SCALABLE DYNAMIC BIG DATA GEOVISUALIZATION WITH SPATIAL DATA STRUCTURE

by

Siqi Gu

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the Degree of

Master of Science

Department of Computer and Information Technology

West Lafayette, Indiana

May 2020

THE PURDUE UNIVERSITY GRADUATE SCHOOL

STATEMENT OF COMMITTEE APPROVAL

Prof. Baijian Yang, Chair

Department of Computer and Information Technology

Prof. Dominic Kao

Department of Computer and Information Technology

Prof. Yingjie Chen

Department of Computer and Graphics Technology

Approved by:

Dr. Eric T. Matson

Head of the Graduate Program

Dedicated to my grandfather who made me love new technology.

ACKNOWLEDGMENTS

I sincerely acknowledge my thesis committee for the profound guidance and visionary suggestions and my family and roommate for providing me with life and spiritual support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

abbr	abbreviation
API	Application programming interface
CGT	Computer Graphics Technology
CIT	Computer and Information Technology
CoT	College of Technology
DOM	Document Object Model
GB	Giga Byte
GIS	Geographic Information System
HTML	HyperText Markup Language
JS	Javascript
MB	Mega Byte
ms	millisecond
PB	Peta Byte
SDS	Spatial data structure
SQL	Structured Query Language
TB	Tera Byte

GLOSSARY

Big Data – it refers to the term that large or complex data sets are not enough to process in traditional data processing application software. Big data can also be defined as a large amount of unstructured or structured data from various sources.

Geovisualization – short for geographic visualization. Refers to a series of methods for analyzing and visualizing geospatial data.Compared with the visualization of ordinary data, it is more challenging to convert information with geographic location into a graphical form that is easy to understand.

Spatial data structure – It refers to the logical structure of spatial data suitable for computer storage, management and processing, and is the organization and coding form of spatial data in the computer.

ABSTRACT

Comparing to traditional cartography, big data geographic information processing is not a simple task at all, it requires special methods and methods. When existing geovisualization systems face millions of data, the zoom function and the dynamical data adding function usually cannot be satisfied at the same time. This research classify the existing methods of geovisualization, then analyze its functions and bottlenecks, analyze its applicability in the big data environment, and proposes a method that combines spatial data structure and iterative calculation on demand. It also proves that this method can effectively balance the performance of scaling and new data, and it is significantly better than the existing library in the time consumption of new data and scaling.

CHAPTER 1. INTRODUCTION

1.1 Background

With the advent of the Big Data era, traditional statistical charts are difficult to visualize complex data. Data visualization has become more and more popular as a new research area in recent years. Successful visualization, if done beautifully, is simple but rich in meaning, allowing observers to gain insight and produce new understanding at a glance (Gorodov & Gubarev, 2013). Among them, geovisualization is the last mile of geographic big data applications, covering different scales, ranging from small houses to large amounts of global landscape data. In essence, geovisualization develops human spatial thinking capabilities, making it easier for people to discover complex relation-ships hidden behind spatial locations, providing a clear under-standing of hidden phenomena and shortening search time.

When it comes to map expressions, it is natural to think of cartography. In fact, cartography as a form of expression of a map is similar to the visualization of geographic information. The difference between them is very sensuous and subtle: geovisualization integrates data visualization, cartography, image analysis, exploratory data analysis, and visual analysis (Andrienko, Andrienko, & Gatalsky, 2003), the results of which should guide and ultimately provide insights that help assist decision making. The nuances of the two are not in the representation of the map language, but in the value orientation of the final result.

1.2 Problem

Comparing to traditional cartography, big data geographic information processing is not a simple task at all, it requires special methods and methods. Graphic thinking is a very simple and natural way of data processing. Therefore, it can be said that image data representation is an effective method to simplify data understanding and provide sufficient support for decision making. However, for big data geographic information, most classical data representation methods become inefficient or unsuitable for specific tasks.

In this case, a large problem is that it is impossible to dynamically render millions of data points on a single image (Gorodov & Gubarev, 2013). Usually we can reduce the magnitude of the data set by preprocessing. However, when we are dealing with geographic data, we meet a new requirement that geographic data is scalable. We may want to observe the same data set on different scales. It's hard and uneconomic to preprocess and store the data in every corner and every scaling ratio, especially when the data is dynamically changing in each second.

1.3 Research Question

Can the proposed method with SDS and On-demand loading able to improve the time consumption when dealing with highly dynamic big datasets with scaling function against existing libraries like D3.js or Plotly?

1.4 Scope

This question is important to those national information management departments or multinational corporations. In many cases, we need to monitor the data change trend both in a nationwide level and a specific single point. For example, we may want to see whether traffic violation has some rule in state level, and when we find the overall rule, we may want to find out how exactly is a hot point composed. It's quite useful to offer such a kind of scalable geographic bigdata visualization.

Therefore, the purpose of this study is to research and summarize the existing algorithms and practices to develop and optimize a new geographic information visualization method. This method should be able to perform high-frequency dynamic updates in a big data environment, and there should not be too much stuck when zooming, and at the same time provide a smooth and inspiring browsing experience at a different zoom ratio with a reasonable memory usage. This method should be able to run on the current general business environment through simple deployment and provide a set of APIs for users to configure and use in the form of external libraries.

1.5 Assumptions

- This study assumes that the zoom ratio between the maximum scale and the minimum scale required by the user exceeds 16 times.
- This study assumes that users need to visualize data with only zero or one continuous or discrete data dimension in addition to geographic information. That is, this study does not consider the problem of visualization switching between different attributes of the data.
- This study assumes that there is no bottleneck in the computation, processing, and rendering required for visualization in terms of memory, storage, and network latency. That is, it is assumed that the computer or array used for visualization always has enough memory, storage, and network bandwidth is large enough for processing power, and network latency is negligible for processing time.
- This study does not consider the processing of complex source data. It is assumed that the user has decided which data items and attributes need to be visualized and extracted them.

1.6 Limitations

- Most of the commonly used data visualization libraries are now open source, but some commercial software itself is not open source. Although there are usually articles describing their internal principles, they are generally unclear. Therefore, for this part of the software that is not open source, only some phenomenological comparisons of external performance can be made, and it is impossible to study the algorithms that may be used.
- Due to the limitation of the experimental environment, the performance of a single computer used in this study will not exceed the common performance of small servers, and the number of computers in the array used will not exceed single digits.
- Actual commercial raw data is usually not disclosed for reasons such as privacy. The datasets available on the network are generally not too dense in time. New highly dynamic information may be tested only through simulated data.

1.7 Delimitations

- The proposed visualization method may not support too many data types for the time being.
- The proposed visualization method will not use too complicated map backgrounds, such as roads, rivers and other information, because reading and visualizing such background information may itself take a lot of time. Research will focus on the visualization of the data points themselves.
- The proposed method supports only one visualization platform.

CHAPTER 2. REVIEW OF LITERATURE

2.1 Typical representations of geovisualization

To be intuitive and impressive to show the spatial distribution of geographic information data, you can use dot density map, category map, choropleth map, proportional symbol map, blending map, heat map, binning map, edge bundled map, and other methods to render the data.

2.1.1 Dot Density Map

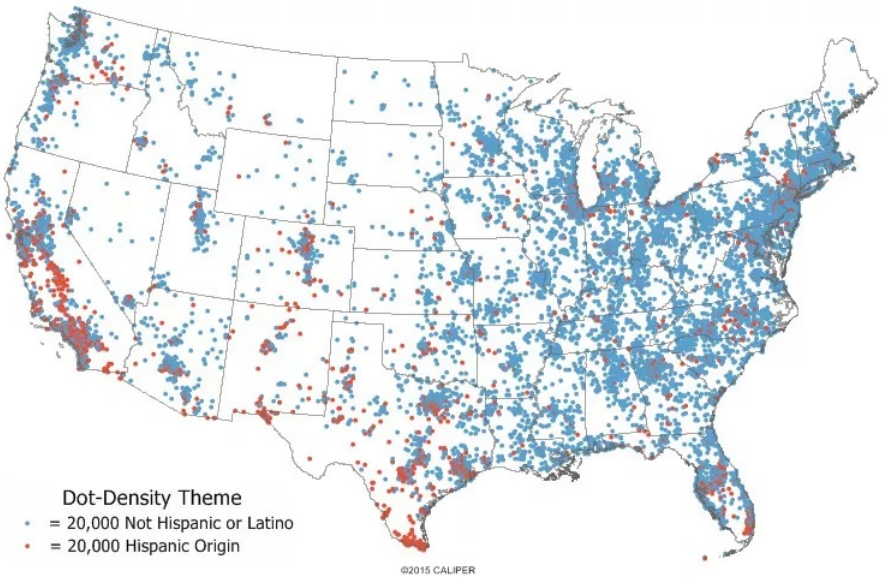


Figure 2.1. The distribution of the Hispanic and Non-Hispanic population

Source: *What is a dot-density map?* (n.d.). Retrieved November, 2019 from: `https://www.caliper.com/glossary/what-is-a-dot-density-map.htm`. (Online)

Map a dot density map is the uniform rendering of data into the same color, shape, or size. Dot density maps are simple and intuitive and can show information about the spatial distribution of some locations.

A dot density map is the simplest way to visualize geographic information, and its data dots only indicate that there is one data at a certain location. If the data itself is Boolean, a dot density map can well represent the existence of this data point. For example, in Figure 2.1, the author set 20,000 people with a point that visually shows distribution of the Hispanic and Non-Hispanic population in United States.

The problem is that a single data point needs to be properly sized. In areas where data points are dense, data points fill the entire area, and the data point density exceeds the upper limit that can be resolved on the graph. No more details can be given when there are too many data points or too dense at some locations.

2.1.2 Category map

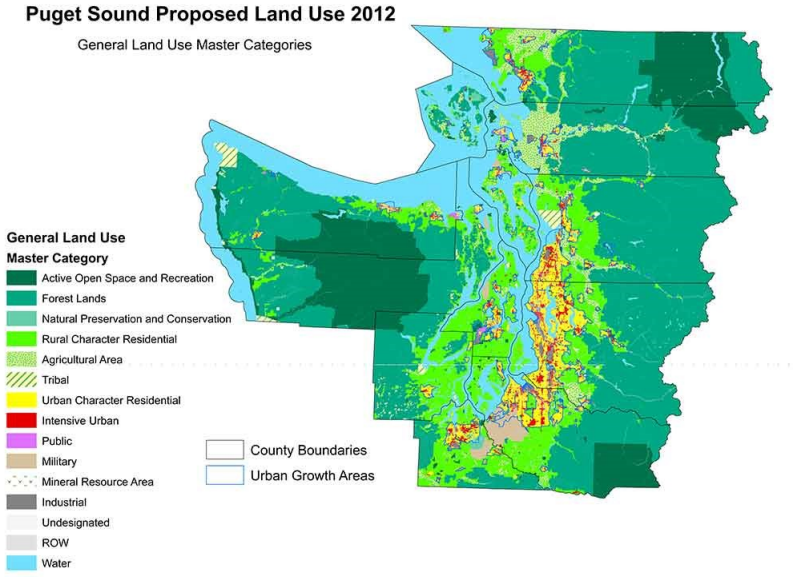


Figure 2.2. Puget Sound Proposed Land Use 2012

Source: *Puget Sound Mapping Project* (2012). Retrieved November, 2019 from: `https://www.commerce.wa.gov/serving-communities/growth-management/puget-sound-mapping-project/`. (Online)

A category map is a rendering of different colors according to different categories or rendering using different shapes and sizes.

For example, in Figure 2.2, The authors used different colors to indicate the different types of land that Washington State was planning.

However, if the size of the different categories of areas differs greatly, or the number of types of categories are large, the category map will face many problems.

As the amount of information increases, it is desirable to increase the resolution of the category map accordingly to accommodate important concepts in space. This leads to an increase in the visual load of the category map. In a limited display window, a large amount of information is closely clustered, and it is difficult to see the local details clearly in the window (Yang, Chen, & Hong, 2003).

2.1.3 Choropleth Map

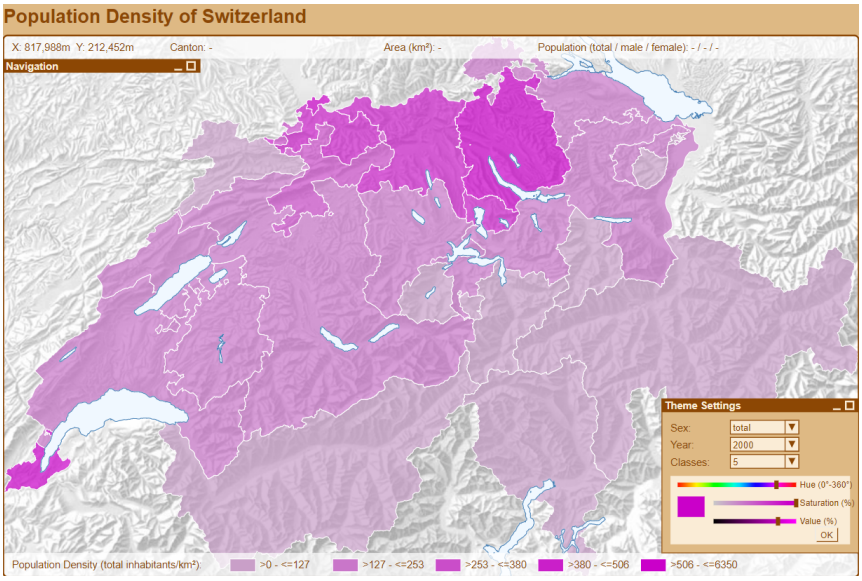


Figure 2.3. Population Density of Switzerland

Source: Schnabel, O. (2008) *Presentation of Thematic Data*. Retrieved November, 2019 from: `http://www.carto.net/schnabel/pop_dens/`. (Online)

A Choropleth Map, which assigns different colors according to the number of one field. For example, a color greater than zero is assigned to a color, and a color greater than one hundred and twenty-eight is assigned to a color, and different intervals are given different colors.

There are usually four progressive segmentation methods of equal interval, equal quantity, natural segmentation and equal standard deviation.

Category map and Choropleth Map have similarities. If the data is quantative and the data span is very large, such as from 1 to 10000, and there are decimals in it, try to use a choropleth graph; if the data volume is a character-based categories, then you can consider using a classification graph.

In Choropleth Map, defined regions are important to a discussion. The size and specificity of the displayed regions should depend on the variable being represented. Problems such as ecological fallacy and modifiable area unit problems (MAUP) can lead to major misunderstandings in situations where the real world partition may not conform to the desired pattern, so other technologies are preferred (Howard, McMaster, Slocum, & Kessler, 2008). While using smaller and more specific areas can reduce the risk of ecological fallacy and MAUP, it can make the map look more complicated. Although representing specific data in large areas can be misleading, it can make the map clearer and easier to interpret and remember.

2.1.4 Proportional Symbol Map

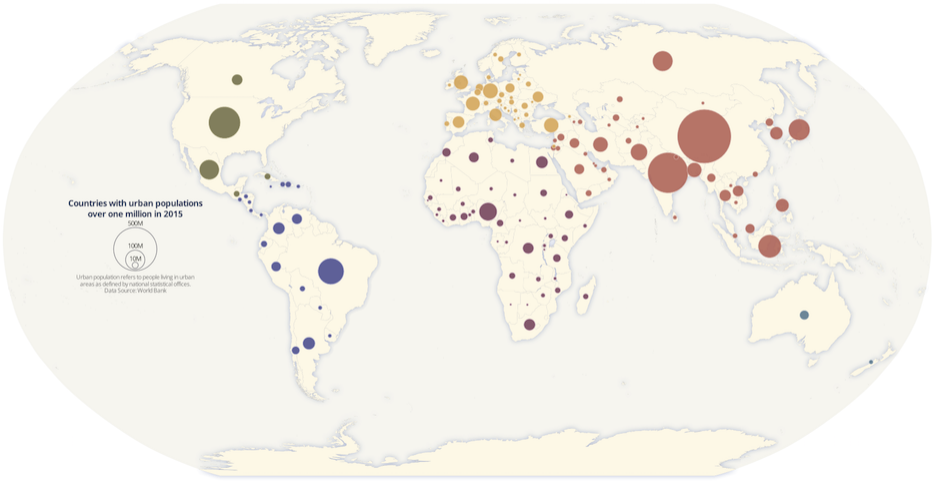


Figure 2.4. 2015 Urban Populations

Source: Akella, M. (2016). *Visualize 2015 Urban Populations with Proportional Symbols*. Retrieved November, 2019 from: `https://carto.com/blog/proportional-symbol-maps`. (Online)

A scale symbol diagram is a thematic map that uses graph symbols of different sizes to represent quantitative variables. The height, length, area or volume of a symbol varies from location to location, depending on the variables they represent.

There are two ways to create a proportional symbol: absolute scaling and range scaling. When using absolute scaling, the area of each symbol on the map is scaled by its value in the data. Through range grading, you can use the classification method to divide the value into multiple ranges, where the size of the symbol is determined according to the range in which the symbol falls (Dent, Torguson, & Hodler, 1999).

One problem is that when partitions are dense and there are more data points, using symbols of different sizes can be very messy. At the same time, there is often overlap between symbols, which makes some symbols difficult to identify. It is possible to reduce the sign size to avoid overlap, but this will reduce the degree of discrimination between different size symbols.

2.2 Modern Improvement

With the requirements of modern data visualization, in order to better display the laws of data clearly and effectively in the case of large amounts of data, some new technologies have been developed to improve the effect of traditional methods. Most of these methods are basically the same as the traditional methods, but they achieve better display results through some optimizations of selection and rendering. Compared with traditional methods, this method is more beautiful and can carry a larger amount of data, which is impressive. Many of its ideas are worth learning from. Here are some popular visualizations that work well with large amounts of data.

2.2.1 Blending Mode

Figure 2.5. Facebook Global User Relationship Map

Source: Nowak, M., & Spiller, G. (2017). *Two Billion People Coming Together on Facebook*. Retrieved November, 2019 from: `https://newsroom.fb.com/news/2017/06/two-billion-people-coming-together-on-facebook/`. (Online)

Blending Mode is a kind of modern rendering method. One of the most famous examples is the Facebook global user relationship diagram above. The color of this picture is blue. In many places, such as the United States and Europe, the color is very bright.

Blending mode does not create a new graphical layout, but rather a rendering mode that is very efficient when the data is large. The single-valued data is converted into brightness by blending, which greatly improves the performance of single-valued images in dense situations.

$$\text{Normal: } f(a,b) = b \tag{2.1}$$

$$\text{Multiply: } f(a,b) = a \times b \tag{2.2}$$

$$\text{Screen: } f(a,b) = 1 - (1-a)(1-b) \tag{2.3}$$

Here are some common ways to blend. Equalation (2.1) is Normal, which is use two lines or two points as the input, output the color of the point or line at above. To achieve a high-brightness effect, you can use (2.2) or (2.3) to do the calculations. For example, using (2.2) , a and b are multiplied to make an output, and then rendered. The overlapping area will be highlighted a lot.

2.2.2 Heat Map

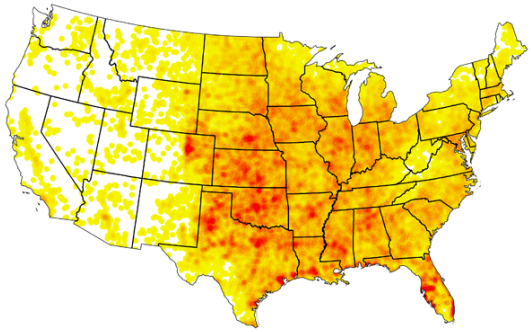


Figure 2.6. A heat map of tornado locations from 1950 to the present

Source: Understanding the Heat Map (2015). Retrieved November, 2019 from: `https://cartographicperspectives.org/index.php/journal/article/view/cp80-deboer/1420`. (Online)

The heat map is to render the data through the gradient of the color, so that the user can see the association between the data sets at a glance. The higher the color temperature, the denser the data points.

One of the biggest functions is to highlight the degree to which a point gathers. The more points gather, the higher the color temperature.

To some extent, the heat map is similar to the Choropleth Map when areas are relatively small in Choropleth Map. The heat map represents the density using a relatively red-to-blue hue representing the heat. This approach is more clear than the traditional Choropleth Map.

Figure 2.6 shows an example of a heat map depicting the prevalence of tornadoes. Google's geo developer blog (Yeap & Uy, 2014) describes these maps as "geospatial data on a map by using different colors to represent areas with different concentrations of points — showing overall shape and concentration trends".

The modern heat map acquires discontinuous point data and displays it as continuous. This method does not apply to all data. While it makes sense to map altitude or temperature to a continuous surface, data that does not continuously change with space may not. In addition, too few points on the surface usually result in large errors.

2.2.3 Binning Map

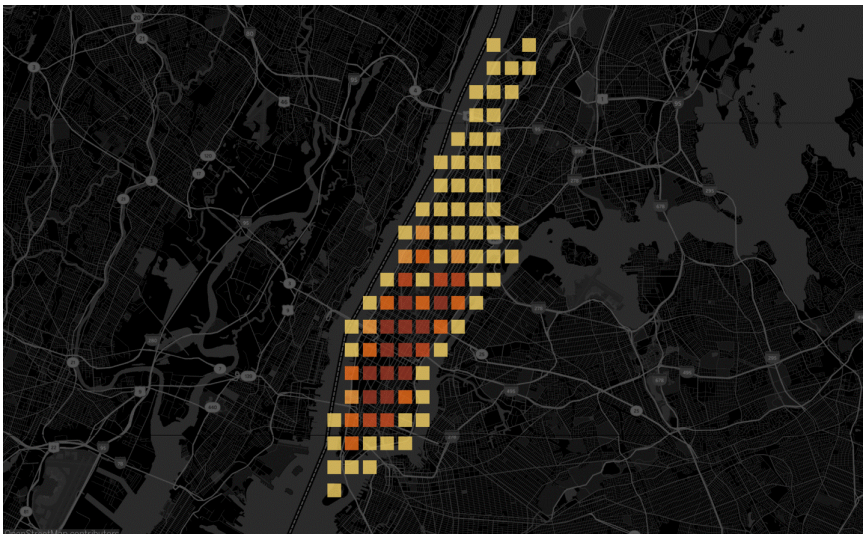


Figure 2.7. Taxi cab pick-up locations in Manhattan

Source: *Taxi cab pick-up locations in Manhattan.* (2015). Retrieved November, 2019 from: `https://www.tableau.com/about/blog/2017/11/data-map-discovery-78603`. (Online)

Dots distribution maps are a good way to understand the spatial distribution pattern of data. In order to figure out these types of patterns, you need to leave enough space between the various dot markers to clearly see the starting position of one data cluster and the location where another data is terminated. When you have a lot of data and don't even see the map, people choose to use the binning map (Battersby, 2017).

These points are spatially aggregated into a polygonal area to view the data set instead of a single point. The binning map is also very similar to the Choropleth Map. Just its area is manually divided. When we have more geographic information details of statistics than traditional regional statistics, we can have more choices in how to express.

2.2.4 Edge Bundled

Figure 2.8. US migration graphs

Source: Ersoy, Hurter, Paulovich, Cantareiro, and Telea (2011)

The edge that can encode the relational data in the map is an important visual primitive for encoding the data in the information visualization study. However, when data becomes very large, visualization often suffers from visual clutter, as thousands of edges can easily overwhelm the display and mask the underlying pattern (Zhou, Xu, Yuan, & Qu, 2013). A number of edge bundling techniques have been proposed to reduce visual clutter in visualization.

Some methods include edge bundling and visual clustering (Zhou, Yuan, Qu, Cui, & Chen, 2008) algorithms using force-directed methods (Zhou, Yuan, Cui, Qu, & Chen, 2008), Geometry-based (Cui, Zhou, Qu, Wong, & Li, 2008) edge bundling algorithm and winding road method (Lambert, Bourqui, & Auber, 2010). The selection of specific methods is not fixed and needs to be considered in combination with specific problems and actual effects.

2.3 Geovisualization Problems in Big Data Set

The so-called BigData can be understood as a huge data set, and its capacity grows exponentially. For traditional visualization methods, the data set may be too large, too ”raw” or too unstructured (Gorodov & Gubarev, 2013). This makes traditional mapping methods likely to face many problems.

2.3.1 Visual Noise

Even the simplest data point representation can get a mess on the screen.

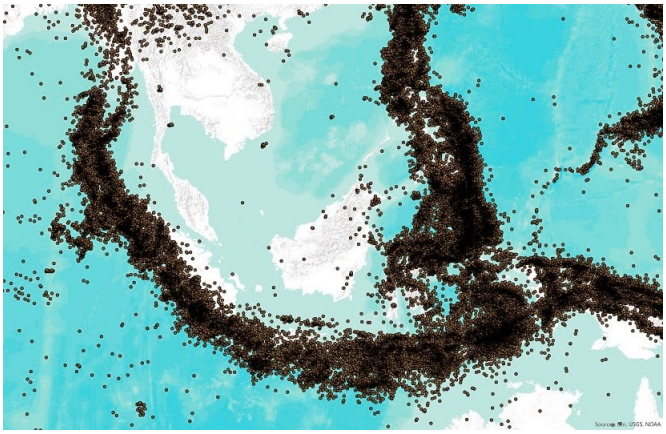


Figure 2.9. Earthquake Density

Source: Lehner, G. (2019). *Introduction to Dynamic Feature Binning in ArcGIS Pro*, Retrieved November, 2019 from: `https://www.esri.com/arcgis-blog/products/arcgis-pro/mapping/introduction-to-dynamic-feature-binning-in-arcgis-pro/`

Most of the objects in the dataset are too close to each other and cannot be divided into separate objects on the screen monitor. In Figure 2.9, The high number of point features make it difficult to determine earthquake density in certain locations (Lehner, 2019). Without any pre-processing tasks, analysts can't get a bit of useful information from the entire data visualization.

2.3.2 Large Image Perception

One of the ways to solve the last problem is to use the largest possible screen for distribution. But even if the screen is large enough, it still faces the limitation of human perception. After reaching this level of perception, humans simply lose the ability to obtain any useful information from the perspective of data overload. With the rapid growth of data, it will be difficult for humans to understand the data and its analysis (Gorodov & Gubarev, 2013).

2.3.3 Information Loss

The visible data set can be reduced by some of the aggregation methods described earlier. Although the above problems are solved, these methods lead to another problem, that is, information loss. These methods aggregate data in some geographic or other relevance.

But it also caused some loss of information that was not noticed during the aggregation. When analysts are unable to notice some interesting hidden objects, using these methods can mislead analysts, and sometimes complex aggregation processes can consume a lot of time and performance resources.

2.3.4 High Performance Requirements

Geovisualization is not limited to static geographical data set, so the above issues become more important in dynamic visualization. For example, the review site Yelp receives dozens of reviews every second during peak hours. Analysts may want to observe trends in real-time data to understand user preferences. But adding data to a huge data set that may have been subjected to complex preprocessing can be a huge challenge.

2.3.5 High Rate of Image Change

The display of high-frequency changing data can also pose challenges. Many geographic visualization methods require a certain amount of calculation with correlation between data, and even a certain correlation in the graph as a whole. High-frequency changing data will cause the image to be frequently re-rendered. At the same time, people's response speed to high-frequency changes is also limited.

As mentioned above, many real-time dataset changes can be very rapid. When the person observing the data is unable to respond to the amount of data changes or the intensity of the display, the benefits of real-time data are greatly offset.

2.3.6 Large Scale Zoom

For geographic big data, a natural requirement is to observe local data when needed. However, since the overall visualization is pre-aggregated, extracting local information is not a simple amplification, and may require partial image generation. Moreover, due to the amount of data of local data, the distribution pattern, etc. may be different from the whole, and it may be encountered if the same visualization method is adopted.

Overall, the increase in data volume and rapid updates have made it difficult to visualize analysis.

2.4 Some Inspiring Visualization Approaches

2.4.1 Customizing Computational Methods

Choo and Park (2013) discussed the interaction between precision and convergence in " Customizing computational methods for visual analytics with big data ".

The computational time required hinders real-time interactive visualization of big data.

In order to solve this problem, the author proposed Customizing Computational Methods. Reducing the precision of calculation was the easiest way to increase the speed of computation. Thus, the accuracy of the calculation could be determined more carefully based on human perception and screen resolution.

Another option was iterative level interactive visualization. Its purpose was to visualize intermediate results in various iterations and let users interact with these results in real time. If the user zoomed in on a particular area, the 2D coordinate information of the data item in that area must be represented in a finer granularity. It was possible to iteratively improve the accuracy of the calculation results.

2.4.2 Scalable Computing

Beyer et al. (2013) described a system for interactively exploring PB-level volume data of neural tissue generated by high-throughput electron microscopy in ” Exploring the connectome: Petascale volume visualization of microscopy data streams.” The visually driven system allowed users to process multiple volumes and incomplete data, limit most calculations to a small portion of the data, and use multi-resolution virtual memory for scalable computing.

By applying similar techniques, it was possible to distribute the rendering of geographic images on different hosts. Pre-processing was performed in advance during the data collection process to reduce the performance pressure of the host used by the terminal for display.

2.4.3 Distributed Parallel Processing

Kim, Ji, and Park (2014) proposed a cloud based on visualization methods to visualize the inherent relationships of users on social networks. The method used a correlation matrix to represent the hierarchical relationship of social network user nodes. This approach used cloud-based Hadoop for distributed parallel processing of visualization algorithms that could accelerate big data on social networks.

2.5 Spatial Database

Spatial database is such a database. It provides a spatial data type in its data model and query language, supports spatial data type in its execution, and at least provides spatial data indexing and storage functions.(Güting, 1994) Scalable geographic data visualization necessarily requires the use of spatial databases as the basis, because only spatial databases can efficiently access data points within a specified geographic area.

In the spatial database, the spatial data structure has many different options according to different needs.

2.5.1 Hierachical Structure

Hierarchical structure is a tree structure. It mainly expresses the hierarchical relationship between data elements, which is usually a one-to-many relationship. The data in each layer is related to multiple data elements in the next layer. There is only one root node in the tree structure called the master, and the remaining nodes are called members, and each node is also the master of the next node, such as the classic quadtree and R-tree.

A quadtree is a tree data structure with four sub-blocks on each node. The quadtree is often used for the analysis and classification of two-dimensional spatial data. It divides the data into four quadrants. The data range can be square or rectangular or any other shape. This data structure was developed by Raphael Finkel and J. L. Bentley in 1974.

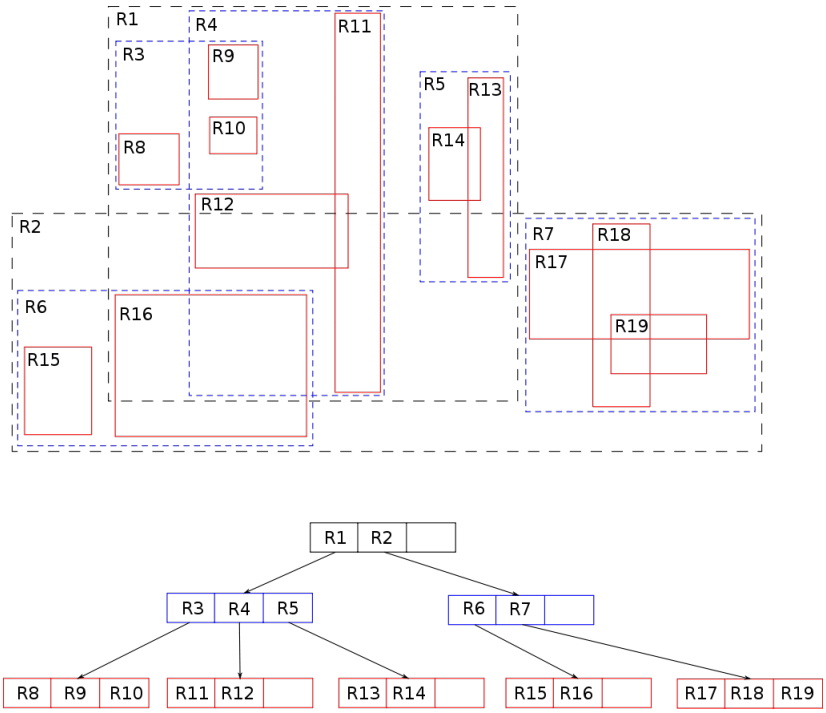


Figure 2.10. R-tree example

Source: Skinkie, Radim B. (2010). *An example of a simple R tree on a two-dimensional rectangle*, Retrieved March, 2020 from: `https://commons.wikimedia.org/wiki/File:R-tree.svg`

R-tree is a tree data structure used for spatial data storage. For example, index multi-dimensional data such as geographic location, rectangle and polygon. The R tree was proposed by Antonin Guttman in 1984.

The core idea of R-tree is to aggregate nodes with short distances and express it as the minimum circumscribed rectangle of these nodes in the upper layer of the tree structure. This minimum circumscribed rectangle becomes a node in the upper layer. The ”R” in the R tree stands for ”Rectangle”.

R-tree does not guarantee worst-case performance, but it generally performs well on real data.

2.6 Aims and Objectives

In the face of highly dynamic big data, traditional visualization methods are overwhelmed, both in terms of expressiveness and performance. Although many people have developed visualization solutions for common large data sets, these visualization solutions also face new challenges when facing geographic big data.We may want to observe the same data set at different scales. This scope of observation may include a span from global to one block.

In the above, we introduced a lot of existing visualization methods for large data sets. Some of them have been optimized for terabytes and even petabytes of data sets. However, they are either poorly dynamic or require significant recalculation once scaled. So far, no one has provided a universal and effective geographic information visualization solution that is scalable in a highly dynamic big data environment.

So, my research question is that, Can the proposed method with SDS and On-demand loading able to improve the time consumption when dealing with highly dynamic big datasets with scaling function against existing libraries like D3.js or Plotly?

CHAPTER 3. RESEARCH METHODOLOGY

3.1 Overall Approach

The main goal of this research is to propose a new method of geographic data visualization based on existing methods, which should support the addition of highly dynamic data in real time on the basis of good zooming effects. Then prove that this method has less time and space consumption than other existing methods. In other words, faster calculation speed and less memory consumption.

3.1.1 Summarize existing methods

First, I need to generalize and summarize the existing visualization methods. The research object includes not only the geovisualization methods, but also other common visualization methods, so it is expected to find out the possibilities that can be used to improve and implement the dynamic scalable geovisualization.This part mainly includes a qualitative analysis of existing methods and their applicability to the expected high dynamic big data.Of course, none of the existing tools and solutions can deal well with highly dynamic geographic big data. Therefore, an important task is to find the bottleneck of the methods. Some quantitative methods may be used here, such as the measurement of effective cpu time and the measurement of peak memory usage.

3.1.2 Analysis of actual needs

On the other hand, we need to study the different needs of geographic visualization at different scales. One of the most common visualization schemes that is most easily applicable to large data sets is heat maps. But the nature of the heat map itself determines that it does not naturally support scaling. Unlike general data visualization, geographic visualization has more practical significance in spatial distribution. Maybe we need to visualize the data in different proportions in different modes. For example, we may want to use a large-scale heat map and a small-scale scatter plot. For some methods, this switch may not be supported, or due to the underlying design, this switch may consume a lot of resources. Through the operation of the underlying logic and the combination of different methods, these problems can be solved.

3.1.3 Find a solution

After research, it entered the development stage. The development phase mainly includes the selection of development tools and the final actual development. Due to the low-level rewriting of the visualization logic, the development tools do not use the existing common visualization tools directly, but still need to learn from the algorithms and development logic of some of the open source tools. During the development process, many different solutions may be generated, and their performance and effects should be quantified and compared. In this step, the performance of the newly developed solution should be compared with existing methods and existing commercial tools in the same environment configuration to find a research direction that can be further optimized. This step will require a lot of actual testing and quantitative analysis. Finally, I hope to find feasible and effective solutions through these studies.

3.2 Practical Visualization Tools

Many big data visualization tools run on the Hadoop platform. Software with visual and interactive capabilities to visualize data has been developed. Although distributed computing has its technical advantages in terms of processing speed for big data, it is inevitable that a finalized output is necessary because of data visualization. Since general data visualization methods such as heat maps are localized and do not have long-distance associations, distributed computing will not bring about a fundamental change in principle in visualization itself, but only for computing speed. Therefore, although many big data visualization tools support distributed technology, in this research, its distributed technology will not be the main research object. All principle research and actual testing will be performed on a single computer. It is expected that a visualization solution that has advantages on a single computer will also gain advantages in distributed computing.

It is worth noting that the previous survey found that a large number of visualization tools are more focused on less large data sets and richer visualization effects. These tools are actually more aimed at small and medium users and designers, rather than enterprise data analysis departments that really deal with big data. Rich and diverse visualization effects and efficient algorithms to deal with big data may be doomed to no simple implementation, after all, different visualization effects will correspond to very different preprocessing and rendering logic, and for tools that pursue universality, implementing these functions is not a priority.

In addition, many visualization tools now choose the web front end as the platform. This is due to the rich content advantages and ease of use of the web. But the web front end is a client-based rendering system, and the client cannot always be expected to have the performance required to process big data. The web itself is not a platform that focuses on performance. Although many modern technologies such as WebAssembly and Shadow Dom have accelerated the javascript operation and dom rendering speed to a level close to native software, its dom tree-based rendering system still brings image rendering itself. Here comes some unnecessary overhead.

However, not many tools can meet the dynamic and scalable geographic information visualization needs. Some visualization tools (Sucharitha, Subash, & Prakash, 2014) may be useful.

Dygraphs is a fast, flexible collection of open source JavaScript diagrams that help discover and understand opaque data sets.

Tableau has three main products for processing large data sets and also embeds the Hadoop infrastructure.

Rave is a fast adaptive visualization engine that IBM is developing. IBM has embedded visualization capabilities into its business analytics solutions. RAVE and scalable visualization capabilities help with effective visualization.

Other tools include the Google Chart API, Flot, D3, and Visual.ly etc. These tools are often more flexible and provide a more customized visualization method to apply specific optimizations for specific problems. But these tools will require more programming work which need to be done by professionals.

Considering that in actual production, the data source sometimes comes from different servers, or even has geographical distribution characteristics itself, distributed processing may be very effective and necessary. Hadoop platform may be a good choice. However, distributed design will not be adopted in this project.

3.3 Program Structure

3.3.1 Front-end

The structure of the visualization program mainly includes three major parts, which are front-end rendering, data processing, and database. Among them, the choice of visualization tools will greatly affect the efficiency of front-end rendering. For example, if I add a Dom element to the front end for each additional data point, the front end needs a lot of time to restructure the Dom tree.If you use canvas based on OpenGL, you can get the advantages of graphics card acceleration. In addition, whether the front end needs to follow the data update completely in real time, how to perform partial updates, whether it needs to accumulate updates, and how to control the refresh rate need to be considered. The front end also needs to assume many control functions. Such as accepting external input to zoom and move the image. On some web-like platforms, these operations are well-defined in advance. But if on other platforms, these operations may not be well defined. I think this should not be the main direction of this research. These interactions are not the reason for limiting the performance of big data visualization.

In summary, if local software is used, the most direct and simplest output method is image sequence output. This is the most direct output method at the bottom, and the program outputs the image sequence directly to the display. However, this output method does not naturally have the function of accepting operation input, which requires pre-programming control. If you use a web platform, the most direct and efficient is to use canvas.

3.3.2 Database

For databases, geographic information naturally has a two-dimensional spatial data structure, so this also limits its query and calculation methods. Under this two-dimensional spatial data structure, the traditional entry-based SQL database cannot provide a sufficiently effective data retrieval speed. For the rendering of a single image, this may not be an important issue. Even if all data points are irregularly distributed within the range, the rendering order will not affect the rendering time of a single image. However, once scaling is involved, the situation is completely different. When zooming, the program needs to retrieve the data points in the area, and then use the data in the area to re-render the calculation. In theory, data in the same area can be retrieved from adjacent locations through preprocessing to speed up retrieval. However, another problem faced in this study is that the situation assumed in this study is that a large amount of dynamic data is stored. Of course, the stored dynamic data will not have spatial regularity. This makes it impossible to pre-process in advance, because newly entered data cannot be pre-processed. Therefore, the linear retrieval time means that every time you zoom, regardless of the size of the area, you need to traverse all the data. This makes the choice of data storage method a very important difficulty. I may need to find a new space-based data storage structure, such as a quadtree or R tree, and this data structure also needs to be able to store intermediate results to a certain extent, so that different scales of access are fast enough.

In summary, as an important key point, a spatial database needs to be found. Theoretically, through the reasonable use of spatial database, the range access to spatial data points can reach O(log (n)) complexity. The specific design principles are discussed below.

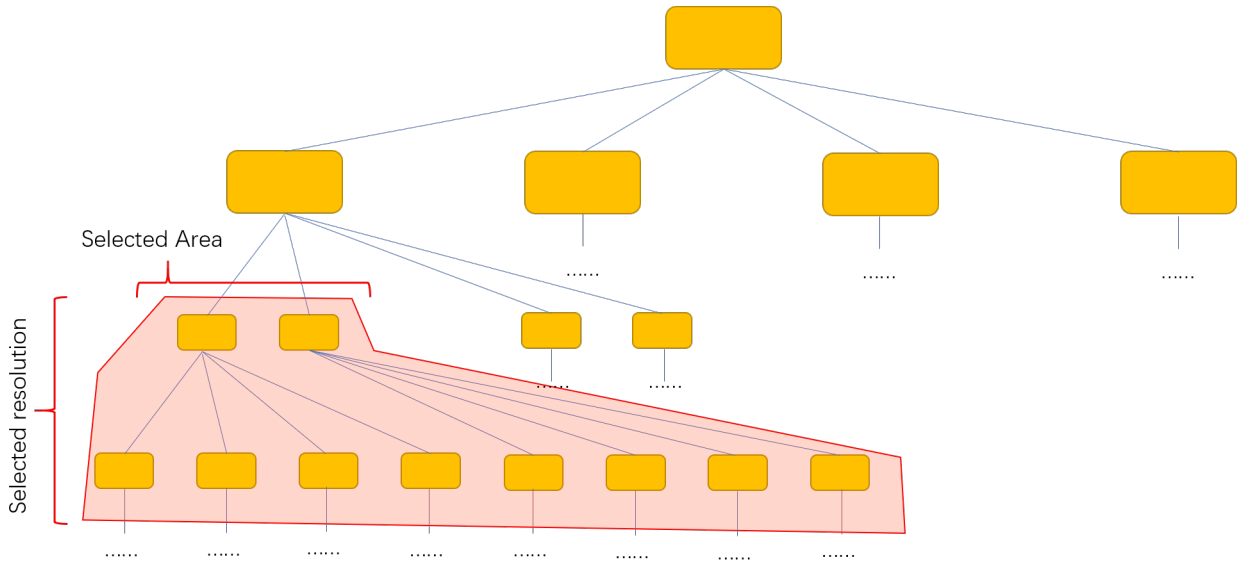


Figure 3.1. Database design, taking quadtree as an example

Take the quadtree as an example. A spatial database structure that can quickly support scaling should be a multi-level hierarchical quadtree. If you want to read all the data in a certain area, it is actually equivalent to selecting one or several subtrees from the quadtree. The data under this subtree is the data we need. On the other hand, because the resolution of the output image is limited, in fact, we do not need to read the entire subtree to the bottom. We store a sum of all the data under it at each level of the subtree. When the size of the area represented by a subtree is already less than the resolution, we don't need to read the lower part, we just need to read the value stored by itself. In this way, for a quadtree, only 10 layers can be read to support a 1024x1024 image output. The number of layers to be read is determined according to different resolution requirements. But for the same resolution, this means that a larger scale will not slow down the reading of data, because we only need to read ten layers of data, instead of reading all specific data points in the area. For determining the resolution, regardless of the scale of the zoom, no matter what area is selected, the time to read the data is basically fixed. This can solve the problem that more data points are read more slowly under normal circumstances.

Some other tree structures are mentioned in the previous literature review. The R tree and its deformations are the most widely used data structures for spatial databases. It is a balanced tree. Compared with the quadtree, its biggest advantage is that the area of its subtrees is not fixed, but the area The density of the interior points varies. The denser the data, the more bifurcation of the R tree. This means that the space utilization efficiency of R is higher, and the worst query time of the R tree is better when targeting uneven data. But this also means that the area divided by the R tree is also uneven. This will slightly increase the linear coefficient of the query time when the number is small. In the worst case, space utilization efficiency and access time will be fairly well guaranteed.

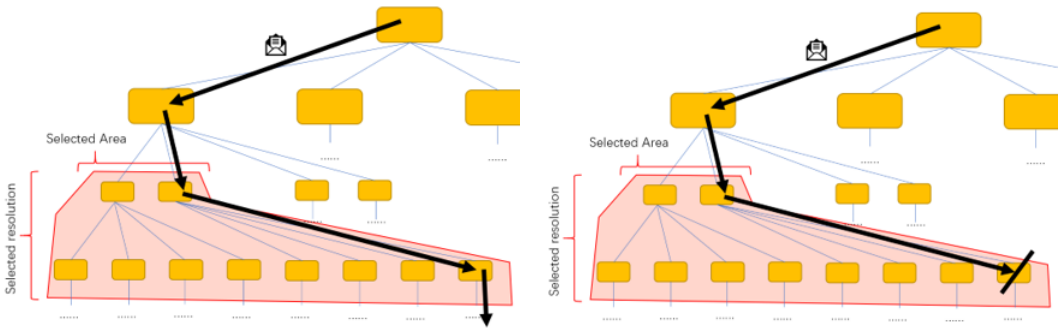
The k-d tree is also a commonly used spatial data structure. The k-d tree is a binary tree that divides the data in half in one dimension at a time. The k-d tree has better applicability to high-dimensional data, but is more suitable for static data, because the k-d tree has no balance maintenance mechanism, and it is easy to lose balance when inserting dynamic data.

3.3.3 Data Processing

Data processing is mainly divided into two parts. On the one hand, data preprocessing, this part includes extracting useful information from the collected raw data. This mainly depends on the specific business logic and the need for visualization, and there will be no general solution, so it will not be discussed here. According to the previous discussion, our final rendering target will be a heat map or a scatter plot, so we assume that after all the data items we have input have been processed, they only contain a two-dimensional geographic location parameter and an independently Dimensional continuous parameters. Such data is sufficient to generate a heat map.

The other hand is how to facilitate the subsequent display by preprocessing the data. Among them, a part of preprocessing has been explained in the data structure part, and the conversion of data access into a specific data structure is itself a kind of preprocessing. If it is a static visualization, this preprocessing is sufficient. However, because we want to support highly dynamic features, we are facing some new challenges. Beyer et al. (2013) is very worthy of reference. On the other hand, it is very important to iterative calculation, which is to show the intermediate results of the calculation to the player. Among them, the second has brought great inspiration to my design.

Still taking the quadtree as an example, when there is a large amount of dynamic data input, we do not need to let every data fall directly to the bottom of the quadtree. If n units of data are stored in a unit of time, the total amount of data is m. If we want all of this data to fall into the bottom of the quadtree, the time complexity is $O(n \log m)$. In the case of big data, when m is very large, $\log m$ cannot be ignored.



(a) Data fall to the bottom (b) Data fall to the level required for rendering

Figure 3.2. The data need not fall to the bottom of the quadtree.

In fact, we only need to let the data fall to the level we need to render, and we can complete the rendering. For example, if our scale is enlarged by 4 times, that is, the root node of the subtree of the entire rendering area is on the third layer, and we need to render at 1024 * 1024 resolution, then we only need to let the data drop 10 + 3 layers. These data can be piled up on the 13th layer, and we don't need to let the data fall further until we need to zoom in again. It is worth noting that when we zoom in, the data that is not in the rendering area only needs to fall to the branch point between them and the data in the area. This means that not all data needs to fall to the specified number of layers. Suppose the root node of the area we need to render is on the kth layer. Assuming that the data points are evenly distributed, only $\frac{1}{4^k}$ of the data needs to fall to the specified number of layers. Other data will only fall in fewer layers.

If the resolution is 1024 * 1024, the root node of the rendering area is in the kth layer, and the data of total n is evenly distributed, of all the n data, three-quarters will only fall to the second layer, three-quarters of the other quarter will fall to the third layer, and so on. The total number of falls can be calculated. The calculation complexity is:

$$O\left(\left(\frac{2}{3}-\frac{k}{4^k}+\frac{28}{3}\times\frac{1}{4^k}\right)\times n\right) \tag{3.1}$$

The greater the magnification, the lower the complexity of data processing, and it will tend to $\frac{2}{3}n$ soon That is to say, even in the case of a large amount of data influx, the rendering of new data can achieve a nearly linear time complexity, regardless of the original amount of data.

We know that, under natural circumstances, the data generation rate is not uniform, but will have some peaks at certain times, and sometimes less. For example, data peaks in the catering industry will be reached during meals. The linear rendering time complexity of this method determines that the image can be rendered efficiently even during the peak time of data influx. Those data points that did not fall to the bottom will be processed when the operation is idle or when the zoom ratio is adjusted.

3.3.4 Overall programming structure

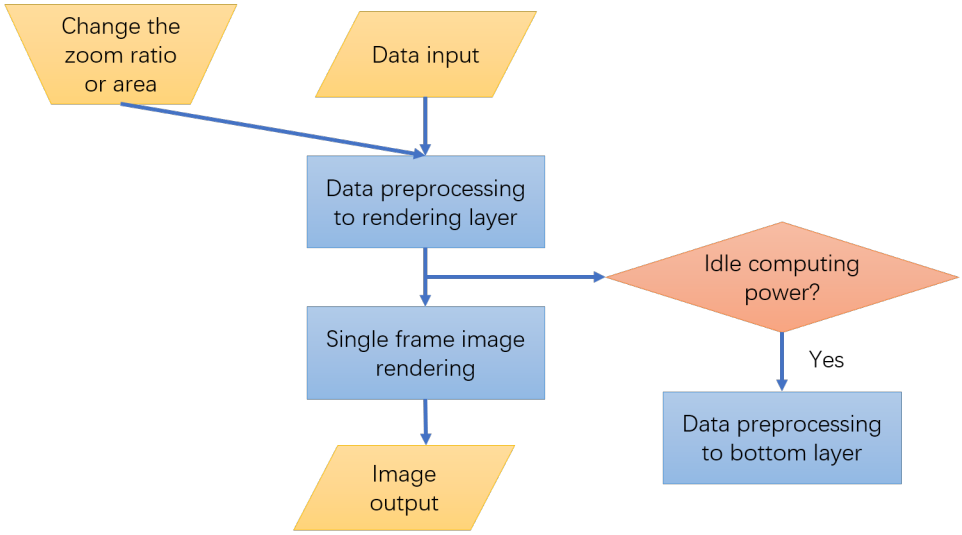


Figure 3.3. Data rendering process

Overall, when the data input is obtained, the data is pre-processed, which is also stored in the quadtree. But the depth of the deposit depends on the rendering requirements and remaining computing power. In idle time, the data that has not been saved to the bottom of the quadtree is processed. When changing the zoom scale and area, you need to calculate some data that has not been saved to the bottom layer. The final rendering will directly take the data of the specified layer of the quad tree in the specified area for rendering.

3.4 Data Sources

The analysis part itself is based on existing academic research and open source libraries. Most of these software are open source software. Now, most software developers choose to use open source to publish visualization tools and seek help from the open source community. Therefore, I can directly study the calculation principle of these open source software, and analyze the applicability and bottleneck of these software in the scenario described in this article. Some commercial software is closed source, so these software are only suitable for performance comparison. The specific rendering details are unknown.

On the other hand, research requires some geographic information datasets with different characteristics and scales for experimental data. Some commonly used data sets include traffic violation data published by the government, some Internet companies such as Yelp's business review data (YelpInc., n.d.), and flight delay data. Take the business review data set released by Yelp as an example, which includes 5,200,000 user reviews for more than 174,000 businesses. Reviews include detailed geographic information about the business. This is a good data set that can be used for actual testing.

However, due to privacy, merchants with actual business data rarely publish very large volumes of raw data. But if necessary, PB-level dynamic geographic data can be manually manufactured. The artificially produced data may differ from the actual data in some distribution rules, but only considering that it is used for visual performance testing, these virtual data should be able to achieve the required functions.

CHAPTER 4. DATA ANALYSIS

4.1 Testing platform and conditions

Table 4.1. *Testing platform*

Operating system	Windows10 x64
CPU	Intel Core i7-9750H 2.60GHz
RAM	16GB
GPU	NVDIA GTX 1660 Ti

The platform used for all the tests below is my personal laptop. Among them, I shielded all other cores of the CPU except Core 0. This is to avoid performance inequities that some software supports multi-threading and some software does not support.

All the data listed below are the average of ten actual tests. Ten tests for each data used different test data sets, including 8 sets of random data, a set of Yelp review data, and a set of traffic violation data. It is worth noting that the test results show that the standard deviation of all test data is within 10%, which means that for different data, the performance of these software is relatively stable. At the same time, it also ensures the reliability of the comparison between the data.

4.2 Analysis of existing software

4.2.1 JS library

Many existing data visualization software have chosen JS as the platform. One of the biggest advantages of the JS platform is to directly interface with rich web pages. Data visualization results can be displayed directly on the browser to any user who opens the webpage.

If convenience is an advantage of the JS platform, performance will become a bottleneck on the JS platform. First of all, JS is based on the browser platform. Now commonly used browsers, such as chrome or firefox, do not support JS multi-threading, making the CPU performance can not be fully utilized. In addition, as we all know, browsers such as Chrome are very memory intensive. In terms of rendering, these JS libraries are often based on DOM trees. One advantage of using the DOM tree is that the DOM tree itself is rich in information, and comes with support for many operations, which is itself a feature of HTML, making it very convenient to describe and manipulate some graphical elements. The disadvantage of the DOM tree is performance. If many modern JS engines can support rates close to native programs, the operation of the DOM tree is very slow. Although JS libraries such as JS have used many new technologies, such as cumulative updates, Shadow DOM, etc. to speed up DOM tree operations, the speed of DOM tree operations is still a bottleneck.

Some commonly used JS-based data visualization tools that support big data include D3.js, Echarts, amCharts, etc. Many high-level visualization libraries are not considered here. The main reason is that many high-level visualization libraries are essentially packaged by low-level visualization libraries. Many advanced visualization libraries will focus on the beauty of visualization or the natural and smooth human-computer interaction, rather than the performance that this study focuses on.

4.2.1.1 JS library static heat map performance test

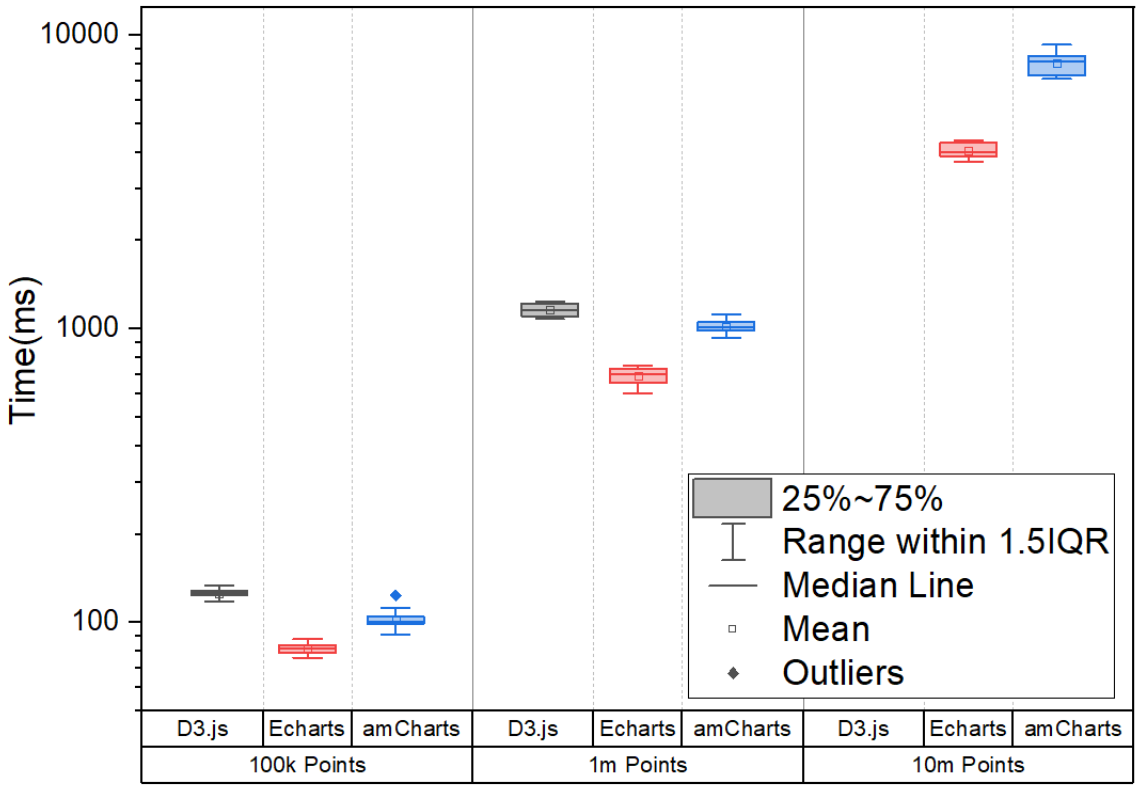


Figure 4.1. JS library static heat map performance

Detailed data can be found in Appendix A. The data in Appendix A includes the subdivision time-consuming data of the js library image rendering in different stages.

A brief explanation about why in the time spent in image visualization, the sum of the various items does not equal the total time. This is because the time of itemization is measured using browser breakpoints. The sum of the actual average measurement results does differ from the total time. Guess this difference is due to the mechanism of the JS engine inside the browser. But the trend it reflects is still accurate and does not affect performance bottleneck analysis.

As you can see, the performance of each JS library is different. The overall performance of D3.js and amCharts is relatively close, while the overall performance of Echarts is better than the first two. D3.js crashed at 10 million data points and could not display the image. No way to fix the crash was found. The crash occurred during the processing of the data, and I was not sure which step had the problem.

In the actual test, I also tested some other JS libraries. Either the performance is worse or it is not representative. Here, the data of the three most representative libraries are selected for explanation.

First look at the time of data reading. In terms of data reading, all three libraries are read in cvs format, and it has been loaded into memory in advance.. The time counted here is the time it takes after the data is read until it is converted to the built-in type of the library. The read time is roughly in a similar range, and it increases as the amount of data increases. It can be considered that when the amount of data is large enough, the data reading time mainly depends on the amount of data, and it should increase linearly until the memory is occupied. This is not surprising, but also shows that data reading is not the bottleneck of rendering.

Then, we look at the data processing time. The data processing time basically increases linearly with the amount of data. This itself is consistent with the characteristics of heat maps. Echarts performed slightly better than D3.js and amCharts.

Finally, it is the image rendering time that shows the significant difference. Echarts uses canvas rendering, while D3 and amCharts use SVG. This is not obvious when the amount of data is small, but it is quickly reflected after the amount of data increases. At the same time, in terms of memory usage, Echarts is significantly better than D3 and amCharts. It should also be because SVG depends on the DOM tree, causing a lot of additional memory overhead. This shows that for static heat map rendering, the front-end time consumption of rendering is a key bottleneck. Real pure image output like canvas can save a lot of time and space. The DOM tree brings freedom of operation, but sacrificed performance is unacceptable in the case of big data.

4.2.1.2 JS library static heat map zooming test

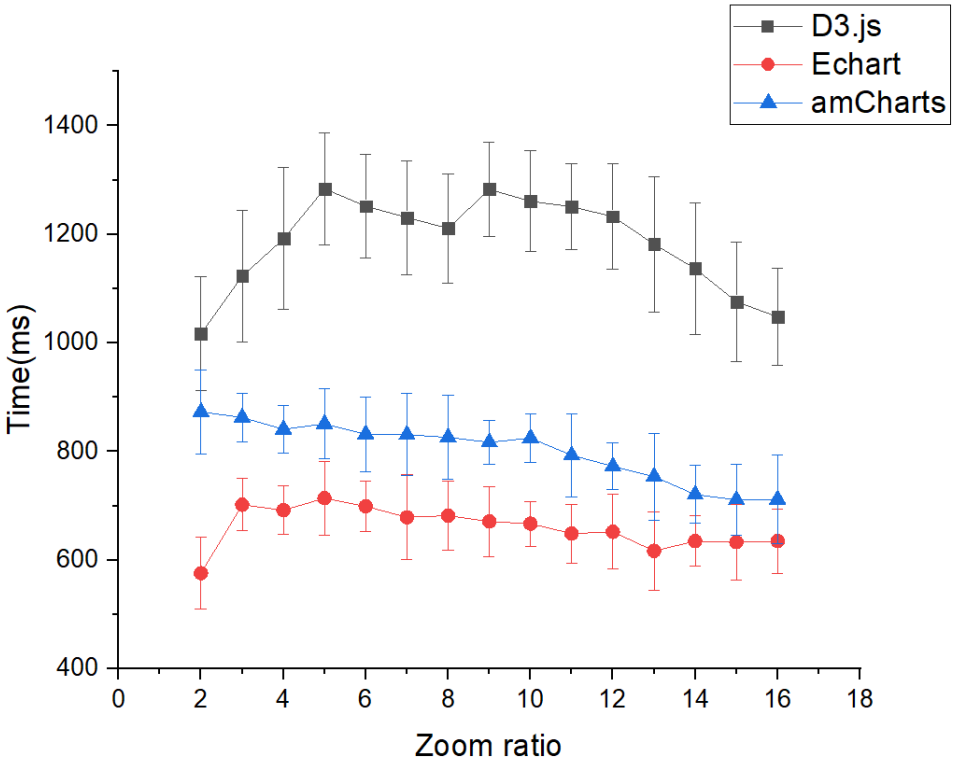


Figure 4.2. JS library static heat map zooming test with one million data

Subsequently, the rendering time difference of the static heat map under different zoom factors was tested.

It can be clearly seen that in fact the zoom ratio is not much different from the rendering time. To make matters worse, both the data processing time and image rendering time at different zoom ratios are close to the time consumption of the first screen. This actually means that these JS libraries are not pre-optimized for zooming. Each time you zoom, the program rescans all the data to find out which ones need to be displayed, and then recalculates each grid point of the heat map. No intermediate data is saved.

The animation effect of these JS libraries in zooming is quite good, but the performance is not optimized at all, probably because they were not originally developed for commercial-grade big data scenarios. But maybe with the popularization of big data in the future, the JS library will also increase the corresponding functions.

4.2.1.3 JS library heat map adding data test

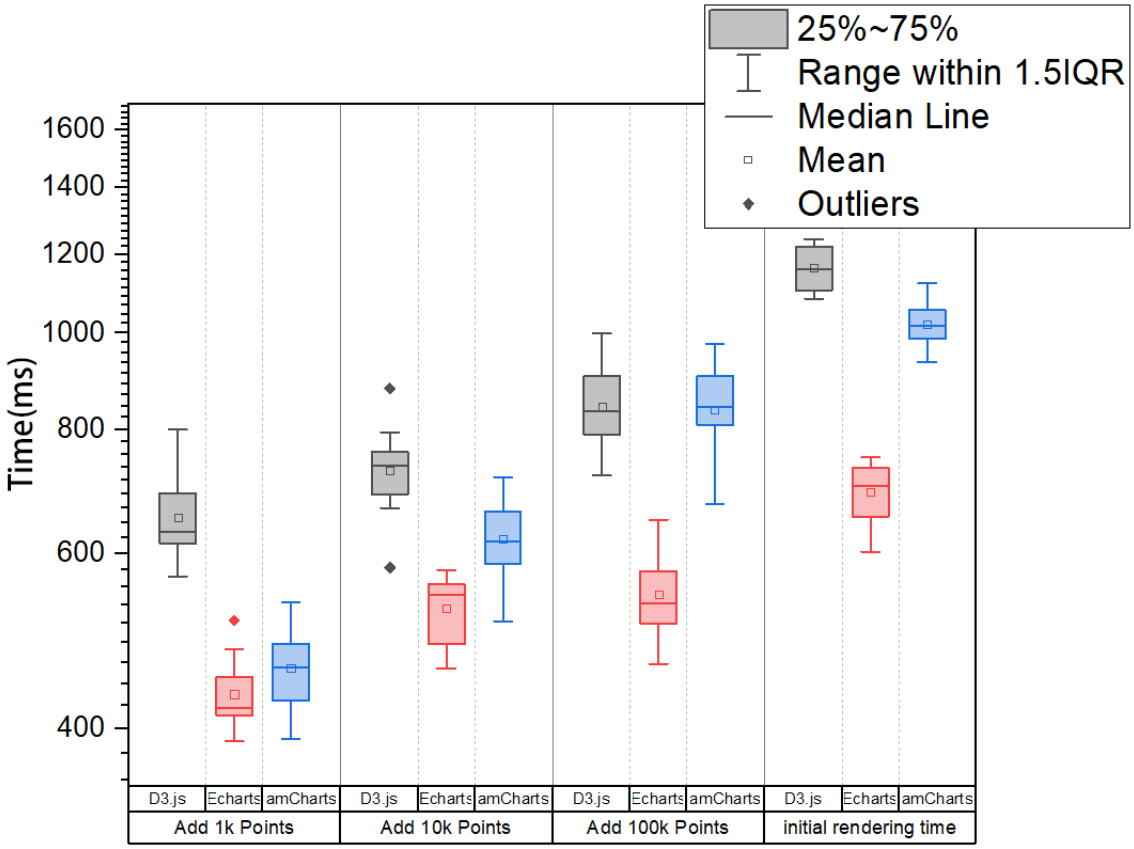


Figure 4.3. JS library heat map adding data test with one million data

It can be seen that when adding new data points, when there is less new data, the time consumed by these JS libraries is slightly reduced, but it is still on the same order of magnitude. When more new data is added, the time consumed by these JS libraries is actually very close to the initial rendering time of the entire screen.

That is to say, although when adding data, these JS libraries can partially reuse the previous calculation results, but there is no fundamental difference in order of magnitude, and the operation is still performed on data that far exceeds the amount of new data points.

4.2.2 Native software

In contrast, the native visualization library is not as rich as the visualization library selection of the JS platform, and a lot of it is large-scale commercial software and is not open source. But usually they show more clearly support for big data. So there is reason to expect them to perform better. Moreover, from the front-end perspective, these local software directly output images, and can even be accelerated by the graphics card. Local software usually executes faster.

Some representative software include Plotly, Tableau, Lyra, etc.

4.2.2.1 Native software static heat map performance test

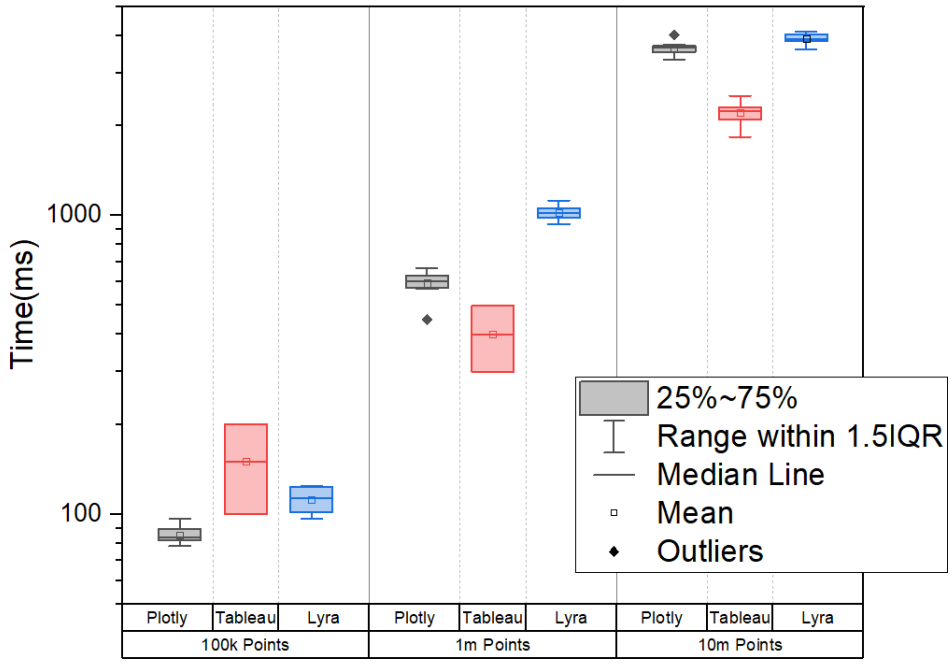


Figure 4.4. Native software static heat map performance test

One problem with using existing native software is that it is difficult to monitor where to go. Native software, even if it is open source, is actually compiled when it is run and cannot be interrupted at will to calculate process time consumption. Not to mention that these softwares are quite large, and some are still proprietary software. Therefore, only the time required to complete rendering a picture is counted here. Because Tableau is a closed-source software, it can only measure the time used manually, so the measurement accuracy is only 0.1 seconds.

It can be seen that the time used by Plotly and Lyra is about the same. Tableau has obvious advantages in time, but the memory it uses is very large compared to the other two.Four to five times the other two. Specific memory usage can be viewed in Table A.4

4.2.2.2 Native software static heat map zooming test

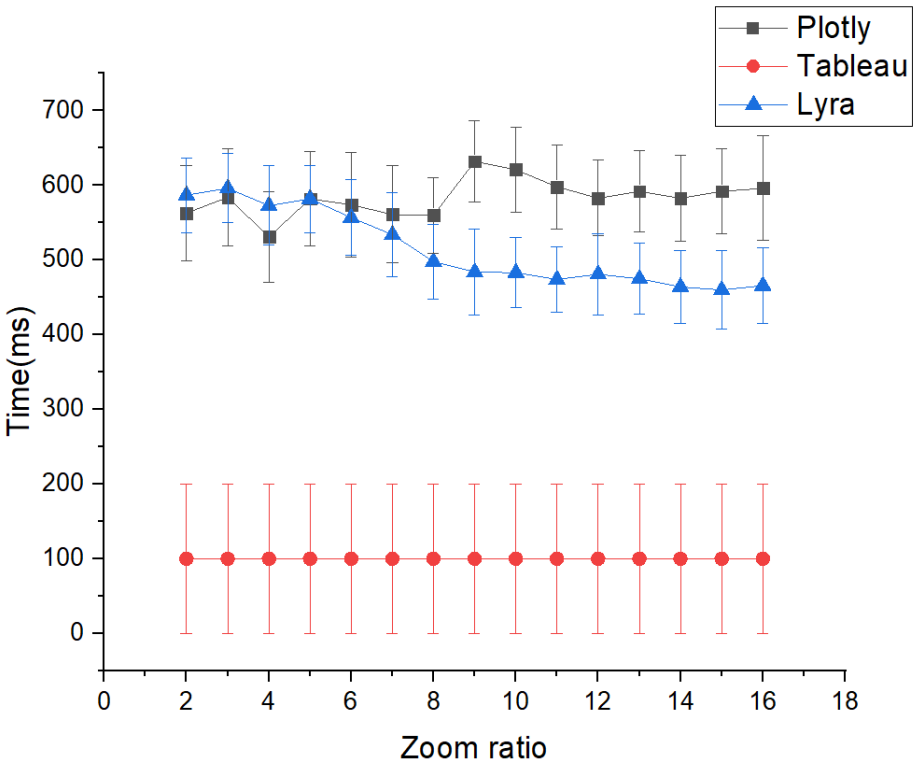


Figure 4.5. Native software static heat map zooming test

Tableau's rendering time after zooming out stood out. In fact, Plotly and Lyra's rendering time after scaling is almost the same as the initial rendering. This shows that they may still read all the data to recalculate and render. Tableau needs little time to re-render after zooming, which inevitably means that Tableau has calculated more data in advance than it needs to render the current heat map.

Tableau did not publish more details on the calculation principles within its software, but it claimed to use PostGIS(Tableau, 2020). PostGIS is an open source program that uses R-tree and GiST to achieve spatial indexing, which greatly accelerates the speed of orthogonal queries. Tableau may also have a built-in spatial database, which makes it unnecessary to traverse all data points when accessing scaled data, thereby greatly increasing the speed.

4.2.2.3 Native software heat map adding data test

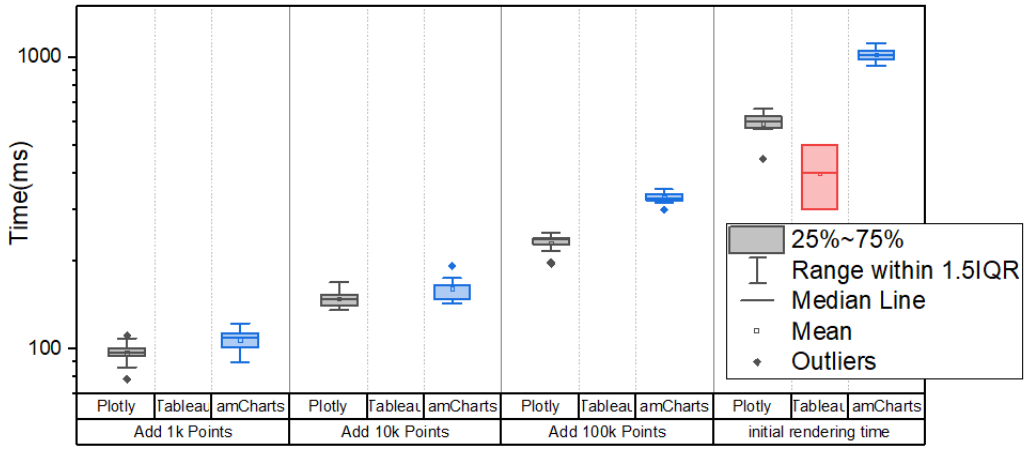


Figure 4.6. Native software heat map adding data test

Although Tableau has many excellent performances in previous tests, Tableau does not support adding data dynamically. Tableau can import incremental data through an external database, but in principle this is equivalent to regenerating the entire project. This may be because Tableau is mainly concerned with report business and data analysis, and these businesses do not need to add data dynamically.

The quick secret of Tableau is that after it gets the data, it performs a lot of preprocessing, and these complex preprocessing are not compatible with dynamically adding data, or it is too complicated to do so. The end result is that although Tableau can efficiently support scaling, it does not support dynamic data at all.

On the contrary, Plotly and Lyra did a good job of adding data. Their performance is close, and the time used when adding data is significantly less than the initial rendering. Because they directly read the position of the data when adding the data, and add it to the corresponding pixel. For heat maps, this is efficient when adding data. But the cost of doing this is that they cannot be scaled. If scaling is performed, all pixels need to be completely recalculated.

In summary, most web-based JS-based data visualization libraries have similar performance and are not very good. Only rendering to canvas has slightly better performance. However, it supports more diverse and beautiful graphics, and is not optimized for spatial data type scaling and new data insertion. The local visualization library has slightly better performance. It has certain optimization for the scaling and real-time insertion of the spatial data type of the heat map, but there is no visualization tool that optimizes both.

4.3 Performance of the new method

In the research method section, I have proposed a method based on spatial data structure. Through on-demand loading and iterative calculation, this method can take into account both scaling performance and the ability to dynamically add new data. According to this method, a visual tool prototype was developed using c ++ for testing. According to this method, a visual tool prototype was developed using c ++ for testing. The rendering front end uses image sequence output directly to avoid performance differences caused by the output platform and rendering method. The file reading also uses a unified interface, so I can focus on testing the performance differences brought by the data structure and data processing. The impact of using this method on actual performance is discussed below.Later, I also transplanted this method to the js platform to compare with the js library to illustrate the platform independence of its advantages.

4.3.1 Comparison between the new method and the traditional method

First, we will compare the impact of using and not using spatial data structures, and using and not using on-demand loading on visualization.

Table 4.2. *Whether to use spatial data structure performance comparison*

	Without SDS	With SDS	With SDS & On-demand
One hundred thousands data points			
First screen time consume	32ms	214ms	54ms
4x Zooming	31ms	23ms	24ms
16x Zooming	29ms	23ms	23ms
Add one thousand data	14ms	16ms	14ms
Add ten thousands data	23ms	34ms	22ms
Add one hundred thousands data	36ms	215ms	42ms
Zooming when adding data	68ms	243ms	53ms
Memory usage	13MB	75MB	77MB
One million data points			
First screen time consume	354ms	1032ms	427ms
4x Zooming	332ms	54ms	53ms
16x Zooming	311ms	52ms	54ms
Add one thousand data	12ms	46ms	16ms
Add ten thousands data	24ms	83ms	14ms
Add one hundred thousands data	37ms	268ms	58ms
Zooming when adding data	394ms	278ms	78ms
Memory usage	49MB	454MB	472MB
Ten million data points			
First screen time consume	3212ms	14212ms	4921ms
4x Zooming	2912ms	66ms	63ms
16x Zooming	2764ms	62ms	65ms
Add one thousand data	17ms	62ms	25ms
Add ten thousands data	26ms	83ms	31ms
Add one hundred thousands data	44ms	352ms	63ms
Zooming when adding data	3401ms	314ms	103ms
Memory usage	311MB	2215MB	2314MB

The test items still include the initial screen rendering time, zoom time and new data point time.But an additional test is the rendering time of zooming while adding new data points.The test also used data sets from 100,000 to 10 million data points as test data.

First look at the initial screen rendering time-consuming. After using the spatial data structure, the initial screen rendering time is greatly increased compared to the linear table data structure. This is because the spatial data structure performs a lot of preprocessing when loading data, which makes the time consumption increase greatly. However, after using the iterative operation of on-demand loading, the initial screen rendering time returned to the time close to the linear table. When the initial screen is loaded, only the data structure is constructed to the required level, the screen is rendered, and the construction of the spatial data structure is continued when the operation is idle.

Look at the memory usage. In terms of memory usage, after using spatial data structures, memory footprint has also increased significantly. And regardless of iterative loading, the memory footprint is close.

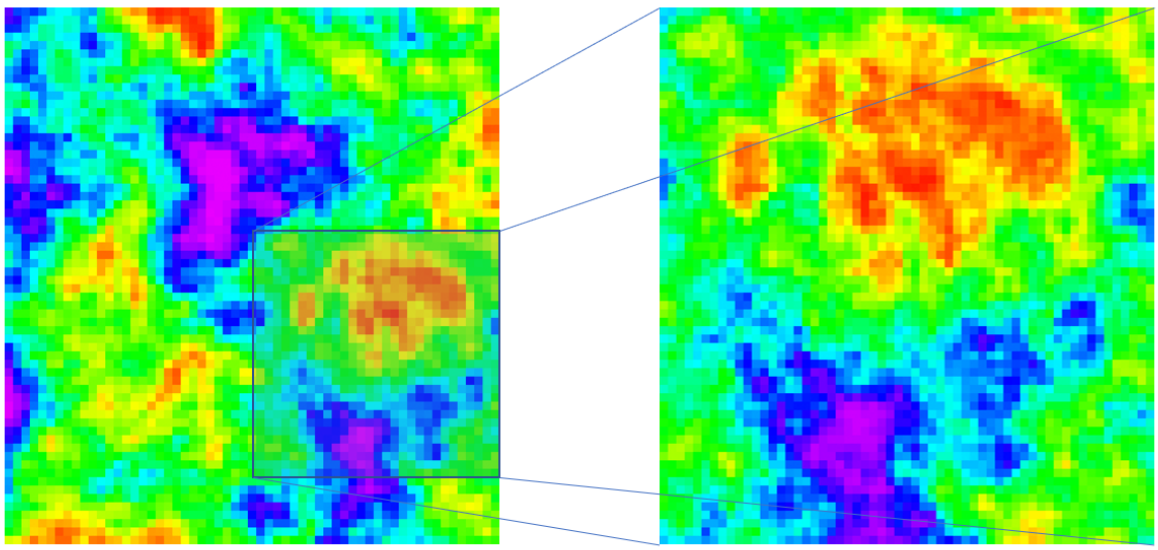


Figure 4.7. Zoom image output example of my method

Then came the impressive zoom performance. When the amount of data is small, whether to use the spatial data structure is not significant. However, as the amount of data increases, the rendering time of the method without the spatial data structure is almost the same as the initial screen rendering, but the method using the spatial data structure takes a relatively short time to zoom and is almost not affected by the increase in the amount of data.

In terms of increasing data, as with the initial screen rendering, the time consumption will increase slightly after using the spatial database. However, as long as iterative loading is used, this part of the increase can be eliminated.

Finally, add data while zooming. It can be seen that without using the spatial data structure, we face the same problem as simple scaling, that is, the rendering time is almost the same as the initial screen rendering. This is unacceptable when the amount of data is large.

However, after using the spatial data structure, especially after using on-demand loading, the program has quite good performance in scaling and adding data. In fact, in the case of 10 million basic data points, every 100,000 additional data points can be rendered in about 0.1 seconds. This means that even if there is an influx of 100,000 data per second, the program can update the screen and zoom at a refresh rate of ten frames per second.

4.3.2 Comparison between the new method and existing libraries

Compared with existing software, the significance of comparing absolute values is not very great. Because the running platform, environment, and some internal algorithms are not the same. The visual test program written by myself is very streamlined and targeted, but the existing software may do some additional processing and operations for compatibility. However, the analysis of its relative time-consuming growth as the data increases is still obvious.

First compare the scaling performance.

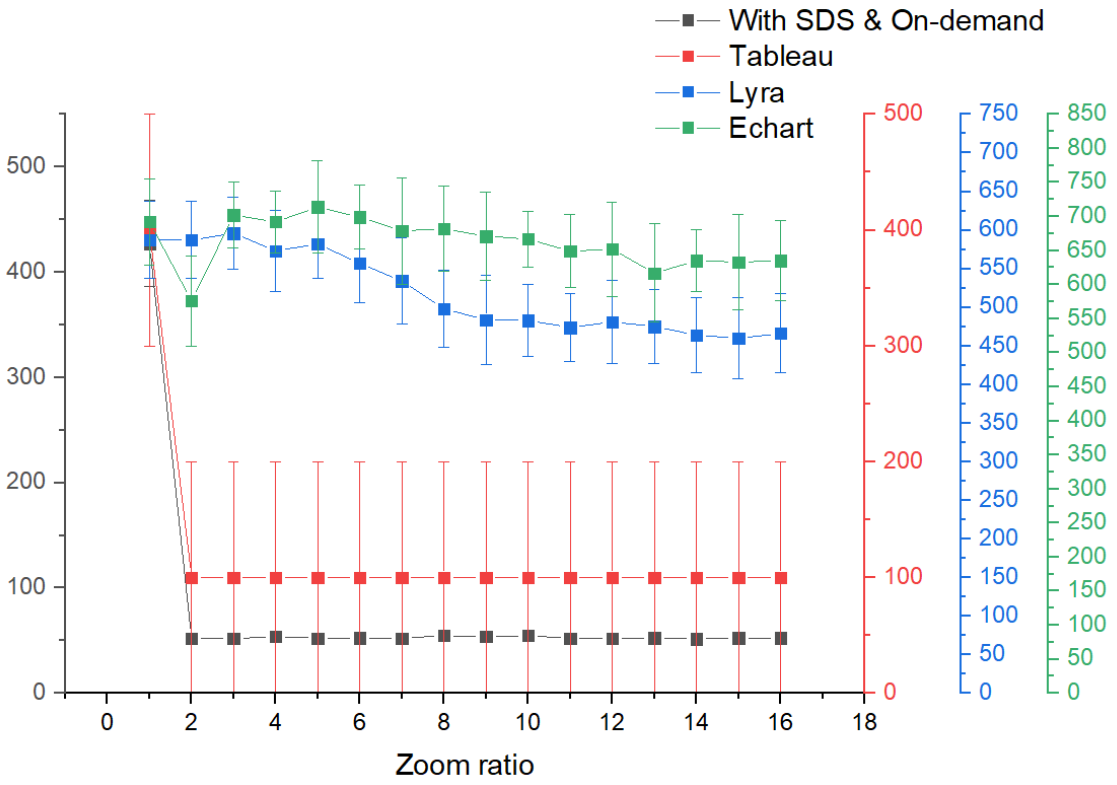


Figure 4.8. Relative zoom time improvement

First, the Y axis is scaled proportionally, so that the initial screen rendering time of all curves is aligned, which can eliminate the deviation caused by the different running speeds of different languages. It can be seen that the new method of using SDS and on-demand loading is the same as Tableau. The time consumption of zooming is much smaller than that of the initial screen rendering, and the curves of Lyra and Echart are close to horizontal.This is because Tableau also uses the SDS method, so it is similar in performance to the method I implemented using c ++.

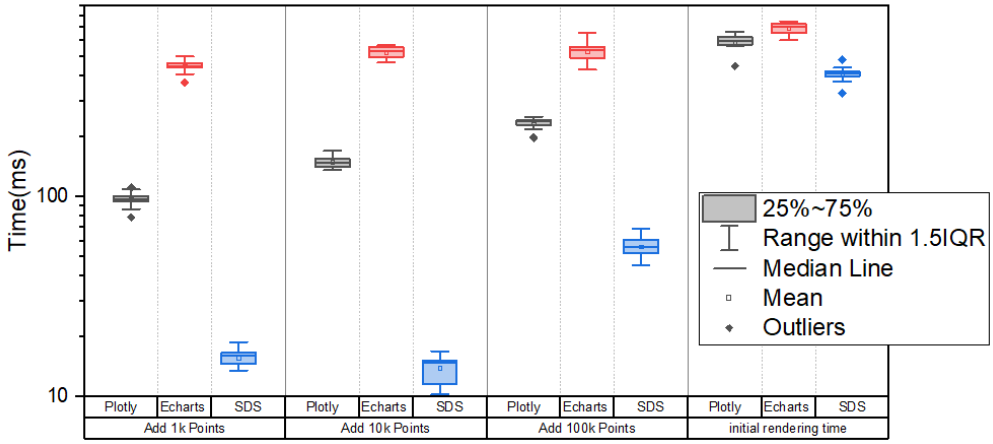


Figure 4.9. Relative adding time improvement

You can then view the performance when adding data. The initial screen rendering time of the three is similar, but when adding data, the method using SDS and on-demand is also better than the existing method. This is because on-demand loading requires recalculation when adding data, rather than recalculating all data. This also brings excellent performance in adding data while scaling.

In fact, existing tools do not fully support dynamic big data and scaling at the same time. The software for big data display is actually recalculating the rendering completely when zooming. With the exception of Tableau, tableau does not accept dynamically added data.

The new method uses a spatial database and adds iterative operations, which takes into account both scaling performance and the performance of dynamically adding data. However, the price is the use of more memory than existing software. But this kind of memory usage is acceptable because the space complexity of memory usage is O(nlogn), and the impact of a log entry is acceptable. However, when there is little data, the use of spatial databases introduces unnecessary time overhead and may even be slower than existing software.

Therefore, when the total amount of data exceeds one million, the use of spatial databases will significantly improve the scaling performance compared to most existing software. When the number of new data points reaches more than tens of thousands per second, the spatial database is an excellent method that takes into account the scaling performance and the performance of the new data. After using iterative on-demand computing, even if it reaches hundreds of thousands of new data per second The picture refresh rate can also be well guaranteed. Specifically, on the test platform, 100,000 new data are added per second while zooming, and the rendering can still achieve a frame rate of 10 frames per second.

CHAPTER 5. CONCLUSION

5.1 Implications and contributions

This paper presents a new visualization method of geographic big data combining spatial data structure and iterative computing on demand.

In this area, existing researches have a gap. So far, no one has provided a universal and effective geographic information visualization solution that can be scaled in a highly dynamic big data environment.

In this area, existing researches have a gap. So far, no one has provided a universal and effective geographic information visualization solution that can be scaled in a highly dynamic big data environment. Existing libraries and software can't take good care of scaling performance and new data performance, let alone scaling and data addition at the same time.

The method proposed in this paper not only takes into account the scaling performance and the performance of the newly added data, but also proves through experiments that when the amount of data reaches more than 100,000, the visualization rendering speed is greatly improved. When the amount of data is large, it significantly surpasses the performance of existing software and methods. And it is foreseeable that when the amount of data is larger, the spatial data structure will become an inevitable choice in geographic visualization.

This research is important to those national information management departments or multinational corporations. In many cases, we need to monitor the data change trend both in a nationwide level and a specific single point. For example, we may want to see whether traffic violation has some rule in state level, and when we find the overall rule, we may want to find out how exactly is a hot point composed.At this time, the ability to scale on highly dynamic data sets is very important.

This research may also inspire more efficient practices for big data visualization, reminding researchers to pay attention to new expression and performance challenges brought by changes in the visualization requirements of the same data set in different scenarios. Today's visualization libraries tend to focus attention and expressiveness and visualization effects, but these visualization effects may not be a simple matter to balance performance with big data.

5.2 Future works

This article only discussed and experimented with heat maps. But there are many more common types of visualizations. The methods used in this article are usable for visualization methods with localized data. For example, Category map or Choropleth Map can be stored in a structure such as quadtree or R tree, and can be calculated on demand. For other different types of visualization, different optimization methods may be required in a highly dynamic big data environment.

An important research direction in the future may include related big data. As mentioned in Blending mode, how to visualize geographic big data with links between data points which is highly dynamic and scalable is still a large challenge. This may not only apply to existing simple spatial data structures. One possible data structure is the Hypergraph Based Data Structure. The concept of hypergraphs is an extension of the concept of graphs, which can better solve problems such as relational data structures that cannot express implicit relationships and lack integrity; tree-like data structures are not suitable for reflecting non-hierarchical relationships; mesh-like data structures are not suitable for reflecting non-history Hierarchy and other issues.

Another research direction includes how to make the scaling and data addition in the big data environment more beautiful. In the case of insufficient computing power, if the frame rate is very low, the output of some intermediate frames may help the zoom effect to be smoother.

REFERENCES

Akella, M. (2016). *Visualize 2015 urban populations with proportional symbols.* Retrieved November, 2019 from: `https://carto.com/blog/proportional-symbol-maps`. (Online)

Andrienko, N., Andrienko, G., & Gatalsky, P. (2003). Exploratory spatio-temporal visualization: an analytical review. *Journal of Visual Languages & Computing*, *14*(6), 503–541.

Battersby, S. (2017). *Data map discovery: How to use spatial binning for complex point distribution maps.* Retrieved November, 2019 from: `https://www.tableau.com/about/blog/2017/11/data-map-discovery-78603`. (Online)

Beyer, J., Hadwiger, M., Al-Awami, A., Jeong, W.-K., Kasthuri, N., Lichtman, J. W., & Pfister, H. (2013). Exploring the connectome: Petascale volume visualization of microscopy data streams. *IEEE computer graphics and applications*, *33*(4), 50–61.

Choo, J., & Park, H. (2013). Customizing computational methods for visual analytics with big data. *IEEE Computer Graphics and Applications*, *33*(4), 22–28.

Cui, W., Zhou, H., Qu, H., Wong, P. C., & Li, X. (2008). Geometry-based edge clustering for graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, *14*(6), 1277–1284.

DeBoer, M. (2015). *Understanding the heat map.* Retrieved November, 2019 from: `https://cartographicperspectives.org/index.php/journal/article/view/cp80-deboer/1420`. (Online)

Dent, B. D., Torguson, J. S., & Hodler, T. W. (1999). *Cartography: Thematic map design* (Vol. 5). WCB/McGraw-Hill New York.

Ersoy, O., Hurter, C., Paulovich, F., Cantareiro, G., & Telea, A. (2011). Skeleton-based edge bundling for graph visualization. *IEEE transactions on visualization and computer graphics*, *17*(12), 2364–2373.

Gorodov, E. Y., & Gubarev, V. V. (2013). Analytical review of data visualization methods in application to big data. *Journal of Electrical and Computer Engineering*, *2013*, 22.

Güting, R. H. (1994). An introduction to spatial database systems. *the VLDB Journal*, *3*(4), 357–399.

Howard, H. H., McMaster, R. B., Slocum, T. A., & Kessler, F. C. (2008). Thematic cartography and geovisualization. , 85–86.

Kim, Y.-I., Ji, Y.-K., & Park, S. (2014). Social network visualization method using inherence relationship of user based on cloud. *International Journal of Multimedia and Ubiquitous Engineering*, *9*(4), 13–20.

Lambert, A., Bourqui, R., & Auber, D. (2010). Winding roads: Routing edges into bundles. In *Computer graphics forum* (Vol. 29, pp. 853–862).

Lehner, G. (2019). *Introduction to dynamic feature binning in arcgis pro.* Retrieved November, 2019 from: https://www.esri.com/arcgis-blog/products/arcgis-pro/mapping/introduction-to-dynamic-feature-binning-in-arcgis-pro/. (Online)

Nowak, M., & Spiller, G. (2017). *Two billion people coming together on facebook.* Retrieved November, 2019 from: https://newsroom.fb.com/news/2017/06/two-billion-people-coming-together-on-facebook/. (Online)

Puget sound mapping project. (2012). Retrieved November, 2019 from: https://www.commerce.wa.gov/serving-communities/growth-management/puget-sound-mapping-project/. (Online)

Schnabel, O. (2008). *Presentation of thematic data.* Retrieved November, 2019 from: http://www.carto.net/schnabel/pop_dens/. (Online)

Sucharitha, V., Subash, S., & Prakash, P. (2014). Visualization of big data: its tools and challenges. *International Journal of Applied Engineering Research*, *9*(18), 5277–5290.

Tableau. (2020). *Connect to spatial data in a database.* https://help.tableau.com/current/pro/desktop/en-us/maps_spatial_sql.htm. (Accessed April 1, 2020)

What is a dot-density map? (n.d.). Retrieved November, 2019 from: https://www.caliper.com/glossary/what-is-a-dot-density-map.htm. (Online)

Yang, C. C., Chen, H., & Hong, K. (2003). Visualization of large category map for internet browsing. *Decision support systems*, *35*(1), 89–102.

Yeap, E., & Uy, I. (2014). *Marker clustering and heatmaps: New features in the google maps android api utility library.* Retrieved November, 2019 from: http://googlegeodevelopers.blogspot.com/2014/02/marker-clustering-and-heatmaps-new.html. (Online)

YelpInc. (n.d.). *Yelp open dataset.* `https://www.yelp.com/dataset`. (Accessed April 1, 2020)

Zhou, H., Xu, P., Yuan, X., & Qu, H. (2013). Edge bundling in information visualization. *Tsinghua Science and Technology*, *18*(2), 145–156.

Zhou, H., Yuan, X., Cui, W., Qu, H., & Chen, B. (2008). Energy-based hierarchical edge clustering of graphs. In *2008 ieee pacific visualization symposium* (pp. 55–61).

Zhou, H., Yuan, X., Qu, H., Cui, W., & Chen, B. (2008). Visual clustering in parallel coordinates. In *Computer graphics forum* (Vol. 27, pp. 1047–1054).

APPENDIX A. EXISTING LIABRARY PERFORMANCE

Table A.1. *JS library static heat map performance test*

	D3.js	Echarts	amCharts
Authorization type	Open Source	Open Source	Private
Rendering front-end	SVG	Canvas	SVG
One hundred thousands data points heat map total time	126ms	82ms	102ms
• Data reading time	40ms	32ms	42ms
• Data processing time	62ms	34ms	32ms
• Image rendering time	41ms	20ms	40ms
• Memory usage	43MB	60MB	60MB
One million data points heat map total time	1182ms	692ms	1023ms
• Data reading time	140ms	120ms	124ms
• Data processing time	450ms	219ms	240ms
• Image rendering time	620ms	153ms	380ms
• Memory usage	205MB	79MB	184MB
Ten million data points heat map total time	Crashed	4390ms	8050ms
• Data reading time	1253ms	1004ms	1540ms
• Data processing time		2310ms	4230ms
• Image rendering time		230ms	1080ms
• Memory usage		352MB	1029MB

Table A.2. *JS library static heat map zooming test*

	D3.js	Echarts	amCharts
One million data points 2x zooming single screen refresh time	1017ms	576ms	873ms
• Data processing time	420ms	193ms	250ms
• Image rendering time	452ms	120ms	346ms
• Memory usage	200MB	80MB	192MB
One million data points 4x zooming single screen refresh time	1192ms	692ms	841ms
• Data processing time	490ms	201ms	252ms
• Image rendering time	372ms	102ms	304ms
• Memory usage	200MB	88MB	201MB
One million data points 16x zooming single screen refresh time	1048ms	635ms	712ms
• Data processing time	439ms	204ms	275ms
• Image rendering time	420ms	133ms	334ms
• Memory usage	205MB	87MB	203MB

Table A.3. *JS library heat map adding data test*

	D3.js	Echarts	amCharts
Totally one million data points before adding			
Add one thousand data	654ms	437ms	468ms
• Data processing time	210ms	183ms	245ms
• Image rendering time	220ms	124ms	145ms
Add ten thousands data	735ms	524ms	624ms
• Data processing time	193ms	177ms	252ms
• Image rendering time	317ms	113ms	228ms
Add one hundred thousands data	832ms	539ms	841ms
• Data processing time	325ms	223ms	223ms
• Image rendering time	482ms	120ms	346ms

Table A.4. *Native software static heat map performance test*

	Plotly	Tableau	Lyra
Authorization type	Open Source	Private	Open Source
How to use	R	GUI	GUI
One hundred thousands data points heat map total time	83ms	<200ms	113ms
• Memory usage	102MB	305MB	124MB
One million data points heat map total time	589ms	400ms	634ms
• Memory usage	211MB	529MB	146MB
Ten million data points heat map total time	3582ms	2200ms	3928ms
• Memory usage	329MB	1329MB	245MB

Table A.5. *Native software static heat map zooming test*

	Plotly	Tableau	Lyra
One million data points 2x zooming single screen refresh time	563ms	<200ms	587ms
• Memory usage	231MB	692MB	142MB
One million data points 4x zooming single screen refresh time	531ms	<200ms	573ms
• Memory usage	213MB	758MB	150MB
One million data points 16x zooming single screen refresh time	596ms	<200ms	466ms
• Memory usage	257MB	783MB	152MB

Table A.6. *Native software heat map adding data test*

	Plotly	Tableau	Lyra
Totally one million data points before adding			
Add one thousand data	96ms	Doesn't Support	105ms
Add ten thousands data	146ms	Doesn't Support	159ms
Add one hundred thousands data	246ms	Doesn't Supports	325ms