:1612.04183.
- Davis, A.J.J., 2016. Motor - the async Python driver for MongoDB and Tornado or asyncio. URL: <https://github.com/mongodb/motor>.
- Delaunay, B., 1934. Sur la sphère vide. *Bull. Acad. Sci. URSS* 1934, 793–800.
- Docker, I., 2016. Docker - Build, Ship, and Run Any App, Anywhere. URL: <https://www.docker.com>.
- Doi, M., Tanaka, M., Fukugita, M., Gunn, J.E., Yasuda, N., Ivezić, Ž., Brinkmann, J., de Haars, E., Kleinman, S.J., Krzesinski, J., French Leger, R., 2010. Photometric Response Functions of the Sloan Digital Sky Survey Imager. *Astronomical Journal* 139, 1628–1648. doi:10.1088/0004-6256/139/4/1628, arXiv:1002.3701.
- FriendFeed, 2016. Tornado Web Server. URL: <https://github.com/tornadoweb/tornado>.
- Górski, K.M., Hivon, E., Banday, A.J., Wandelt, B.D., Hansen, F.K., Reinecke, M., Bartelmann, M., 2005. HEALPix: A Framework for High-Resolution Discretization and Fast Analysis of Data Distributed on the Sphere. *Astrophysical Journal* 622, 759–771. doi:10.1086/427976, arXiv:astro-ph/0409513.
- Granger, B., 2014. ipyleaflet: An IPython Widget for Leaflet Maps. URL: <https://github.com/ellisonbg/ipyleaflet>.
- Gunn, J.E., Carr, M., Rockosi, C., Sekiguchi, M., Berry, K., Elms, B., de Haas, E., Ivezić, Ž., Knapp, G., Lupton, R., Pauls, G., Simcoe, R., Hirsch, R., Sanford, D., Wang, S., York, D., Harris, F., Annis, J., Bartozek, L., Boroski, W., Bakken, J., Haldeman, M., Kent, S., Holm, S., Holmgren, D., Petrvick, D., Prosaio, A., Rechenmacher, R., Doi, M., Fukugita, M., Shimasaku, K., Okada, N., Hull, C., Siegmund, W., Mannery, E., Blouke, M., Heidtman, D., Schneider, D., Lucinio, R., Brinkman, J., 1998. The Sloan Digital Sky Survey Photometric Camera. *Astronomical Journal* 116, 3040–3081. doi:10.1086/300645, arXiv:astro-ph/9809085.
- Hunter, J.D., 2007. Matplotlib: A 2D Graphics Environment. *Computing in Science and Engineering* 9, 90–95. doi:10.1109/MCSE.2007.55.
- ICKE, V., van de Weygaert, R., 1987. Fragmenting the universe. *Astronomy and Astrophysics* 184, 16–32.
- International, E., 2009. ECMA-262 ECMAScript Language Specification. *JavaScript Specif.* 16, 1–252. URL: <http://www.ecma-international.org/publications/standards/Ecma-262.htm>.
- Jones, E., Oliphant, T., Peterson, P., et al., 2001–. SciPy: Open source scientific tools for Python. URL: <http://www.scipy.org/>. [Online; accessed 28.12.16].
- Kaiser, N., Aussel, H., Burke, B.E., Boesgaard, H., Chambers, K., Chun, M.R., Heasley, J.N., Hodapp, K.W., Hunt, B., Jedicke, R., Jewitt, D., Kudritzki, R., Luppino, G.A., Maberry, M., Magnier, E., Monet, D.G., Onaka, P.M., Pickles, A.J., Rhoads, P.H.H., Simon, T., Szalay, A., Szapudi, I., Tholen, D.J., Tonry, J.L., Waterson, M., Wick, J., 2002. Pan-STARRS: A Large Synoptic Survey Telescope Array, in: Tyson, J.A., Wolff, S. (Eds.), Survey and Other Telescope Technologies and Discoveries, pp. 154–164. doi:10.1117/12.457365.
- Kruskal, J.B., 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. Am. Math. Soc.* 7, 48–48. doi:10.1090/S0002-9939-1956-0078686-7.
- Laureijs, R., Gondoin, P., Duvet, L., Saavedra Criado, G., Hoar, J., Amiaux, J., Auguères, J.L., Cole, R., Cropper, M., Ealet, A., Ferruit, P., Escudero Sanz, I., Jahnke, K., Kohley, R., Maciaszek, T., Mellier, Y., Oosterbroek, T., Pasian, F., Sauvage, M., Scaramella, R., Sirianni, M., Valenziano, L., 2012. Euclid: ESA's mission to map the geometry of the dark universe, in: Space Telescopes and Instrumentation 2012: Optical, Infrared, and Millimeter Wave, p. 84420T. doi:10.1117/12.926496.
- LSST Science Collaboration, Abell, P.A., Allison, J., Anderson, S.F., Andrew, J.R., Angel, J.R.P., Armus, L., Arnett, D., Asztalos, S.J., Axelrod, T.S., et al., 2009. LSST Science Book, Version 2.0. ArXiv e-prints arXiv:0912.0201.
- McKinney, W., 2010. Data structures for statistical computing in python, in: van der Walt, S., Millman, J. (Eds.), Proceedings of the 9th Python in Science Conference, pp. 51 – 56.
- MongoDB, I., 2016. MongoDB for GIANT Ideas. URL: <https://www.mongodb.com/>.
- Moolekamp, F., Mamajek, E., 2015. Toyz: A framework for scientific analysis of large datasets and astronomical images. *Astronomy and Computing* 13, 50–57. doi:10.1016/j.ascom.2015.10.001, arXiv:1506.08930.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, É., 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* URL: <http://hal.inria.fr/hal-00650905>.
- Pérez, F., Granger, B.E., 2007. IPython: A system for interactive scientific computing. *Comput. Sci. Eng.* 9, 21–29. URL: <http://ieeexplore.ieee.org/document/4160251/>, doi:10.1109/MCSE.2007.53.
- Project Jupyter Contributors, 2016. ipywidgets: Interactive HTML Widgets. URL: <https://github.com/ipython/ipywidgets>.
- Ragan-Kelley, M., Perez, F., Granger, B., Kluyver, T., Ivanov, P., Frederic, J., Bussonier, M., 2014. The Jupyter/IPython architecture: a unified view of computational research, from interactive exploration to communication and publication. *AGU Fall Meeting Abstracts*, D7.
- Ramella, M., Boschin, W., Fadda, D., Nonino, M., 2001. Finding galaxy clusters using Voronoi tessellations. *Astronomy and Astrophysics* 368, 776–786. doi:10.1051/0004-6361:20010071, arXiv:astro-ph/0101411.
- SDSS Collaboration, Albareti, F.D., Allende Prieto, C., Almeida, A., Anders, F., Anderson, S., Andrews, B.H., Aragon-Salamanca, A., Argudo-Fernandez, M., Armengaud, E., et al., 2016. The Thirteenth Data Release of the Sloan Digital Sky Survey: First Spectroscopic Data from the SDSS-IV Survey MAppling Nearby Galaxies at Apache Point Observatory. ArXiv e-prints arXiv:1608.02013.
- Smei, S.A., Gunn, J.E., Uomoto, A., Roe, N., Schlegel, D., Rockosi, C.M., Carr, M.A., Leger, F., Dawson, K.S., Olmstead, M.D., Brinkmann, J., Owen, R., Barkhouse, R.H., Honscheid, K., Harding, P., Long, D., Lupton, R.H., Loomis, C., Anderson, L., Annis, J., Bernardi, M., Bhardwaj, V., Bizyaev, D., Bolton, A.S., Brewington, H., Briggs, J.W., Burles, S., Burns, J.G., Castander, F.J., Connolly, A., Davenport, J.R.A., Ebelke, G., Epps, H., Feldman, P.D., Friedman, S.D., Frieman, J., Heckman, T., Hull, C.L., Knapp, G.R., Lawrence, D.M., Loveday, J., Mannery, E.J., Malanushenko, E., Malanushenko, V., Merrelli, A.J., Muna, D., Newman, P.R., Nichol, R.C., Oravetz, D., Pan, K., Pope, A.C., Ricketts, P.G., Sheldon, A., Sandford, D., Siegmund, W., Simmons, A., Smith, D.S., Snedden, S., Schneider, D.P., SubbaRao, M., Tremonti, C., Waddell, P., York, D.G., 2013. The Multi-object, Fiber-fed Spectrographs for the Sloan Digital Sky Survey and the Baryon Oscillation Spectroscopic Survey. *Astronomical Journal* 146, 32. doi:10.1088/0004-6256/146/2/32, arXiv:1208.2233.
- The Astropy Collaboration, A., Robitaille, T.P., Tollerud, E.J., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., Davis, M., Ginsburg, A., Price-Whelan, A.M., Kerzendorf, W.E., Conley, A., Crighton, N., Barbary, K., Muna, D., Ferguson, H., Grollier, F., Parikh, M.M., Nair, P.H., Günther,

- H.M., Deil, C., Woillez, J., Conseil, S., Kramer, R., Turner, J.E.H., Singer, L., Fox, R., Weaver, B.A., Zabalza, V., Edwards, Z.I., Bostroem, K.A., Burke, D.J., Casey, A.R., Crawford, S.M., Dencheva, N., Ely, J., Jenness, T., Labrie, K., Lim, P.L., Pierfederici, F., Pontzen, A., Ptak, A., Refsdal, B., Servillat, M., Streicher, O., 2013. Astropy: A Community Python Package for Astronomy. *Astron. Astrophys.* Vol. 558, id.A33, 9 pp. 558. doi:10.1051/0004-6361/201322068, arXiv:1307.6212.
- The Dark Energy Survey Collaboration, 2005. The Dark Energy Survey. ArXiv Astrophysics e-prints arXiv:astro-ph/0510346.
- Tyson, J.A., 2002. Large Synoptic Survey Telescope: Overview, in: Tyson, J.A., Wolff, S. (Eds.), *Survey and Other Telescope Technologies and Discoveries*, pp. 10–20. doi:10.1117/12.456772, arXiv:astro-ph/0302102.
- van de Weygaert, R., 2007. Voronoi Tessellations and the Cosmic Web: Spatial Patterns and Clustering across the Universe. ArXiv e-prints arXiv:0707.2877.
- Voronoi, G., 1908. Nouvelles applications des paramtres continus la thorie des formes quadratiques. premier mmoire. sur quelques proprits des formes quadratiques positives parfaites. *Journal fr die reine und angewandte Mathematik* 133, 97–178. URL: <http://eudml.org/doc/149276>.
- W3C, 2011a. Cascading Style Sheets (CSS) Snapshot 2010. Technical Report. URL: <http://www.w3.org/TR/css-2010/>.
- W3C, 2011b. Scalable Vector Graphics (SVG) 1.1 (Second Edition). URL: <http://www.w3.org/TR/SVG/>.
- W3C, 2014. HTML5. URL: <http://www.w3.org/TR/html5/>.
- van der Walt, S., Colbert, S.C., Varoquaux, G., 2011. The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering* 13, 22–30. doi:10.1109/MCSE.2011.37.
- York, D.G., Adelman, J., Anderson, Jr., J.E., Anderson, S.F., Annis, J., Bahcall, N.A., Bakken, J.A., Barkhouser, R., Bastian, S., Berman, E., Boroski, W.N., Bracker, S., Briegel, C., Briggs, J.W., Brinkmann, J., Brunner, R., Burles, S., Carey, L., Carr, M.A., Castander, F.J., Chen, B., Colestock, P.L., Connolly, A.J., Crocker, J.H., Csabai, I., Czarapata, P.C., Davis, J.E., Doi, M., Dombeck, T., Eisenstein, D., Ellman, N., Elms, B.R., Evans, M.L., Fan, X., Federwitz, G.R., Fiscelli, L., Friedman, S., Frieman, J.A., Fukugita, M., Gillespie, B., Gunn, J.E., Gurbani, V.K., de Haas, E., Haldeman, M., Harris, F.H., Hayes, J., Heckman, T.M., Hennessy, G.S., Hindsley, R.B., Holm, S., Holmgren, D.J., Huang, C.h., Hull, C., Husby, D., Ichikawa, S.I., Ichikawa, T., Ivezić, Ž., Kent, S., Kim, R.S.J., Kinney, E., Klaene, M., Kleinman, A.N., Kleinman, S., Knapp, G.R., Korienek, J., Kron, R.G., Kunysz, P.Z., Lamb, D.Q., Lee, B., Leger, R.F., Limmongkol, S., Lindenmeyer, C., Long, D.C., Loomis, C., Loveday, J., Lucinio, R., Lupton, R.H., MacKinnon, B., Mannery, E.J., Mantsch, P.M., Margon, B., McGehee, P., McKay, T.A., Meiksin, A., Merelli, A., Monet, D.G., Munn, J.A., Narayanan, V.K., Nash, T., Neilsen, E., Neswold, R., Newberg, H.J., Nichol, R.C., Nicinski, T., Nonino, M., Okada, N., Okamura, S., Ostriker, J.P., Owen, R., Pauls, A.G., Peoples, J., Peterson, R.L., Petracick, D., Pier, J.R., Pope, A., Pordes, R., Protopiano, A., Rechenmacher, R., Quinn, T.R., Richards, G.T., Richmond, M.W., Rivetta, C.H., Rockosi, C.M., Ruthmansdorfer, K., Sandford, D., Schlegel, D.J., Schneider, D.P., Sekiguchi, M., Sergey, G., Shimabukuro, K., Siegmund, W.A., Smee, S., Smith, J.A., Snedden, S., Stone, R., Stoughton, C., Strauss, M.A., Stubbs, C., SubbaRao, M., Szalay, A.S., Szapudi, I., Szokoly, G.P., Thakar, A.R., Tremonti, C., Tucker, D.L., Uomoto, A., Vanden Berk, D., Vogelsley, M.S., Waddell, P., Wang, S.i., Watanabe, M., Weinberg, D.H., Yanny, B., Yasuda, N., SDSS Collaboration, 2000. The Sloan Digital Sky Survey: Technical Summary. *Astronomical Journal* 120, 1579–1587. doi:10.1086/301513, arXiv:astro-ph/0006396.