HMS Summer Institute in Biomedical Informatics

Mark Keller *

Summer 2019

Abstract

Abstract here...

1 Introduction

1.1 Background

Single-cell genomics and transcriptomics enable cell-level measurement of quantities such as gene expression, mutations, and methylation. Single-cell data reveals cellular heterogeneity and captures the functional states of individual cells that would be hidden by measurements of bulk cell populations that only capture averages [1]. Cell types can be teased apart in single-cell data using computational approaches such as dimensionality reduction and clustering based on gene expression profiles [2, 3].

Following the model of the human genome project which focused resources across institutions to discover a consensus human genome sequence, projects such as ENCODE [4], GTEx [5], the Human Cell Atlas [6], and the 4D Nucleome [7] have been envisioned and executed. The NIH-sponsored Human BioMolecular Atlas Program (HuBMAP) is using this model to map the human body at the single cell level in a limited number of subjects. HuBMAP aims to develop spatial mappings of cells and molecules, with new coordinate frameworks that allow querying across levels, from organ to tissue to cell to molecule. Learning from the struggles of past biomedical and genomic data collection efforts that have developed data portals and visualization tools as afterthoughts, HuBMAP was conceived with the HuBMAP Integration, Visualization, and Engagement (HIVE) group responsible for articulating data access needs from the start [8].

Visualization of 2D embeddings of cells can facilitate identification of distinct cell populations that group into clusters or subclusters [9, 10]. However, such embeddings are dependent on the dimensionality reduction method employed and its parameters. Often, multiple dimensionality reduction methods are compared, including PCA, t-SNE, and UMAP [11, 12]. Technologies that quantify gene expression may also preserve spatial information, for example RNA flourescence *in situ* hybridization (FISH). In some tissues, it is likely that cells located close to each other are of the same type [2]. Spatial information can be visualized as images or as points in a spatial coordinate system, or overlapping layers of different data types aligned to the same coordinate system.

*advised by Professor Nils Gehlenborg

Vitessce (`http://vitessce.io`) is a web-based visualization tool for single-cell experiment data, including spatial data from multiple modalities and scatterplot data from arbitrary dimensionality reduction methods. Visualizations in Vitessce are interactive, with customizable zoom levels, viewer sizes, tooltips, and color encodings. To display spatial data, including cells and molecules, Vitessce leverages performant geospatial web technologies. Visualizations within Vitessce have been developed as modular components intended to make them reusable by external applications that require specific functionality provided by Vitessce but focused on different domains or goals.

1.2 Related work

The development of web-based visualization tools for spatial single cell transcriptomics is currently very active, in both open- and closed-source models. Existing open-source and academic tools include Pagoda2 [13], UCSC Cell Browser [14], the Single Cell Viewer [15], SCope [16], Giotto [17], Blue Brain Cell Atlas [18], Allen Cell Explorer [1], and cellxgene [2]. Several commercial tools are available or have been announced, including those from 10x Genomics and BioTuring. Tools for static visualization of single-cell data are also popular, with use cases including creating publication-ready figures or fitting into traditional analysis workflows.

Many of these existing tools focus on a particular aspect of single-cell data visualization, but none support the multi-view linked spatial, scatterplot, and heatmap views that Vitessce supports. Additionally, the interfaces of tools like Single Cell Viewer are not user-friendly, requiring users to scroll and change pages many times to view different aspects of their data. Vitessce focuses on user experience to allow customization and arbitrary simultaneous views of data, while also recognizing that certain use cases may be out of scope and better accommodated by individual Vitessce components rather than the app as a whole.

1.3 Contributions

My work this summer focused on implementing features to support cell selection, cell set management, and linked plot hover events, as well as fixing miscellaneous bugs and "tech debt".

When analyzing clusters of cells and molecules, users may need to make a selection of plot elements to create a set of cells or molecules that fall within a particular area of the plot. Prior to this summer, a rectangle selection tool had already been implemented in Vitessce. However, clusters are not typically rectangle-shaped and spatial data of interest can be of any shape. To address this, one of my first tasks was to develop a polygon selection tool, allowing a user to select points to use as polygon vertices to create a boundary around a potentially irregularly shaped set of cells or other items. In developing this tool, I used `turf` [3] and `nebula.gl` [4], JavaScript libraries containing utilities for working with geospatial data. To integrate these with the Vitessce spatial and scatterplot components, heavy customization of the selection code provided by `nebula.gl` was required to ensure that the interface was consistent with the rectangular selection tool. Additionally, using `nebula.gl` forced us to upgrade the version of one of its peer dependencies, `deck.gl` [5], which illuminated bugs that we reported and were quickly fixed by the `deck.gl` maintainers. This

[1] `https://allencell.org`
[2] `https://chanzuckerberg.github.io/cellxgene/`
[3] `http://turfjs.org`
[4] `http://nebula.gl/`
[5] `http://deck.gl`

was my first time reporting bugs in open source software and interacting with external maintainers to help with the bug fix process.

Once cell sets have been selected, either via one of the selection tools or as a representation of members of clusters resulting from a clustering algorithm, it is useful to be able to organize and update them. The second main focus of my summer was building a cell set management component to enable users to organize cell sets. While this component motivated by cell sets, organization of sets is important for other types of items, so the component was developed at an abstract level to support other set types, for instance genes or molecules.

Prior to writing any code for the set management component, I first created UI mockups in Sketch, a UI design and prototyping application, in order to get feedback regarding the features and layout that the component would support. Through this process, we aimed to get clarification about the desired interactions or workflow for managing cell sets, which was successful prompted me to develop the component based on the agreed-upon mockup.

The first iteration of the cell set component allowed users to store sets resulting from interactions with the rectangle and polygon selection tools. This component supported the operations of naming and renaming sets, viewing set size, and deleting sets. Additionally, I implemented a feature to store saved sets to the user's web browser to allow them to return to a previous state. To enable sharing and long-term saving, I implemented features to import and export sets from a JSON file. This required developing a simple JSON schema to which the imported and exported data would conform.

While the first iteration of the cell set manager component was useful, through our evaluation of its functionality, we realized that it would be necessary to allow hierarchical organization of sets. A hierarchy can support organization of sets and subsets, particularly those representing clusters and subclusters at multiple levels of nesting. This prompted a second iteration of the component, mostly rewriting the code but learning from the flat first implementation. To display the tree of sets, I explored multiple open source JavaScript UI libraries containing tree layout components: `ant-design` [6], `blueprint` [7], `atlaskit tree` [8]. I ultimately chose to use `ant-design` because it was the only tree component supporting checking and dragging tree nodes. As a bonus, the library comes with additional UI components including tabs and popups, and is widely used so will likely continue to be supported for an extended period of time.

In developing the hierarchical cell set manager, I first developed tree and tree node utilities in JavaScript to serve as the internal representation of the hierarchy of sets. Then, I added UI features that were present in the flat cell set manager, including renaming and deleting sets. Next, I connected the dragging functionality provided by the tree component to the internal tree representation. I implemented updated import and export functionalities supporting the hierarchical representation of sets, requiring me to modify the flat import/export JSON schema. To allow coloring of sets, I added a color picker to each tree node, triggering an event which updates the scatterplot and spatial plots upon color change.

Cell sets can be viewed using a "view" popup menu and choosing the current set or a particular level of its descendants. This menu is dynamic to accommodate tree nodes at different levels. These view options allow multiple cell sets at the same level of the hierarchy to be viewed simultaneously, distinguished by their differing colors. For example, if the clusters discovered by a particular clustering algorithm are stored under the same parent node, the visualization of all clusters can

[6] `https://ant.design/components/tree/`

[7] `https://blueprintjs.com/docs/#core/components/tree`

[8] `https://atlaskit.atlassian.com/packages/core/tree`

be done by selecting the parent node and choosing to view the 1st level descendants.

Within the cell set manager, I also implemented a tab interface to ease the management of deeply nested tree nodes. From the view menu of a tree node, there is an option to open the node as a new tab, in which the children of the node of interest are displayed as the top level of the hierarchy, simplifying the interface. These tabs act like web browser tabs that a user would expect, and can be viewed and closed in the same way.

It is useful to be able to derive new sets from the existing sets that may have been the result of a selection interaction or a clustering algorithm. Another part of the cell set manager component that I implemented this summer is the set operations interface. This interface supports taking unions, intersections, and complements that become new sets. One example of the usage of these operations in the context of cell sets would be to manually create a new set of all neurons from multiple sets representing different neuron types. These operations can be performed on multiple operands by using the checkboxes corresponding to each tree node.

I added support for linked cursor hover events across plots to be able to show contextual information when a user hovers on a particular cell. On plots peripheral to the currently-hovered plot, emphasis will be placed on the cell over which the cursor rests in the hovered plot. This allows a user to quickly identify the location of a particular cell across the different visualizations: scatterplots, spatial views, and gene expression heatmaps.

In addition to highlighting the same cell across plots, hover events allow us to show tooltips containing more information about the cell of interest. A tooltip is a message positioned near the mouse in a graphical user interface. In the context of Vitessce, tooltips may contain a cell's ID or list the names of the clusters and subclusters to which it belongs. To ensure that tooltips do not overflow a particular plot component's boundaries, I implemented collision detection to be able to flip the position of a tooltip relative to the cursor when approaching the edge of a plot.

My implementation of tooltips and hover emphasis markings was done by rendering contextual elements in the DOM rather than in a visualization layer directly, both performance reasons and to keep the visualization clean to in preparation for potential downloading. To synchronize HTML element positions with positions of data points in visualization layers, we perform projections from the data coordinate space to the browser coordinate space. This approach also follows the decoupling paradigms followed throughout Vitessce, making the extra elements modular and composable.

Along with the development of individual features for Vitessce, I also helped to reduce tech debt and streamline some aspects of the code base. This included adhering to the recent additions to the React library documentation regarding the implementation of new components as JavaScript functions rather than JavaScript classes. However, because pure JavaScript functions do not allow for the storage of component state variables (as classes do), to "hook into" React's state functionality, usage of additional functions called "hooks" is necessary. React hooks can make code more readable by allowing code blocks corresponding to a particular task to be grouped together, while class-based components require separation of code based on component lifecycle events. I migrated our code for styling Vitessce from CSS to Sass [9], allowing us to leverage CSS preprocessing for nesting styles, computed variables, and style reuse via mixins. In the development of the cell set manager, I wrote unit tests for each of its functions, as well as tests for the import/export schema validation. My final task for the summer was exploring a data serialization tool called Protobuf [10] developed by Google for data input and output in a language-agnostic way. Improved

[9] `http://sass-lang.com`

[10] `http://developers.google.com/protocol-buffers`

serialization and compression could improve loading times of large datasets in Vitessce. The first part of this exploration process entailed creating a small pipeline to make a full trip: from a Protobuf schema to a Python data structure to a serialized data structure sent the browser to be decoded into a JSON data structure.

These updates are invisible to Vitessce users but will help future developers to easily understand, add, and update functionality in Vitessce.

2 Discussion

It has been a fun experience to spend the summer immersed in a lab focused more on methods and infrastructure development than hypothesis-based research. While I enjoy traditional software development, visualization research software and infrastructure development is particularly exciting to me, as it moves fast and serves as a critical part of the translational, educational, and exploratory functions of research. The field of life sciences research software development is currently struggling with the problems of reproducibility and long-term maintenance, issues important to the adoption of such software [19, 20]. Software development practices in academia seem to lag far behind those in industry, possibly due to differences in incentives. For instance, the concept of "continuous integration" has existed since 1991, yet articles as recently as 2017 were still treating it as a foreign concept to life sciences researchers [21, 22]. Continuous integration and testing require more time and effort up front but prevent software from regressing and decrease the cognitive overhead for maintenance later. It is promising that projects like Vitessce, HiGlass [23], and others in the lab are committed to these best practices.

Vitessce adopts both general UI technologies and specific geospatial technologies for the single-cell visualization domain. While end users typically do not need to think about the particular technology used to implement a visualization tool, it is something I thought about this summer. Web technologies currently provide a good balance of accessibility, performance, and development experience for visualization, and will continue to do so in the near future. Web browsers support time-tested visualization technologies such as SVG, WebGL, and the canvas element, while improving the development experience with recent standards like WebAssembly and modern JavaScript language features. The community around open-source JavaScript libraries is large and very active. Vitessce is implemented using the React JavaScript framework originally developed for Facebook but now distributed as open source [11]. React is a framework that simplifies the development of modular, component-based web applications.

Vitessce performs spatial visualization by leveraging libraries developed with a focus on cartography and mapping. The open source Uber-maintained Deck.gl JavaScript library is powerful because it implements reactive updates that are similar to those used by React. Deck.gl performs diffing on the data passed to each view layer to detect updates and invalidate the current state, just as React performs diffing on the virtual DOM in response to data changes to determine when to update the browser's DOM.

The development of Vitessce has focused on separating logic into components that can operate independently to enable them to be imported and re-used by other projects, including HuBMAP data portals and tissue viewers. To achieve this goal, Vitessce does not maintain a global state that is used by child components. Instead, there is an event-based mechanism, with wrapper subscriber components that pass state down to children along with publisher update functions. This allows external usage of components to be done by setting up custom publishing and subscription

[11] `https://reactjs.org`

wrappers rather than using the Vitessce-specific wrappers. A drawback of this approach is that events are asynchronous, so implementing a history mechanism may prove to be difficult, as there is no one source of truth for events.

To help others to adopt open source software, packaging and distribution are important to think about. In the case of the Vitessce components, we transpile the JavaScript code and distribute it on NPM. The CSS code for styling the components must also be shared along with the JavaScript code. Because of my work transitioning to a more modular styling system using SCSS files, we can distribute modular stylesheets for users that are using only a specific component or a small set of components.

3 Conclusions

Conclusions here...

3.1 Reflection

Acknowledgements

References

[1] Ehud Shapiro, Tamir Biezuner, and Sten Linnarsson. Single-cell sequencing-based technologies will revolutionize whole-organism science. *Nature Reviews Genetics*, 14(9):618, 2013.

[2] Oliver Stegle, Sarah A Teichmann, and John C Marioni. Computational and analytical challenges in single-cell transcriptomics. *Nature Reviews Genetics*, 16(3):133, 2015.

[3] Darren J Burgess. Spatial transcriptomics coming of age. *Nature Reviews Genetics*, 20(6):317, 2019.

[4] ENCODE Project Consortium et al. The ENCODE (ENCyclopedia of DNA elements) project. *Science*, 306(5696):636–640, 2004.

[5] John Lonsdale, Jeffrey Thomas, Mike Salvatore, Rebecca Phillips, Edmund Lo, Saboor Shad, Richard Hasz, Gary Walters, Fernando Garcia, Nancy Young, et al. The genotype-tissue expression (GTEx) project. *Nature genetics*, 45(6):580, 2013.

[6] Aviv Regev, Sarah A Teichmann, Eric S Lander, Ido Amit, Christophe Benoist, Ewan Birney, Bernd Bodenmiller, Peter Campbell, Piero Carninci, Menna Clatworthy, et al. Science Forum: The Human Cell Atlas. *Elife*, 6:e27041, 2017.

[7] Job Dekker, Andrew S Belmont, Mitchell Guttman, Victor O Leshyk, John T Lis, Stavros Lomvardas, Leonid A Mirny, Clodagh C O'shea, Peter J Park, Bing Ren, et al. The 4D nucleome project. *Nature*, 549(7671):219, 2017.

[8] Michael P Snyder, Shin Lin, Amanda Posgai, Mark Atkinson, Aviv Regev, Jennifer Rood, Orit Rosen, Leslie Gaffney, Anna Hupalowska, Rahul Satija, et al. Mapping the Human Body at Cellular Resolution–The NIH Common Fund Human BioMolecular Atlas Program. *arXiv preprint arXiv:1903.07231*, 2019.

[9] Bo Wang, Junjie Zhu, Emma Pierson, Daniele Ramazzotti, and Serafim Batzoglou. Visualization and analysis of single-cell RNA-seq data by kernel-based similarity learning. *Nature methods*, 14(4):414, 2017.

[10] Patrik L Ståhl, Fredrik Salmén, Sanja Vickovic, Anna Lundmark, José Fernández Navarro, Jens Magnusson, Stefania Giacomello, Michaela Asp, Jakub O Westholm, Mikael Huss, et al. Visualization and analysis of gene expression in tissue sections by spatial transcriptomics. *Science*, 353(6294):78–82, 2016.

[11] Svetlana Ovchinnikova and Simon Anders. Exploring dimension-reduced embeddings with Sleepwalk. *bioRxiv*, page 603589, 2019.

[12] Etienne Becht, Leland McInnes, John Healy, Charles-Antoine Dutertre, Immanuel WH Kwok, Lai Guan Ng, Florent Ginhoux, and Evan W Newell. Dimensionality reduction for visualizing single-cell data using UMAP. *Nature biotechnology*, 37(1):38, 2019.

[13] Blue B Lake, Song Chen, Brandon C Sos, Jean Fan, Gwendolyn E Kaeser, Yun C Yung, Thu E Duong, Derek Gao, Jerold Chun, Peter V Kharchenko, et al. Integrative single-cell analysis of transcriptional and epigenetic states in the human adult brain. *Nature biotechnology*, 36(1):70, 2018.

[14] Tomasz J Nowakowski, Aparna Bhaduri, Alex A Pollen, Beatriz Alvarado, Mohammed A Mostajo-Radji, Elizabeth Di Lullo, Maximilian Haeussler, Carmen Sandoval-Espinosa, Siyuan John Liu, Dmitry Velmeshev, et al. Spatiotemporal gene expression trajectories reveal developmental hierarchies of the human cortex. *Science*, 358(6368):1318–1323, 2017.

[15] Shuoguo Wang, Constance Brett, Mohan Bolisetty, Ryan Golhar, Isaac Neuhaus, and Kandasamy Ravi. Single Cell Viewer (SCV): An interactive visualization data portal for single cell RNA sequence data. *BioRxiv*, page 664789, 2019.

[16] Jasper Wouters, Zeynep Kalender Atak, Liesbeth Minnoye, Katina Spanier, Maxime De Waegeneer, Carmen Bravo Gonzalez-Blas, David Mauduit, Kristofer Davie, Gert Hulselmans, Ahmad Najem, et al. Single-cell gene regulatory network analysis reveals new melanoma cell states and transition trajectories during phenotype switching. *bioRxiv*, page 715995, 2019.

[17] Ruben Dries, Qian Zhu, Chee-Huat Linus Eng, Arpan Sarkar, Feng Bao, Rani E George, Nico Pierson, Long Cai, and Guo-Cheng Yuan. Giotto, a pipeline for integrative analysis and visualization of single-cell spatial transcriptomic data. 2019.

[18] Csaba Erö, Marc-Oliver Gewaltig, Daniel Keller, and Henry Markram. A Cell Atlas for the Mouse Brain. *Frontiers in Neuroinformatics*, 13:7, 2019.

[19] Björn Grüning, John Chilton, Johannes Köster, Ryan Dale, Nicola Soranzo, Marius van den Beek, Jeremy Goecks, Rolf Backofen, Anton Nekrutenko, and James Taylor. Practical computational reproducibility in the life sciences. *Cell systems*, 6(6):631–635, 2018.

[20] Peter Belmann, Johannes Dröge, Andreas Bremges, Alice C McHardy, Alexander Sczyrba, and Michael D Barton. Bioboxes: standardised containers for interchangeable bioinformatics software. *Gigascience*, 4(1):47, 2015.

[21] Grady Booch. *Object Oriented Design with Applications*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1991.

[22] Brett K Beaulieu-Jones and Casey S Greene. Reproducibility of computational workflows is automated using continuous analysis. *Nature Biotechnology*, 35(4):342, 2017.

[23] Peter Kerpedjiev, Nezar Abdennur, Fritz Lekschas, Chuck McCallum, Kasper Dinkla, Hendrik Strobelt, Jacob M Luber, Scott B Ouellette, Alaleh Azhir, Nikhil Kumar, et al. HiGlass: web-based visual exploration and analysis of genome interaction maps. *Genome biology*, 19(1):125, 2018.