Visliu

Responsive Visualization of Points in Space: Sampling, Clustering, Partitioning

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
Akshay Jain
May 2015

_____

Professor Dennis Shasha, Research Advisor

_____

Dr. Manpreet Singh Katari, Second Reader

Acknowledgements

I would like to thank my research advisor, Professor Dennis Shasha, for inspiring this work and especially for his guidance and patience along the way.

Thanks also to Dr. Manpreet Katari, my second reader for introducing me to Virtual Plant which brought me in contact with the use of interactive visualization for data analysis and research.

Finally, I want to thank my parents for their endless support in everything I do.

Table of Contents

Future Work

Bibliography

.

Interactive visualization

“I tell you and you forget. I show you and you remember. I involve you and you understand.”
Confucius, 500 BC

“A picture is worth a thousand words. An interface is worth a thousand pictures.”
Ben Shneiderman, 2003

1. Introduction

Humans have a total of 5 senses through which they interact with the world : sound, sight, touch, smell and taste. Of these, smell and taste are pretty limited in the sense that we have not yet found any efficient way to communicate using them. Touch is limited by the requirement of tangible objects in close proximity. Sound and Sight are known to be most advanced and are a major part of most of our daily activities. Through language we have even connected them to activate the same areas of the brain using subvocalization. But these two are still quite different. Sight has multiple orders of magnitude higher ability to process signals than sound and unsurprisingly has a larger portion of brain dedicated to it than any other single task.[1]

We like to define Interactive Visualization as the field of study which aims to use sight as a more efficient means for interacting with and understanding the world around us. A key feature of a good interactive visualization system is that it tries to use cognitive intuition to convey information.

In this paper we survey some of the major developments in interactive visualization till now and then combine some of these ideas to build a web system for interactive visualization of points in space. We also use some ideas of our own that enable a responsive interface even with sufficiently large dataset sizes.

2. Present state of things

Fueled by continuous developments in hardware technologies and an exponential increase in data collected, Interactive Visualization technologies have developed a great deal in the past few decades. Although a lot of effort and time has been spent on developing better visualization technologies, there is still a long way to go.

Here we are going to look at few of the technologies and systems that have recently been developed or come into the spotlight for Interactive Visualization on the web.

Base technologies for rendering on the browser:

a. **Canvas** (HTML5)[2] : Pretty recent; Natively supported by most browsers, but limited by the number of elements that can be rendered at a time.

b. **SVG**[3] : More than a decade old; Natively supported by most browsers but limited by the number of elements that can be rendered at a time. This limit is lower than that of HTML5.

c. **WebGL**[4] : Most recent; Variable and limited support from different browsers. Brings the power of the client GPU to the browser. Limited by the number of devices having a GPU and no standard browser support. Chrome and Firefox have a whitelisted list of GPUs that they allow WebGL to run on.

Libraries/Services for Interactive visualization on the web:

a. **Data-Driven Documents**[5] (D3) (2011):
   i. Javascript library for DOM manipulation on client side.
   ii. Uses SVG at the core.
   iii. Requires learning HTML, javascript, D3's API and SVG.
   iv. Biggest contribution are the tools that make the connection between data and graphics easy.

b. **Google Charts**[6] (2010):
   i. Google's own visualization tool that is nicely integrated with BigTable and other Google cloud services.
   ii. Lets users create a chart from some data and embed it in a webpage.

c. **Highcharts**[7] (2009):
   i. Javascript library for visualizations and animations.
   ii. Offers an easy way to create charts and embed them in your web page.
   iii. Based on HTML5 Canvas.

d. **Three.js**[8] (2010 , stable :2015):
   i. Javascript library for 3D visualizations using WebGL.
   ii. Abstracts the presently messy interface to WebGL.
   iii. When it works(browser and client need to support WebGL), provides abilities to render and interact with a large number of objects at the same time.
   iv. User experience varies depending on the GPU and available client resources.

e. **Bokeh**[9] (2013, still in beta):
   i. Python Interactive visualization library that targets presentation on browser.
   ii. Coding is totally in python.
   iii. Is trying to get graphics in the style of D3.js but uses canvas instead of SGV, so it can render more objects at a time.

f. **Datawrapper**[10] (2012):
   i. Javascript library for creating interactive embeddable charts.
   ii. Visualization is created on the server but it provides a set of interactive abilities which run on the client through its library.

g. **Flot**[11] (2007):
   i. A plotting library for jQuery that uses the HTML5 canvas.
   ii. Simple and good-looking interactive features, but very limited in functionality.

3. Challenges

With widespread interest in visualization at present, there are tons of new systems and libraries for visualization but all of them face these challenges and most of them refuse or fail to address them properly.

a. **Data size**: With web visualization, this is the biggest bottleneck right now. The browser even with native canvas support cannot render or manipulate even medium sized datasets. Some systems overcome this by doing aggregation on the server and updating client with a static image. Others need you to take care of your aggregation yourself before rendering your charts. These ways are either hacky or put extra work on the user that should be the responsibility of the visualization system.

b. **Network**: For libraries that work with a remote server, all data transfer happens in a go and slow or unreliable network leads to bad user-experience.

c. **Computation power**: Most libraries are client side and are limited by the computation capability of the client in providing responsive interaction on bigger than trivial data sizes or provide a very limited set of operations.

Pad and children

1. Multiscale Zooming Interfaces:

The ideas behind multiscale zooming interfaces can be traced back to Sutherland's Sketchpad system[14] developed by him in 1963, but it was not until much later that systems started using zooming as a primary feature in interactive visualization. "Pad"[13], developed by Ken Perlin and David Fox at NYU was a big milestone in realizing the possibilities of zooming interfaces. Pad was followed by multiple systems that took forward the ideas of semantic zooming and filter lenses to build next level systems that have since evolved to become the conventions for interactive data visualization and analysis in their respective fields.

Let us first go over what we mean by zooming in the context of visualizations and then we will briefly look at few of the applications that came out from it.

Let us consider a bunch of 2D objects (marks, icons, etc) displayed on the screen of a monitor. Now there is no actual depth in this setup but the effect of enlarging or shrinking the size of objects with a corresponding increase in inter-object distances on pressing a button or rolling a wheel gives us the illusion of depth. It works very well and uniformly for everyone because it has a direct correspondence with what we experience in daily life. Because of how our vision works, we move closer to an object to see it in more detail and move away to take in the whole view. Since moving closer enlarges an object in our view and moving away shrinks it, on a stationary 2D display, just the act of enlarging or shrinking something is perceived in the same way by the brain. Just the enlarging and shrinking of an object without any changes in its representation is called geometric zooming. Pad introduced semantic zooming which more closely resembles the actual world. With semantic zooming, each object has set magnifications at which its representation changes. When you zoom in, more details are shown while zooming out hides the least important details. A smooth transition or movement of objects on the screen while zooming or panning is very important here as that eases the task of keeping track of objects of interest between two representations.

2. Applications :

In this section we will go over a few significant areas where the concepts introduced by Pad were initially applied. We will also look at a few techniques that came out to complement or directly as a result of the Pad.

a. Macroscope[12]:

Henry Lieberman, working at the Media Laboratory at MIT came up with a nice technique that improves on the traditional zoom and pan. Macroscope introduces multiple translucent layers that show iteratively zoomed out views from the present scale and thus helps the user to avoid the problem of losing visual context. Henry points out that multiple zooming and panning operations might cause the viewer to lose the context from his memory. Macroscope consists of combining multiple layers of information on a single display, using translucency, focus and other image processing techniques to visually combine layers while retaining the integrity of the individual components.

b. Portal Lenses in Pad++[15] :

Portals or magic filters were first described in the original Pad paper and later improved on by Pad++. These basically consist of small windows on the workspace which can transform the default appearance or behavior of an object. Along with the capability to communicate with the object during rendering to change how it appears, these can also filter user events which pass through it.

Today Portal Lenses can be seen in use everywhere. An android phone's screen can be thought of having a landscape lens when a user tilts it to a horizontal position. The view elements rearrange to fill up the newly updated dimensions. Google maps with its multiple views of the same location is another major example. Almost all major document editors (MS word, OpenOffice, Google Docs) contain these lenses in the form of visual representations of a table of numbers in the form of different kinds of charts.

c. Network Collaboration[16] :

Infinite zoomable workspaces have tremendous potential as a long distance collaborative work tool because it allows multiple users on a network to work together in the same apparent information space. Different users could share and view multiple applications while assigning each user a desired degree of interaction.

d. Device user-interface[17] :

Zoomable interfaces can be used as an alternative to traditional windowing systems to view and navigate through large hierarchical databases.

In these systems, searching through the directory structure involves zooming in and out of the directory tree. Though top operating systems continue to use windowing systems as their default, zoomable interfaces have started appearing on touchscreen based mobile system due to a more zoom friendly hardware interface.

e. Web browser[18] :

Zooming interfaces have also shown the capability of being used as an alternate browser interface. Instead of having a single page visible at a time, multiple pages and the links between them are depicted on a large zoomable information surface. Benefits over the present interface include easily surfing through session history and moving back and forth. Many browsers including firefox have already experimented with the idea, but are yet to bring it to the masses.

f. Multiscale Editing :

Availability of easy and intuitive interfaces to create and edit zoomable visualizations is an important part of increasing the usage of zoomable interfaces. “MuSE” [19], developed by George W Furnas and Xiaolong Zhang at the University of Michigan, addressed a number of challenges in creating systems for multi-scale editing. While MuSE proposed some nice strategies to deal with some of the challenges, their solution to problems related to creating and editing connections between objects at different zoom level and creating guided tours were pretty complex.

Prezi[20] is a modern presentation system which has gained a big user base in a short time. It uses an infinite zoomable information surface as the workspace and provides state of the art tools to create and edit animated guided tours through a rich variety of media objects. It is less powerful than MuSE in the type and variety of edits that can be made, but it makes-up for that by making the most used common tasks as intuitive as possible.

g. Techniques for Time-Oriented data[22] :

This paper introduces multiple techniques to visualize and interact with time-series data. These include a qualitative-quantitative hybrid representation of time-series data, more intuitive representation of high-frequency data, and efficient timeline interactions.

Visliu

“Visual Information-Seeking Mantra :Overview first, zoom and filter, then details-on-demand.”
[Shneiderman, 1996]

1. Motivation

After extensive survey of web visualization systems out there, we found that there are a lot of things in which they fall short. We have created a system which uses a server-client architecture to enable responsive visualization of points in space for the client using a combination of clustering, partitioning and sampling along with the learnings from Multiscale Zoomable Interfaces to produce a nice user-experience that lets users navigate and perform basic operations on their data. Instead of trying to compete with the professional systems out there in what they are good at, we have tried to address their most common limitations so as to explore the boundary of what is possible with the present set of technologies in interactive visualization.

2. User-view

Visliu consists of a single page that consists of a 1000x1000px canvas element in the center which we will refer to as ‘workspace’ from this point onwards. It has an operations menu on the left hand side and an options menu in the header between the title and the the workspace (Figure 1).

The user first chooses a data file from the drop down menu on the top left of the page. The input file consists of two tab separated floating point numbers per line. On clicking “Load Data” button, the ‘top level’ representation of the points gets displayed. Top level representation consists of two types of objects, points and clusters. While points are rendered as small circles and represent single data-points from the input data-set, clusters are large filled squares and represent a large number of points in the small area covered by the filled square. We will discuss them in more detail in the implementation section.

There is a slider in the header menu to switch between pan and select modes of interaction. While ‘Select’ mode allows you to select one or multiple objects from the workspace which can then be operated on by using the options on the left, ‘Pan’ mode lets you click and drag over the workspace in a zoomed state to explore the neighborhood of the present view. How to use zoom and pan for data exploration, perform operations and adjust visualization configuration will be explained next:

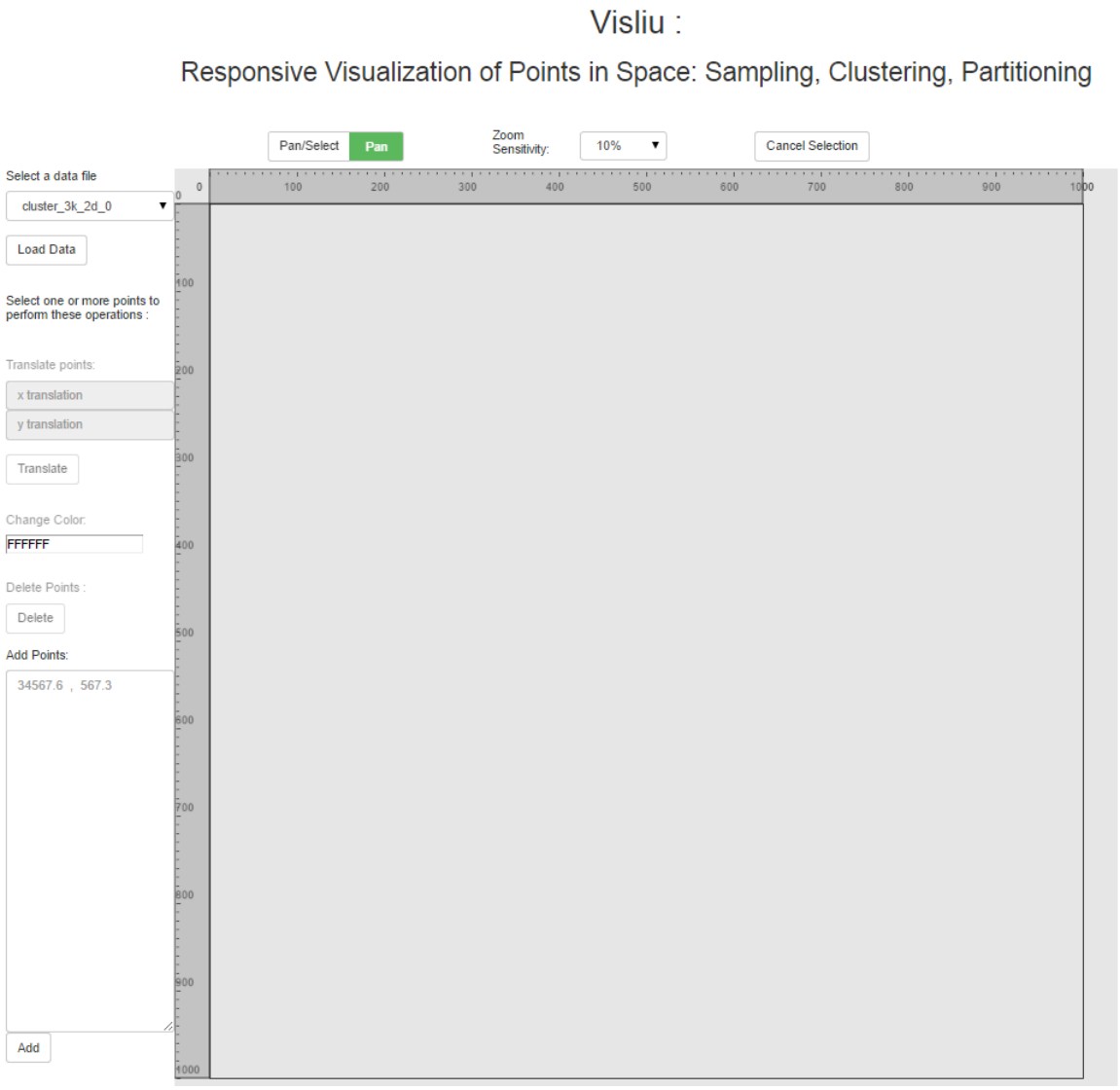

Figure 1: *User-view of the Visliu Web Interface*

a. Zoom and Pan

Much of our effort has gone into making zooming and panning work in a way that is intuitive for the user.

When a data-set it loaded by the user, the server calculates a high-level view of the whole dataset by clustering points in areas with point-density greater than a threshold density. The client displays clusters as filled squares on the workspace(Figure 2).

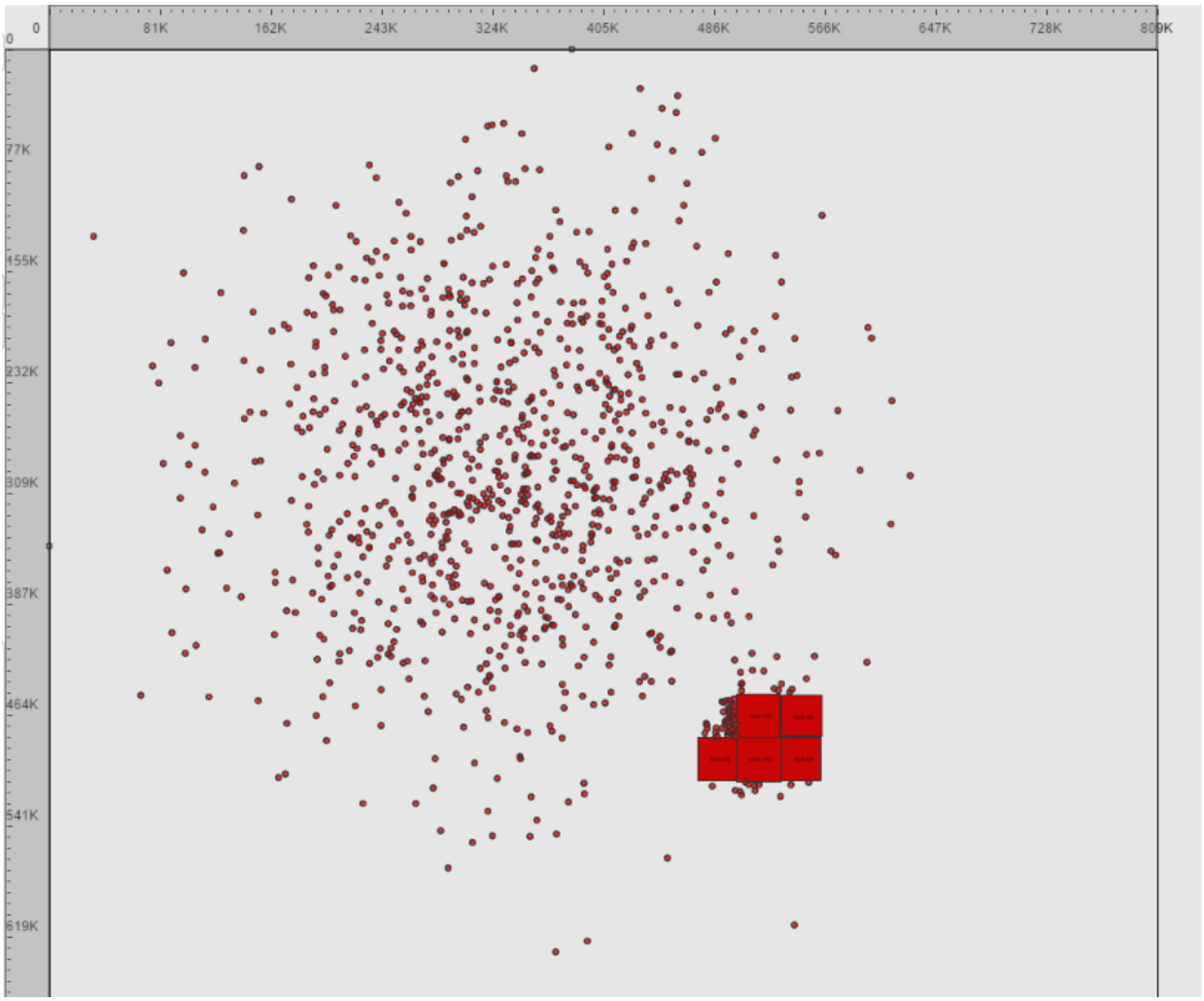

Figure 2: *High-level view of a data-set with 3,000 points*

The user can use scrolling on a mouse or two-finger vertical swipes on a modern touchpad to zoom in and out of the view at whichever point they desire.

If the user zooms into a cluster square,the square starts getting larger accordingly but also starts getting less opaque. The opacity of the square reaches 0 at 4x the initial size of the cluster. This is done so that the details of objects inside the cluster become clearly visible. A cluster could itself contain more clusters and points. The size of each cluster is displayed on it.(Figure 3) This is an implementation of the concept of semantic zooming as defined for Pad++ systems and a key to effective data exploration in zoom based user interfaces. We will talk more about it in the implementation section.

The user can click and drag the mouse over the canvas to do panning across the information space in a zoomed state.

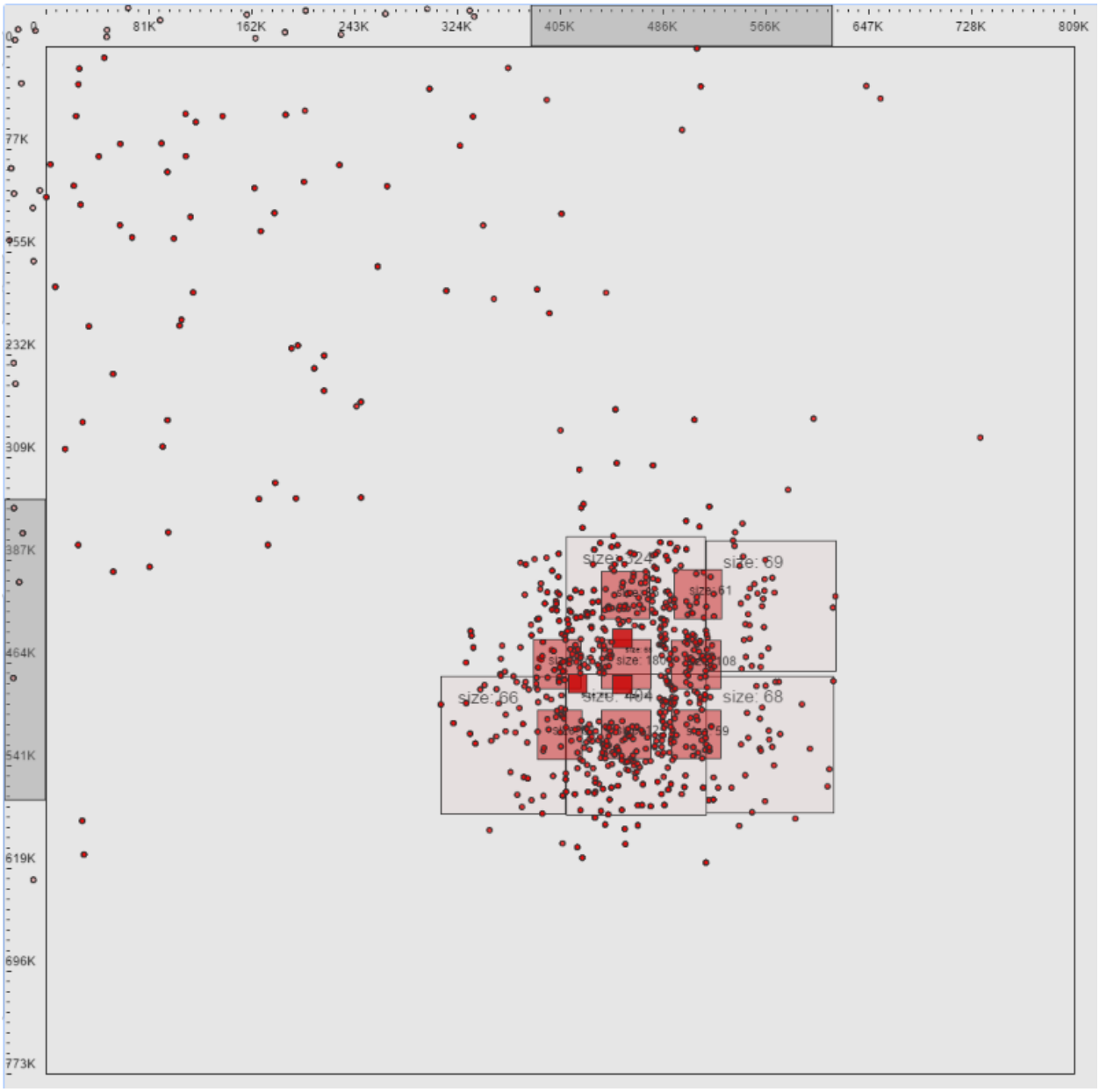

Figure 3: *Visliu workspace with 3 levels of clusters seen. Outermost level is already transparent, next level is partially transparent but is already displaying its internal objects, Innermost level is still opaque and will display its constituent points on further zoom.*

Visliu also provides an option of changing the zoom sensitivity in the header menu which lets the user change the rate at which the view zooms in or out depending on their hardware and preference. This is also helpful as a user might want to zoom in pretty slowly for dense datasets or when he is trying to find patterns in the data-set and fast when he knows the subset of the data he wants to zoom look at. The zoom

sensitivity varies from 10% to 100% which indicates the percentage by which the distances between points scale up or down with a single step of mouse wheel or touchpad event.

Another feature that helps the user keep track of where they are with respect to the overall data is the grey colored dynamic axis lining the top and left side of the workspace(Figure 3). This is superimposed on top of a static axis which ranges from the respective minimum and maximum values of the two dimensions in the dataset. The translucent grey coloring shrinks and enlarges with the zoom level and moves left/right or up/down depending on the center of the present view in the zoomed state. We have found this quite helpful in our experiments on exploring and operating on data-sets with Visliu.

b. Operations

The system has multiple operations that can be performed on selected subsets of data. These are: translation, recoloring, point deletion and point addition. The first step to performing an operation is to switch to Select mode by clicking on the Pan/Select switch in the header menu of the page. Once it is in select mode, the user can use mousedown and drag actions on the workspace to select an arbitrary subset of points. A translucent blue area appears during mouse drag to indicate the points that are going to get selected if user leaves the mouse button at that instant. Multiple click and drags add to the present selected point set. This allows the user to select an arbitrary set of points to perform an operation on. As soon as a subset is selected, the operations listed below the Load Data button which were disabled before get activated and can be used:

1. Translate : Consists of two input fields, x and y. The selected points get shifted in the present view by these values on clicking the Translate button.

2. Change Color: Clicking on the change color textbox displays a full color pallet. Clicking on any color in the pallet changes the color of the selected points to that.

3. Delete Points: Clicking the “Delete” button removes the selected points from the workspace.

4. Add Points: Consists of a text-area that can be given space separated values, two in each line. Clicking on the “Add” button adds these points to the visualization.

c. Configuration

Due to the variety to possible point distributions in 2D space, there is no single optimal configuration to display them all. Also, there are user-preferences on point sizes, cluster sizes, etc. To address these, we have made some global server configurations available to the user to view and adjust according to their needs or preferences. Here is the list of modifiable configurations(Figure 4):

- Default point color (default : Red)
- Default cluster color (default : Red)
- Threshold Point Density for clustering (default : 50 points per 20px side square)
- Maximum clusters per level (default : 100)

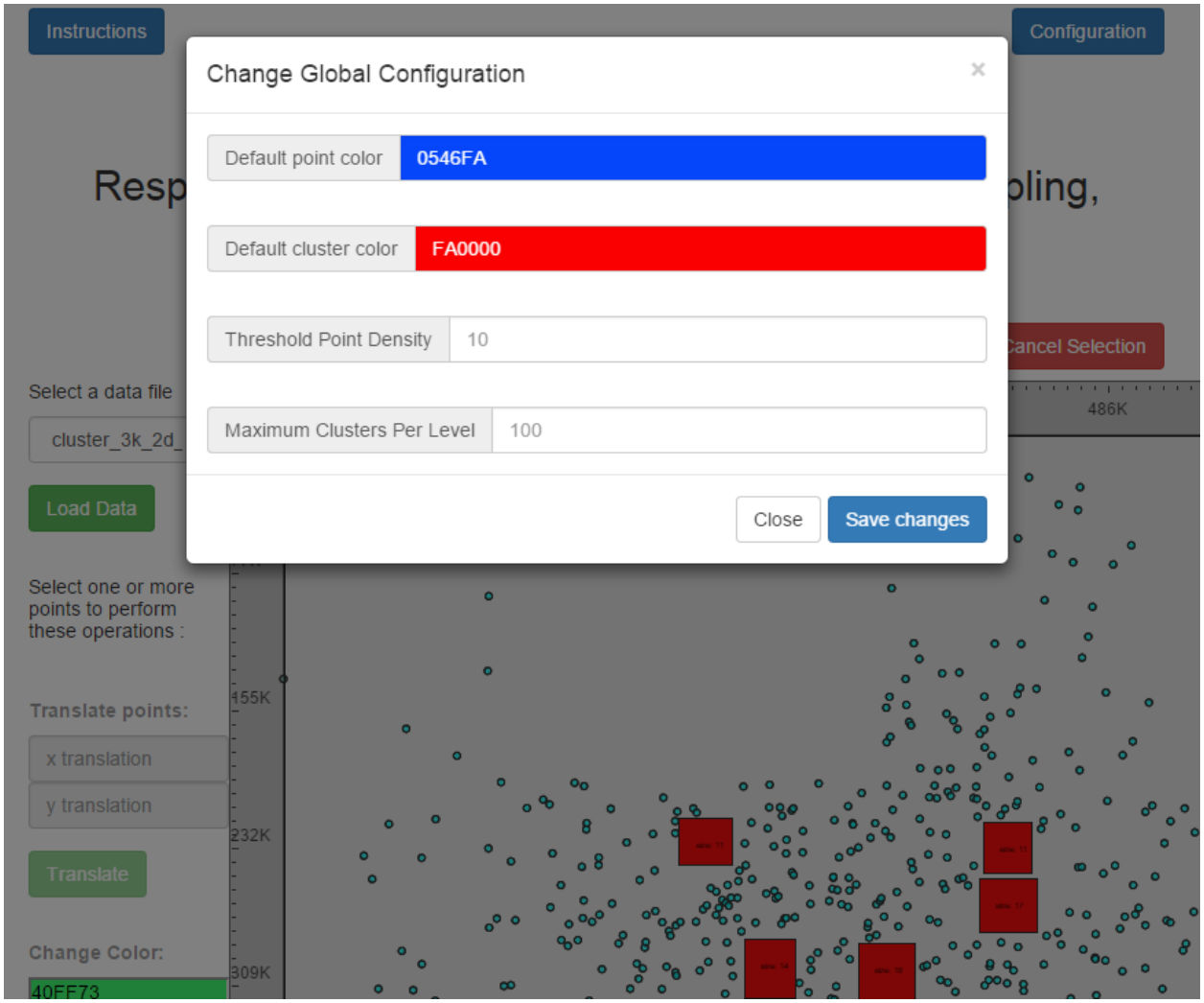

Figure 4: *Menu to change global Configuration of the Visualization in Visliu*

3. Implementation

a. Technology Stack

i. **javascript** :

Javascript is the language of the web. Though it has been ridden with problems and controversies for the major part of the time it has been around, in the last few years it has gained a pretty large following. At present, with major improvements in javascript interpreters, it runs pretty fast and has a sea of libraries available online which do everything from replacing all ‘x’s in a string to ‘y’s, to simulating the linux kernel in the browser, to providing all the tools to make online 3D massively multiplayer role-playing games.

We use javascript both for the front-end as well as the back-end. This provides us with the ability to use the same code for data operations on both the client and the server for automatic load distribution.

ii. **canvas**:

The canvas element is a part of the HTML5 specification by the world wide web consortium and is supported by all major browsers right now. The canvas allows for dynamic, scriptable rendering of 2D shapes and bitmap images. It updates a built-in bitmap and does not have a built-in scene graph as SVG does. Therefore, Canvas is considered as a lower level API on which something like SVG can be built. This also makes Canvas more powerful in terms of number of objects it can create and manipulate simultaneously compared with SVG.

Visliu’s workspace consists of a 1000x1000 px canvas element.

iii. **node.js**:

Node.js is an open source, cross-platform runtime environment for server-side and networking applications. It provides an event-driven architecture and a non-blocking I/O API that optimizes an application's throughput and scalability. Node.js uses the Google V8 JavaScript engine to execute code, and a large percentage of the basic modules are written in JavaScript.

We use node.js for two reasons. Firstly it lets us use javascript on the server-side, secondly, the event-driven architecture and the non-blocking I/O API suits our requirements quite well.

iv. **Processing.js:**

Processing is a programming language which was designed to create and manipulate visualizations, images, and interactive content. It has been used for a long time by researchers and practitioners alike to create interactive visualizations. Processing.js is a javascript port of this language. It basically uses canvas and javascript in the background to provide an API identical to the original Processing language.

We felt this API to be preferable for designing and manipulating our visualization system. We do not however use the event handling API of this library as that was not performing as expected in our experiments.

v. **socket.io:**

This is a javascript framework that enables real-time bi-directional event-based communication. It uses websockets internally which are radically more efficient than AJAX for client server communication. The WebSocket Protocol is an independent TCP-based protocol. Its only relationship to HTTP is that it's handshake is interpreted by HTTP servers as an Upgrade request. The WebSocket protocol was standardized by the IETF as RFC 6455 in 2011, and the WebSocket API in Web IDL is being standardized by the W3C.

Socket.io contributes in a major way in making our system functional by making constant server-client communication faster and easy to implement and manage.

vi. **Other technologies**

We use Bootstrap for giving a better look to buttons and text fields. Jade templating language is used to create the dynamic webpage on the backend. Heroku’s cloud hosting service is used to host the app for public access.

b. Architecture Overview

Figure 5 lists the important design components of Visliu. Figure 6, 7, 8, 9, 10, 11 and 12 go into the details of each of these components and explain how they function.

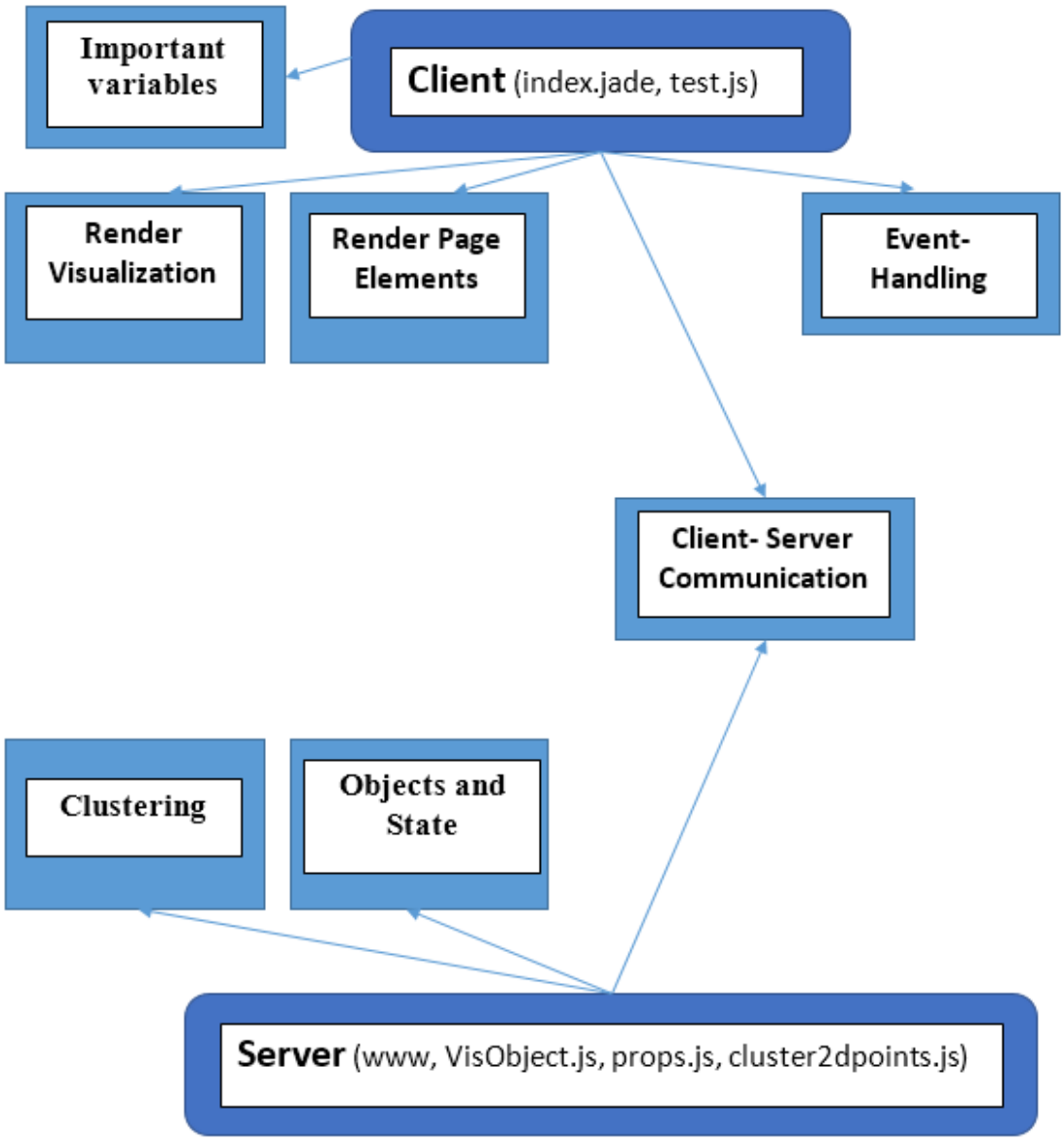

Figure 5: *Major components of Visliu's design*

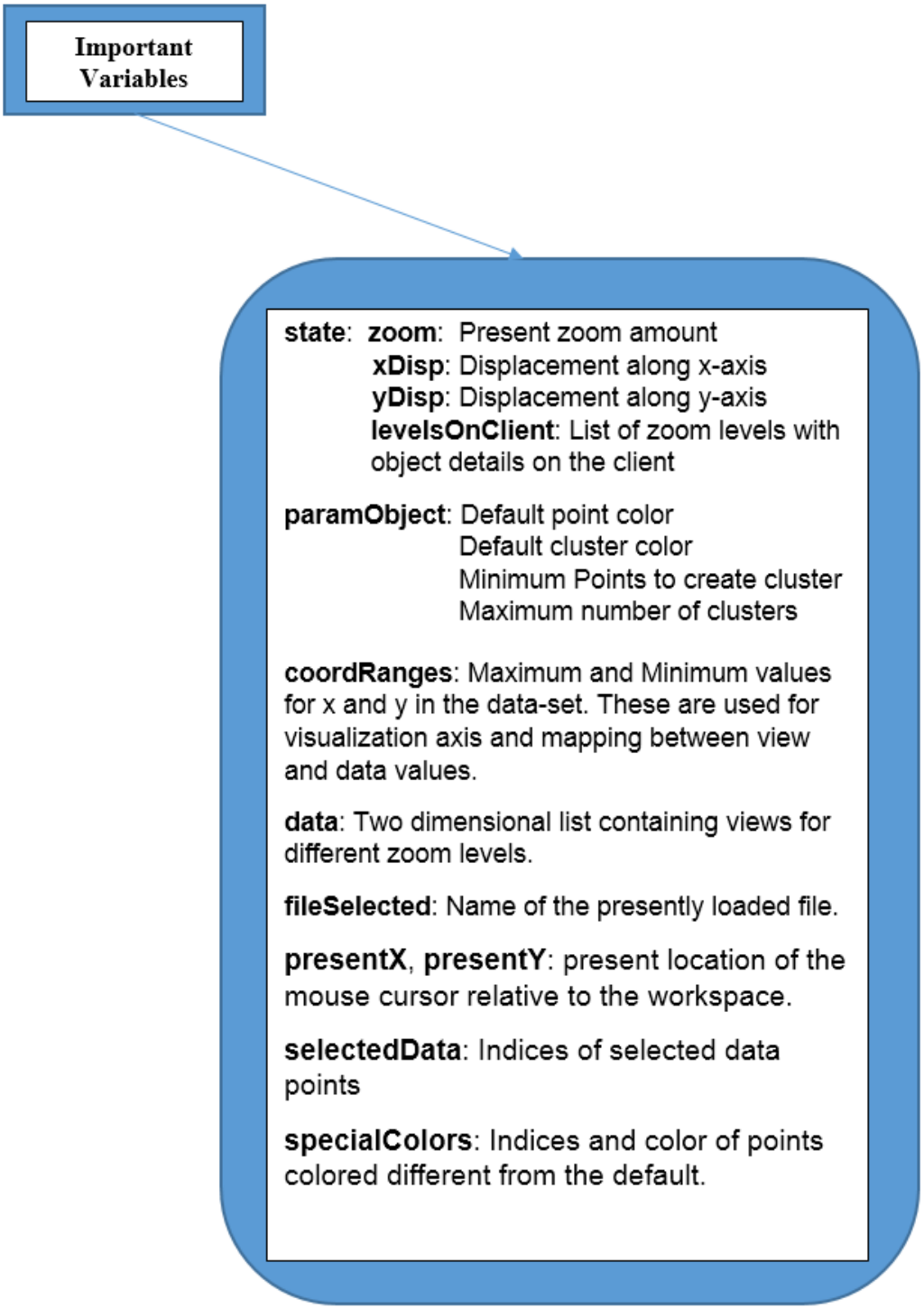

Figure 6: *Description of important variables on client*

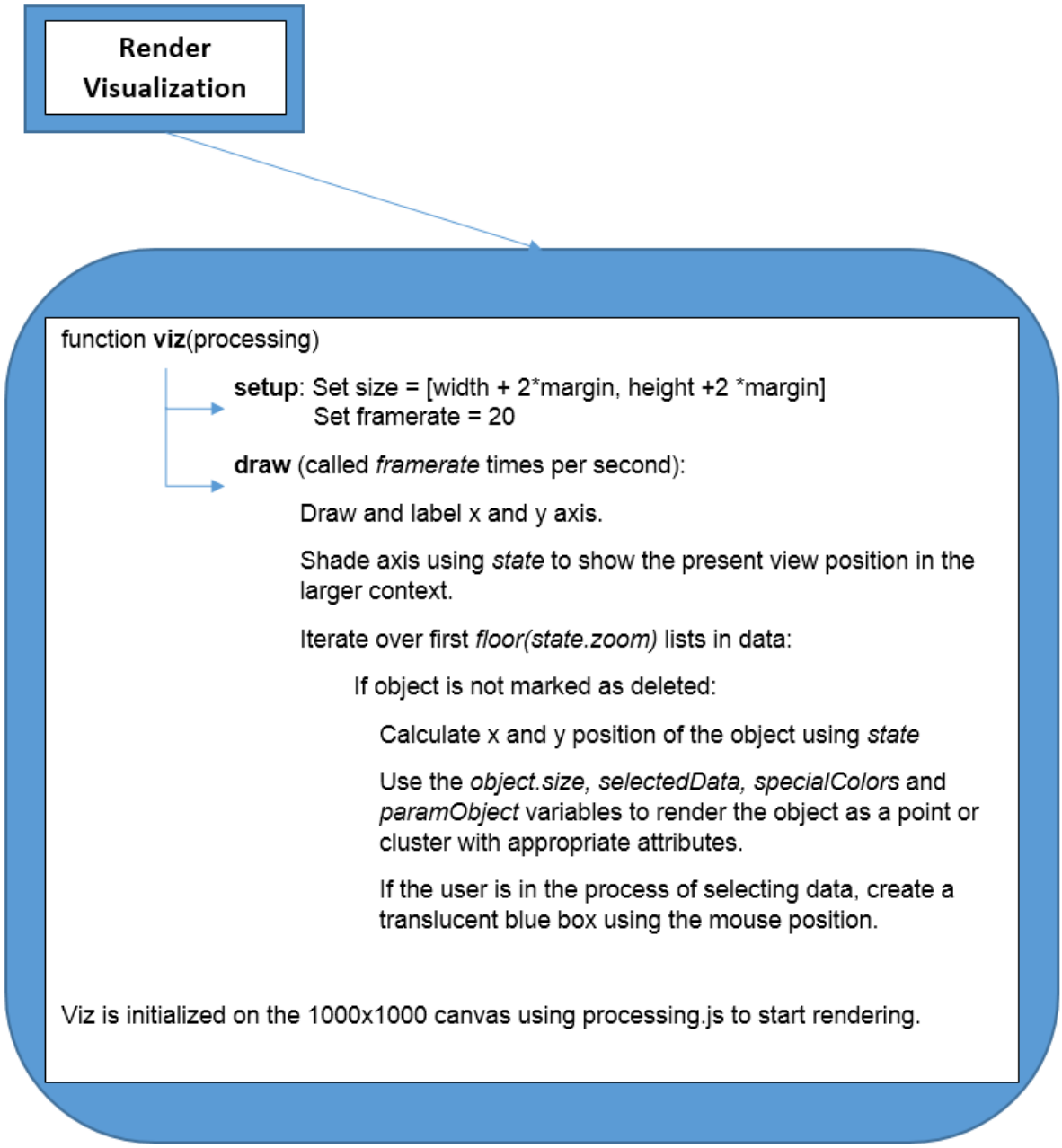

Figure 7*: Overview of Visualization rendering loop on the client*

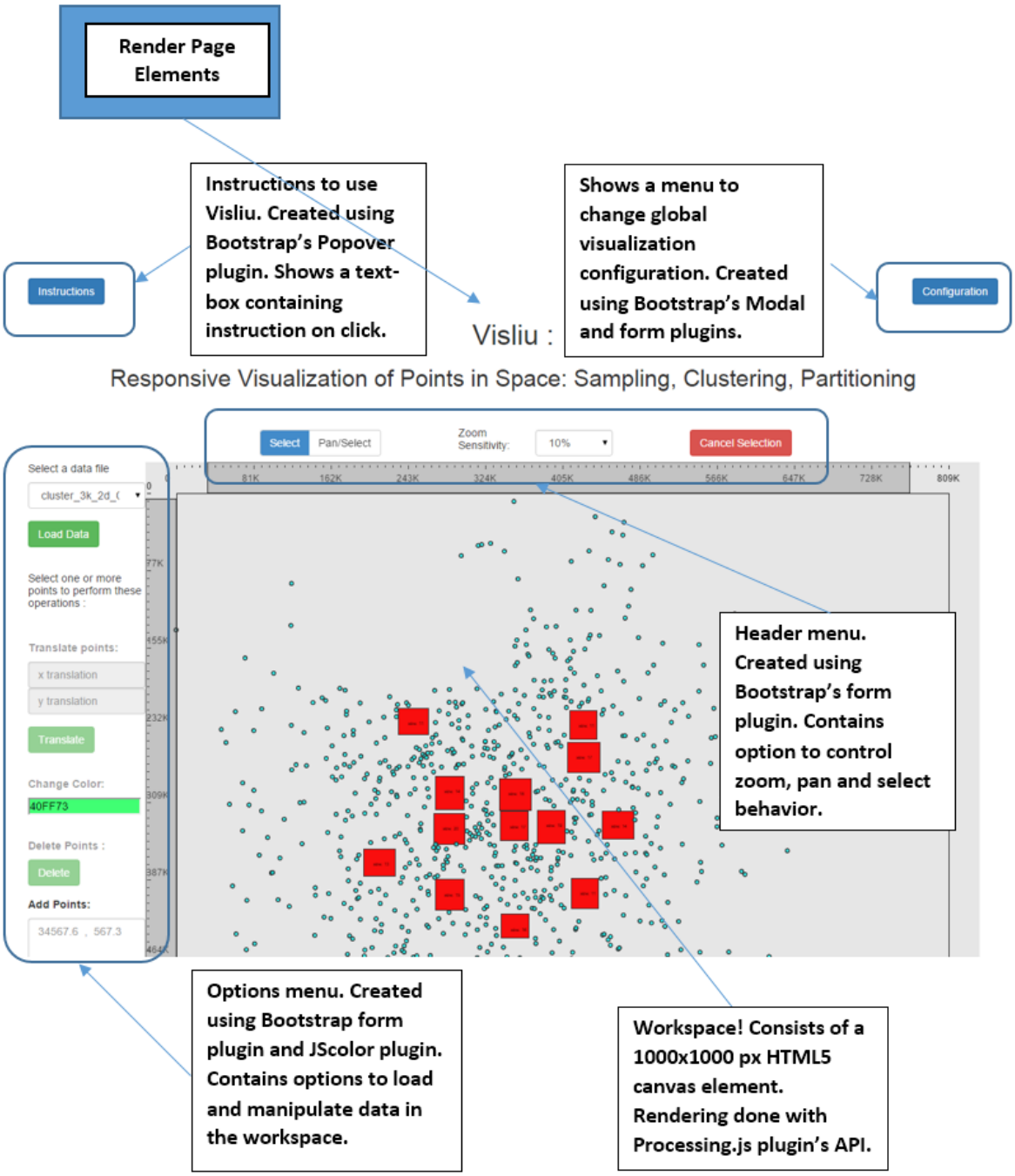

Figure 8: *Description of various elements on the page and how were they made.*

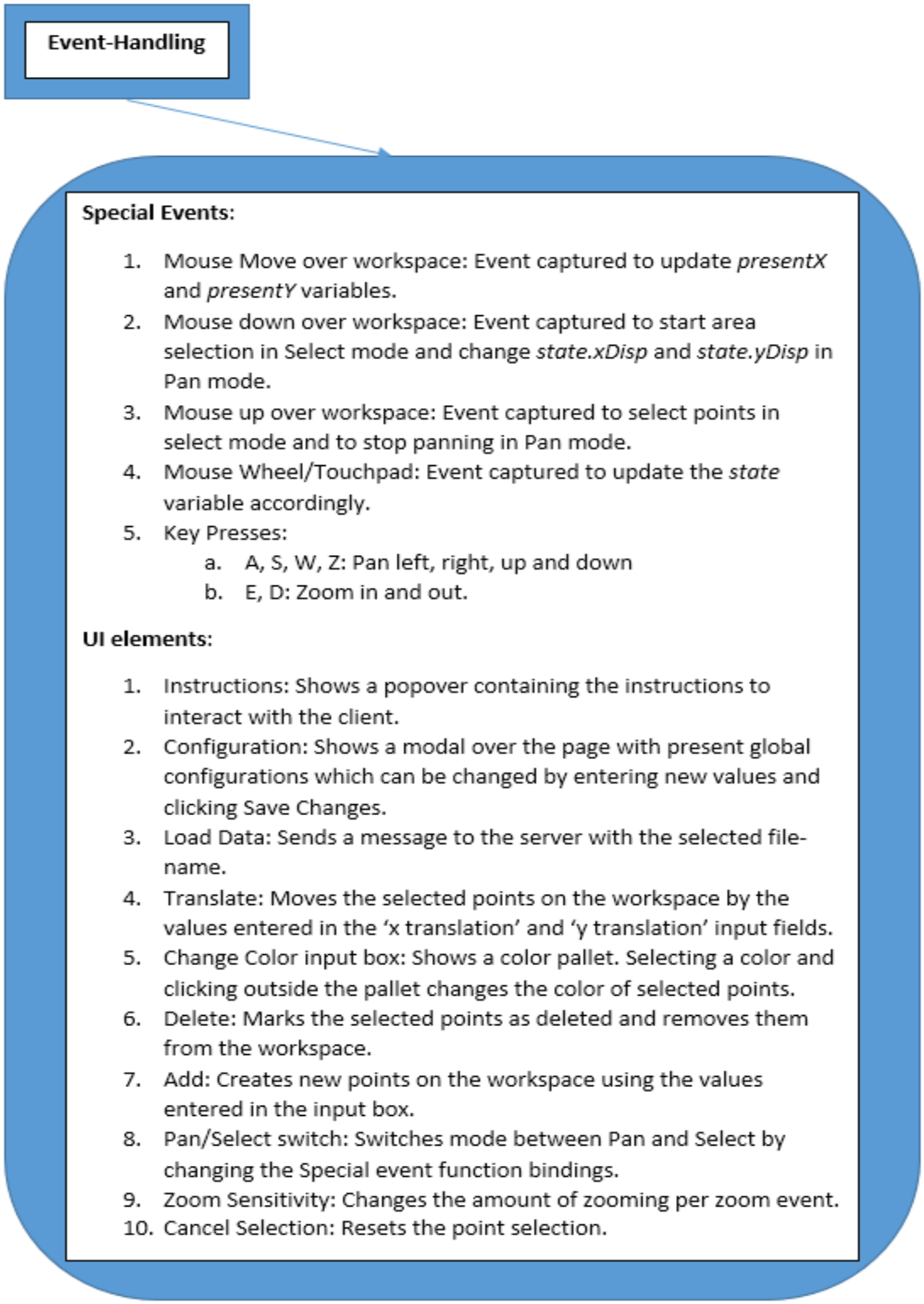

Figure 9: *Description of various user-events that trigger functions on the client*

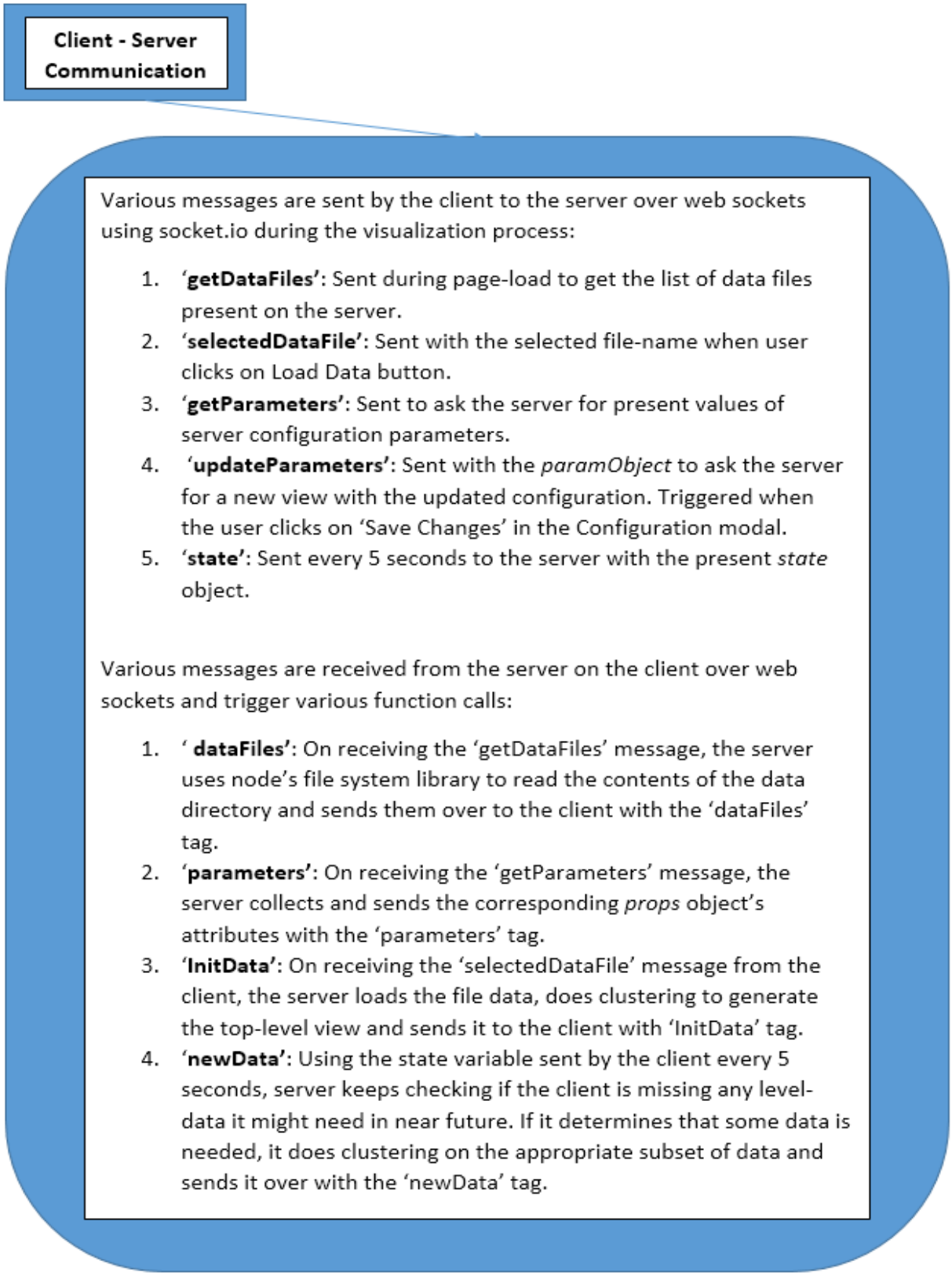

Figure 10: *Details of messages passed between the server and client and their function.*

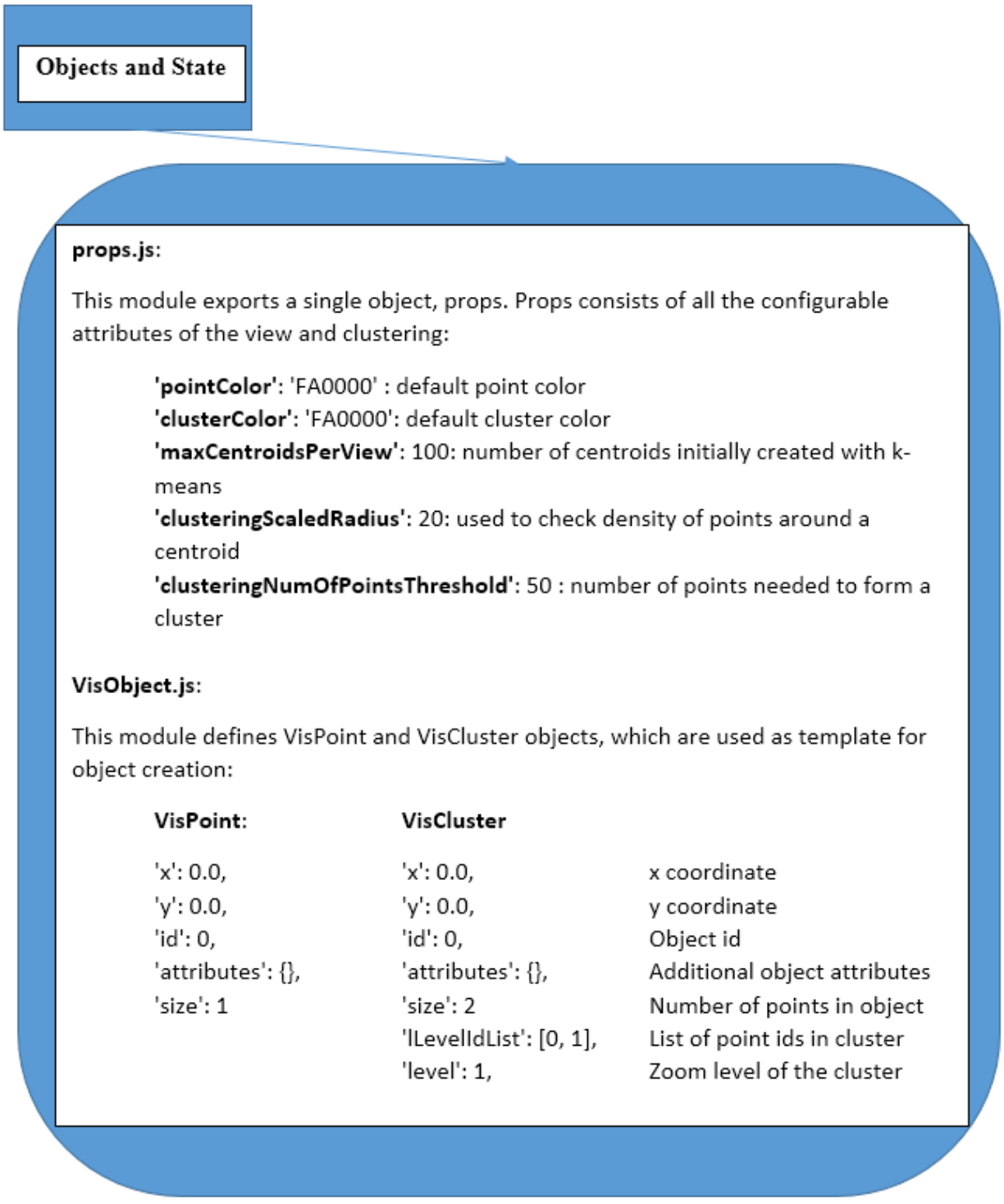

Figure 11: *Description of the Props and VisObject modules on the server.*

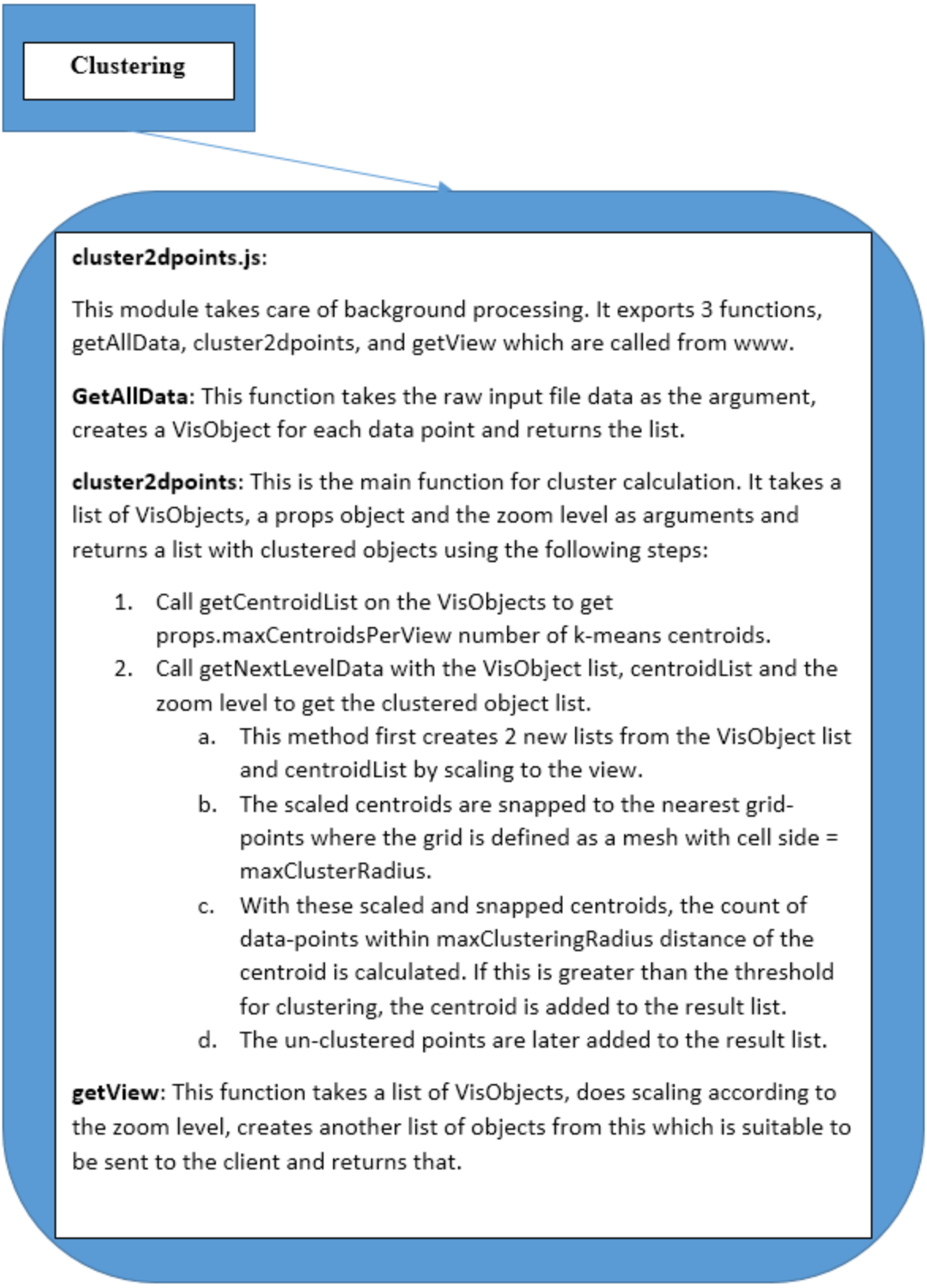

Figure 12: *Description of the data processing done on the server to create the views.*

4. Challenges

Here we will list down a few of the challenges we faced during the design and implementation of Visliu.

a. **Smooth zooming :** Changing the view in coarse steps was easy since the view only needs to be updated to the next level step determined by the server. To achieve smooth transition, we had to implement an update on points for every scroll event received from the client. The state object with a high enough refresh frame rate of the workspace made this possible. Though the scrolling works nicely on using a mouse with a scroll wheel, due to the variation in the touchpad hardware on different machines, the zoom speed is not consistent across browsers or machines. We have tried to make-up for that partially by providing an option for zoom sensitivity in the header options, a truly consistent zooming experience cannot be achieved without some kind of support from the browser to get a normalized scroll amount value which seems to be missing at the moment.

b. **Clustering :** We tried multiple clustering methods to create clusters for the visualization and there was always a trade-off between speed and quality. Since Visliu is an interactive system, we could not give up on speed much in favor of better quality. In the end, we came up with our own clustering method which uses a combination of k-means, grid alignment and threshold point density to get good clusters on our test data-sets without giving up on the ability of the system to calculate these in near real-time in response to user actions. Pre-calculating nearby clustering levels and keeping them cached on the client also goes a long way in hiding this computation.

c. **Network Communication :** Initially with ajax for client-server communication, the network communication was the major bottleneck to a responsive visualization system. Ajax is not just slow, but hard to manage. With some research we came across the recently developed websocket protocol which still runs on TCP but removes the large overhead from AJAX. Visliu now uses websockets though socket.io library.

d. **Javascript and asynchronous function calls :** Javascript was a blessing in the sense that it was fast enough for what we wanted to do and worked both on the client as well as the server, but the non-blocking I/O model of node created a lot of transient errors during the development which were pretty hard to debug.

5. Applications

While Visliu was started with an aim to explore the capabilities and challenges of interactive visualization on the web and without any particular application in mind, along the way we have discovered multiple areas and occasions on which it can be used: medical reports, financial data, as an educational tool, sensor data, and 2D simulation data.

One application that stands out is for pre-analysis of experiment results **:** Many experiments and research projects generate two-dimensional data as intermediate or final results which need to be analyzed multiple times for manually tuning and improving the experiment. Visliu can be used as a quick way to inspect these results both in the larger context as well as to find patterns in specific parts of the results.

With automatic clustering, Visliu has the capability to let the user work with much larger datasets than normally supported by web visualization systems. Though locally installed visualization software like Tableau[22] still hold the edge in terms of being able to use all of a system's resources with native code to create a wide-variety of pretty complex visualizations, with a few powerful servers and a fast network, Visliu could compete with locally installed visualization softwares on low power client machines like tablets, and ultra-thin notebooks that are gaining popularity over more-powerful but bulky machines. The ability to visualize and interact with point-space data can be provided to everyone in an organization or even freely online.

6. Code and Public access link

The code for Visliu is publicly available at :
https://github.com/shaqal/visel

The working app has been hosted with Heroku on an Amazon server at:
https://immense-atoll-3270.herokuapp.com

Future Work

While the system works right now, it is not really production ready. Here are a few improvements that can be made to it to get it to work at full potential:

1. **Parallel Clustering with MPI:** While Visliu works with a maximum of around 20k points right now, which is already more than most web visualization systems can handle, the biggest bottleneck is the single thread clustering with javascript on the node server that gets too slow above 20k points to be made-up for by our level caching strategy. If we could offload the clustering to a program implemented in C with MPI, it can utilize all the cores of the server or even multiple servers to get Visliu working with much larger datasets. The MPI program can keep using our clustering strategy or use a more compute intensive, but better hierarchical clustering strategy.

2. **Rendering with a variable frame-rate:** Visliu does a lot of processing on the client and although browsers are more powerful now than they have ever been, freeing up load on the client machine allows for faster response times to user-events and an overall better user experience. Frame-rates can be adjusted using the user interaction history for the last few minutes.

3. **User-defined Operations:** We expect future use of Visliu in two forms. First is for just a visual inspection of the data to detect patterns and derive insights. Second is to actually perform operations on the data to experiment and try to reach some kind of a goal. With the few basic operations Visliu provides right now, it falls short of its full potential for the second kind of use. Adding the ability for the user to define their own operations by providing them with the basic constructs to define these operations will make Visliu pretty useful for the second kind of use-case.

4. **Input-files**: Adding the ability for Visliu to take input files from Cloud-storage services like S3, Dropbox and BigTable will add a great deal to the usability of the system.

Bibliography

[1] David Marr, 2010. Vision: A Computational Investigation into the Human Representation and Processing of Visual Information. Edition. The MIT Press.

[2] Canvas API - Web API Interfaces | MDN. 2015. Available at: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API.

[3] SVG | MDN. 2015. Available at:https://developer.mozilla.org/en-US/docs/Web/SVG.

[4] WebGL | MDN. 2015. Available at:https://developer.mozilla.org/en-US/docs/Web/WebGL.

[5] D3.js - Data-Driven Documents. 2015. Available at: http://d3js.org/. .

[6] Google Charts - Google Developers. 2015. Available at: https://developers.google.com/chart/.

[7] Highcharts. 2015. Available at: http://www.highcharts.com/.

[8] three.js - Javascript 3D library. 2015. Available at: http://threejs.org/.

[9] Bokeh. 2015. Available at:http://bokeh.pydata.org/en/latest/.

[10] Datawrapper. 2015. Available at:https://datawrapper.de/.

[11] Flot: Attractive JavaScript plotting for jQuery. 2015. Available at: http://www.flotcharts.org/.

[12] Lieberman, H, (1994). A multi-scale, multi-layer, translucent virtual space : Information Visualization. In IEEE Conference. San Juan, Puerto Rico, 27 August. USA: The Printing House. 124-131.

[13] Perlin, K and Fox, D (1993). Pad: an alternative approach to the computer interface.. In Computer graphics and interactive techniques . Anaheim, CA, USA, 1 September. New York: ACM New York, NY. 57-64.

[14] Sutherland, Ivan E. "Sketch pad a man-machine graphical communication system." Proceedings of the SHARE design automation workshop 1 Jan. 1964: 6.329-6.346.

[15] Bederson, B, 1996. Pad++: A zoomable graphical sketchpad for exploring alternate interface physics.. Journal of Visual Languages & Computing, 32, 3.

[16] Welz, G, 1995. Zooming Through Information Space on PAD++. Peripheral Visions, 1, 8.

[17] Bederson, B, Stead, L & Hollan, J 1994. ‘Pad++: Advances in multiscale interfaces.’ In Conference companion on Human factors in computing systems. Boston, MA, USA, 28 April. New York: ACM New York, NY, USA, pp.315-316.

[18] Bederson, Benjamin B et al. "Zooming web browser." Electronic Imaging: Science & Technology 25 Mar. 1996: 260-271.

[19] Furnas, George W, and Xiaolong Zhang 1998 'MuSE: a multiscale editor',Proceedings of the 11th annual ACM symposium on User interface software and technology, 1(1), pp. 107-116..

[20] Prezi - Presentation Software. 2015. Available at: http://www.prezi.com/.

[21] Bade, R, Schlechtweg, S and Miksch, S (2004). Connecting time-oriented data and information to a coherent interactive visualization. In SIGCHI Conference on Human Factors in Computing Systems. Vienna, Austria, 25 April. New York: ACM New York, NY, USA . 105-112.

[22] Business Intelligence and Analytics | Tableau Software. 2015. Available at:http://www.tableau.com/.