The Genealogy of D3 Code

Lucas Throckmorton
Stanford University
353 Serra Mall, Stanford, CA 94305
lucast@stanford.edu

Cathy Zhu
Stanford University
353 Serra Mall, Stanford, CA 94305
cathyzhu@stanford.edu

ABSTRACT
In this paper we describe a data exploration tool for visualizing similarities and influences between snippets of D3 code sourced from github gists. We collected raw code snippets and metadata from the open API of http://blockbuilder.org, D3 blocks aggregator, and ran sets of code snippets through MOSS, a software similarity detector. We visualize similarity relationships between code snippets using a timeline graph; code is represented as nodes, and relationships between any two pieces of code are represented as links. We show metadata for each node and link on selection and enable zoom and concentration on a single thread of ancestry. Further work would include integrating more data via a database server, enabling more filters and queries, and calculating per-ancestry thread metadata.

Author Keywords
D3, MOSS, author's kit, conference publications.

ACM Classification Keywords
H5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

INTRODUCTION
Within the D3 community, code sharing is widely encouraged via github gists, publicly available on bl.ocks.org. Members of the D3 community have also built tools on top of gists like blocksplorer or blocksbuilder to enable searching for different visualization samples. We would like to investigate the impact of code sharing by identifying primary source examples and tracing out how these examples are modified to generate new examples. We achieve this by plotting out a timeline of code similarity amongst snippets.

RELATED WORK
There are numerous existing projects for visualizing code dependencies, as well as for visualizing open source communities, developer interaction, and the evolution of code within a single repository over time. Our project seeks to visualize the genealogy of D3 code, which involves comparing numerous code snippets from different authors over time and looking at the similarity chain. Our data contains nodes (code snippets) that directionally relate to each other and timestamp data of when they were created. Our work is informed by both research projects on code visualization such as ClonEvol[1] and Software Storyline[2], as well as techniques and infographics on timeline and graph visualizations such as the sankey, the cluster dendrogram, the Evolution of the Web infographic, and the Linux Distro Timeline.

METHODOLOGY

Data Source
We scraped several sets of D3 code and metadata from github gists and ran them through MOSS, identifying links between snippets with more than 10 lines of similar code. We ignored similarity between code snippets by the same author. We determine ancestry by timestamp, meaning that given two pieces of similar code, we say the code created at an earlier time "influenced" the code created at a later date. Our per code snippet metadata includes the gist's title, author, creation time, and primary D3 API calls (tagged via crowdsourcing). Per-link metadata generated from MOSS includes the specific lines that are similar between two snippets of code as well as the confidence of each similar paragraph.

Visualization
We created an exploration tool for our dataset with a big picture view in the form of a timeline tree and metadata cards below our main visualization that shows pertinent metadata for each node and link. Nodes are colored by author and edges are colored on an ordinal scale, depending on the number of lines of similar code were found. We also enable manipulation of the main view via zoom, drag, and selection. Double-clicking on a single node enables selection of and focus on the node's lineage--i.e. all of its ancestors and descendants.

RESULTS
We implemented a timeline tree with metadata cards below our primary visualization. A full overview of data set allows the user to identify nodes (gists) and threads

[1] ClonEvol visualizes the evolution of a single repository by the lines of code and files added, modified, or deleted over time.

[2] Software Storyline plots the behaviors (commit, pull request, commenting, etc.) and interactions of developers within the same open source codebase over time.

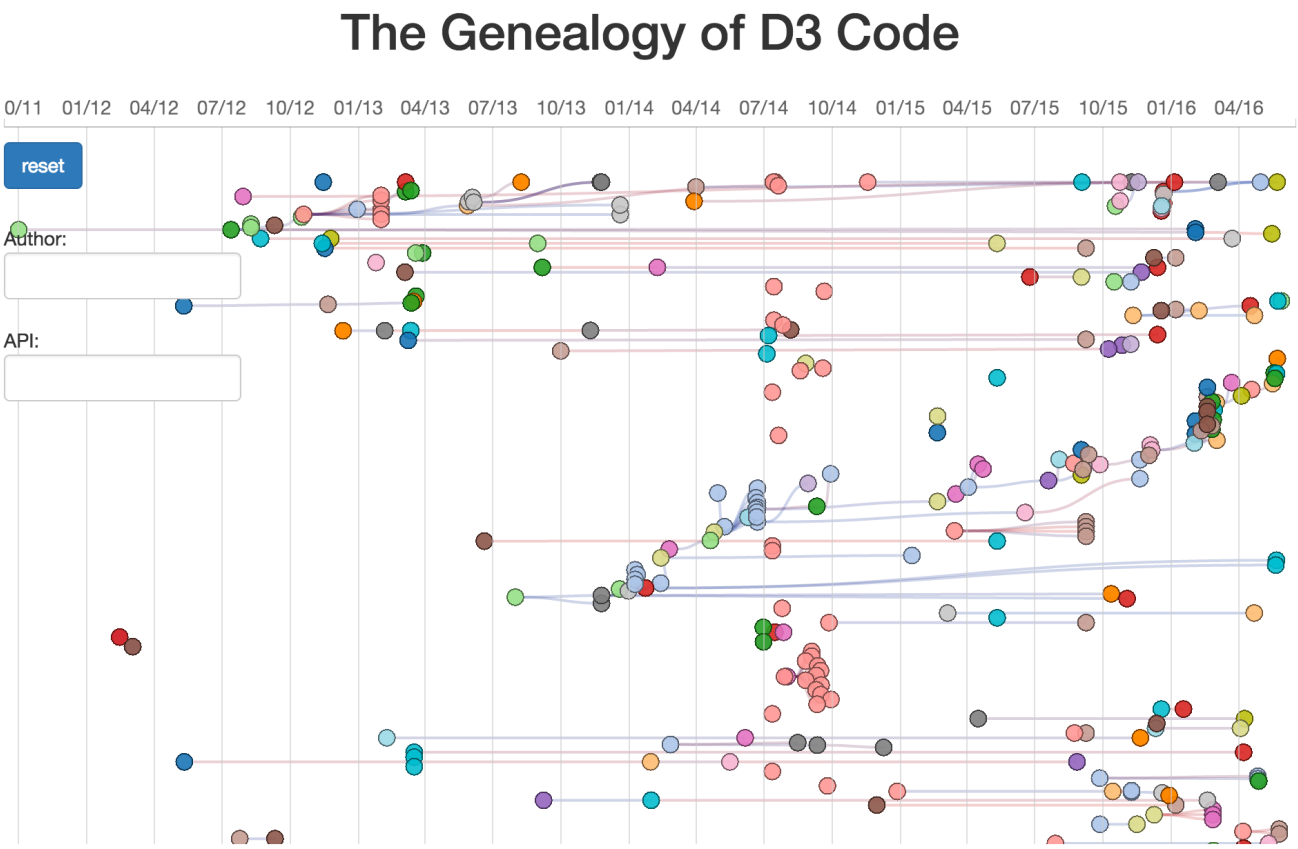

Figure 1. Overview

(similarity relationships) of interest. Nodes are colored by author, and edges are colored on a sliding scale. Similarity between any two gists is indicated by color decreases as links shift from red to blue (figure 1).

Hovering a node provides information on the gist that a user can then use to further filter the dataset.

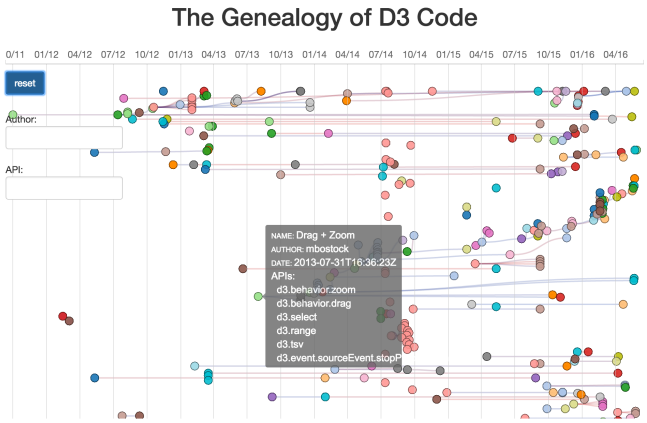

Figure 2. Tooltip on hover

By double clicking on a node, a user can zoom into a single thread of interest, ignoring all other data.

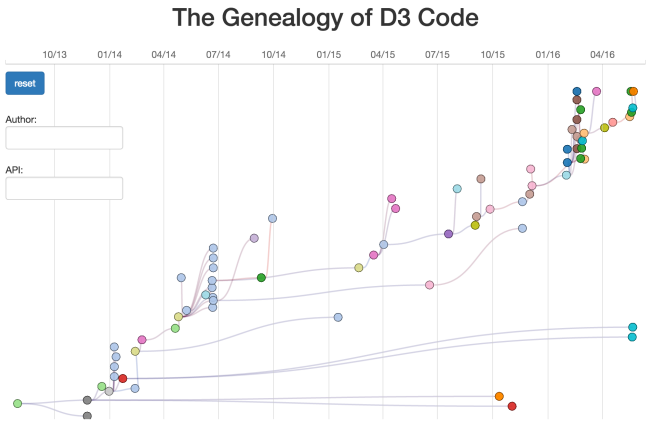

Figure 3. Selecting a single ancestry thread

Alternatively, a user can filter by author to view only threads that contain at least one gist by the author.

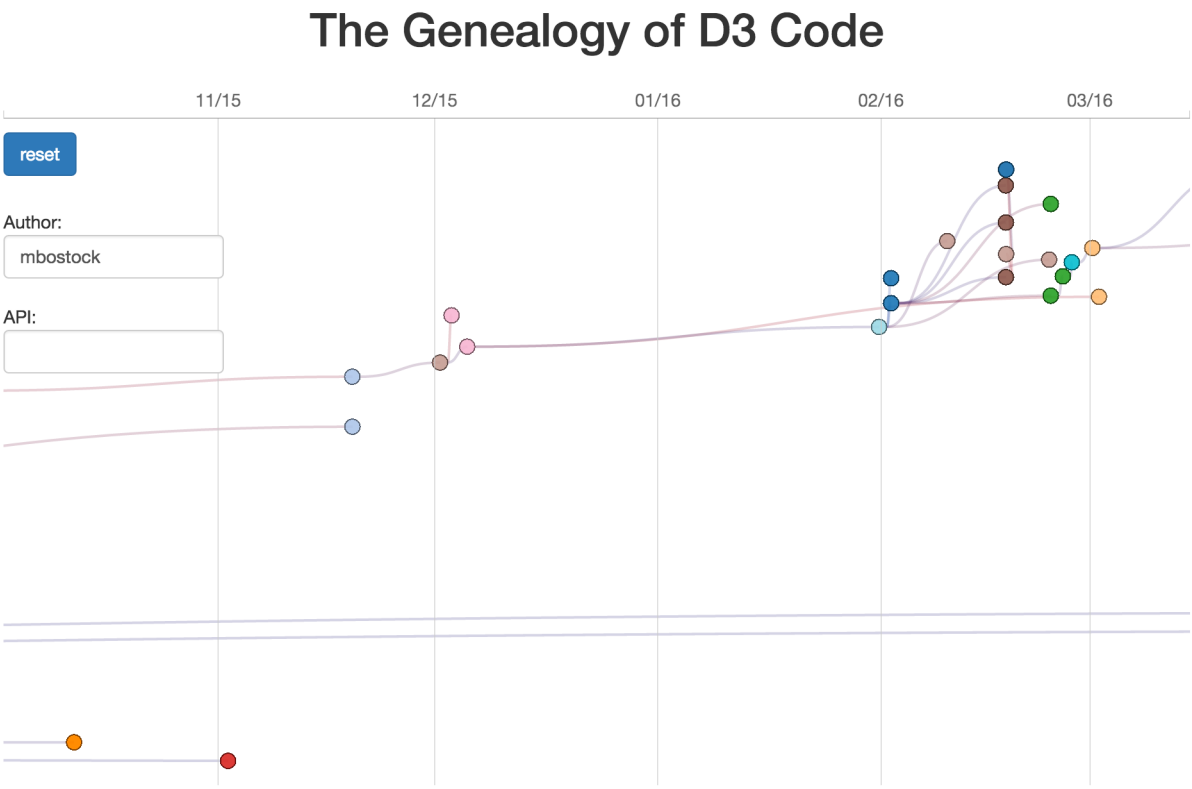

Figure 4. Querying by author and interactive zoom

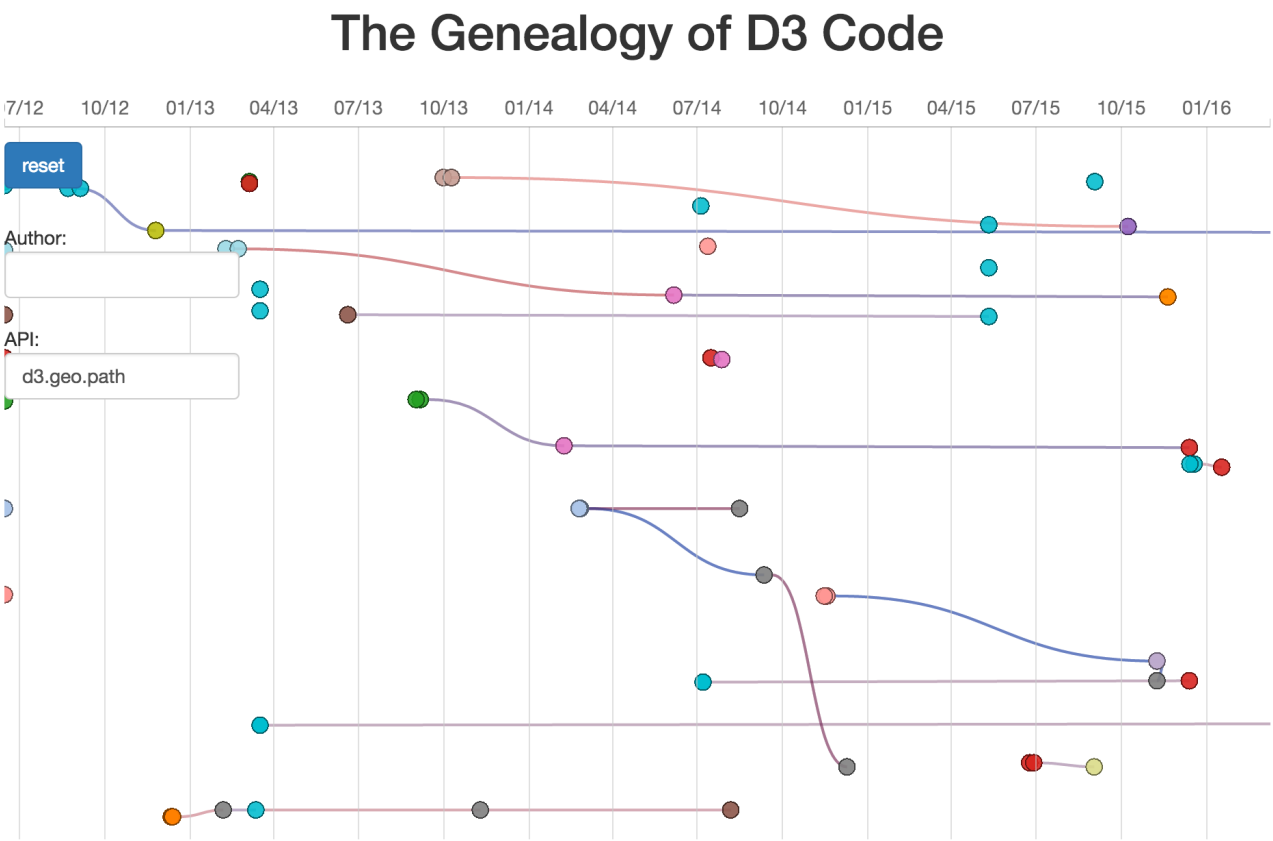

Figure 5. Filter by API call

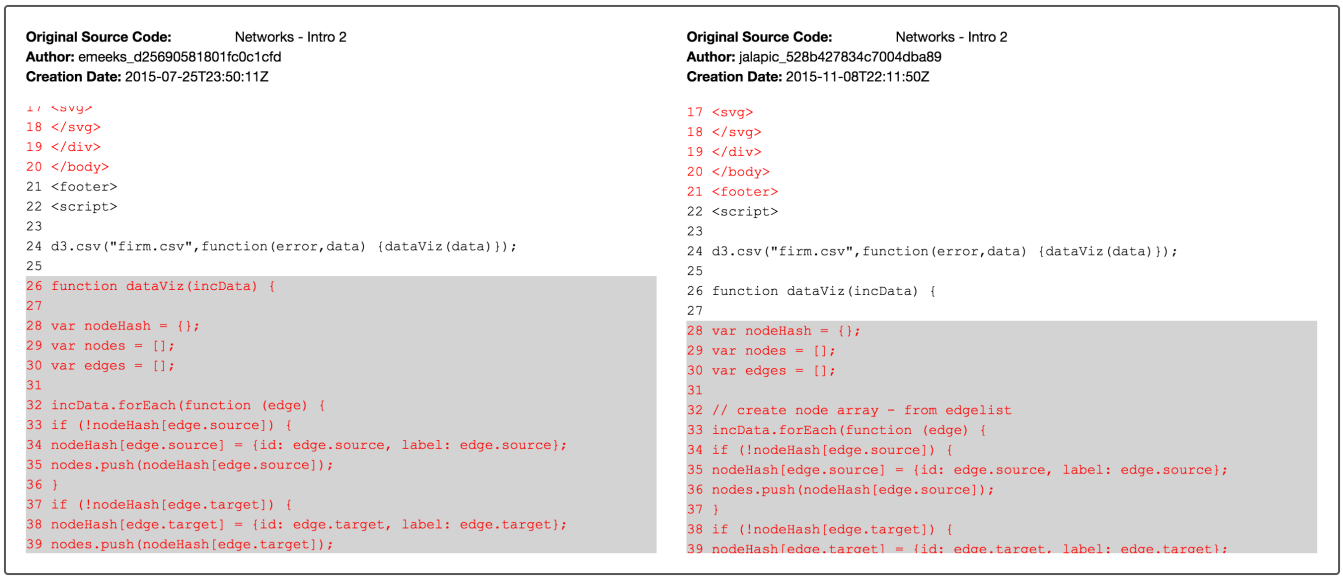

Figure 6. Metadata cards showing similar lines between two gists.

Users can scroll and zoom into areas of interest to view data at higher detail. Users can also filter by api call. Unlike filtering by author, this will only show gists that use that api call, filtering all other from the visualization.

Once a user identifies a snippet or ancestry thread of interest, they can select individual links to view all identified similarities between gists in order to trace the code's usage through time.

Discussion

Our work has introduced a new way of visualizing code that was used and changed over time by different authors and identified needs for future visualizations. The particular problem of code use and evolution amongst a community poses several unique challenges that prior related works have not addressed.

Our visualization enables viewers to learn about the following:

Identify gists of interest

Our tree visualization makes it visually easy to identify gists that have many descendants, or many ancestors.

Evaluating author impact

Since we color code by author and enable queries by author, it is easy to identify prolific authors (with many gists) and impactful authors (whose work has many descendants) within any dataset. Furthermore, after identifying an author of interest, a user can filter across the entire dataset using the author or API calls to learn more about the author's work or to find other authors who have done similar work.

Tracing the evolution of a single code snippet

By double-clicking on any node, we zoom in on the node's lineage (all ancestors and descendants). By having detailed data cards highlighting similar lines of source code, we facilitate tracing copied snippets over time.

During our poster session, potential users showed interest in the following use cases for our dataset and visualization.

1. Identifying the most commonly copied code snippets, this may represent opportunity for d3 to encapsulate the functionality into a standalone library.
2. Identifying the most prolific authors, and authors whose snippets are most authoritative (i.e. copied the most).
3. Getting a big picture overview of the history of D3 via the evolution of API calls, tracing the rising and falling productivity of authors, etc.

We can optimize for 1 and 2 by integrating features like statistical metadata and ranking. We need additional data processing functionality to extract the exact snippets of code copied, but for the most part this just requires summary data and ranking that is easily implemented with a database server. For any of our data to accurately represent the state of D3 code, we will also need larger datasets.

Given that most data sets for any API call contains at least hundreds of nodes and thousands of links, visualizing this on a single web page can be busy and difficult to read. We tried to identify redundant links to create a cleaner visualization, and found two types of relationships that cause single nodes to have multiple parents:

1. The case where multiple gists copy the same snippet of code. These links are a large percentage of all links in the visualization: For n nodes containing the same code snippet, MOSS would generate n choose 2 similarity

links, whereas n-1 links should be sufficient to visualize this relationship.

2. At times, authors may draw inspiration from multiple projects for different pieces of functionality. In this case, when a snippet has multiple parents, all of them are important pieces of information. These edges represent a relatively small amount of edges in the visualization.

In one version of our visualization, we elected to reduce the number of relationship links present in the visualization by enforcing single parentage, eliminating about half of the links from the graph. This ensured that all redundant edges were removed, though some edges from the second, non-redundant category were also removed. Single-parentage visualization provided an effective way to produce a cleaner and easy to read visualization, however further work needs to be done on the parentage filtering methodology to avoid removing relevant data.

FUTURE WORK

More Data

We currently have our data as a JSON file imported via JavaScript. D3 currently does not handle more than 800 or so links worth of data containing raw code at a suitable latency (data display, manipulation, and redrawing takes too much time, occasionally crashes). Hosting this on a server with an actual database will probably help with information hiding (we can live query for detailed information instead of front-loading JSON with raw code snippets). It will also enable us to calculate per-thread metadata, which would be of interest. More data would look something like the total of 30,000 gists available on block builder or a complete scrape of d3-related github gists.

Selection and Metadata

As mentioned above, we currently support metadata for nodes and links only. We are interested in other selections of the data that may have summary metadata. For example, we may select a single thread of ancestry and view the number of unique code snippets in the thread, the number of unique authors, average or median lines copied, etc. Or we may be interested in metadata associated with a particular author or API call.

Author Impact

Some of our potential users expressed an interest in evaluating author impact. Our author-based coloring and querying features are a good starting point for this. We can extend this to display summary data for each author (e.g. total gists, total descendants, number of unique ancestors, for their work), enable querying and ranking (e.g. query for top authors by total gists or total descendants), as well as experiment with visually effective ways to identify an author's impact (e.g. gray out other nodes to highlight an author's work on selection).

Refining Ancestry Filtering

Per discussion, we are interested in reducing redundant similarity links. We need to refine our parent filtering algorithm to take into account the relevant snippets of code within each gist.

Code Snippet Tracing

Per discussion, we are interested in implementing code snippet tracing to identify the most popular code snippets. As a baseline, this would involve more data processing to extract the similar code snippets from similar gists and computing occurrence counts and rankings. Further extension might allow users to select a particular code snippet and view its evolution throughout the timeline.

ACKNOWLEDGMENTS

We thank our professor Maneesh Agrawala, and the CS 448B teaching staff for an excellent Data Visualization class.

REFERENCES

1. Michael Ogawa and Kwan-Liu Ma. 2010. Software evolution storylines. In Proceedings of the 5th international symposium on Software visualization (SOFTVIS '10). ACM, New York, NY, USA, 35-42. DOI=http://dx.doi.org/10.1145/1879211.1879219
2. Shixia Liu, Yingcai Wu, Enxun Wei, Mengchen Liu, Yang Liu, "StoryFlow: Tracking the Evolution of Stories," IEEE Transactions on Visualization and Computer Graphics, vol. 19, no. 12, pp. 2436-2445, Dec., 2013
3. PM, Anvaka, [Github Repository] (2015), https://github.com/anvaka/pm
4. Codegraph, Facebook, [Github Repository] (2015), https://github.com/facebook/pfff/wiki/CodeGraph
5. CodeFlower, fzaninotto, [Github Repository]. (2014), https://github.com/fzaninotto/CodeFlower
6. The Linux Distribution Timeline, [Poster] (2012), http://futurist.se/gldt/
7. Evolution of the Web, [Webpage] (2012), http://www.evolutionoftheweb.com/
8. Flow Visual Tracer, [Webpage] (2016) http://findtheflow.io/
9. ClonEvol: Visualizing Software Evolution with Code Clones (A. Hanjalic, Proc. IEEE VISSOFT 2013), http://www.cs.rug.nl/~alext/PAPERS/VISSOFT13/paper.pdf
10. Mike Bostock, Cluster Dendrogram (2016), [Github Gist]. http://bl.ocks.org/mbostock/4063570
11. Mike Bostock, Collapsible Tree [Visualization]. http://mbostock.github.io/d3/talk/20111018/tree.html
12. Mike Bostock, Sankey Diagrams (2012), [Github Gist]. https://bost.ocks.org/mike/sankey/