Data Visualization in Virtual Reality

Directed Research Project under Dr. Saty Raghavachary

University of Southern California

Ishaan Vasant

1. Introduction

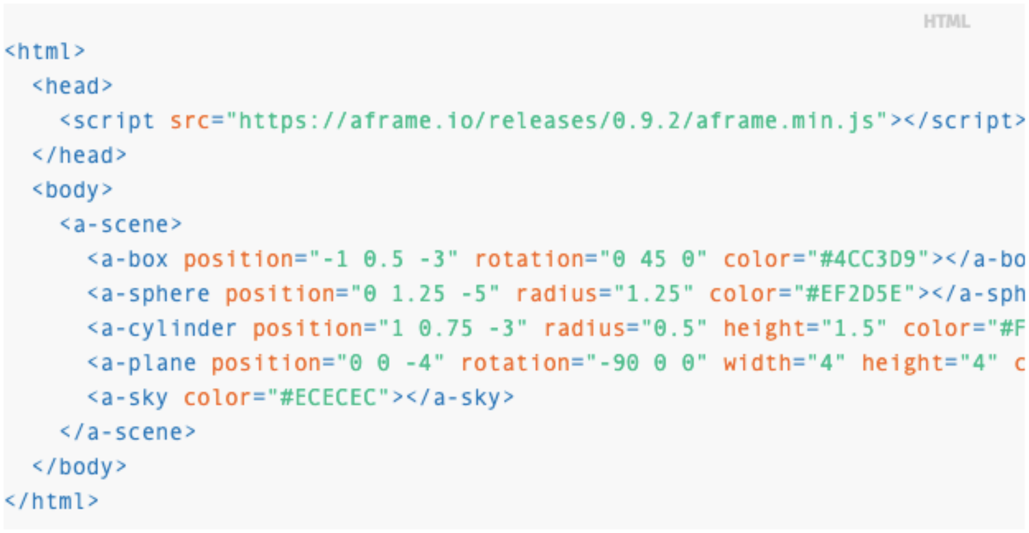
Data visualization, simply put, is the graphical representation of data. It has become a massive and integral part of data science today. Key stakeholders respond better to clean and clear visuals as opposed to raw data, as they help in understanding information quickly and effectively. There are a large number of visualization libraries currently in use with matplotlib (Python) and ggplot2 (R) being the standouts. However, these libraries have not yet made a significant breakthrough into the virtual reality space. As technology makes promising strides into virtual and augmented reality, it is essential that users can experience data visualizations in AR/VR as well. With this in mind, the objective of this project is the exploration and creation of data visualizations in virtual reality.

2. WebVR

The goal is to make it easier for everyone to get into VR experiences, no matter what device they have. WebVR is picked as the optimal medium for this very reason. WebVR is an experimental JavaScript application programming interface (API) that provides support for virtual reality devices. It is an open specification that also makes it possible to experience VR in a browser. One needs two things to experience WebVR: a headset and a compatible browser.

3. A-Frame

A-Frame is a web framework for building virtual reality (VR) experiences. A-Frame is based on top of HTML, making it simple to get started. But A-Frame is not just a 3D scene graph or a markup language; the core is a powerful entity-component framework that provides a declarative, extensible, and composable structure to three.js. A-Frame can be developed from a plain HTML file without having to install anything. Alternatively, one can create an .html file and include A-Frame in the <head> as shown in Fig 1.



```html
<html>
  <head>
    <script src="https://aframe.io/releases/0.9.2/aframe.min.js"></script>
  </head>
  <body>
    <a-scene>
      <a-box position="-1 0.5 -3" rotation="0 45 0" color="#4CC3D9"></a-bo
      <a-sphere position="0 1.25 -5" radius="1.25" color="#EF2D5E"></a-sph
      <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5" color="#F
      <a-plane position="0 0 -4" rotation="-90 0 0" width="4" height="4" c
      <a-sky color="#ECECEC"></a-sky>
    </a-scene>
  </body>
</html>
```

Fig 1: Example HTML file with A-Frame

3.1 Key Features of A-Frame

- VR Made Simple: A <script> tag and an <a-scene> is all that needs to be included. A-Frame will handle 3D boilerplate, VR setup, and default controls. No installation or build steps are required.

- Declarative HTML: HTML is easy to read, understand, and copy-and-paste. Being based on top of HTML, A-Frame is accessible to everyone - web developers, VR enthusiasts, artists, designers, educators, makers.

- Cross-Platform VR: VR applications can be built for Vive, Rift, Windows Mixed Reality, Daydream, GearVR, and Cardboard with support for all respective controllers. Even if one does not have a headset or controllers, A-Frame can still be used as it works on standard desktop and smartphones.

- Performance: A-Frame is optimized from the ground up for WebVR. While A-Frame uses the DOM, its elements don't touch the browser layout engine. 3D object updates are all done in memory with little garbage and overhead. The most interactive and large scale WebVR applications have been done in A-Frame running smoothly at 90fps.

- Visual Inspector: A-Frame provides a handy built-in visual 3D inspector. One can open up any A-Frame scene, hit <ctrl> + <alt> + i, and fly around to peek under the hood!

- Components: Hit the ground running with A-Frame's core components such as geometries, materials, lights, animations, models, shadows, positional audio, text, and controls for most major headsets. To add to that, there are hundreds of community components including environment, state, particle systems, physics, multiuser, oceans, teleportation, super hands, and augmented reality.

4. D3.js

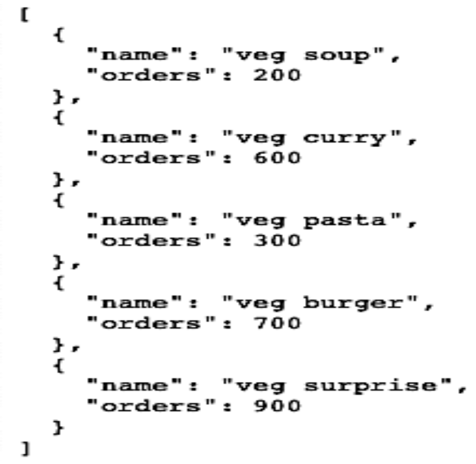
A-Frame is built on top of the DOM so most libraries and frameworks work including D3.js, which is a JavaScript library for manipulating documents based on data. D3 helps bring data to life using HTML, SVG (Scalable Vector Graphics), and CSS (Cascading Style Sheets). D3's emphasis on web standards gives the user the full capabilities of modern browsers without tying themselves to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM (Document Object Model) manipulation. For example, one can use D3 to generate an HTML table from an array of numbers. Or, use the same data to create an interactive SVG bar chart with smooth transitions and interaction.

D3 is not a monolithic framework that seeks to provide every conceivable feature. Instead, D3 solves the crux of the problem: efficient manipulation of documents based on data. With minimal overhead, D3 is extremely fast, supporting large datasets and dynamic behaviors for interaction and animation. D3's functional style allows code reuse through a diverse collection of official and community-developed modules.

D3 employs a declarative approach, operating on arbitrary sets of nodes called selections. D3 provides numerous methods for mutating nodes: setting attributes or styles; registering event listeners; adding, removing or sorting nodes; and changing HTML or text content. These suffice for the vast majority of data visualization needs.

5. Implementation of Visualization I

The objective of this visualization is to read data from a URL that contains a static JSON file and create a bar chart in VR. The initial point of the camera would be at the center of the chart with the various bars placed around it. Two files would be required to construct this visualization – an HTML file that defines the body of the page using A-Frame entities, and a Javascript file that performs the visualization after reading the data. The JSON file that contains the data to be visualized is depicted in Fig 2.

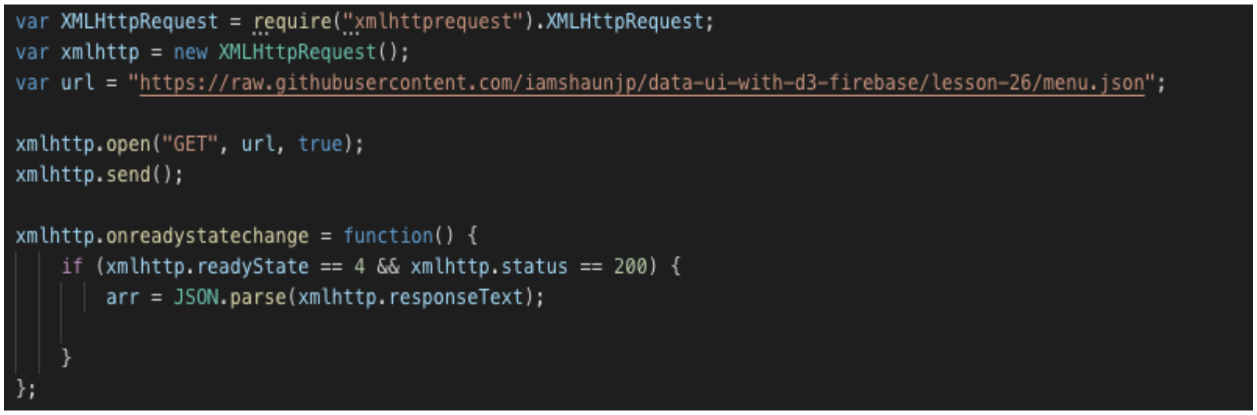


```
[
  {
    "name": "veg soup",
    "orders": 200
  },
  {
    "name": "veg curry",
    "orders": 600
  },
  {
    "name": "veg pasta",
    "orders": 300
  },
  {
    "name": "veg burger",
    "orders": 700
  },
  {
    "name": "veg surprise",
    "orders": 900
  }
]
```

Fig 2: Static JSON file

5.1 XMLHttpRequest (XHR)

In order to read data like the JSON file above from a URL, it is necessary to use XMLHttpRequest. XMLHttpRequest is a built-in browser object that allows to make HTTP requests in JavaScript. Despite having the word "XML" in its name, it can operate on any data, including JSON, HTML or plain text. One can upload/download files, track progress and use XMLHttpRequest (XHR) objects to interact with servers. The user can retrieve data from a URL without having to do a complete page refresh. This enables a web page to update just part of a page without disrupting what the user is doing. XMLHttpRequest is used heavily in AJAX programming.



```
var XMLHttpRequest = require("xmlhttprequest").XMLHttpRequest;
var xmlhttp = new XMLHttpRequest();
var url = "https://raw.githubusercontent.com/iamshaunjp/data-ui-with-d3-firebase/lesson-26/menu.json";

xmlhttp.open("GET", url, true);
xmlhttp.send();

xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        arr = JSON.parse(xmlhttp.responseText);

    }
};
```

Fig 3: Code Fragment - XMLHttpRequest

In Fig 3 above, the following steps are carried out: -

1. An XMLHttpRequest object is created
2. The required URL is assigned to a variable
3. The XHR object is initialized with the 'GET' method and URL specified
4. The connection is opened and the request to the server is sent
5. After listening to the response, the response text is read
6. The response text is converted from string to JSON using JSON.parse and stored in a variable
7. The variable is used in the function that handles the data visualization

5.2 Browserify

After formulating the visualization function and linking the Javascript file with the HTML file, an error still occurred. The reason is the presence of the 'require' method in the first line of the code snippet. The 'require' method is used to include the module in the project and use it as the browser based XHR object. The issue is that while Node.js has the 'require' method defined, browsers don't. With Browserify, one can write code that uses 'require' in the same way that one would use it in Node. In order to use this functionality, these steps were followed: -

1. ‘xmlhttprequest’ is installed from the command line using npm
2. All the required modules are recursively bundled up starting at main.js (the working JS file) into a single file called bundle.js with the browserify command
3. A single <script> tag is included in the HTML file with bundle.js specified as the source

5.3 Visualization Function

The function that takes care of the DOM manipulation and data visualization gets the data from the XHR API. The data is pushed into arrays and is then scaled in order to fit the screen. The A-Frame ‘a-scene’ is selected and the ‘a-cube’ bars are appended to it. Following this, the various attributes of the bars such as the height, width, position and rotation were defined. The ‘mouseenter’ and ‘mouseleave’ animations were also defined using the ‘transition’ method – this facilitates the expansion and lighting up of a bar when hovered upon. Lastly, the labels that go underneath the bars in the visualization were composed, along with the required attributes. A snapshot of the final visualization is displayed in Fig 4.

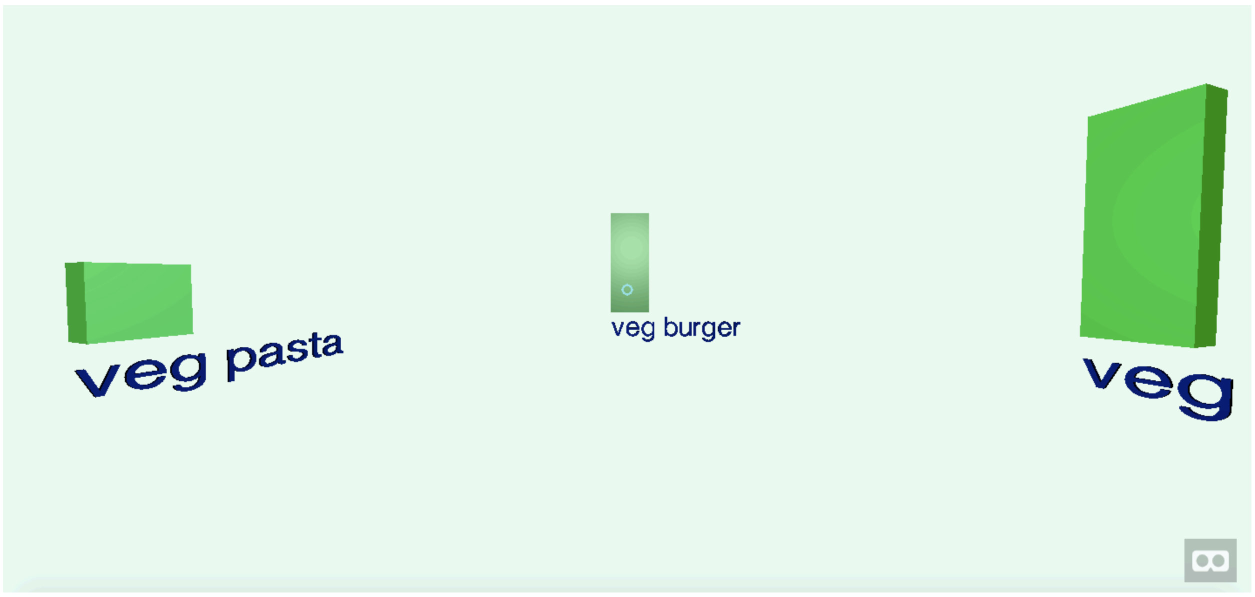


Fig 4: VR Visualization I

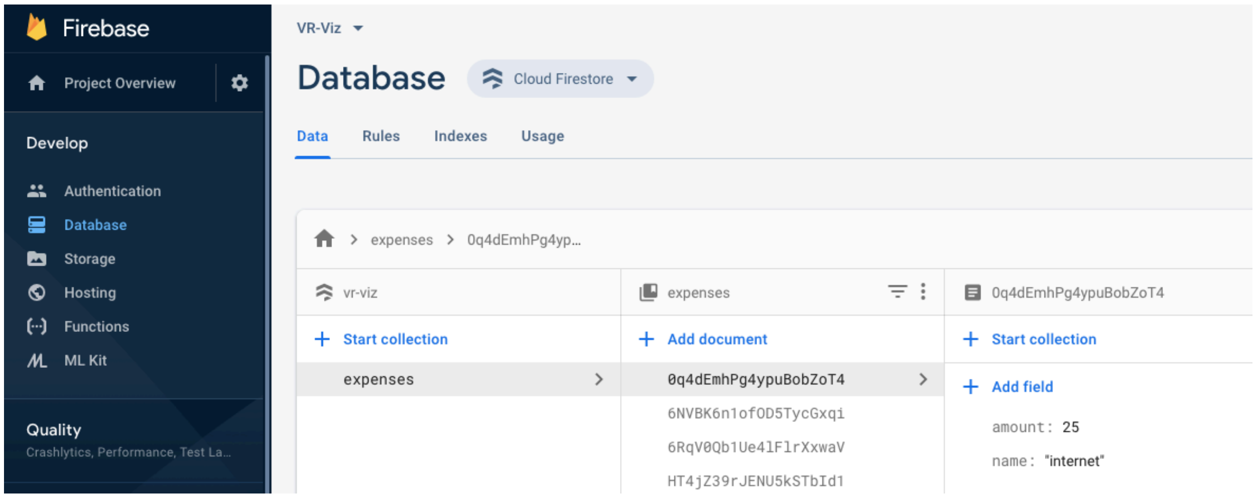
6. Implementation of Visualization II

The first visualization can only read data from a static JSON file. It would be more practical and useful if the visualization could read data that changes dynamically. This is the objective of the second visualization. It will borrow the basic structure and functions of the first visualization, while adding the ability to update itself in real-time.

6.1 Cloud Firestore

In this visualization, since the data is not being sourced from a static JSON file, the use of XMLHttpRequest and Browserify is not required. Instead, the data is obtained from Google Firebase's Cloud Firestore. Cloud Firestore is a flexible, scalable NoSQL cloud database for mobile, web, and server development from Firebase and Google Cloud Platform. It keeps the data in sync across client apps through real-time listeners and offers offline support for mobile and web so that the user can build responsive apps that work regardless of network latency or Internet connectivity.

Cloud Firestore uses data synchronization to update data on any connected device. However, it's also designed to make simple, one-time fetch queries efficiently. The Cloud Firestore data model supports flexible, hierarchical data structures. It stores data in documents, organized into collections. Documents can contain complex nested objects in addition to sub collections. For this visualization the 'expenses' collection with multiple documents is used, as seen in Fig 5.



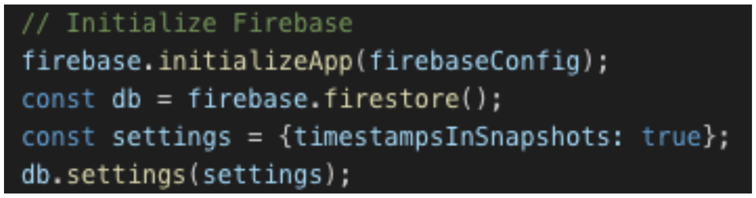
expenses

0q4dEmhPg4ypuBobZoT4
6NVBK6n1ofOD5TycGxqi
6RqV0Qb1Ue41FlrXxwaV
HT4jZ39rJENU5kSTbId1

amount: 25
name: "internet"

Fig 5: Cloud Firestore Database

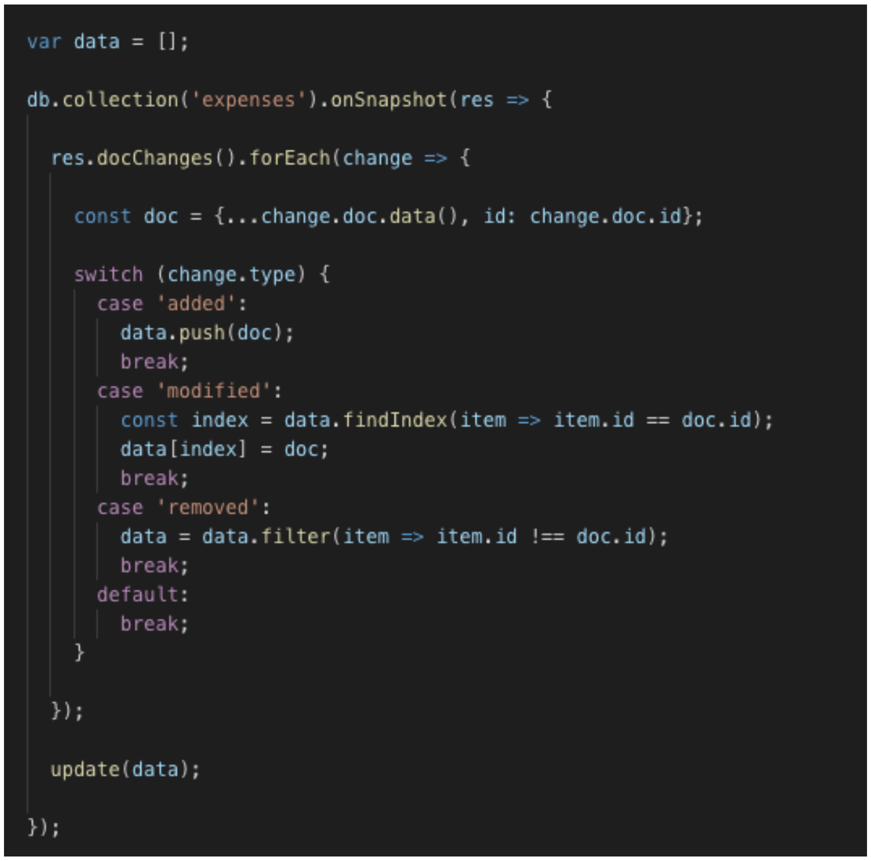
For the visualization to access the data within the Cloud Firestore database, a <script> is included in the HTML file. In it, the user's web app Firebase configuration details are specified. The database is initialized and accessed as 'db' using the code in Fig 6.



```
// Initialize Firebase
firebase.initializeApp(firebaseConfig);
const db = firebase.firestore();
const settings = {timestampsInSnapshots: true};
db.settings(settings);
```

Fig 6: Code Fragment - Firebase Initialization

The code in Fig 7 below explains how the program deals with changes in data. Each entry in the collection is of type ‘added’, ‘modified’ or ‘removed’. The switch statement in the code deals with each of these cases by manipulating the input data. Once the data is redefined, it is passed into the function that handles the visualization.



```
var data = [];

db.collection('expenses').onSnapshot(res => {

  res.docChanges().forEach(change => {

    const doc = {...change.doc.data(), id: change.doc.id};

    switch (change.type) {
      case 'added':
        data.push(doc);
        break;
      case 'modified':
        const index = data.findIndex(item => item.id == doc.id);
        data[index] = doc;
        break;
      case 'removed':
        data = data.filter(item => item.id !== doc.id);
        break;
      default:
        break;
    }

  });

  update(data);

});
```

Fig 7: Code Fragment - Switch Statement for Dynamic Data Manipulation

6.2 Visualization Function

As alluded to earlier, the main designing function of the second visualization is borrowed from the first. The scene, bars and labels are constructed in a similar fashion with the same attributes. One added feature is that the color of the bars is now based on a condition – purple if the value is greater than 50, and blue otherwise. However, the key added component in this visualization is the text annotation, which is fashioned as a second label that is seen above each of the bars. These annotations represent the height of the bar and appear only when the mouse hovers over a bar. This allows the viewer to comprehend the relative values and visualize the data more effectively.

The same visualization was first created in regular D3, without the use of A-Frame VR. This was done to test the interaction with the Cloud Firestore database and to be used as a reference for the VR visualization. The 2D visualization is displayed in Fig 8.

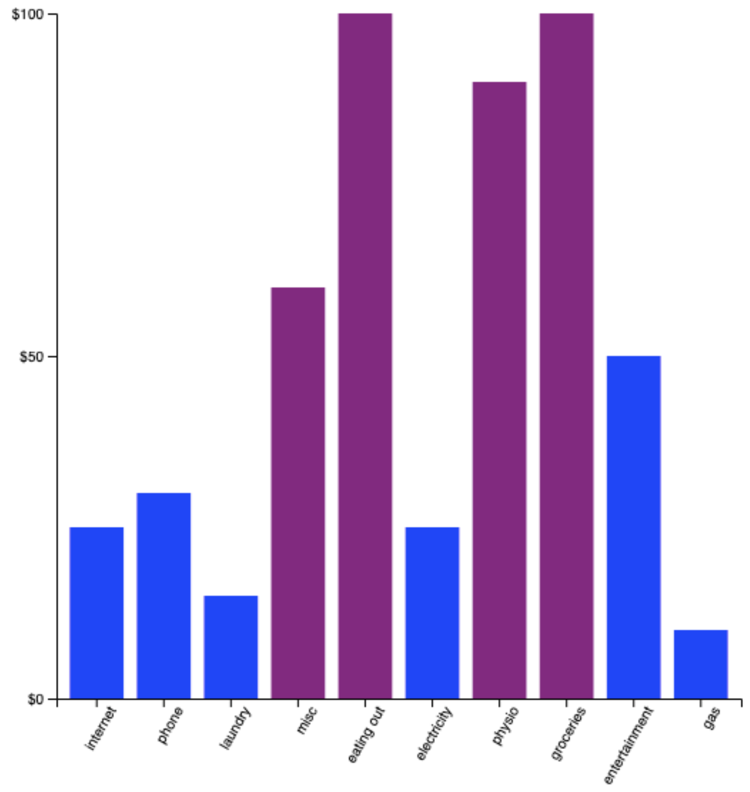


Fig 8: 2D version of Visualization II

Once it was observed that the above visualization updated itself in real-time, it was translated to the virtual reality space as presented in Fig 9 below: -

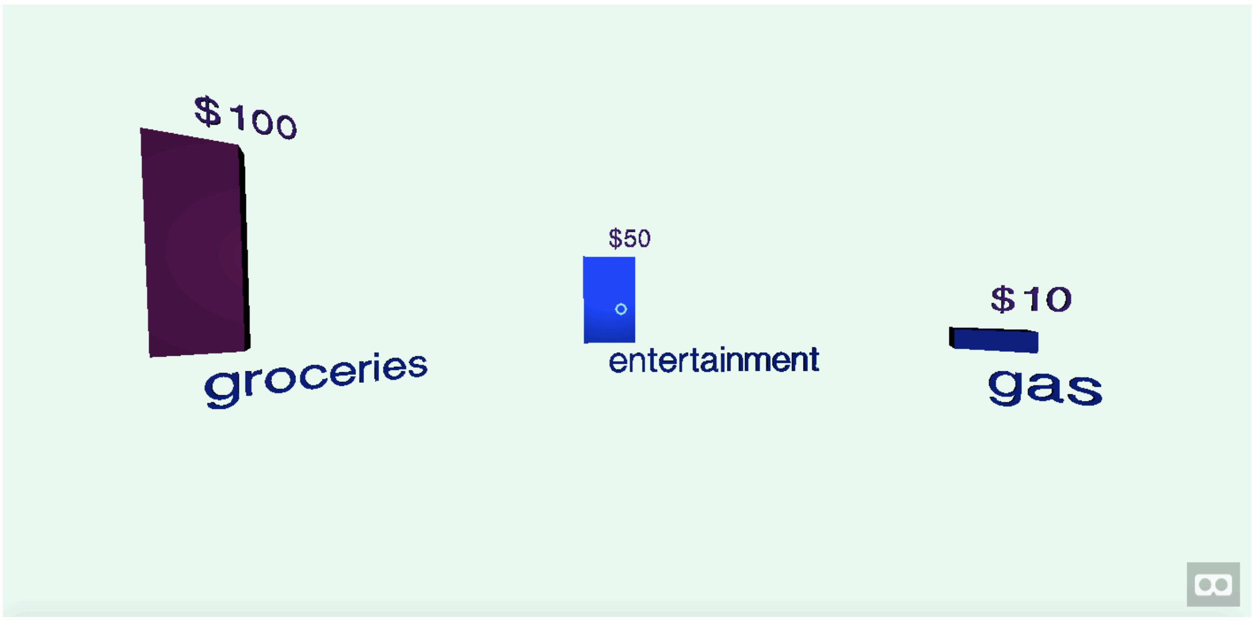


Fig 9: VR Visualization II

7. Conclusion

The objective of this project was to study the possibilities of visualizing data in virtual reality. To achieve this, the key components that needed to be understood were WebVR, A-Frame and D3. WebVR was the medium through which the user could experience the visualizations, A-Frame provided the tools to set up the VR scene and entities, while D3 enabled DOM manipulation and definition of various visualization attributes.

In Visualization I, the data was read from a static JSON file using XMLHttpRequest and Browserify, and its corresponding bar chart was generated. Visualization II emulated the first in basic functionality but sourced the data from Google Firebase's Cloud Firestore. Along with being able to read dynamic data from a database, Visualization II also had custom annotations that appeared when the mouse hovered over a bar. This allowed the user to understand the values and relative heights of the bars, thereby experiencing an effective visualization of the data. Therefore, it was seen that with the right tools and approach, data can be effectively visualized in the virtual reality space.

8. Future Work

This project only dealt with bar charts using A-Frame's 'a-cube' entity. A lot of different types of visualizations can be built using various other entities. For example, pie charts and donut charts can be crafted using 'a-cylinder' and 'a-torus' entities respectively. The current visualizations are demonstrated on the web using an HTML file from a local directory. In future, these can be published online as websites using the services of Google, GitHub, GoDaddy, etc. These would allow the ability to share visualizations across the internet and would also facilitate collaboration.

During the course of this project, it was discovered that there exists very little material on the subject at hand. Data visualization has not yet made a serious entrance into the virtual and augmented reality domain. This research project, therefore, can serve a starting point in the conversation which can hopefully be taken forward in the near future.

9. References

- https://webvr.info/
- https://aframe.io/docs/0.9.0/introduction/
- https://blockbuilder.org/enjalot/1fd196cd99f8d58a56d3
- https://raw.githubusercontent.com/iamshaunjp/data-ui-with-d3-firebase/
- https://javascript.info/xmlhttprequest
- http://browserify.org/
- https://d3js.org/
- https://firebase.google.com/docs/firestore/