

University of Denver

Digital Commons @ DU

Electronic Theses and Dissertations

Graduate Studies

2022

Geovisualization and Open-Source Web Mapping of Big Origin-Destination Data, a Test Case

Joseph Hiebert
University of Denver

Follow this and additional works at: <https://digitalcommons.du.edu/etd>



Part of the [Geographic Information Sciences Commons](#), [Human Geography Commons](#), [Migration Studies Commons](#), and the [Other Geography Commons](#)

Recommended Citation

Hiebert, Joseph, "Geovisualization and Open-Source Web Mapping of Big Origin-Destination Data, a Test Case" (2022). *Electronic Theses and Dissertations*. 2054.
<https://digitalcommons.du.edu/etd/2054>

This Thesis is brought to you for free and open access by the Graduate Studies at Digital Commons @ DU. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of Digital Commons @ DU. For more information, please contact jennifer.cox@du.edu, dig-commons@du.edu.

Geovisualization and Open-Source Web Mapping of Big Origin-Destination Data, a Test Case

Abstract

Migration plays a key role in determining the health and success of cities, counties, and countries. It also plays a key role in determining the health and wellbeing of the individuals and families that undergo a migration event. This has led many scholars to map and study global migration patterns to understand how and why people move. While migration data are powerful, the origin-destination (O-D), tabular format of the data can be hard to interpret. To make O-D data more powerful, geographers can lean on computer cartography and new geovisualization techniques to help decision makers make sense of large, complex datasets. This research seeks to determine which visualization methods are applicable to represent O-D migration flows at multiple spatial scales via a cost-effective and scalable web-based mapping tool.

Document Type

Thesis

Degree Name

M.A.

Department

Geography

First Advisor

Paul Sutton

Second Advisor

Jing Li

Third Advisor

Guiming Zhang

Keywords

Cartography, Geovisualization, Migration, Open-source, Origin-destination, Visualization

Subject Categories

Geographic Information Sciences | Geography | Human Geography | Migration Studies | Other Geography

Publication Statement

Copyright is held by the author. User is responsible for all copyright compliance.

Geovisualization and Open-Source Web Mapping of Big Origin-Destination Data, a
Test Case

A Thesis

Presented to

the Faculty of the College of Natural Sciences and Mathematics

University of Denver

In Partial Fulfillment

of the Requirements for the Degree

Master of Arts

by

Joseph Hiebert

June 2022

Advisor: Dr. Paul Sutton

©Copyright by Joseph Hiebert 2022

All Rights Reserved

Author: Joseph Hiebert

Title: Geovisualization and Open-Source Web Mapping of Big Origin-Destination Data, a Test Case

Advisor: Dr. Paul Sutton

Degree Date: June 2022

Abstract

Migration plays a key role in determining the health and success of cities, counties, and countries. It also plays a key role in determining the health and wellbeing of the individuals and families that undergo a migration event. This has led many scholars to map and study global migration patterns to understand how and why people move. While migration data are powerful, the origin-destination (O-D), tabular format of the data can be hard to interpret. To make O-D data more powerful, geographers can lean on computer cartography and new geovisualization techniques to help decision makers make sense of large, complex datasets. This research seeks to determine which visualization methods are applicable to represent O-D migration flows at multiple spatial scales via a cost-effective and scalable web-based mapping tool.

Acknowledgements

The author thanks Ilya Boyandin of Teralytics for creating flowmap.gl, a robust open-source web mapping library that was critical for the completion of this project.

Table of Contents

Abstract	ii
Acknowledgements.....	iii
Table of Contents	iv
List of Figures	ii
Chapter One: Introduction	1
Background Information	2
Chapter Two: Methods	7
Project Design.....	7
Coding Tools.....	8
Data Structures.....	10
Code Logic and Implementation.....	13
Chapter Three: Results.....	20
Chapter Four: Discussion.....	25
Challenges.....	27
Opportunities for Improvement	29
Chapter Five: Conclusion	31
References.....	32
Appendix.....	36

List of Figures

Figure 1: Importing code libraries into application coding environment	13
Figure 2: Example of const declaration applied to the getInitialState function	14
Figure 3: componentDidMount function to call and mount location and flow data.....	16
Figure 4: Render method	18
Figure 5: Initial View of Web Application Prototype	21
Figure 6: Hover functionality over Denver County, CO, USA	23
Figure 7: Zoom in and hover over individual flow line functionality	24

Chapter One: Introduction

The United States of America is the most immigrated to country in the world. In 2018, 44.8 million immigrants lived in the United States, accounting for 13.7% of the nation's population (Budiman et al. 2020). This has led many scholars to map and study global migration patterns to understand why and how people come to the United States and other developed areas. However, less attention has been paid to internal migration patterns in the United States. This is problematic as internal migration has been shown to affect population and employment growth and decline within countries, the efficient functioning of economies and housing markets, and allows individuals and families to achieve their goals and aspirations (Bell et al. 2015; Bernard et al. 2014). Geographers can offer insights into to the spatial patterns related to internal migration movement by giving policymakers maps that synthesize large datasets related to internal movement of citizens between regions. This will lead to an increased understanding to why citizens are leaving or coming to a region. This is especially powerful for rural communities that have been experiencing population stagnation or decline over the past century.

Rural communities in the United States are in the midst of a century-long trend of emigration. In 1910, 54.5% of US residents lived in rural areas. In 2010, only 19.3% of US residents live in rural areas and the trend is persisting (US Census Bureau 2016). While scholars have shown that this trend is often attributable to the effects of the decentralization of manufacturing, regional restructuring, changes in rural amenities and

residential preference, expansion of state colleges, etc. (Frey 1987, Johnson and Beale 1994), these explanations do not explain localized changes over time. Local policymakers with intimate knowledge of local municipalities would benefit from a mapping tool that showed immigration and emigration over time to understand which push and pull factors are affecting the decisions of residents to leave or stay. For example, some municipalities such as Tulsa, Oklahoma or Bentonville, Arkansas have observed they are losing residents to tech focused cities (Tulsa Remote 2021, Finding NWA 2021). In response to this trend, the cities have offered incentives, such as \$10,000 moving bonuses or free bikes, to attract remote tech workers to the area and grow the local economy and tech expertise in the area. Often the cost of these incentives is recouped in just one year via direct and indirect economic benefits (Finding NWA 2021). Other municipalities, including rural areas, are likely interested in exploring the feasibility of these programs for their own areas, yet these policymakers often lack the data to garner support for such a program. Migration scholars and local policymakers alike would benefit from a web map tool to analyze internal migration in the United States quickly and effectively, however, such a tool does not currently exist.

Background Information

Internal migration is an important part of the history and identity of the United States, not only for cultural reasons, but for short and long-term economic development reasons as well (Rodriguez-Pose and Berlepsch 2014). Between 1783 and 1898 the United States, through purchase, conquest, and treaty gained 3 million square miles of land sparking a great westward migration (Kim and Margo 2003). Migrants, typically farmers, headed to

the rural west for large, cheap tracts of land and the hope of building new lives and communities. The United States today is experiencing an opposite migration pattern. In 1910, 54.5% of US residents lived in rural areas. In 2010, only 19.3% of US residents live in rural areas and the trend is persisting (US Census Bureau 2016). While these numbers only offer a crude glimpse into the demographics of rural communities, the nature of this migration trend is well documented, showing only short periods of reversal in the early 1970s and early 1990's (Beale, 1977, Johnson and Beale 1994, Barcus, 2004). Given the persistence of this trend, surprisingly little has been published on the systemic causes of the trend and how those causes affect or are affected by the decisions of migrants. Likewise, little attention has been given to analyzing this data at a more granular level through the usage of geographic information systems or web mapping.

Migration plays a key role in determining the health and success of cities, counties, and countries. It also plays a key role in determining the health and wellbeing of the individuals and families that undergo a migration event. While the reasons that migrants decide to move can be complex and motivated by a variety of economic, societal, and environmental conditions, it is generally assumed people move to better their overall wellbeing (Fuguitt and Brown 1990, Green 2019, Martin and Sunley 2007). This suggests that areas that suit the needs and desires of many people, or areas that possess ample "pull" factors, will experience population growth through in migration. The opposite is true for "push" areas that have poor economic prospects, environmental disamenities, or other factors that cause citizens to look at relocating to more attractive locales (Dorigo and Tobler 1983, Krugman 1999). This implies that migration data are

important indicators of the success of a city or region. While politics, environmental factors, and economic conditions may differ in type and quality across a region or country, migration data tell us immediately what areas are attracting, or repelling migrants. Local scholars and policymakers can then use this data and their knowledge of the area to help craft better economic, social, and environmental policy.

Veenendaal et al. (2017) acknowledge that much progress has been made in leveraging spatial data for location-based insights, largely driven by advancements in high-speed communication networks, mobile wireless environments, cloud computing, and Web GIS services. Yet challenges remain as barriers including “linking information around geographic location, analyzing and handling big geospatial data, discovering and interpreting intelligent information” persist today (Veenendaal et al 2017). While migration data are powerful, the origin-destination (O-D), tabular format of the data can be hard to interpret. To make O-D data more powerful, geographers can lean on computer cartography and new geovisualization techniques to help decision makers make sense of large, complex datasets. While there are a variety of approaches that have been proposed for the best way to display origin-destination data, the most common is to leverage flow maps that clearly communicate the migration of citizens from one geography to another. The width or color tone of the flow line can communicate the quantity or intensity of the migration between geographies. The orientation of the flow arrow communicates the origin – destination relationship between geographies of the flow, with the arrow pointing to the destination and away from the origin (Tobler 1987). This methodology has proved effective at displaying relatively small datasets containing about 50 O-D entries, yet

when datasets grow larger than this aggregation is necessary to correct for occlusion and cluttered display (Tobler 1987, Zhu and Guo 2014). There are different methods to aggregate large O-D datasets, each with advantages and drawbacks so picking the right method for each application is crucial to the success of a flow map.

Flow map aggregation methods aim to generalize flow patterns between geographies in one of two ways: using spatial unit-based aggregation or flow-based aggregation. Spatial unit-based aggregation combines small spatial units into larger regions to reduce the number of flow lines on a map. This method is used in flow maps when data availability and computer processing capabilities are limited but can suffer from the modifiable areal unit problem (Zhu and Gao 2014). Even when spatial community structures and larger regions are properly defined, problems persist in the form of information loss (decreased granularity of data) and flow patterns may vary simply due to different aggregations (Zhu et al 2019). Flow-based aggregation mitigates these issues by generalizing the flows via clustering methods, yet the problem of data loss may persist depending on the clustering method chosen. The method generally reduces the data size of large datasets but retains the important flow patterns between geographies (Jenny et al. 2016, Zhu and Gao 2014, Zhu et al. 2019).

The United States contains 3,143 counties or county equivalents. To analyze county level migration trends over a 20-year period, one would need to comb through over 163,000 county dyadic pairs containing over 3.2 million county year observations containing data on over 343 million migrants (Hauer and Byars 2019). This is an onerous task for the time-strapped policymaker or scholar that wants quick insights into the trends

and rates of migrations of regions or counties. This presents a clear opportunity for flow-based aggregation to generalize a large internal migration dataset. In this research, I outline the process of how to develop a web-based mapping tool that can synthesize large O-D datasets and visualize the data in a way that is meaningful and allows for further analysis.

To assist other scholars and local policymakers, my web-based internal migration flow mapping tool for my MA thesis attempts to answer the following research questions:

- What visualization methods are applicable to represent origin-destination internal migration flows at multiple spatial scales in the US via a web-based mapping tool?
- How does one design a tool that is cost-effective and scalable for large (over 500,000 records) O-D datasets

Chapter Two: Methods

Project Design

According to Robinson et al.'s (2005) user centered design framework, the first stage of web-app development is project design and scoping. This step involves assessing the needs and requirements of end users, also referred to as "the client". Effective web applications capture users' attention, are easy to use and navigate, and reliably deliver functionality in a timely manner (Huber and Demetz 2019, Ricker and Roth 2018, Roth 2017). To accomplish this, Shneiderman's (2003) seminal article on effective user interface design for webpages was used to guide application design choices.

To make the web map application engaging and informative, design priorities were to give users the ability zoom, interact, and search for regions of interest to learn more about the immigration and emigration flows at varying spatial scales. To make the web map appearance free from excess buttons that could distract the user and limit visibility of flow lines, the decision was made not to add zoom boxes that allow a user to zoom in and out to regions of interest. Rather, this functionality is achieved via scroll functions on a mouse or trackpad, or with a double click of the mouse. This decision was influenced by You et al.'s (2007) assessment of web map zoom and pan functions where they found "no significant difference between the zoom modes or between pan button layouts." The authors of this paper also found via survey of study participants that zoom button functions were not easy to use even though the icons clearly represented the function.

Another important aspect of web map design is making the map interactive and searchable, so users can explore data and regions that are of interest to them. Most users of web maps today are familiar with both mouse and touch screen input methods for interacting with mapping applications. The advancement of rendering speed and continuous control and continuous display navigational modes have led users to be able to connect with web maps and their associated data in real time to gain on demand insights (Wu et al. 2011, Huber and Demetz 2019, Ricker and Roth 2018, Roth 2017). This led to the development of an app with established navigation functionality that gives users the ability to pan to new map views and hover over migration flows and location nodes of interest. As the user hovers over an area of interest, for example Denver County, CO, all migration flows both in and out are highlighted in white, while other flow lights are darkened and rendered in the background. This same hover function can be applied to individual flow lines to observe migration flows between two locations.

Coding Tools

This research originally proposed using ESRI software to create a web application capable of implementing all design choices outlined above. However, ESRI tools, such as the distributed flow mapping tool, do not currently support the mapping of origin destination data with more than one origin point. My dataset has many origin points and requires multiple lines to be drawn to many destinations, meaning ESRI products were not able to be utilized for my application. Once it was determined that ESRI products were insufficient to implement the design choices outlined in the research proposal and project design, open-source coding options were explored. A variety of coding languages

were examined to understand which languages allowed for the most amount of customization when it comes to web mapping. Most open-source code libraries for web mapping are based on JavaScript due to its popularity in user interface development for web pages. According to the University Consortium for Geographic Information Science and Technology Body of Knowledge, JavaScript is often used for web mapping due to its ability to add automation, animation, and interactivity into web pages and mapping applications (Swift 2020). This claim was verified by the fact that many other coding languages and libraries leverage JavaScript and can be used in conjunction with established JavaScript techniques and best practices. JavaScript is particularly powerful for building user interfaces when paired with an open-source JavaScript library called React.

React was developed by Facebook and released for public use in 2013. React allows developers to build single-page and mobile applications by building reusable components (Facebook 2022). This means that once a component of a web application has been developed, it can be reused to accomplish other goals. For example, once a legend box has been developed, you can reuse the legend box for other tasks, such as creating a menu bar to change the dataset being visualized on the map, or to change colors of flow lines. React also has the advantage of efficiently updating and rendering data, while being very easy to debug due to its efficient structure.

After reviewing many JavaScript-based coding platforms, it was determined that a combination of the React and JavaScript coding languages would allow the design goals outlined above to be achieved. Given the time constraint of the project and lack of

experience with JavaScript and React, writing an application completely from scratch was not an option for this project. This meant a web mapping library of tools would need to be identified to serve as a foundation for app development. This library needed to be customizable and scalable with the capability to handle large amounts of data with good rendering speed.

After searching GitHub and other open-source code repositories, a WebGL-based code library called flowmap.gl was identified as a code library capable of performing the tasks outlined above. This code library was developed by Ilya Boyandin of Teralytics, a developer of mobility and transportation coding solutions (Teralytics 2022). This library is written in Typescript, a syntactical superset of JavaScript, and allows developers to create custom webapps in JavaScript and React by calling pre-developed app components and tools. While this greatly increases the speed of development for web apps, getting a working prototype is no simple task and requires a solid understanding of geographic data types and the front-end coding logic of JavaScript, React, and other languages like TypeScript and ES6.

Data Structures

To make use of the flowmap.gl library, a solid understanding of the data structures used by the tools needed to be determined. The geographic data needed to be organized such that large amounts of polygon, point, and line data could be read and rendered. To understand the geographic data structure, I looked at a developer's example source data and found that the location polygon and centroid data were stored in a JavaScript Object Notation (JSON) file with the following format:

```
{"type":"FeatureCollection","features":[{"type":"Feature","properties":{"abbr":"ZH",  
"name":"Zürich","no":1,"centroid":[8.654789284720719,47.412606122294044]},"ge  
ometry":{"type":"Polygon","coordinates":[[[8.808068406840686,47.2208912891289  
15],[8.793098109810982,47.223080508050806], etc. ]]]}],
```

This JSON file designates the identifier of the polygon (abbr), name of the polygon (name), the coordinates of the center of the polygon (centroid), and the coordinates of each vertex in the polygon (coordinates, not all coordinates shown here in this example). For this project, these location polygons are county boundaries, with centroid coordinates generated and added to a location JSON file in ArcPro. Formatting of this JSON file was necessary to meet the data structure requirements of this library as well. When ESRI's ArcGIS Pro exports a JSON file, there are extra brackets and apostrophes that are assigned in the coordinates field and these extra characters must be found and deleted. To clean the location JSON file, a combination of web-based JSON data formatting tools and Microsoft's Notepad+ application were used to find and replace these characters.

To improve render speed the geometry of the polygons needed to be simplified. To accomplish this, a tool called MapShaper was used (MapShaper 2022). MapShaper allows users to import polygon shapefiles and decrease the number of vertices in each county boundary. This makes for less detailed boundaries, yet for the purposes of this tool, cadastral levels of map accuracy are not needed. Render speeds are 95% faster after making this change and the user is still able to accurately determine which county they are in.

The format of the “flows” data is quite similar to the location file, but instead of writing in geometry, the file references the locations file geometry for rendering of the lines:

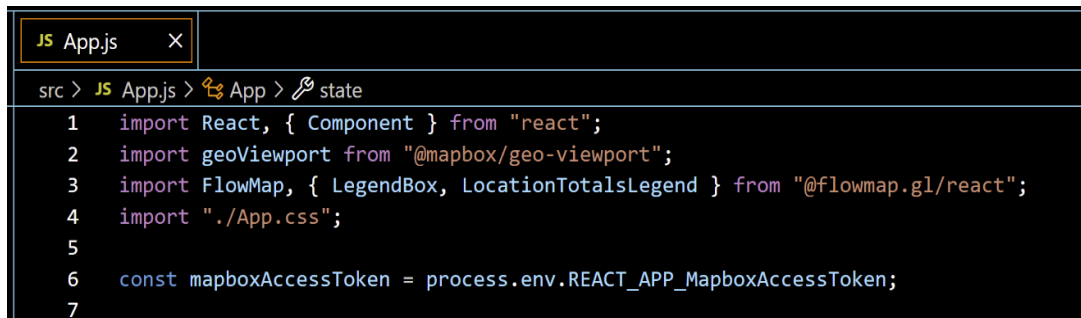
```
[{"origin":"ZH","dest":"ZH","count":66855},{ "origin":"ZH","dest":"BE","count":1673},{ "origin":"ZH","dest":"LU","count":1017}, etc.]
```

This data format allows for painless fetching of data and allows for each attribute of the data to be called in an efficient way that tells tool how to render the data. This will be discussed further below with the discussion of code logic. My flow data are derived from Internal Revenue Service (IRS) tax return data. If a citizen submits a tax return with a different address from the year prior, that citizen and any dependents will be counted as an out migrant in the county they are leaving and an in migrant in the county they move to (Hauer and Bauer 2019). My prototype web application is showing all migration in 2010 that has an origin in a Colorado county (1,817 flow lines), but this application platform can support even more flow data while maintaining good rendering speed and performance.

Once the data were formatted correctly, development of a simple, static web application began. The goal of this initial development was to create a proof of concept for the web application design. To build this application, Visual Studio Code was used as a text editor, GitHub was the version control manager, code repository, and final hosting site. NPM served as a package manager, which is the default package manager for the JavaScript runtime environment Node.js. These systems allowed for real time visualization of application by hosting a development server locally on my machine.

Code Logic and Implementation

The first step in the coding process is to identify the code packages needed to create the application and call them into the application coding environment (app.js file). This is done by analyzing the code libraries that will be used and calling these libraries into the coding environment via the import function. Specific tools or functions can be called in by name from a library by using braces or by calling in a whole file as shown in Figure 1. The import is a crucial first step as this establishes the connection between the scripts in the library and the functions called in the application. Without the import step, most of the functions called in the app.js would be undefined and would fail to render. The next step after the importing libraries and tools is establishing parameters and properties of variables that will be called later in the application.

A screenshot of a code editor window titled "JS App.js". The editor shows the following code:

```
src > JS App.js > App > state
1 import React, { Component } from "react";
2 import geoViewport from "@mapbox/geo-viewport";
3 import FlowMap, { LegendBox, LocationTotalsLegend } from "@flowmap.gl/react";
4 import "./App.css";
5
6 const mapboxAccessToken = process.env.REACT_APP_MapboxAccessToken;
7
```

Figure 1: Importing code libraries into application coding environment

To establish default values of various application elements, the const declaration is used. For example, to establish the map extent shown when the application first renders, a const declaration is used to define a bounding box. This is shown in Figure 2, where `const bbox = [-109.06, 36.99, -102.04, 41.0]`. This defines the parameters of the function “getInitialViewState”. When the getInitialViewState function is called in the render function, this bbox declaration is read and the initial map extent is centered over

the state of Colorado, the region of interest for the migration data used. The const declaration was also used to define the colors used for locations, flow lines, and legend items. It also works as a method to call in the MapBox basemap API access code, which is stored in an environment file for security. The next step in the coding process is unique to React and involves declaring a component that can be called in an entry point file, such as index.js.

```
6  const mapboxAccessToken = process.env.REACT_APP_MapboxAccessToken;
7
8  const getInitialViewState = () => {
9    const bbox = [-109.06, 36.99, -102.04, 41.0];
10   const {
11     center: [longitude, latitude],
12     zoom,
13   } = geoViewport.viewport(
14     bbox,
15     [window.innerWidth, window.innerHeight],
16     undefined,
17     undefined,
18     512
19   );
20   return {
21     longitude,
22     latitude,
23     zoom,
24     bearing: 0,
25     pitch: 0,
26   };
27 };
28
```

Figure 2: Example of const declaration applied to the getInitialViewState function

In React it is necessary to declare either a class-based or functional component than can be called by the index.js file to render the user interface. For this project, a class-based component is used so the “export default class App extends Component {“ line is written to create the environment where the user interface will be defined. Once the class-

based component is defined, the next step is establishing the default property values of the state object. This is similar to const declarations, yet state is establishing component initial values that can be modified using built in React functions. In other words, the state is an object that holds some information that may change over the lifecycle of the component. For the purposes of this application, the state values that are defined here are locations and flows. Each of these state components are set to null as a default and the values are updated during the mounting phase, which is the last function to be called. Utilizing null as the initial state for locations and flows is a great way to check for errors when developing code, because if no flows or location centroids and boundaries are rendered on your development map, you know there is an error either fetching the flow and location data or in the render method.

Once default state values are declared, a lifecycle mounting method can be established to fetch the data from the public folder where the data are stored in JSON format. To do this, the `componentDidMount` function was used. This function is used in class-based components and is run last in the lifecycle process. This means that once all other components of the user interface are rendered (the basemap, the legend, etc.), the location and flow data are called and rendered. This is a great way to render data because the other components render very quickly and need to be presented first to make sense of the flow and locations once they appear. It also lets the end user know the app is loading correctly and that other data is likely to appear soon.

The anatomy of the `componentDidMount` function shown below in Figure 3 shows how the locations and flows get mounted and set to the state. First, the fetch

function is called, and the first argument is the URL which houses our data. Given the data are on the project GitHub repository, the public root folder path needs to be defined using the `process.env.PUBLIC_URL` method. The file path is then completed by defining the data folder path to the JSON file. The response of this data call is then stored using the `.then` method and finally the state is defined using the `this.setState` method.

```
86
87   componentDidMount() {
88     fetch(
89       `${process.env.PUBLIC_URL}/data/US_Counties_AND_Centroids_NoPR_5.5.json`
90     )
91       .then((response) => response.json())
92       .then((json) => this.setState({ locations: json }));
93
94     fetch(`${process.env.PUBLIC_URL}/data/CO_Origin_flows.json`)
95       .then((response) => response.json())
96       .then((json) => this.setState({ flows: json }));
97   }
98
```

Figure 3: `componentDidMount` function to call and mount location and flow data

The last element to define in the class component is the render method. This is where the structure and properties of the UI are defined. The first step of the render method is to establish the state variables. This tells React to look at the flows and locations properties of the state and to allow them to be called by name in the return statement below. As noted previously, the state is null until the mounting phase (the last step in the lifecycle method for class-based components in React), so locations and flows will be rendered last. In the return statement, there are two classes being called: `FlowMap` and `LegendBox`. The `FlowMap` and `LegendBox` classes are components created by Ilya Boyandin of Teralytics. These classes are able to be called and defined here because the classes along with all available properties were imported at the beginning of our coding process (`import FlowMap, { LegendBox, LocationTotalsLegend } from`

"@flowmap.gl/react"). This makes the code in the section much cleaner as all the functionality we are calling into this return function is defined in the FlowMap class. It is possible to alter the FlowMap class to meet custom requirements, such as speed of the animation or altering clicking behavior. Other properties can be edited by defining the function properties using the const method as described earlier with the getInitialState function. The LegendBox class defines the attributes and location of the legend box. Figure 4 below shows the render method.

```

99     render() {
100       const { flows, locations } = this.state;
101       return (
102         <>
103           <FlowMap
104             flows={flows}
105             locations={locations}
106             animate={true}
107             pickable={true}
108             colors={DARK_COLORS}
109             getLocationId={(l) => l.properties.GEOID}
110             getLocationCentroid={(l) => l.properties.Centroid}
111             getFlowOriginId={(f) => f.origin}
112             getFlowDestId={(f) => f.dest}
113             getFlowMagnitude={(f) => f.count2010}
114             initialState={getInitialViewState()}
115             mapboxAccessToken={mapboxAccessToken}
116           />
117           <LegendBox
118             bottom={35}
119             left={10}
120             style={{ backgroundColor: "#CCC", color: "#000" }}
121           >
122             <LocationTotalsLegend colors={DARK_COLORS} />
123           </LegendBox>
124         </>
125       );
126     }
127   }
128

```

Figure 4: Render method

In the FlowMap class, twelve properties are defined, and this is what gives the flows and locations their appearance and functionality. The first two properties identify the location of the flow and location data, which are assigned to the state. The next property uses a Boolean operator to animate the flow lines with properties designated in the FlowMap layer class. The settings of the animation, such as how quickly the animation moves, the size of the tick marks that make up the animation, and the duration

of the animation, can be manipulated in the `AnimateFlowLinesLayer` class. The `pickable` property uses a Boolean operator to determine whether the flow lines and locations are selectable when the mouse hovers over an area. Just as the animation settings can be altered, the `pickable` conditions can be changed for custom applications. The next six properties after `pickable` listed in figure 4 define the colors and tell the `FlowMap` class where the location and flow data are and how to read in the data. The IDs of the locations and flows are identified along with the centroid coordinates and the number of people migrating from one location to another to determine the thickness (magnitude) of the flow lines. The last two properties of the `FlowMap` class set the initial map view extent when the page loads and read in the basemap by calling the MapBox API key.

When the application is functioning fully and with good render speed in the development environment, the `app.js` file can be committed to GitHub. Once changes are committed, the command “`npm run deploy`” is run to create a build package that optimizes the application for the web and allows for successful deploy to GitHub Pages, the application hosting site.

Chapter Three: Results

The results of this application are best viewed on the web either on a desktop or a mobile device at the following URL: <https://jh-gis.github.io/flowmap.gl-example/>. Figure 5 shows the initial view of the webpage which is displaying all migration in 2010 that has an origin in a Colorado county. Flow lines are colored and layered by their magnitude (number of people moving from one county to another). Lower magnitude flow lines are shown in darker blue, while higher magnitude flow lines are light blue or white and draw above the darker lines. This draws users' attention to areas where greater numbers of people are moving and improves interpretation of the map. User's can also consult the legend box to understand which counties are experiencing more incoming or outgoing migration patterns. The magnitude of the location centroid also gives users a relative sense of the migration occurring in that county (larger location circles indicate more migration, either incoming or outgoing).

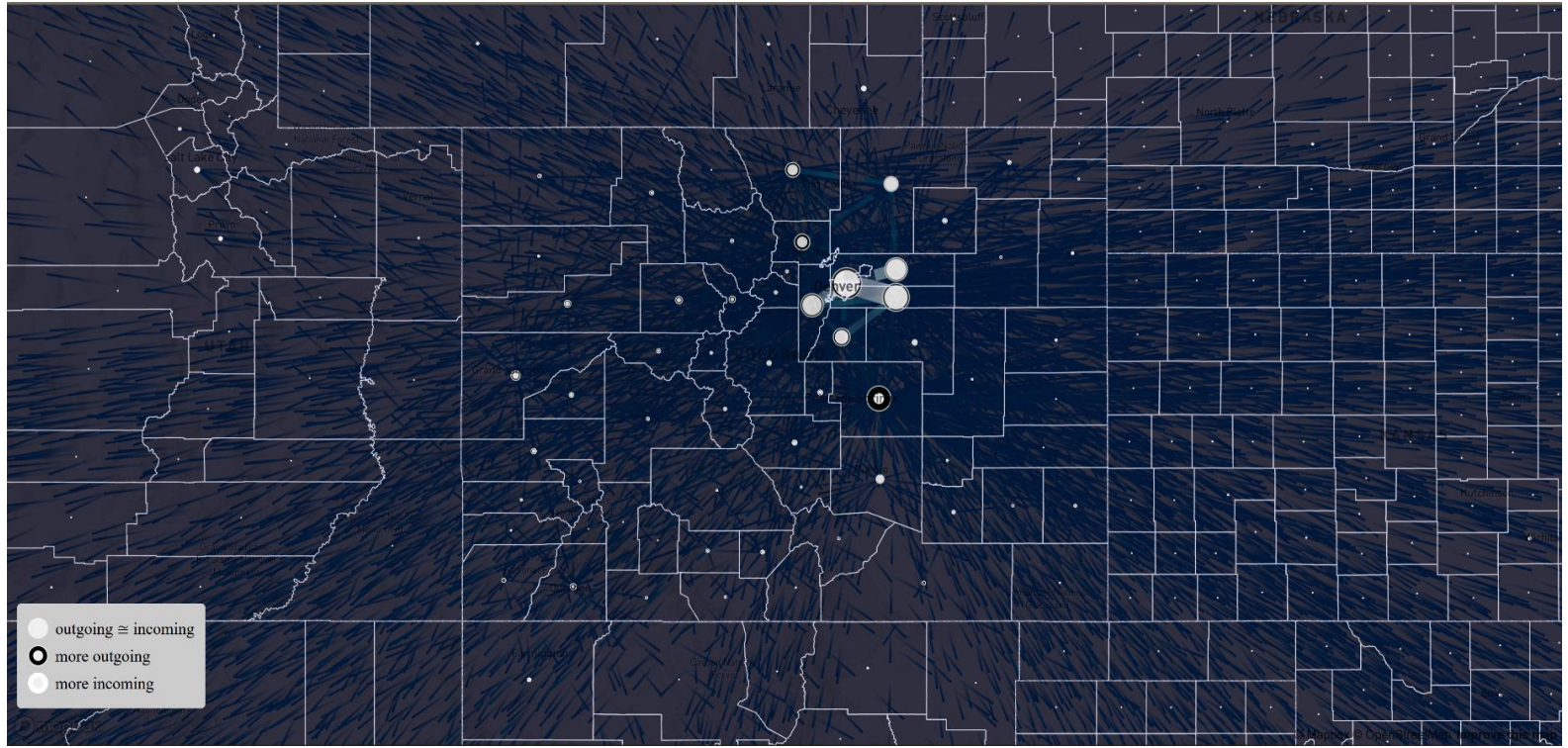


Figure 5: Initial View of Web Application Prototype

Once the user interface has rendered fully, the user can begin exploring the map and data. Users can zoom in and out using a scroll wheel, trackpad, or double clicking on an area (zoom in only for double click). As a user zooms into an area, the size of the location centroid circles and flow lines adjust to the zoom level and maintain their relative size. Flow lines and locations centroids increase in size as the user zooms out and decrease in size as the user zooms in. Users are also able to use their mouse to hover over a location centroid to highlight all incoming and outgoing migration flows. If a user hovers over an individual flow lines, that flow line is highlighted. These functionalities are shown in figures 6 and 7. This application also animates the flow lines, with all lines displaying moving tick marks. The tick marks are larger for higher magnitude flows, but all lines animate at the same speed to avoid disorienting the user.

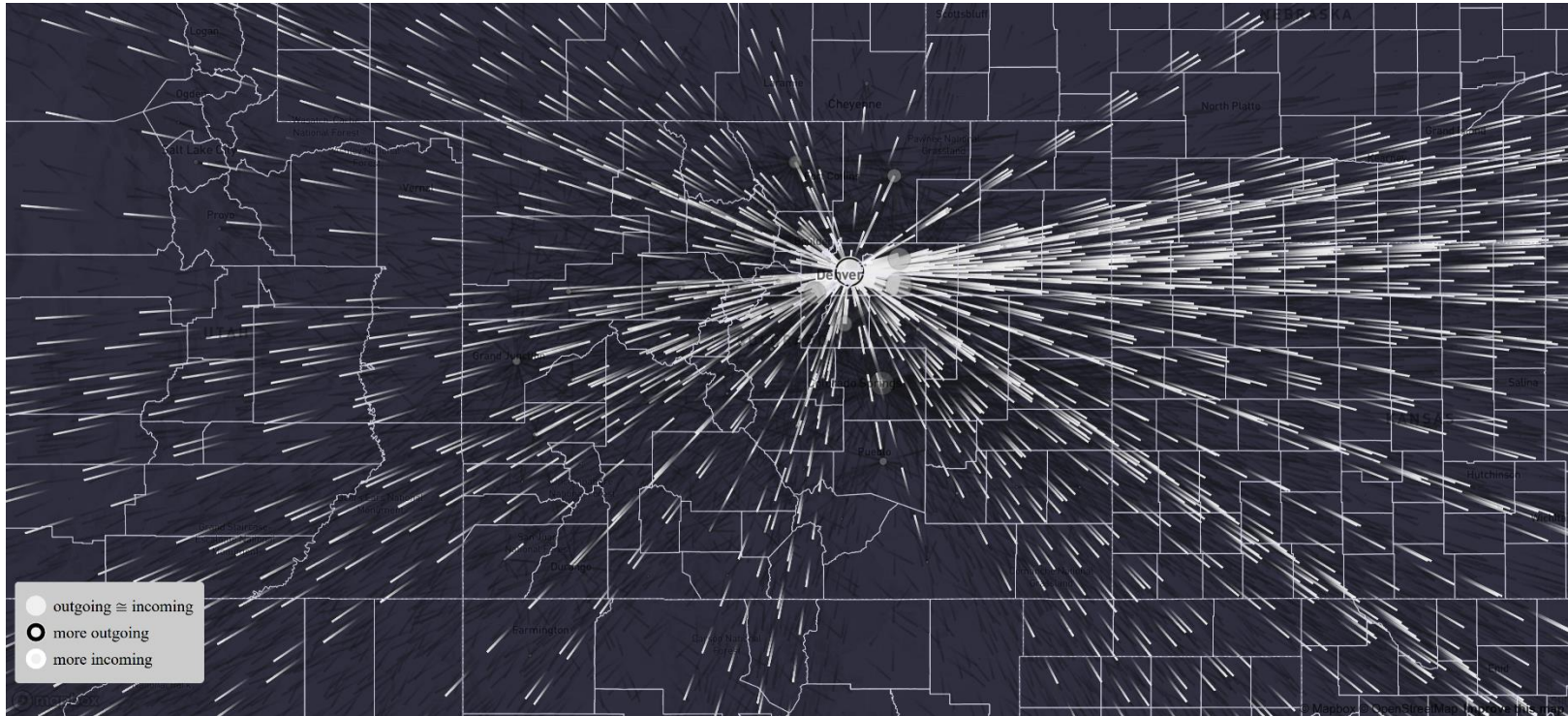


Figure 6: Hover functionality over Denver County, CO, USA

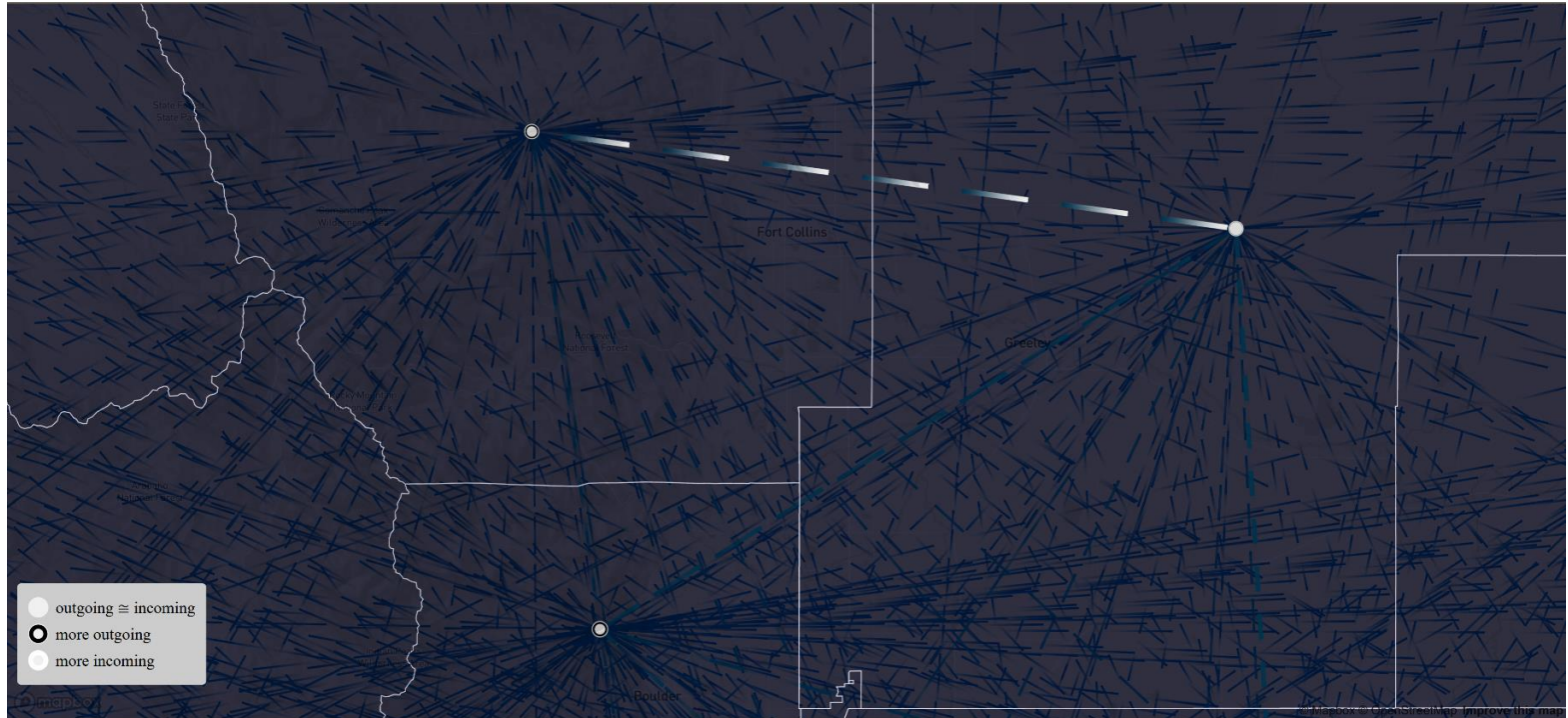


Figure 7: Zoom in and hover over individual flow line functionality

Chapter Four: Discussion

This prototype shows that large amounts of migration data can be efficiently rendered and displayed on the web. These data and visualizations can offer insights into the movements of people and can be leveraged for decision making by city planners, demographers, and the public. At a glance, users can get a relative comparison of migration flows around the country and understand which areas are gaining and losing the most citizens. The layering, coloring, and animation of the flow lines allows large amounts of data, typically represented in graphs and tables, to be presented in a digestible way that suits visual consumers of data and user seeking quick insights. The interactivity of this map also serves as a foundation for a tool that can be leveraged for further insights and time series trend analysis via pop-up and query functions that can be added in future work. This tool presents a template for engaging audiences that are not motivated by or not savvy with large Excel spreadsheets.

To understand the impact this tool can have on decision making, consider Denver County migration in 2010 (figure 6). Users of the tool can see that Denver is experiencing net emigration by looking at the small, black ring in the county centroid. Emigrants are moving all over the country, yet the predominant flows are to the east and west. Users can gain further insight by highlighting individual flows between two counties of interest, gaining a relative sense of magnitude in migration. This visualization can be used by users of various education and interest levels and can give policymakers insight into areas

that have ample pull or push factors. Using demographic and economic data, policymakers can complete the story of why people are moving to certain areas and leaving others. Sinar (2015) explains that users of data visualizations that utilize position, length, density, and color can detect patterns in 200 to 250 milliseconds, further validating the merits of an origin-destination mapping tool. However, further research needs to be performed to understand exactly how much data can be digested by an end user and how managerial audiences can form actionable conclusions from the data (Pandey et al. 2014).

This application is scalable and cost-effective for very large O-D datasets. The application was tested and can support the rendering of over 5,000 flow lines, with the main render speed bottleneck being the polygon data (county and state boundaries), not the flow lines themselves. The tool can likely support even more flow lines with good render speed, but animation and hover functionality may make the tool less comprehensible to end users. This cognition problem can be mitigated by introducing flow line clustering via logic that combines flow lines based on zoom level. The only costs associated with developing an open-source mapping application are time, computer expenses, and possibly hosting costs if the application receives high amounts of web traffic. This tool highlights the benefits of using open-source code to develop custom solutions to geovisualization challenges, but it should be noted that there are also barriers to widespread adoption of open-source applications.

Challenges

While the methods of this project may seem straightforward and efficient, there are many challenges associated with developing a web application with open-source code. Challenges are accentuated when the developer lacks substantial experience with the coding languages and environments being used as well. For this project, an intermediate to advanced level of experience with JavaScript, React, TypeScript and GitHub are necessary for understanding the code logic used in the code libraries on which this application is built. Then, an understanding of how to apply these code libraries to a custom application is required to leverage the power of these tools. While many challenges experienced in the building of this application could be mitigated with more coding and application development experience, there are other challenges that all web developers face when leveraging open-source code.

Open-source code offers the advantages of being transparent, customizable, and free to use, yet for some users and developers the challenges and shortcomings outweigh the benefits. Documentation can be limited in open-source code projects as community contributors are often more interested in contributing to the code, rather than writing about how to code works (Gacek and Arief 2004). There was almost no documentation associated with the flowmap.gl code libraries used in this project, except for a few readme documents and the occasional comment in the backend code libraries. Some of these challenges can be mitigated by an active coding community that can respond to questions on forums and issue discussion boards, but this support is not reliable and often gets less available as libraries age (Stol and Ali Babar 2010).

The lack of reliable maintenance can also be a hinderance to widespread implementation of open-source code for web application development. Older JavaScript and React libraries are often not maintained very well as new tools are being developed all the time, often with new coding techniques and tools. While creating this application, the author of the flowmap.gl code library began posting a newer library to GitHub. When requesting help interpreting code in the FlowMap library, the author recommended I use his new library, rather than helping me troubleshoot an issue with his old library. This was not feasible as it would have forced large amounts of code the code to be rewritten and the time constraint of this master's thesis project did not allow for that change.

In addition, code libraries are often reliant on many dependencies (other code libraries or tools that are also constantly being updated and altered). When dependencies are not updated or code logic is not updated to reflect changes in dependencies, functionality can be lost. This limits the reliability of open-source code as it ages and may cause developers to consider using paid services that they know will be serviced by a team of developers through time. Other free services, such as hosting sites, suffer from similar problems, yet the advent of the freemium business has mitigated some of these issues (Zeleti et al. 2014). For example, Heroku, a free to use hosting service that also offers paid services that add extra features and storage space, suffered from a security breach that left the connection between GitHub and Heroku severed for about two weeks. This outage was investigated by a large team that can focus on keeping Heroku active and current, due to the revenue produced by the freemium model. Due to this outage,

however, I had to figure out new deployment method (GitHub Pages) which took time away from development of other tools such as search and popups.

Opportunities for Improvement

While this web application accomplishes the goal of being a proof of concept for a new way to represent large amounts of O-D data, there are many opportunities for improvement and future work. Users of web maps, such as Google Maps and Apple Maps, expect certain functionality to be present. The ability to search for areas of interest is a crucial navigational function that should be added to this application. Currently, MapBox does not support a geocoder (search bar) that can be used with the flowmap.gl library and syntax used in this application. There are possible workarounds to this, but due to time constraints this task was not completed before this conclusion of this project.

Users would benefit from other functionality not currently available on this prototype. A pop-up window that renders when hovering over the flow lines and locations would give a more detailed look into the number of in and out migrants, rather than a line that just shows relative differences. This pop-up functionality would be further improved by offering alternative visualizations such as charts or graphs, giving users more ways to interact with this data. Originally, this project intended to make time series data available in a pop-up that would allow users to identify trends over time for a particular migration flow. If time series data is made available, users could also select what year of migration data they wish to be visualized on the flow map. Adding this data would require more data cleaning following the JSON format described in the methods but could be done in a couple of hours. To call the time series analysis data, however, a change in logic

associated with the data mounting lifecycle method would be required. While adding this functionality would take more time and effort, more robust insights would be gained pertaining to migration trends and patterns through time.

There are also ways to improve the visualization of the current dataset. The `flowmap.gl` library supports a method of aggregating data to “clusters”. This causes flow lines to be grouped at coarse zoom levels and as a user zooms into an area, more flow lines are visualized. Cluster functionality leads to more a clean and efficient user experience, where the user can discover more detail if wanted, or can gain different insights into broad migration patterns (Koylu and Guo 2017). These new tools and functions would improve the user experience and the insights gained from using the tool.

Chapter Five: Conclusion

This project has shown that large amounts of origin destination data can be effectively displayed at various spatial scales on the web. The geovisualization of big data is both cost-effective and scalable using open-source tools. The performance of these tools will continually improve as other dependent technologies become more capable. As more geospatial data become available via mobile devices, internet of things, and other GIS enabled technologies, the demand for insights derived from this data will grow. While visualization techniques and strategies will surely evolve through time, open-source GIS web developers will continue to create and share new tools to meet demand.

The challenges of documentation, maintenance, and support will likely remain a barrier to widespread adoption of niche, open-source code, but there are still many advantages to using open-source libraries. A geographer with extensive experience in JavaScript, React, or other front-end coding languages can build compelling, interactive geovisualization tools using open-source code and other resources unavailable in popular GIS platforms, such as ArcGIS Pro and QGIS. The creativity, flexibility, and collaborative nature of open-source code will surely push the boundaries of what is possible for geovisualizations and the field of geography.

References

- Barcus, H.R., 2004, Urban-Rural Migration in the USA: An Analysis of Residential Satisfaction. *Regional Studies*, v. 38, pp. 643-657.
- Beale, C.L., 1977, The recent shift of united states population to nonmetropolitan areas. *International regional science review*, v. 2, pp. 113-122.
- Bell, M., Charles-Edwards, E., Ueffing, P., Stillwell, J., Kupiszewski, M., Kupiszewska, D., 2015, Internal Migration and Development: Comparing Migration Intensities Around the World, *Population and Development Review*, v. 41(1), pp. 33–5.
- Bernard A., Bell, M., Charles-Edwards, E., 2014, Life-Course Transitions and the Age Profile of Internal Migration, *Population and Development Review*, v. 40(2), pp. 213–23.
- Budiman, A., Christine, T., Mora, L., Noe-Bustamante, L., 2020, Facts on U.S. immigrants, 2018 Statistical portrait of the foreign-born population in the United States. *Pew Research Center*.
- Dorigo, G., Tobler, W., 1983, Push-Pull Migration Laws, *Annals of the Association of American Geographers*, v. 73, pp. 1-17.
- Facebook, 2022, React, *Facebook Open Source*.
- Finding NWA, 2021, Business, *Finding Northwest Arkansas*, <https://findingnwa.com/incentive/>.
- Frey W.H., 1987, Migration and depopulation of the metropolis: Regional restructuring or rural renaissance? *American Sociological Review*. v. 52, pp. 240-257.
- Fuguitt, G.V., Brown, D.L., 1990, Residential Preferences and Population Redistribution: 1972-1988. *Demography*, v. 27, pp. 589-600.
- Gacek, C., Arief, B., 2004, The many meanings of open source, *IEEE Software*, v. 21, pp. 34-40
- Green, A., 2019, Understanding the drivers of internal migration, *Internal Migration in the Developed World: Routledge*, chp. 2, pp. 1-25.
- Hauer, M., Byars, J., 2019, IRS county-to-county migration data, 1990–2010, *Demographic Research*, v. 40, pp. 1154-1165.

- Huber, S., Demetz, L., 2019, Performance Analysis of Mobile Cross platform Development Approaches based on Typical UI Interactions, *In Proceedings of the 14th International Conference on Software Technologies (ICSOFT)*, pp. 40-48.
- Jenny, B., Stephen, D.M., Muehlenhaus, I., Marston, B.E., Sharma, R., Zhang, E., Jenny, H., 2016, Design principles for origin-destination flow maps, *Cartography and Geographic Information Science*, v. 45, pp. 62-75.
- Johnson, K.M., Beale, C.L., 1994, The recent revival of widespread population growth in nonmetropolitan areas of the US. *Rural Sociology*, v. 59, pp. 655-667.
- Kim S., Margo, R.A, 2003, Historical Perspectives on U.S. Economic Geography. *NBER Working Paper Series*, v. 9594.
- Koylu, C., Guo, D., 2017, Design and evaluation of line symbolizations for origin–destination flow maps, *Information Visualization*, v. 16, pp. 309–331.
- Krugman, P., 1999, The role of geography in development. *International Regional Science Review*, v. 22., 142-161.
- MapShaper, 2022, MapShaper, <https://mapshaper.org/>.
- Martin, R., Sunley, P., 2007, Complexity thinking and evolutionary economic geography. *Journal of economic geography*. v. 7, pp. 573-601.
- Pandey, A. V., Manivannan, A., Nov, O., Satterthwaite, M. L., Bertini, E., 2014, The persuasive power of data visualization, *New York University Public Law and Legal Theory Working Papers*, p. 474.
- Ricker, B., and Roth, R. E., 2018, Mobile Maps and Responsive Design. *The Geographic Information Science & Technology Body of Knowledge*, 2nd Quarter.
- Robinson, A.C., Chen, J., Lengerich, E.J., Meyer, H.G., MacEachren, A.M., 2005, Combining Usability Techniques to Design Geovisualization Tools for Epidemiology, *Cartography and Geographic Information Science*, v. 32, pp. 243-255.
- Rodríguez-Pose, A., von Berlepsch, V., 2014, When Migrants Rule: The Legacy of Mass Migration on Economic Development in the United States. *Annals of the Association of American Geographers*, v. 104, pp. 628-651.

- Roth, R. E. (2017). User Interface and User Experience (UI/UX) Design. *The Geographic Information Science & Technology Body of Knowledge*, (2nd Quarter).
- Shneiderman, B., 2003, The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations, *Institute for Systems Research*, T.R. 96-66.
- Sinar, E., 2015, Big Data at Work: The Data Science Revolution and Organizational Psychology, *Taylor and Francis*, p. 117-125.
- Stol, K., Ali Babar, M, 2010, Challenges in using open source software in product development: a review of the literature, *FLOSS '10: Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, pp. 17-22.
- Swift, J., 2020, JavaScript for GIS, *The Geographic Information Science & Technology Body of Knowledge*, 3rd Quarter.
- Teralytics, 2022, flowmap.gl, <https://github.com/teralytics/flowmap.gl>.
- Tobler, W., 1987, Experiments in migration mapping by computer, *The American Cartographer*, v. 14, pp. 155-163.
- Tulsa Remote, 2021, Benefits, *Tulsa Remote*,
https://tulsaremote.com/?utm_source=newsletter&utm_medium=email&utm_campaign=newsletter_axioscities&stream=cities.
- U.S. Census Bureau, 2016, *Measuring America: Our Changing Landscape, 1910-2010 US Censuses*, retrieved from
<https://www.census.gov/library/visualizations/2016/comm/acs-rural-urban.html>.
- Veenendaal, B., Brovelli, M.A., Songnian, L., 2017, Review of Web Mapping: Eras, Trends and Directions, *International Journal of Geo-Information*, v. 6, pp. 1-31.
- Wu, F., Lin, H., You, M., 2011, Direct-touch vs. mouse input for navigation modes of the web map, *Displays*, v. 32, pp. 261-267.
- You, M., Chen, C., Hantsai, L., Hsuan, L., 2007, A Usability Evaluation of Web Map Zoom and Pan Functions, *International Journal of Design*, v. 1, pp. 15-25.

- Zeleti, F., Ojo, A., Curry, E., 2014, Emerging business models for the open data industry: characterization and analysis, *Proceedings of the 15th Annual International Conference on Digital Government Research*, pp. 215-226.
- Zhu, X., Guo, D., 2014, Mapping Large Spatial Flows Data with Hierarchical Clustering, *Transaction in GIS*, v. 18, pp. 421-435.
- Zhu, X., Guo, D., Koylu, C., Chen, C., 2019, Density-based multi-scale flow mapping and generalization, *Computers, Environment and Urban Systems*, v. 77, pp. 1-10.

Appendix

All the code used to create this application is available to view on my GitHub account:

<https://github.com/jh-gis/flowmap.gl-example>