The Small Body Map for the Average Person

Colm Lang
University of San Francisco
San Francisco, CA 94117
cplang@dons.usfca.edu

***Abstract*—With this comprehensive Small Body Map for the Average Person (SBMAP), I provide a digestible, holistic representation of our solar system's many small bodies without sacrificing detail or accuracy. I have successfully created a lightweight interactive visualization for the web by prioritizing data-processing and sampling methods. Namely, with the use of Python's Pandas and Matplotlib along with an implementation of multivariate stratified sampling in TypeScript, I was able to reduce the data-set's bulk while maintaining its characteristics and quirks. This ultimately allowed for a rich, *details on demand* approach for the Birds Eye View visualization and a more responsive Asteroid Dashboard.**

I. Introduction

In the recent years I have developed a keen interest in outer space and with this project I want to bridge the gap between Astrophysicists and the rest of us. When data about outer space is collected it is often either displayed in ways that the average person cannot understand, or stripped of its intricacies to dumb down the content. It is my goal to provide a resource that keeps the depth of NASA's data without needing an advanced degree to understand.

A. Project Objectives

The goal of SBMAP is to produce a digestible holistic representation of our solar system's many small bodies that does not sacrifice detail or accuracy. To accomplish this, the project contains the following:

1) A **birds eye view** of our solar system and where most asteroids orbit
2) Analysis of the **distribution** of the small bodies' orbit distance
3) Analysis of small body **dimensions** and **size**

II. Related Work

The inspiration of this project came from the Small Body Database Visualizations by NASA/JPL. Therein, it had a visualization of every small body's location on 1/1/2018 by P. Chodas [5]. While this visualization was very nice, I thought that adding each body's full orbital path, rather than just showing a singular location, would be significant. This created a necessity for large-scale aggregation, as one million ellipses would hardly be the best way to show a birds eye view of our solar system's many small bodies. It was at this stage that I came across Mike Bostock's D3 Hexbin Observable post [3].

A classmate, Sean Sanchez, referenced a great visualization by Eleanor Lutz titled *A Map of Every Object in our Solar System* [9] that—aptly named—showed a detailed map of each body's position in our solar system. Color coordinated and visually appealing, the visualization proved to be quite inspirational.

When beginning to add transitions to the Asteroid Dashboard, I wanted to explore ways to interpolate the density plots and the contour plot. I imagined the best way to transition between states of contours would be to "lower" and "raise" the different contour sections to their new values. The best source I found to mimic this intuitive transition was done by Paul Murray. In an Observable post, he proposes the tweening of each point's weight between the previous state and the current state [12]. This built upon the very useful *D3 Density Contours* example by Mike Bostock [2]. Bostock's example was incredibly useful in the development of the Asteroid Dashboard's density contour.

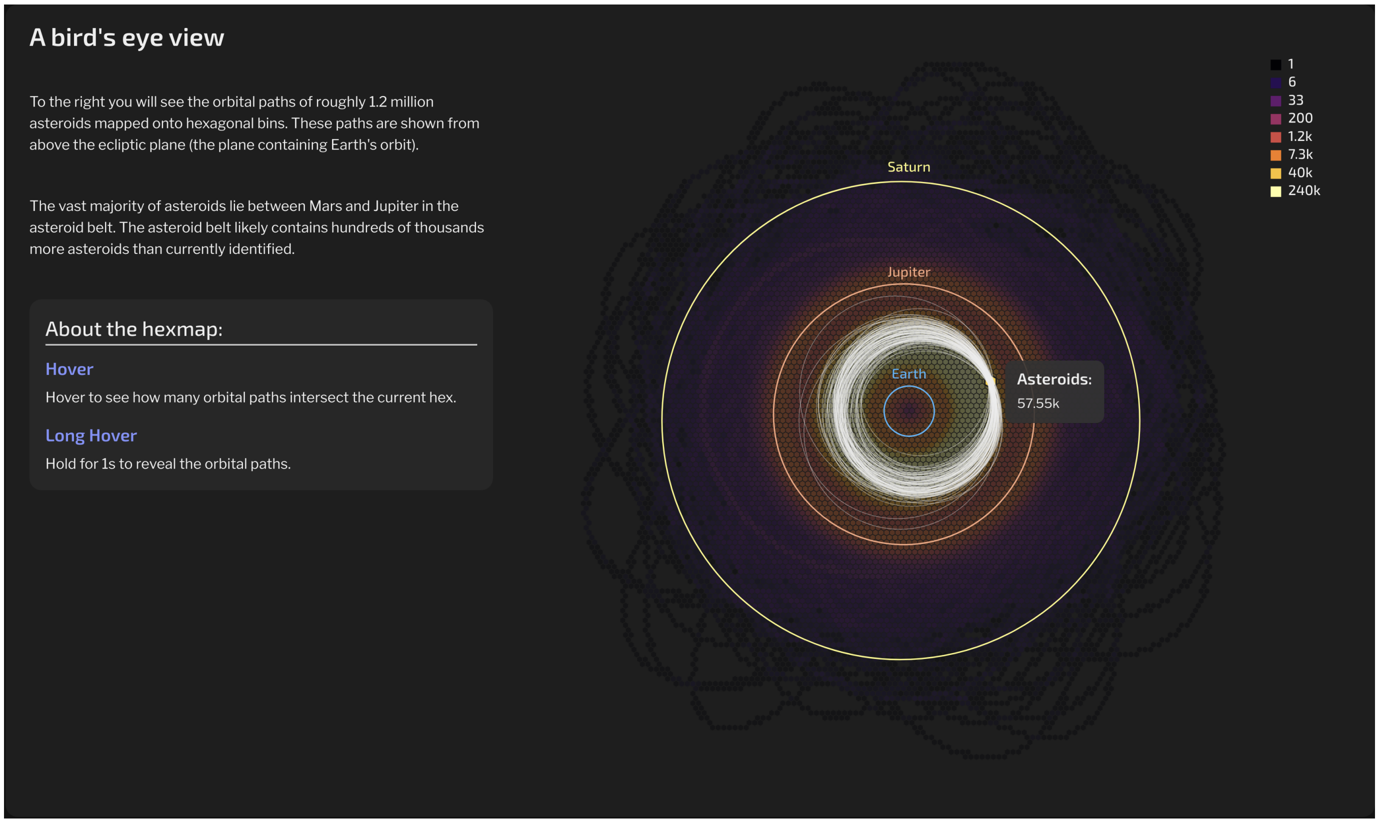
III. Approach

Throughout the development process, I maintained the following strategy: Data first, Concise Data Visualization second, then UI/UX and Sass [15] considerations, and Optimization last. This strategy ensured that the Data Visualization techniques I used would be aptly suited to the data I had. By thinking in this manner, I was able to reasonably tackle each project objective with limited need for big adjustments. A great example of this would be how I handled the data processing of the hexmap visualization.

A. The Hexmap

The birds eye view visualization was ultimately developed using the process laid out above. I began with data-processing by placing each ellipse into bins and aggregating the total count of intersections. I used Pandas [13] for data manipulation and Matplotlib's [10] hexbin implementation. First I would calculate the points of each ellipse in the data set then I would get the set of hexes that the ellipse intersected. For each ellipse in that set I would increment the count and add the body's ID to the hexes set of intersected orbits. I chose this approach to allow me to ship a much smaller file to the client. Rather than needing 1.2 million lines, I only needed to send the hexbins and their counts.

Due to this algorithm's computational intensity and the sheer size of the data set (with roughly 1.2 million orbits), it takes roughly 75 hours to complete synchronously. By using Python's Multiprocessing [11], however, I reduced the algorithm run time by 92% (down to 6 hours).


A bird's eye view

To the right you will see the orbital paths of roughly 1.2 million asteroids mapped onto hexagonal bins. These paths are shown from above the ecliptic plane (the plane containing Earth's orbit).

The vast majority of asteroids lie between Mars and Jupiter in the asteroid belt. The asteroid belt likely contains hundreds of thousands more asteroids than currently identified.

About the hexmap:

Hover

Hover to see how many orbital paths intersect the current hex.

Long Hover

Hold for 1s to reveal the orbital paths.

Fig. 1. The Birds Eye View during a hover interaction. Note the orbital paths that have been drawn and the tooltip displaying the total count of asteroids intersecting the hex.

Here is a simple pseudo code demonstration of the data processing done in Python:

```
for body in dataset:
    # get all points based on
    # the body's orbital parameters
    points = get_ellipse_points(body)

    # get all bins based on the points
    bins = get_bins_by_points(points)

    for bin in bins:
        # if intersects
        if bin.count > 0:
            # add id to list
            bin.ids += body.id

    # update intersection totals
    total_bins += bins
```

To supplement user interaction, I wanted to use the visualization technique that I originally planned: ellipses. Done in reasonably quantities, I found that displaying orbital ellipses when a user hovers on a hex was actually quite effective. Therefore, I knew I would need to implement the calculation of orbital intersections into the hexbinning algorithm. Through a simple ID relationship, I added this feature to the data processing algorithm.

When a user hovers a hex to display the intersecting orbits, I use JavaScript's setTimeout function to draw the orbits after one second. Originally, I would draw them immediately with an opacity of zero, then fade them in after a delay of one second. While this worked, it would render the underlying hexes for every hex that was hovered, however brief the hover was. By choosing to use setTimeout, I was able to defer renders until the JavaScript call stack was clear after one second, which made things much faster. To ensure that this wouldn't just create the same issue as before—albeit one second delayed—I cleared the timeout when the user's mouse would exit the hex. This prevents more that one set of hex intersection ellipses from being drawn. I found that this performed much faster and I liked how it ensures a limited amount of ellipses being drawn at a given time.

B. The Asteroid Dashboard

The Asteroid Dashboard displays a Kernel Density Estimation (KDE) plot of orbit distance, a KDE plot of diameter, and a contour plot of orbit eccentricity to orbit distance. This dashboard displays four different asteroid groups with the ability to toggle any/all of them for data exploration.

Therefore, not only does the plot need to draw quickly, it must update quickly as well. Here I handled state using React's *useState* hook [14] to track the current filtered groups to display. While this was a natural choice due to the React environment, filtering each plot using this state was not so straightforward. Using the paradigm set forth in *Towards Reusable Charts* [4] I abstracted the contour plot and KDE plot into their own files. This allowed one to call the data function with new filtered data and each plot would only update the changed information, rather than redrawing completely. Within these files I was also able to implement transition logic for smooth transitions between state.

All of this was only possible with a reasonably sized data set. I knew this before the visualization implementations described above, so I implemented sampling to reduce the data set from 140,000 entries to a more reasonable 10,000 entries. While a random sample would likely be fine, I wanted to ensure that the sample population accurately represented the distribution in all three dimensions (orbit distance, diameter, and eccentricity).

My solution resembles some sort of multivariate stratified sampling. I thought that by grouping the data points by orbit distance, then grouping each group by diameter, then again by eccentricity, one could be sure that each group is similar in all three dimensions. To produce a sample population of 10,000 points I simple take one in every 14 entries to reduce the population.

For the actual transitions in the dashboard I used *interpolatePath* [1] for the KDE plot to remove the unexpected behavior that often accompanies svg path transitions. As mentioned above, I used the weight tweening paradigm put forth by Paul Murray [12] to achieve contour transitions.

C. The Violin Plot

The implementation of the Violin Plot was initially done through binning the bodies into 0.01au bins. This was effective yet limiting because the processing required some hard-coding of values. I would then render a curve that traces the exact count of each bin. After creating multivariate stratified sampling that retained the distributions of the data set, I could reuse the same data that is used for the dashboard for the Violin Plot. This would reduce the bundle size and remove any unnecessary file parsing. It also seemed reasonable to replace the naive binning method with the already implemented KDE for a nicer curve.

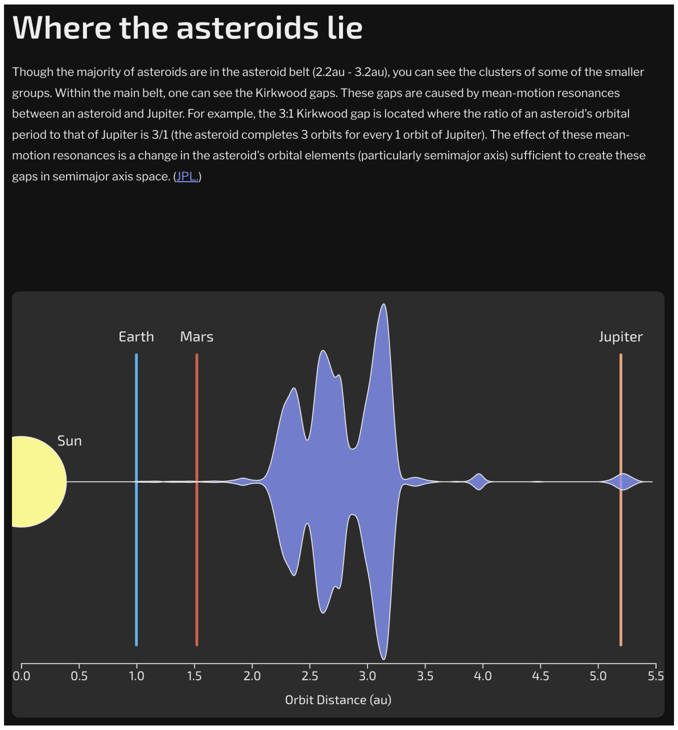
D. The Halley's Comet Scrollama

The Halley's Comet Scrollama was built using Jason Kao's *React Scrollama* library [8] as a foundation. Again, I adopted the *Towards Reusable Charts* paradigm to manage state transitions. With a simple variable keeping track of the view, I could trigger a transition between the 'side' view—seen in Figure 4a—and the 'above' view—seen in Figure 4b.

IV. Results

A. The Hexmap

The Hexmap Visualization shown in Figure 1 very effectively shows a **birds eye view** of our solar system. I find that this coupled with the ability for details on demand makes this visualization very effective. While the user hover interaction was very fast and crisp on Opera, during the feedback session for the Beta Release my group members brought browser issues to my attention. On Chrome, hover interactions were extremely slow and clunky but would seem to speed up after the first few minutes or so. Meanwhile, on Safari this visualization, along with the other visualizations, would not even render. Browser compatibility aside, the hexmap is very fast. Given that there are over 100,000 underlying ellipses that can be drawn in the hexmap, the way I handled their renders was very efficient and ensured that a maximum of 300 were being drawn at a time.



Where the asteroids lie

Though the majority of asteroids are in the asteroid belt (2.2au - 3.2au), you can see the clusters of some of the smaller groups. Within the main belt, one can see the Kirkwood gaps. These gaps are caused by mean-motion resonances between an asteroid and Jupiter. For example, the 3:1 Kirkwood gap is located where the ratio of an asteroid's orbital period to that of Jupiter is 3/1 (the asteroid completes 3 orbits for every 1 orbit of Jupiter). The effect of these mean-motion resonances is a change in the asteroid's orbital elements (particularly semimajor axis) sufficient to create these gaps in semimajor axis space. (JPL.)

Fig. 2. Using a sample of asteroids generated using multivariate stratification, the Violin Plot shows a Kernel Density Estimation of orbital distance. Note the annotations of the Earth, Mars, Jupiter, and the Sun for helpful context.

B. The Violin Plot

The Violin Plot seen in Figure 2 utilized the data sampling and KDE implemented for the Asteroid Dashboard. This reuse aligns with the import I placed on a light client side. In the end, the violin plot served as a great way to visualize the **distribution** of orbit distance among the asteroids. The annotations I provided for the Sun, the Earth, Mars, and Jupiter create invaluable context to actual orbit distance for the asteroids.

C. The Asteroid Dashboard

The Asteroid Dashboard seen in Figure 3 allows the user to investigate the **dimensions** and **size** of each asteroid group. The transitions between each group is done effectively and smoothly, allowing for seamless user interactions. With the

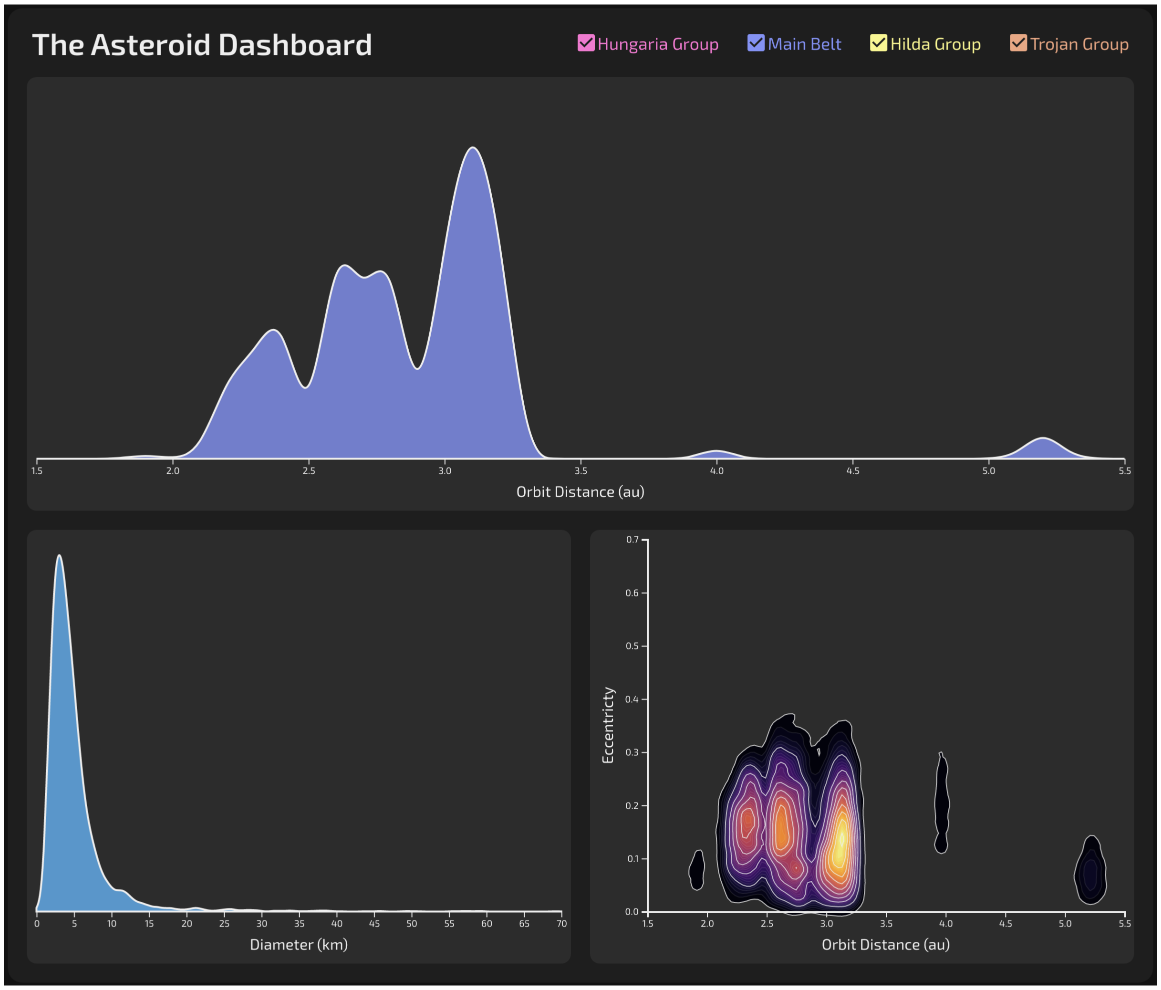


Fig. 3. The Asteroid Dashboard

adaptation of Murray's *Animated Contours* [12] and Beshai's *d3-interpolate-path* [1] I was able to create very satisfying and pleasant transitions, at no cost to performance. Behind the scenes, React's *useState* [14] and Mike Bostock's *Towards Reusable Charts* [4] paradigm ensured that there was no cumbersome or slow code. With this in mind, I am very pleased with the Asteroid Dashboard and I would certainly deem this a success.

D. The Halley's Comet Scrollama

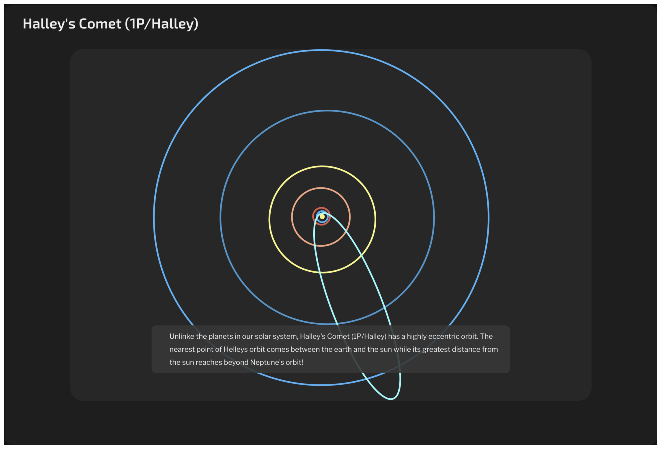
Though not directly aligned with my Project's Objectives, I was very pleased with the Halley's Comet Scrollama shown in Figure 4. I found the scroll-based interaction very satisfying and the transitions I implemented to be both clean and semantic. Moreover, the transition from the overhead view seen in Figure 4a to the side view shown in Figure 4b helps to show the user we are tilting our view to examine a side angle. What is being shown might not be so clear to the user with a transition that is not semantic. However, I must concede that it has far too few steps. The Scrollama would benefit greatly from more steps, both in user experience but also with the information I could provide to the user.

V. Discussion

Overall, I found the approach I took to be very promising. More specifically, I liked the strategy I used for development. Each stage I came to was very manageable and I think that additional feature implementation would be very reasonable using this strategy.

The *Towards Reusable Charts* [4] paradigm that I implemented in the dashboard was extremely effective. Because the visualization technique was unchanging and all that needed to be updated was the underlying data, having an object with a data property kept the update action very simple. This ensured not only that data points were drawn only once, but that each user interaction required the smallest changes possible. Due to the code cleanliness, and re-usability of this approach, I would definitely like to adopt it in the rest of the project.

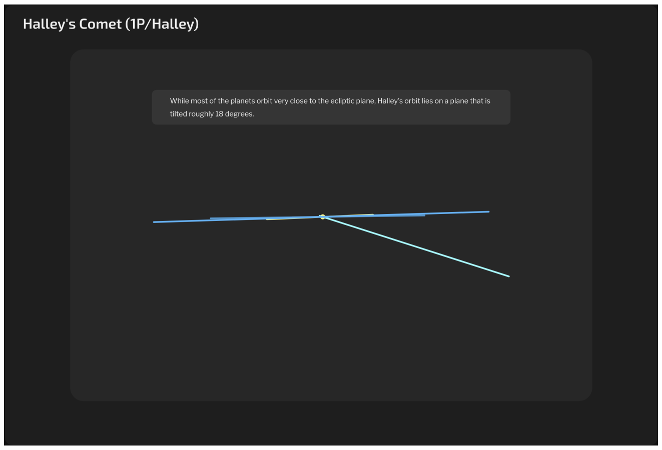
There was much to learn from this project but I found that data processing and the importance of chart re-usability served to be the most significant learning points. React's state management coupled very nicely with the reusable charts. Moreover, I learned so much about the delicate balance between React and D3's manipulation of the DOM [6]. Furthermore,



Halley's Comet (1P/Halley)

Unlinke the planets in our solar system, Halley's Comet (1P/Halley) has a highly eccentric orbit. The nearest point of Helleys orbit comes between the earth and the sun while its greatest distance from the sun reaches beyond Neptune's orbit!

(a) The first step of the Halley's Comet Scrollama. This visualization shows off the incredibly eccentric orbit of 1P/Halley that comes between the earth and the Sun and reaches beyond Neptune's orbit.



Halley's Comet (1P/Halley)

While most of the planets orbit very close to the ecliptic plane, Halley's orbit lies on a plane that is tilted roughly 18 degrees.

(b) Upon scrolling far enough, the orbits shown in Figure 4a transition to display the orbits' inclinations. This shows off 1P/Halley's incredible inclination.

Fig. 4. The two steps of the Halley's Comet Scrollama, with 4a showing an overhead view of our solar system, and 4b showing a side view of our solar system—in reference to the ecliptic plane.

the necessity for clean, fast code presented itself the deeper I got in the project. I learned a lot about how important considerations like code architecture, documentation, and project management are.

VI. Future Work

While I found the visualizations to be quite effective, I would like to add supplementary context features. Additions like a glossary, more annotations, and more tooltips, would be very useful to make SBMAP as accessible for beginners as possible.

Full browser support would be ideal as Opera is not a very common browser. Ensuring that Chrome and Safari users have a fast interactive experience on the hexmap would make SBMAP's experience the best for much more users.

Furthermore, I would like to add more steps to the Halley's Comet Scrollama. I had plans to show the size of 1P/Halley compared to San Francisco as they are quite comparable and to highlight it's orbital period—roughly 76 years. Though it is sad to say, I did not finish these additions purely due to time constraints.

References

[1] Peter Beshai, D3 Interpolate Path, https://github.com/pbeshai/d3-interpolate-path
[2] Mike Bostock, D3 Density Contours, https://observablehq.com/@d3/density-contours
[3] Mike Bostock, D3 Hexbin, https://observablehq.com/@d3/hexbin
[4] Mike Bostock, Towards Reusable Charts, https://bost.ocks.org/mike/chart/
[5] P. Chodas NASA/JPL, Inner Solar System, https://ssd.jpl.nasa.gov/diagrams/
[6] D3, https://d3js.org
[7] Yan Holtz, Data to Viz Ridgeline Plot, https://www.data-to-viz.com/graph/ridgeline.html
[8] Jason Kao, React Scrollama, https://github.com/jsonkao/react-scrollama
[9] Eleanor Lutz (Visualization), Nicholas LePan (Author), A Map of Every Object in our Solar System, https://www.visualcapitalist.com/mapping-every-object-in-our-solar-system/
[10] Matplotlib, https://matplotlib.org/stable/index.html
[11] Multiprocessing, https://docs.python.org/3/library/multiprocessing.html
[12] Paul Murray, Animated Contours, https://observablehq.com/@plmrry/animated-contours
[13] Pandas, https://pandas.pydata.org
[14] React, https://reactjs.org
[15] Sass, https://sass-lang.com