Brown University

CSCI 2980: Reading and Research

2019 Fall, 2020 Spring

Assisting with Scalable Scalable Vector Graphics and VisConnect

Shash Sinha

This report is submitted in partial fulfillment of the requirements for the degree of Master of Computer Science (ScM) at Brown University

Abstract

The aim of this report is to highlight some of my contributions in and around two visualization research projects; I was involved in during my last academic term at Brown University. For the sake of brevity, I have only included significant milestones due to the exploratory nature of the work I conducted. Similarly, I have chosen to provide minimal background context and instead assume readers have looked at the two research papers[1,2] produced from the projects before reading this report.

[1]Scalable Vector Graphics: Fast Drop-In Rendering for D3.js – `https://osf.io/mea2v/`

[2]VisConnect: Distributed Event Synchronization for Collaborative Visualization – `https://osf.io/hkaxc/`

Acknowledgements

I want to express my sincere gratitude to my supervisor, Prof. James Tompkin[1], for introducing me to the visualization field and being a great supervisor in general. I would also like to thank Michail Schwab[2] for enabling me to assist on the insanely cool projects he created; I strive to be as competent at software engineering as he is one day. Lastly, I would like to thank all the other collaborators who worked on either project—I look forward to keeping in touch.

[1]James Tompkin's Department Homepage – `https://cs.brown.edu/people/faculty/jtompki1/`

[2]Michail Schwab's Department Homepage – `https://khoury.northeastern.edu/people/michail-schwab/`

Contents

List of Figures

List of Tables

1. Introduction

The following report has been split up into three sections: Attaining familiarity with SVGs, Assisting on SSVG Project, and Assisting on VisConnect Project. The first two cover work carried out in the first and second halves of Fall 2019, respectively, and the last section covers work contributed during Spring 2020.

Due to the large number of abbreviations used in this document, Table 1.1 is included below for the readers reference.

N.B. If the reader is familiar with the research papers as recommended in the abstract i.e., *Scalable Vector Graphics: Fast Drop-In Rendering for D3.js* [10] and *VisConnect: Distributed Event Synchronization for Collaborative Visualization* [11] the below table should contain minimal abbreviations whose definitions they are not aware of.

Abbreviation	**Definition**
SVG	Scalable Vector Graphics [6]
SSVG	Scalable Scalable Vector Graphics [10]
LWA	Live Website Annotate [See Section 4]
DZI	Deep Zoom Image [8]
PNG	Portable Network Graphics [5]
D3	Data Driven Documents [1]
DOM	Document Object Model [3]
HTML	Hypertext Markup Language [4]
OpenGL	Open Graphics Library [13]
FPS	Frames Per Second
RGBA	Red Green Blue Alpha

Table 1.1: Definitions of abbreviations used in document

2. Attaining familiarity with SVGs

Before I could assist on the SSVG project [10], I needed to become familar with what SVGs were and how they worked. For this I explored a previous project that was created by the same collaborators – EasyPZ.js: Interaction Binding for Pan and Zoom Visualizations [12].

In essence the tool introduced in this project allows users to make any SVG visualization interactive via pan and zoom, for mobile and desktop by adding just a single line of code to the visualization[1]:

```
<script src="https://code.easypz.io/easypz.latest.min.js"></script>
```

Alternatively users can activate advanced pan and zoom techniques on their SVG visualizations with a simple click on the EazyPZ Bookmarklet [12] shown in Figure 2.1.

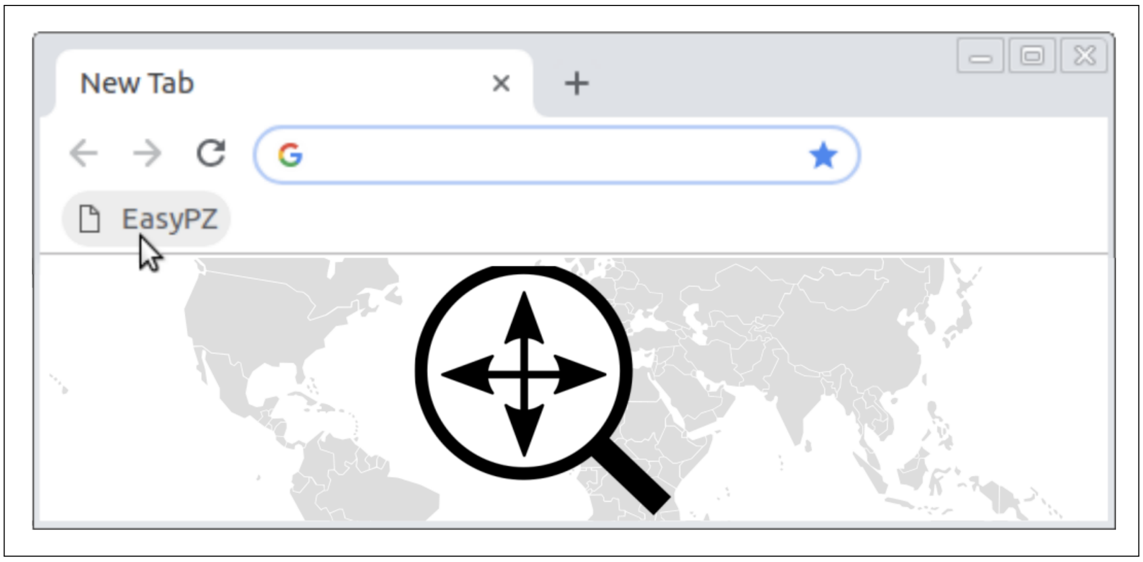


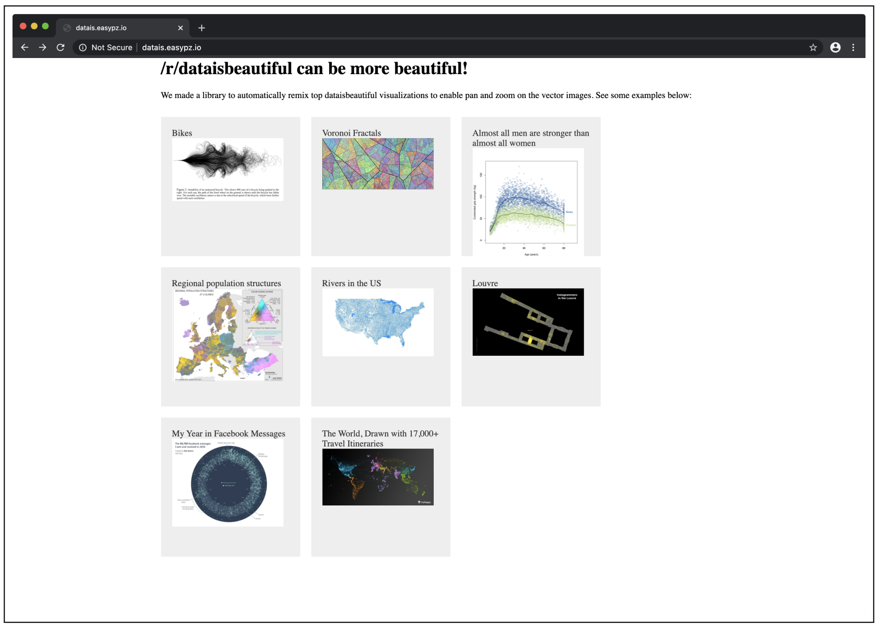
Figure 2.1: The EasyPZ bookmarklet.

2.1 Existing /r/dataisbeautiful examples site

As part of the EasyPZ project a webpage (`http://datais.easypz.io/`) had been created that showcased the tool on a eight popular r/dataisbeautiful[2] Figure 2.2 and Figure 2.3 show how this webpage looks.

[1]EasyPZ.js Pan & Zoom – `https://easypz.io/`

[2]r/dataisbeautiful: Data visualizations subreddit on Reddit – `https://reddit.com/r/dataisbeautiful/`



/r/dataisbeautiful can be more beautiful!

We made a library to automatically remix top dataisbeautiful visualizations to enable pan and zoom on the vector images. See some examples below:

Bikes

Voronoi Fractals

Almost all men are stronger than almost all women

Regional population structures

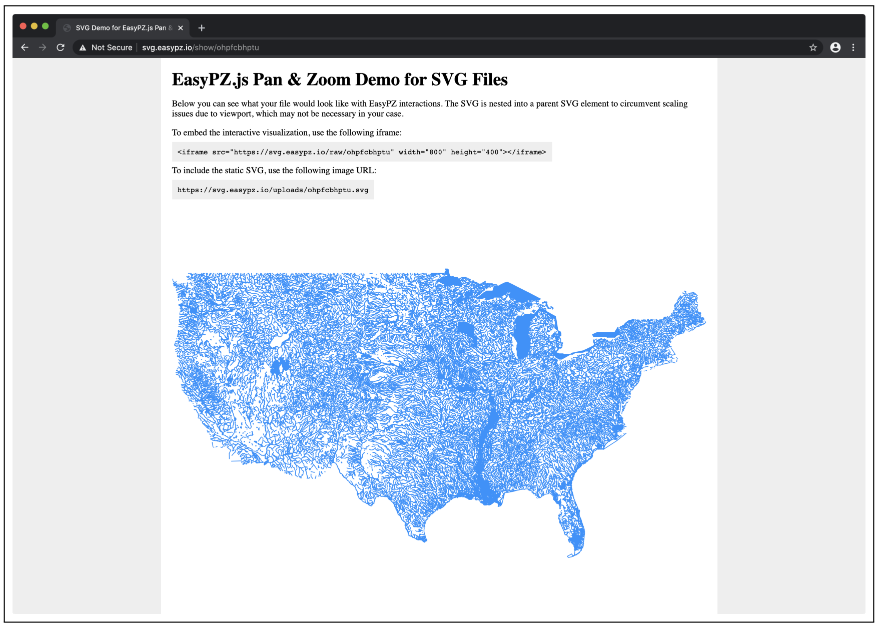
Rivers in the US

Louvre

My Year in Facebook Messages

The World, Drawn with 17,000+ Travel Itineraries

Figure 2.2: The existing EasyPZ r/dataisbeautiful visualizations homepage



EasyPZ.js Pan & Zoom Demo for SVG Files

Below you can see what your file would look like with EasyPZ interactions. The SVG is nested into a parent SVG element to circumvent scaling issues due to viewport, which may not be necessary in your case.

To embed the interactive visualization, use the following iframe:

```
<iframe src="https://svg.easypz.io/raw/ohpfcbhptu" width="800" height="400"></iframe>
```

To include the static SVG, use the following image URL:

```
https://svg.easypz.io/uploads/ohpfcbhptu.svg
```

Figure 2.3: The rivers in US example on the existing EasyPZ website

2.2 Remade /r/dataisbeautiful examples site using Openseadragon

Rather than simply cleaning up the existing website, I instead explored if using OpenSeadragon – An open-source, web-based viewer for high-resolution zoomable images, implemented in pure JavaScript, for desktop and mobile[1], was a viable alternative.

OpenSeadragon supports various zoomable image formats however I used the most popular one – Deep Zoom Image. DZI is a format created by Microsoft back in 2008 that allows images to be represented by a image tiled pyramid (See Figure 2.4 below for a visual example). This format allows the Deep Zoom rendering engine (OpenSeadragon in this case) to grab only the portion of data that is necessary for a particular view of an image [8].

The hypothesis was that if OpenSeadragon worked reasonably well, we could create a service to convert raw SVG visualizations into DZI. The service would also handle hosting all of the output image tiles such that visualization creators could simply replace the raw SVG image references on their websites for a OpenSeadragon HTML snippet that would be generated for them. The benefit of doing this is that these zoom enabled images would have a zoom user interface overlay from OpenSeadragon, which did not exist with EazyPZ.

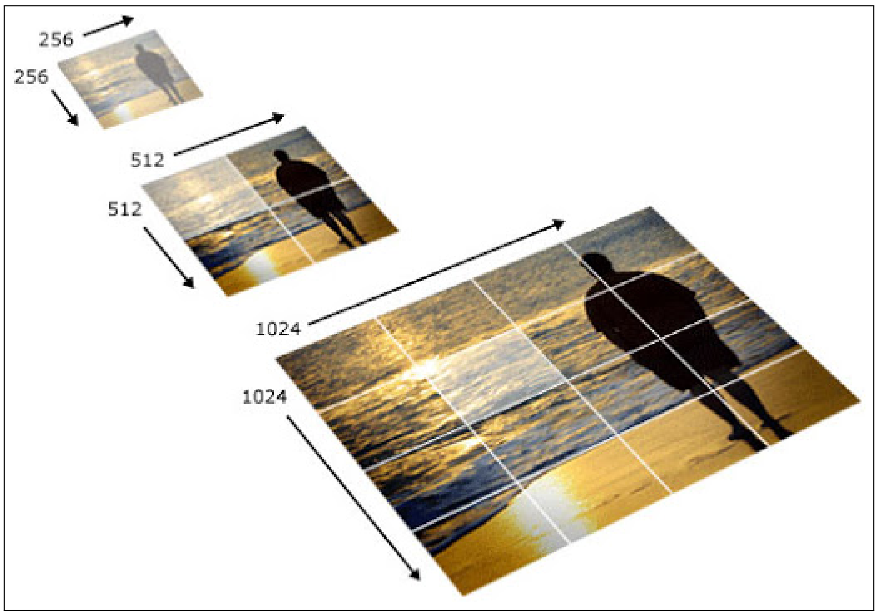


Figure 2.4: Example image tiled pyramid

[1]OpenSeadragon – `https://openseadragon.github.io/`

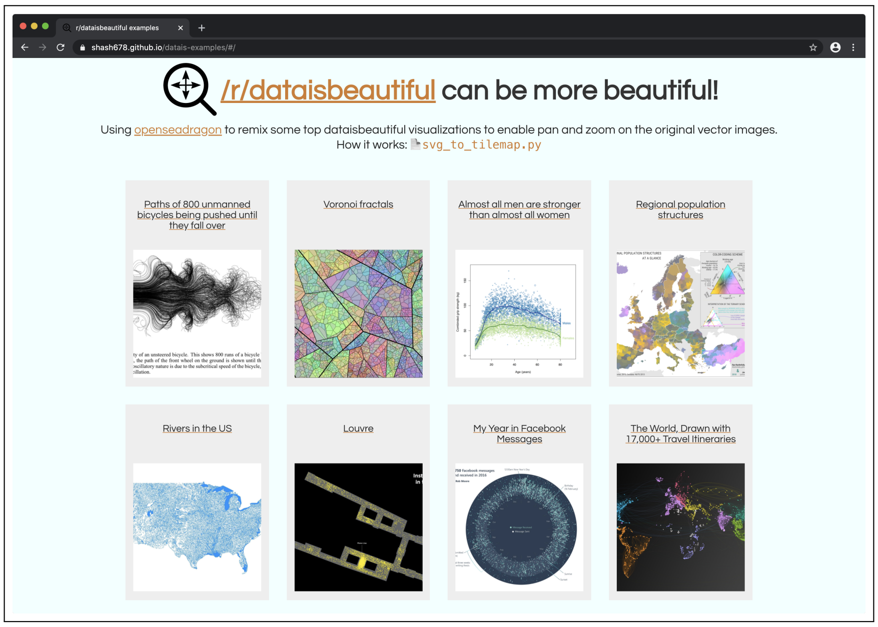
Table 2.1 shows the approach and tools that ended up being used to convert the SVG visualizations into the DZI format compatible with OpenSeadragon.

Pseudocode 2.1 *SVG to OpenSeadragon DZI*

```
For every SVG to be converted call a Python[1] script that:
  1. Coverts the input SVG into a upscaled PNG (10x by default) using cairosvg[2] or
     svgexport[3] if cairosvg fails.
  2. Uses magikslicer.sh[4] to convert the upscaled PNG into DZI format.
  3. Moves all DZI associated files into output folder.
  4. Creates an HTML file which has set up OpenSeadragon correctly and is pointing to
     the expected full web address (URL) of the folder via Python string interpolation.
```

Table 2.1: SVG to OpenSeadragon DZI approach

Figure 2.5 and Figure 2.6 show how the OpenSeadragon based webpage (`https://shash678.github.io/datais-examples/`) laid out using React[5] looks like.



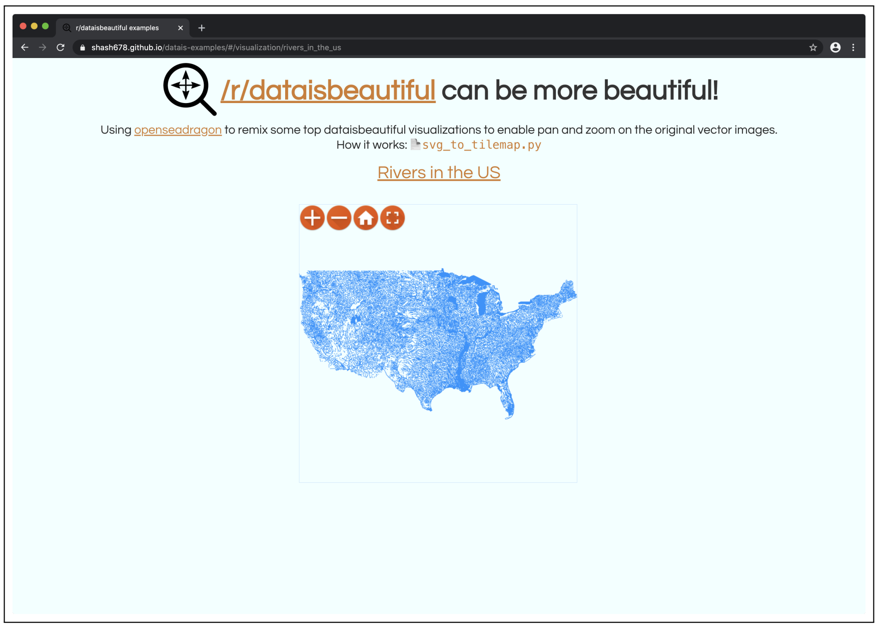
/r/dataisbeautiful can be more beautiful!

Using openseadragon to remix some top dataisbeautiful visualizations to enable pan and zoom on the original vector images.
How it works: svg_to_tilemap.py

r/dataisbeautiful examples
shash678.github.io/datais-examples/#/
Paths of 800 unmanned bicycles being pushed until they fall over
Voronoi fractals
Almost all men are stronger than almost all women
Regional population structures
Rivers in the US
Louvre
My Year in Facebook Messages
The World, Drawn with 17,000+ Travel Itineraries

Figure 2.5: The OpenSeadragon based /r/dataisbeautiful visualizations homepage

[1]Python: Popular scripting programming language – `https://python.org/`
[2]CairoSVG: Uses the Cairo 2D graphics library to efficiently convert SVGs other formats – `https://cairosvg.org/`
[3]Svgexport: Node.js module that uses Puppeteer for converting SVGs – `https://npmjs.com/package/svgexport/`
[4]MagickSlicer: Shell script that generates DZI tiles – `https://github.com/VoidVolker/MagickSlicer/`
[5]React: JavaScript library for building user interfaces – `https://reactjs.org/`


r/dataisbeautiful examples
shash678.github.io/datais-examples/#/visualization/rivers_in_the_us

/r/dataisbeautiful can be more beautiful!

Using openseadragon to remix some top dataisbeautiful visualizations to enable pan and zoom on the original vector images.
How it works: svg_to_tilemap.py

Rivers in the US

Figure 2.6: The rivers in US example on the OpenSeadragon remade website

2.3 Possible future work and current limitations

Unfortunately since I needed to start working on the SSVG project there are multiple issues with the current approach that still need to be addressed:

- Currently, during SVG conversion, the scale factor needs to be manually set when upscaling. Consequently, without a sufficiently large scale factor, the resulting max possible zoom is not enough to make out the smallest details required to interpret some visualizations, e.g., those with tiny text.
- Unfortunately, both `cairosvg` and `svgexport` have difficulty upscaling complex SVGs (at least on the 2018 13 Inch MacBook Pro I was testing on), e.g., upscaling the Voronoi Fractals visualization crashed with any scale factor above 2, which meant the resulting max possible zoom is minuscule.
- In general, it was challenging to come up with a heuristic to dynamically determine from an arbitrary SVG's DOM tree what scale factor is appropriate.
- An approach I was starting to explore that seemed promising was utilizing the minimum *font-size* attribute in the SVG's DOM tree if it existed and if not analyzing the smallest bezier curves and b-splines in the SVG's path elements.
- Another limitation to consider in regards to providing a service eventually is the size of the output DZI folder for a sufficiently zoomable complex visualization currently is quite large.

3. Assisting with SSVG Project

At the point which I joined the SSVG project [10], the core development was essentially complete. My role instead involved establishing that SSVG accomplished one of the primary goals it was built to achieve:

> *To address SVG's main shortcoming, we wish to make the technology scalable to more elements so that more data can be displayed. For a smooth user experience, the rendering performance should be at least 20–30 FPS.* [10]

The way I accomplished this was by first adding support for SSVG to use an alternate WebGL[1] renderer called Stardust[2] to act as an additional yardstick. Following that, I then ran a collection of benchmarks that measured the performance of different approaches in rendering common types of SVG elements.

3.1 Adding support for Stardust.js

Stardust[9] uses the term "mark" for the graphical elements, such as circles, lines etc. So for the three marks: lines, circles and rectangles I had to essentially modify SSVG's renderer to draw the appropriate mark call using the data available in SSVG's virtual DOM at that time.

3.1.1 Limitations and possible future work

Stardust, unfortunately, does not support closed splines[3] so even a basic polygon like a filled-in triangle is not possible natively.

Despite managing to implement a working solution for supporting non-convex polygons from valid path data (using multiple wedge marks for its sub triangles), the performance was not good enough to have polygons in the benchmarks. Future work could involve modifying the Stardust library directly and adding an optimized custom polygon mark natively.

Additionally, Stardust marks only support setting the fill color using RGBA in a somewhat obscure format of an array of four floats, i.e., `[[0,1], [0,1], [0,1]]`. Unfortunately, performing the 3 division operations for every shape (RGBA colors in D3 are represented like `rgba([0,255], [0,255], [0,255], [0,1])`) every rendering "tick" was affecting performance, so I removed the ability to set the color for the benchmarks and hard coded the color of all shapes. Future work could be done to cache the colors in some way and add to SSVGs internal color translation methods to convert from D3/HTML's other color representations to a Stardust compliant format efficiently.

[1]WebGL: Cross-platform standard for a low-level graphics API based on OpenGL – `https://khronos.org/webgl/`
[2]Stardust: A library for rendering information visualizations with GPU – `https://stardustjs.github.io/`
[3]Spline: A special function defined piecewise by polynomials

3.2 Rewriting and running shape rendering benchmarks

Before I joined the project, there existed a few benchmarks for different shapes, however the code for each for the benchmark was difficult to follow, did not have a set structure. Furthermore, results for each benchmark had to be manually copied from the developer console. So I:

- Cleaned up all benchmarks into easy to read ES6 code with descriptive file names that described the tests.
- Made all benchmarks essentially identical other than what SVG element was specifically being tested by the benchmark.
- Added code to automatically save a CSV with the results.
- Created a README file for how to run the benchmarks in the same conditions I ran them in.

Figure 3.2 and Figure 3.3 show the final results showcased in the paper while Figure 3.1 shows what an example benchmark looks like when running.

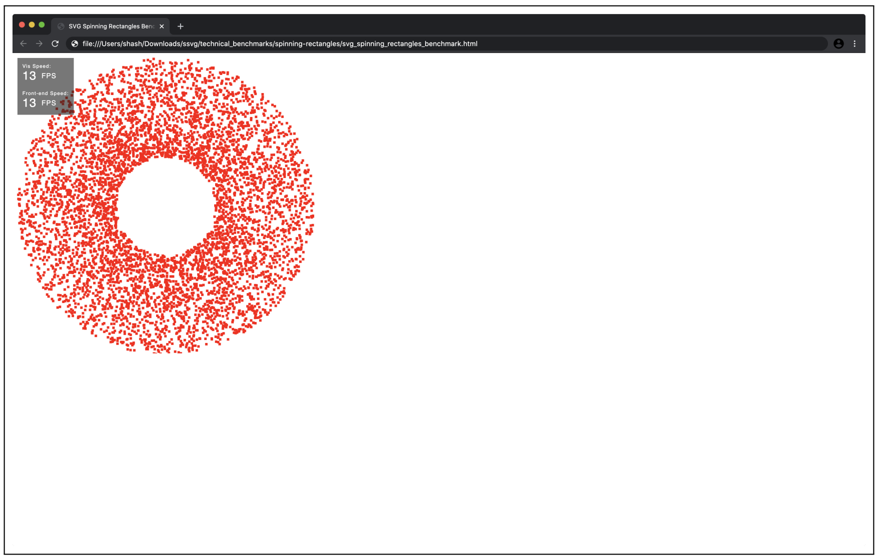

SVG Spinning Rectangles Benc
file:///Users/shash/Downloads/ssvg/technical_benchmarks/spinning-rectangles/svg_spinning_rectangles_benchmark.html
Vis Speed:
13 FPS
Front-end Speed:
13 FPS

Figure 3.1: Spinning rectangles standard SVG benchmark screenshot. Observe that even with only 6000 rectangles (this is printed in the developer console) the FPS is already only 13.

3.2.1 Future work and limitations

Currently each benchmark takes 15 to 30 minutes to run and has to be started manually. It would be ideal if all the benchmarks could be run one after another automatically. If this was doable it would make the benchmark process much easier to perform and perhaps more reliable. Additionally, further investigation into the reason for the dip in FPS at 17000 lines in Figure 3.3 should be conducted as mentioned in the Figure's caption.

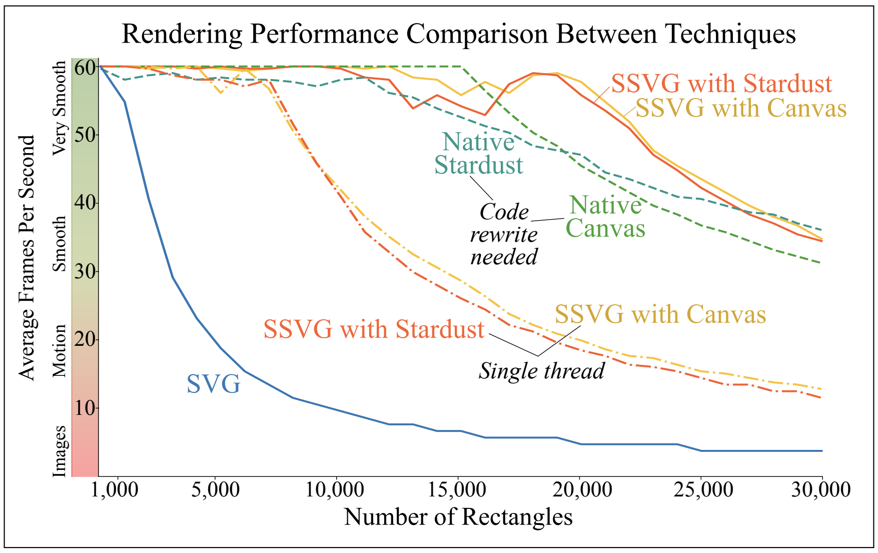


Figure 3.2: Rendering performance comparison between SVG (blue), SSVG (yellow), SSVG with Stardust using WebGL (orange), custom native implementations of the same visualization for Canvas (green) and Stardust (cyan), and a single-threaded version of SSVG and SSVG with Stardust (yellow and orange, dash-dotted) that shows how much multi-threading contributes to SSVG's performance. At 20,000 nodes, the SVG only renders at about 6 FPS, leading to jank and a bad user experience. Meanwhile, both SSVG implementations render at about 55 FPS, on par with and even outperforming custom native implementations of the same visualization in Canvas and WebGL. Data and implementations in supplemental material.

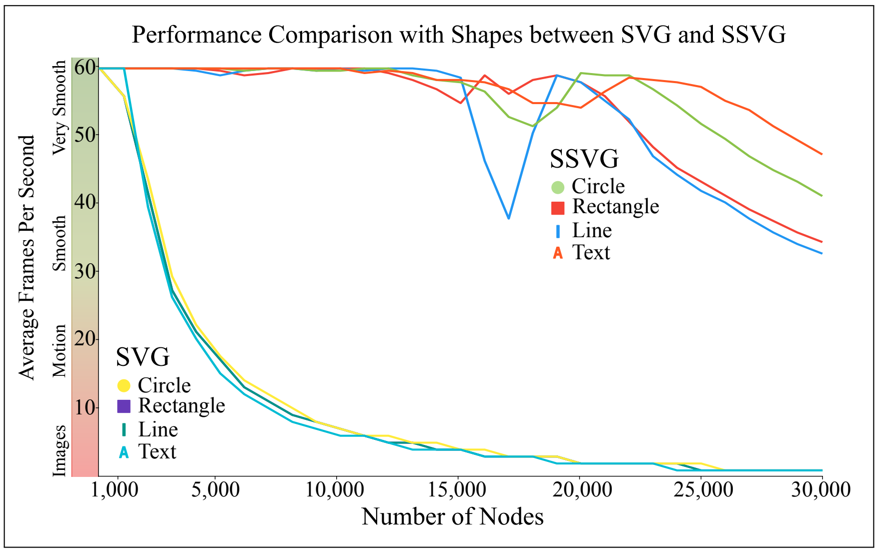


Figure 3.3: Rendering performance comparison between SVG and SSVG for text, circle, rectangle, and line primitives. SSVG outperforms SVG across all tested shapes. Around 17,000 line nodes, SSVG's performance is lower than with more and fewer nodes. This is an anomaly which requires further investigation.

4. Assisting with VisConnect Project

For the VisConnect project [11] various examples needed to be created of how VisConnect could be used by the visualization community. For this purpose I created a Chrome Extension called Live Website Annotate. 4.1 is the relevant section from the research paper produced from the project essentially verbatim and section 4.2 covers some of the current limitations and possible future work that could be conducted both regarding Live Website Annotate and in general in relation to VisConnect.

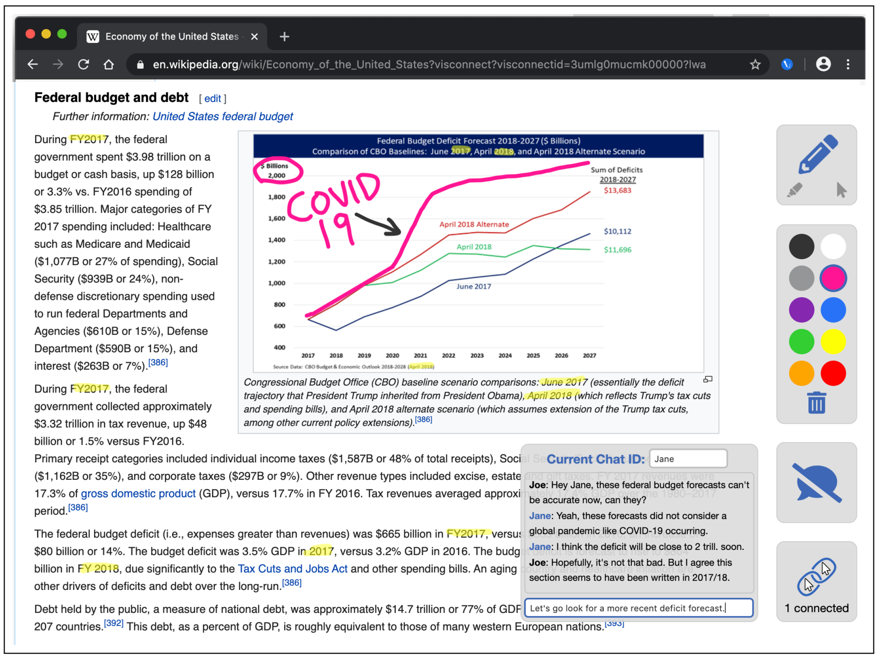
4.1 Real-time Collaborative Annotation

Shared annotations are an ongoing research effort to tackle the challenges of communication and coordination during remote collaboration on visualizations [2, 7]. Collaborative annotation functionalities enable group examinations of visualization and support collective and collaborative sensemaking [14], i.e., people actively pursue shared goals and group interpretations [2]. For example, a new user of a complex interactive visualization may need guidance in learning all the features and functions afforded in the visualization. The tutelage of an expert to guide them with annotations can easily introduce them to the affordances of the tool resulting in the user being more likely to use it again in the future. To illustrate VisConnect's capabilities in synchronized collaborative sensemaking, we created a Google Chrome extension "Live Website Annotate" (LWA) which enables multiple users to collaboratively and simultaneously annotate existing online visualizations and websites.[1] To enable this collaboration, all collaborators need to install the LWA extension from the Chrome Web Store.[2] One collaborator can invite other collaborators to a shared session by creating an invitation link for the current page they are on. Other collaborators can then join the session by entering this link into the extension's menu and share it themselves once connected.

To enable synchronous annotation, the extension adds two stacked SVG elements to a website. The first SVG is used for the user interface, including drawing options for the annotations. The second SVG is used as a canvas for users to draw annotations on.[3] To allow the existing drawing block to be used as a collaborative annotation tool, the typical process for using VisConnect is applied: Load the VisConnect script into the current page, set the `collaboration` attribute to `live` for the the current pages body tag, and replace `d3.drag` with `vc.drag`. We include some other code changes to add a user interface, as shown in Figure 4.1. The annotation methods supported include drawing and highlighting currently in a choice of ten different colors. The tool also includes a chat feature as to enable communication during the drawing process.

Aside from annotations, the LWA extension could be useful for collaborative document review, instruction for novices of complicated user interfaces, or expert data visualizations.

[1]Source code is available at `https://github.com/shash678/Live-Website-Annotate/`
[2]The extension is available at
`https://chrome.google.com/webstore/detail/live-website-annotate/njhclbnmjgcghngbbbacndfcnbhgomac`
[3]The core drawing functionality is based on the existing block at
`https://bl.ocks.org/nitaku/d79632a53187f8e92b15/`


Economy of the United States
en.wikipedia.org/wiki/Economy_of_the_United_States?visconnect?visconnectid=3umlg0mucmk00000?lwa
Federal Budget Deficit Forecast 2018-2027 ($ Billions)
Comparison of CBO Baselines: June 2017, April 2018, and April 2018 Alternate Scenario
$ Billions
2,000
1,800
1,600
1,400
1,200
1,000
800
600
400
2017 2018 2019 2020 2021 2022 2023 2024 2025 2026 2027
COVID 19
April 2018 Alternate
April 2018
June 2017
Sum of Deficits 2018-2027
$13,683
$10,112
$11,696
Source Data: CBO Budget & Economic Outlook 2018-2028 (April 2018)
1 connected

Federal budget and debt [edit]

Further information: United States federal budget

During FY2017, the federal government spent $3.98 trillion on a budget or cash basis, up $128 billion or 3.3% vs. FY2016 spending of $3.85 trillion. Major categories of FY 2017 spending included: Healthcare such as Medicare and Medicaid ($1,077B or 27% of spending), Social Security ($939B or 24%), non-defense discretionary spending used to run federal Departments and Agencies ($610B or 15%), Defense Department ($590B or 15%), and interest ($263B or 7%).[386]

Congressional Budget Office (CBO) baseline scenario comparisons: June 2017 (essentially the deficit trajectory that President Trump inherited from President Obama), April 2018 (which reflects Trump's tax cuts and spending bills), and April 2018 alternate scenario (which assumes extension of the Trump tax cuts, among other current policy extensions).[386]

During FY2017, the federal government collected approximately $3.32 trillion in tax revenue, up $48 billion or 1.5% versus FY2016. Primary receipt categories included individual income taxes ($1,587B or 48% of total receipts), Soci ($1,162B or 35%), and corporate taxes ($297B or 9%). Other revenue types included excise, estate 17.3% of gross domestic product (GDP), versus 17.7% in FY 2016. Tax revenues averaged approxi period.[386]

The federal budget deficit (i.e., expenses greater than revenues) was $665 billion in FY2017, versus $80 billion or 14%. The budget deficit was 3.5% GDP in 2017, versus 3.2% GDP in 2016. The budg billion in FY 2018, due significantly to the Tax Cuts and Jobs Act and other spending bills. An aging other drivers of deficits and debt over the long-run.[386]

Debt held by the public, a measure of national debt, was approximately $14.7 trillion or 77% of GDF 207 countries.[392] This debt, as a percent of GDP, is roughly equivalent to those of many western European nations.[393]

Current Chat ID: Jane

Joe: Hey Jane, these federal budget forecasts can't be accurate now, can they?

Jane: Yeah, these forecasts did not consider a global pandemic like COVID-19 occurring.

Jane: I think the deficit will be close to 2 trill. soon.

Joe: Hopefully, it's not that bad. But I agree this section seems to have been written in 2017/18.

Let's go look for a more recent deficit forecast.

Figure 4.1: A scenario where two collaborators are using Live Website Annotate to scrutinize a graph on Wikipedia. Live Website Annotate provides three main modes: Annotate (currently active)–allows annotating with a solid line, Highlight–allows highlighting with a opaque thicker line, and Interact–allows interaction with the underlying web page. A text based chat is also available to communicate with other connected collaborators.

4.2 Limitations and future work

The main issue I faced when creating this tool was how to deal with collaborators having varying screen sizes and resolutions. After trialing various solutions like dynamically scaling the SVG canvas, aligning the SVG canvas with specific elements in the document body, and rendering the underlying webpage in a standard size iframe. I concluded the most stable solution that allowed for the accuracy desired would be to have all users of the tool have approximately the same window size. To achieve this, since browser vendors have stopped supporting changing a user's browsers window size programmatically for UX concerns, I instead interactively guide users to resize their browser windows to a standard resolution before allowing annotation to begin. This can be seen in Figure 4.2 and Figure 4.3.

This current browser window resizing flow is not ideal UX—if in the future it could be simplified further that would be great.

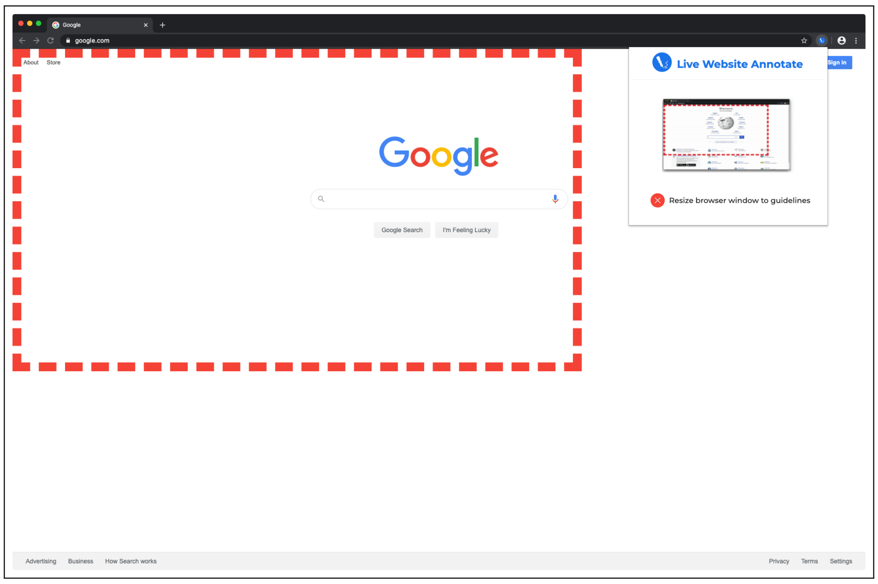


Figure 4.2: What a user would see after enabling the Live Website Annotate extension on the Google homepage when their browser window is too big. Note the browser picture in the extension menu is animated and shows how the browser window should be resized to the guidelines in a loop.

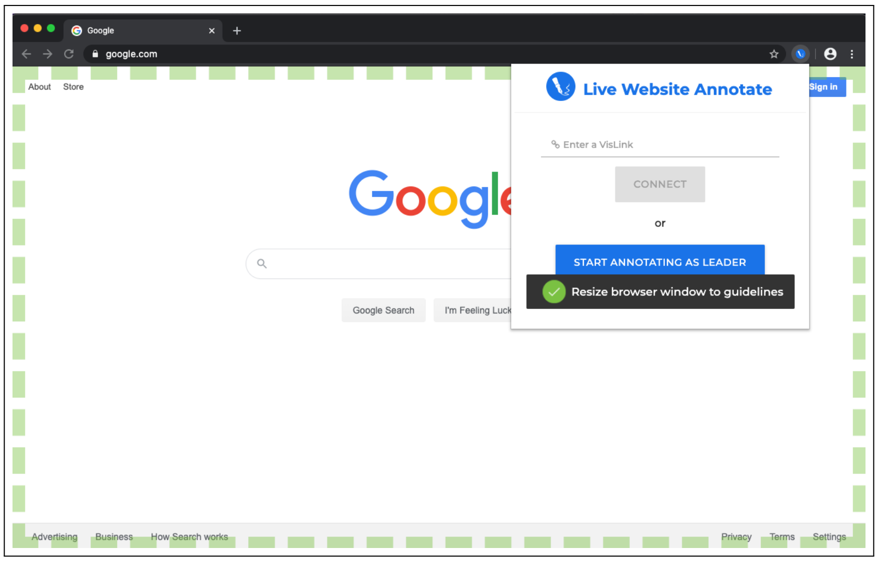


Figure 4.3: What the user from Figure 4.2 would see after resizing their browser window to the guidelines. Note the guidelines are translucent since they are fading out, and similarly the black toast notification in the menu is animated and will disappear soon.

A straightforward improvement that could be made would be to add the ability to move LWA's UI around or to turn it off temporarily. This would solve the issue that for certain websites the relevant content that you might want to annotate might exist in the bottom left portion of the users window behind LWA's UI.

4.2.1 Example of directly collaborative synchronous interaction for visualization

Shortly before the end of the project, I was working on a visualization powered by VisConnect that required more than one person having to work together at the same time. This visualization is shown in Figure 4.4.

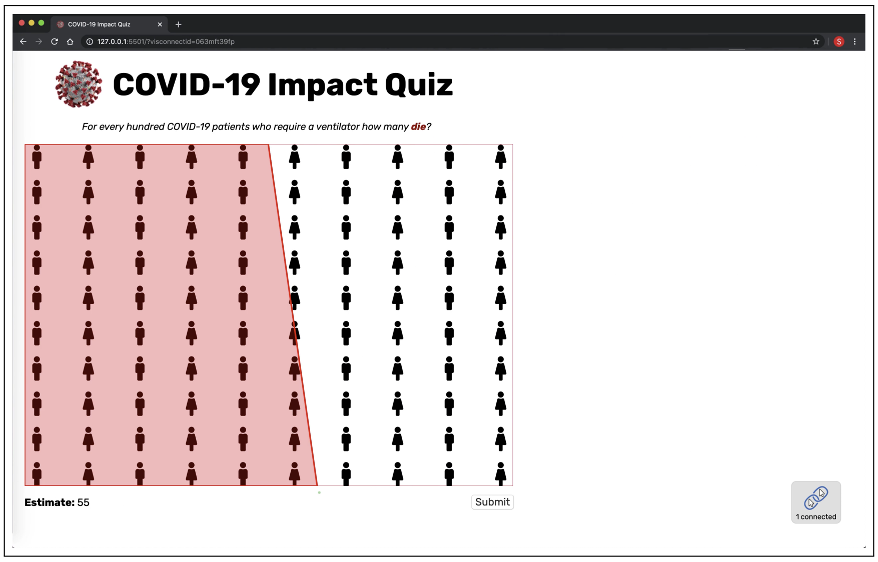

COVID-19 Impact Quiz
127.0.0.1:5501/?visconnectid=063mft39fp
COVID-19 Impact Quiz
For every hundred COVID-19 patients who require a ventilator how many die?
Estimate: 55
Submit
1 connected

Figure 4.4: An alpha version of the interface of a COVID-19 collaborative, interactive quiz where two users are needed to progress. I.e., each user has control over one end of a maneuverable line that can split the pictogram. Each user needs to attach the end of the line that they have control over to the edge of the boundary box. While both line tips are not attached, an area selection is not made, and therefore they cannot submit their answer and progress to the next question/view how they did.

.

Although this exact example is not the most intuitive. The type of interaction it shows, (more than one person having to work together at the exact same time), is a kind of interaction that is really not possible with previous collaborative systems and could be something to pursue in the future to effectively demonstrate VisConnect's benefits.

5. Conclusion

Overall these past two semesters were a gratifying and edifying experience for myself. I learned a lot about the field of visualization and how real-world computer science research is conducted from the people I worked with. I will definitely make sure to be on the lookout for opportunities to implement innovative visualizations and participate in computer science research throughout my career.

References

[1] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. “D^3 data-driven documents”. In: *IEEE transactions on visualization and computer graphics* 17.12 (2011), pp. 2301–2309.

[2] Sabrina Bresciani and Martin J Eppler. “The benefits of synchronous collaborative information visualization: Evidence from an experimental evaluation”. In: *IEEE transactions on visualization and computer graphics* 15.6 (2009), pp. 1073–1080.

[3] The World Wide Web Consortium. *Document Object Model (DOM) Level 3 Core Specification.* 2004 (accessed September 24, 2019). URL: `https://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/`.

[4] The World Wide Web Consortium. *HTML 5.2.* 2017 (accessed September 24, 2019). URL: `https://www.w3.org/TR/2017/REC-html52-20171214/`.

[5] The World Wide Web Consortium. *Portable Network Graphics (PNG) Specification (Second Edition).* 2003 (accessed September 23, 2019). URL: `https://www.w3.org/TR/2003/REC-PNG-20031110/`.

[6] The World Wide Web Consortium. *Scalable Vector Graphics (SVG) 1.1 (Second Edition).* 2011 (accessed September 23, 2019). URL: `https://www.w3.org/TR/SVG11/`.

[7] Petra Isenberg et al. “Collaborative visualization: Definition, challenges, and research agenda”. In: *Information Visualization* 10.4 (2011), pp. 310–326. DOI: `10.1177/1473871611412817`.

[8] Microsoft. *Deep Zoom File Format Overview.* 2011 (accessed September 29, 2019). URL: `https://docs.microsoft.com/en-us/previous-versions/windows/silverlight/dotnet-windows-silverlight/cc645077(v=vs.95)`.

[9] Donghao Ren, Bongshin Lee, and Tobias Höllerer. “Stardust: Accessible and transparent gpu support for information visualization rendering”. In: *Computer Graphics Forum.* Vol. 36. 3. Wiley Online Library. 2017, pp. 179–188.

[10] M. Schwab et al. *Scalable Vector Graphics: Fast Drop-In Rendering for D3.js.* Open Source Framework, 2019. URL: `https://osf.io/mea2v/`.

[11] M. Schwab et al. *VisConnect: Distributed Event Synchronization for Collaborative Visualization.* Open Source Framework, 2020. URL: `https://osf.io/hkaxc/`.

[12] Michail Schwab et al. “EasyPZ. js: Interaction Binding for Pan and Zoom Visualizations”. In: *2019 IEEE Visualization Conference (VIS).* IEEE. 2019, pp. 31–35.

[13] Dave Shreiner and The Khronos OpenGL ARB Working Group. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1.* 7th. Addison-Wesley Professional, 2009. ISBN: 0321552628.

[14] James J Thomas. *Illuminating the path: The Research and Development Agenda for Visual Analytics.* IEEE Computer Society, 2005.