ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

Virtual Reality Data Dashboard

Autor: Francisco Aguilar Hidalgo
Tutor: Jesús M. González Barahona

Curso académico: 2016/2017

Proyecto Fin de Carrera

Virtual Reality Data Dashboard

Autor: Francisco Aguilar Hidalgo
Tutor: Jesús M. González Barahona

La defensa del presente Proyecto Fin de Carrera se realizó el día
2017, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a 2017

Una imagen dice más que mil palabras.

Acknowledgements

En primer lugar, quiero agradecer la oportunidad que Jesús M. Barahona me ha ofrecido para realizar este proyecto y su dedicación en el mismo. Ha sido un gran apoyo durante todo el camino recorrido en este proyecto siempre dándome los mejores consejos posibles. También agradecer a Adrián Alonso y David Moreno del departamento y creadores de THREEDC y VBoard-UI por su colaboración y ayuda.

En segundo lugar, quiero agradecer a todos mis amigos que me han ayudado a lo largo de este camino. Ofreciéndome su apoyo, cariño y ánimos de forma incondicional. Muchos de ellos los he conocido en la universidad y ahora forman una parte muy importante de mi vida. Juntos hemos pasado grandes experiencias durante la carrera y después y espero que siga siendo así por mucho tiempo. A mis amigos de siempre que son un gran apoyo, con ellos he madurado a través de los años y compartido muchas primeras experiencias. En especial agradecer a mi novia Elena por su paciencia y comprensión conmigo ayudándome en todo momento.

Por último quiero agradecer a mi familia, ellos son los que me han guiado hasta aquí. Siempre han estado cuando les he necesitado y han hecho grandes sacrificios por mí. Me han transmitido todos sus valores y enseñado las cosas importantes de la vida. Para mí ellos son lo más importante y una gran parte de lo que soy se lo debo a ellos.

A todos vosotros os dedico mi proyecto.

Abstract

Nowadays data visualization is one of the main ways of analyzing them. The need to analyse large data-sets generated by the current society is very important to help in decision making. Currently the tools that allow data visualizations perform 2D graphics. The main form of diffusion is through web technologies, among other reasons, for its ease of use and accessibility.

The main objective of the project is the creation of a tool that allows creating visualizations of data with 3D graphics in a web environment using current web technologies. This tool also offers user interaction for data exploration and integration with a data analysis engine that enhances the ease of use of the tool. It also aims to offer the same 3D scene on a virtual reality enviroment to have a more inmersive experience. All the code and its examples are open source. The information is on the specific web of the project[1].

[1] https://fran-aguilar.github.io/

Resumen

Hoy en día la visualización de datos es una de las principales vías de análisis de los mismos. La necesidad de poder analizar la gran cantidad de datos generados por la sociedad actual es muy importante para ayudar en la toma de decisiones. Actualmente las herramientas que permiten realizar visualizaciones de datos realizan gráficos en 2D. La forma principal de difusión es mediante tecnologías web, entre otros motivos, por su facilidad de uso y accesibilidad.

El objetivo principal del presente proyecto es la creación de una herramienta que permita crear visualizaciones de datos con gráficos 3D en un entorno web utilizando las tecnologías web actuales. La herramienta también ofrece interacción con el usuario para la exploración de los datos y la integración con un motor de análisis de datos donde se potencia la facilidad de uso de la herramienta. También tiene como objetivo ofrecer esa misma escena 3D en un entorno de realidad virtual para tener experiencia más inmersiva. Todo el codigo y sus ejemplos son código abierto. La información está en la web especifica del proyecto[2].

[2] https://fran-aguilar.github.io/

Contents

Figure index

Acronyms

API	**A**pplication **P**rogramming **I**nterface
VR	**V**irtual **R**eality
UI	**U**ser **I**nterface
3D	**T**hree**D**imensional
URJC	**U**niversidad **R**ey **J**uan **C**arlos I
HTML	**H**yper**T**ext **M**arkup **L**anguage

Chapter 1

Introduction

In this chapter we will describe the problem and introduce the project objetives and its context, in order to clarify its basis before diving into technical details.

1.1 Current context

Today's society is generating more and more data. The analysis of data is therefore more complex and the transformation process is difficult to highlight useful information in order to support decision making. One of the most common ways of facilitating data analysis has been through charts of different types such as pie, bar, or network diagrams. With modern operating systems, it facilitated the creation of simple visualizations from data with programs like Microsoft Excel. Then with the rise of the internet and better web browsers new libraries that allowed the visualization of charts of data appeared. These libraries were multiplatform, they did not need the installation of any software in our computer and they allowed a greater interaction with the user, examples of this are dc.js, 3d.js or highchart.

In recent years, with the bursting of the big data technology market, there has been a greater need for efficient representation systems that transform all that information into useful conclusions. The term data dashboard gains special interest because it allows us to visually see metrics about key points of a business

or a specific process. Mainly the charts used are in 2D but the people do not explore too much a 3D environment where you can rotate the charts or display these same data with 3D graphics. The last year an URJC student explore this field creating his library THREEDC that we will discuss later.

During these last years the use of virtual reality environment is also being explored for commercial purposes since these provide several advantages. A greater immersion as we are totally focused on the content shown in our virtual reality headset. The intensity of the experience is greater than in a 3D scenario since the user's behavior is directly linked to changes in the scene. One of the main reasons for this interest in virtual reality has been: betting of big companies with the launch of very low cost devices like Google CardBoard or Samsung VR. The effort of the standardization of virtual Reality technologies in browser with WebVR API. Also The demanding technical features required by a virtual reality environment such as a delay in response input <20ms or a high frame rate per second can be overcome by the processors of current smartphones makes it accessible by a larger audience.

1.2 Objetives

In this project we have as main objective the creation of a library for web browsers to create 3D charts of data. The library will be based on web standards so it is not necessary to install a plugin to our browser. By relying on web standards we also ensure that a very high percentage of users can use our application regardless of the operating system or web browser used. We will implement existing interactions in 2D charts as they are dynamic charts that depend on the user's actions. In the project we will also visualize our 3D data charts on a virtual reality environment. We want that our 3D scenes don't require additional code when we want to see it on virtual reality. To summarize we list all the objetives below. The project has the following basic objetives:

- Create 3D chart visualizations of data.

- The library must rely on web standards so no additional software must we installed for executing our software. We ensure that our scene are executed on different browsers and different Operating systems if they support the standards. We'll mainly rely on WebGL the standard of 3D graphics representation.
- A way of managing the chart system.
- Achieve functionality and performance similar to existing libraries with filter possibilities.
- Achieve a fast filter response. The user interaction must be as faster as possible and data filter time is important to user's experience.
- The feature to view any 3D scene created by our library on Virtual Reality without additional code.
- Render our charts on a existing scenario, or create a scene from scratch.

Chapter 2

Used Technologies

During this chapter we'll describe the technologies and concepts used through this project. We'll do examples of the most important to my project.

2.1 HTML

HTML[1] is a markup language based on tags in order to create web pages. It defines a basic structure and a code (the HTML code) for the definition of content (text, images, video and so on). Its great popularity is because it is a standard of W3C (World Wide Web Consortium) a organization dedicated to standardization of most of all web technologies and is the most used standard in web pages representation.
It's important to comment some of the new enhancements at its latest review HTML5. The step from HTML 4 to HTML 5 was important because many features was added .Some of these features are the following[2]:

- new semantic elements like <header>, <footer>, and lesssection>tags. These new tags helps search engines to identify the correct web page content.

- Canvas, a two-dimensional drawing surface that you can program with JavaScript.

- Video that you can embed on your web pages without resorting to third-party plugins.
- Geolocation API. whereby visitors can choose to share their physical location with a web application.
- Storage API. Persistent local storage without resorting to third-party plugins.
- Improvements to HTML Web forms.

It also brings new changes and standardization on the 3D technology field. The most important are canvas element and the WebGL standard these features provides to us a way of create 2D and 3D graphics. In addiction the latest versions of web browsers have the GPU's acceleration feature that greatly improve processing time of the graphics.
On the other hand new experimental API that are not standarized at 100 % such as positional audio API and Gamepad API or the webVR API is a signal of the effort that companies are making to bring 3D web sites and new experiences on the web.

2.2 JavaScript

JavaScript[3] is a interpreted programming language, implemented as a part of web browsers. It allows the creation of client-side scripts in order to interact with the user, controls the web browser in which the script is running, asynchronous comunication between client and server and dynamic modification of the structure and the content showing on the web page. The origins comes from Netscape company and its first release comes from its web browser in September 1995 under the name of LiveScript. It was renamed to the actual name at the 2.0B3 version of this web browser.
In November 1996 Netscape submitted it to the ECMA Internacional association in order to consider it a standard on the industry originating the first standardized

version of the language at the ECMA-262 specification (also known as ECMAScript) in June 1997. Actually the version of the standard is 5.1 released in June 2011. But JavaScript is not the unique implementation of this standard there are others like JScript or ActionScript.

One of the main problems of JavaScript comes historically from the differences between the JavaScript engines inside the web browsers because not all the versions of the different browsers has 100 % done the implementation of the standard and it comes the problem, the web developers create their scripts and test on a specific web browser but they have to consider a huge set of different web browsers and operating systems. They usually have to create specific code for specific browsers. Despite of that it is widely used among the web and it is one of the most popular languages right now.

We describe some of the main features at the list below:

- **Dynamic Types** The types of the data are associated with the actual value of a variable instead of its definition. There are no checks that verifies the type safety at runtime so it accelerates the edit-test-debug process and we don't need specific signatures for our functions.

- **Functions** In JavaScript the functions can accept a function as a parameter or return one, the variables defined on the function are restricted to the function's scope.

- **Prototype based** Its a type of object-oriented programming in which the code reuse is performed inherit existing properties from a old object. When you make a new object, you can select that should be its prototype.The *prototype link* is used only in retrieval. If we try to retrieve a property value from an object, and if the object lacks the property name, then JavaScript attempts to retrieve the property value from the prototype object. And if the object is lacking the property, then goes to its prototype object. This is called *delegation*.

- **Constructors** The way of creating constructors is to use a function with the reserved keyword new and on this function we create the object Inheriting properties and methods.

- **Expresive object literal notation** We can create an object simply listing their name value pairs. This notation is the precursor of the data exchange standard JSON (JavaScript Object Notation).

We can see at these examples below some of the key features of JavaScript:

```
//example of a function
function sum(x, y){
return x + y;
}
//a function as a variable
var sumfunction = function(x, y) {
return x + y;
};
var sumresult = sumfunction(1, 2);
//a function that takes anther function as a parameter
function performOperation(op, x, y){
return op(x, y);
};
//a function with an anonimous function inside
var displayClosure = function() {
var count = 0;
return function () {
return ++count;
};
}
//inicialize and declares the variable count
//returns a function that increments the count variable
var inc = displayClosure();
inc(); // returns 1
inc(); // returns 2
var operationresult = performOperation(suma, 2, 2); //4
operationresult = performOperation(funcionSuma, 2, 2); //4
//dynamic typing
```

```
var operationresult = suma("a", 'b'); //"ab"
//Object literal notation
var position = { x: 15, y:10};
//adding a new property to the object
position.z = 18;
//Object's constructor
function Vector(x, y)
{
this.x = x;
this.y = y;
}
var v = new Vector(1, 2);
//we change vector prototype, all the objects with Vector's
    prototype
//are affected
Vector.prototype.min = function(){
return Math.min(this.x, this.y);
}
v.min // 1
```

2.3 WebGL

WebGL[4] is a web standard for 3D graphics representation, is a multi-plataform API that allow us to use a native implementation of OpenGL ES 2.0 a simplified variant of OpenGL designed for embedded systems. For web browser this API will be exposed as a set of JavaScript programming interfaces. Thanks to this native implementation we can render the graphics through PC's hardware so no additional software will be downloaded to show 3D content on a web page. In webGL we must define a context (a canvas region within our HTML) in which the 3D scene will be drawn.We only create that context in web browsers that supports HTML5. That region could be mixed with the rest of the page as another html element and being transparent for the end-user also we have the posibility of interact in a dynamic way with other elements on the page. At the figure 2.1

below we can see the set of initial OpenGL ES primitives. This primitive are the same as the WebGL.

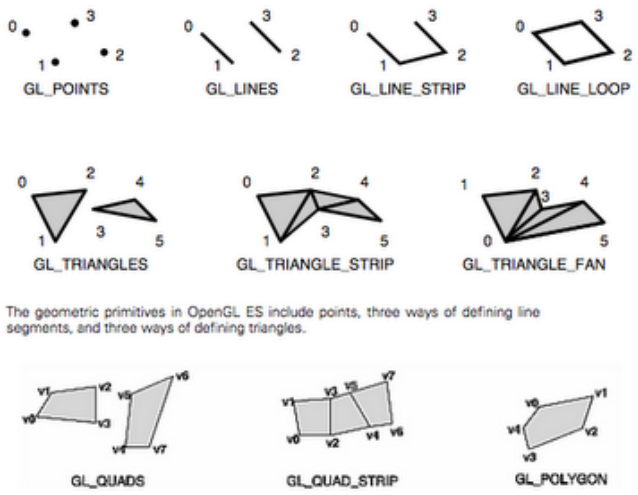


Figure 2.1: OpenGL ES primitives

Its first release was on March 2011 and the current stable version is 2.0 released on January 2017. In these years a lot of libraries based on WebGL and game engines based on WebGl appear in order to make easier the work of building 3D scenes. Today most of web browsers for desktop, smart-phones and tables have an implementation of webGL at least its first version. In the figure 2.2 we can see a chart including usage relative of each browser and webGL support.Red color means no support for that technology, Light green means partial support and green means full support.

In order to render (create a 2D image from a scene) WebGL into a page, an application must, at a minimum, perform the following steps:

1. Create a canvas element.

2. Obtain a drawing context for the canvas.

3. Initialize the viewport.That is the 2D rectangle used to project 3D scene.

4. Create one or more buffers containing the data to be rendered (typically vertices).

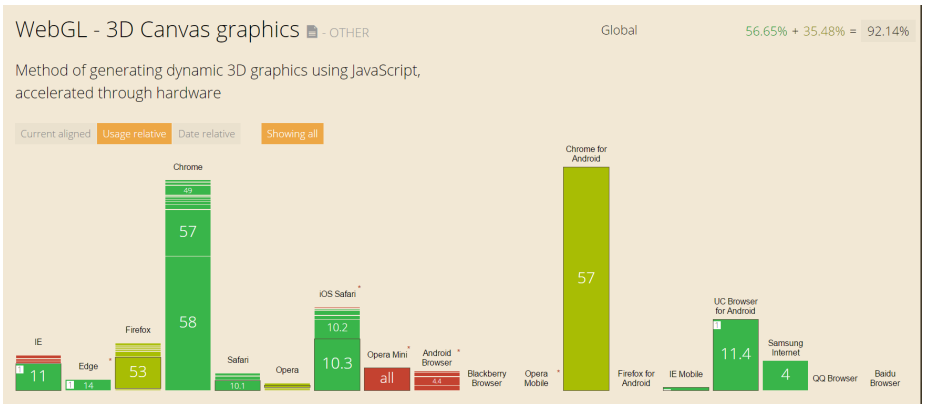


Figure 2.2: Implementation of WebGL in web browsers

5. Create one or more matrices to define the transformation from vertex buffers to screen space.

6. Create one or more shaders to implement the drawing algorithm.

7. Initialize the shaders with parameters.

8. Draw.

This example[1] is too long to include all your code in memory so in footnote section we put a URL of this example. The output of this example is the shown in figure 2.3. We can see that we work on a low level of abstraction and we need to find a library that simplifies some WebGL task and allows us higher level of abstraction.

[1] https://github.com/tparisi/WebGLBook/blob/master/Chapter%201/example1-1.html

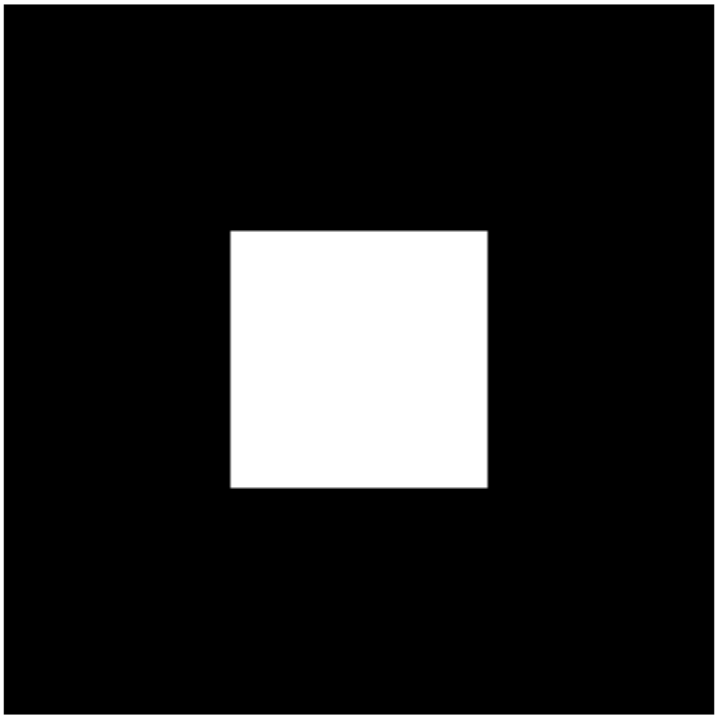


Figure 2.3: Basic Example of WebGL

2.4 Virtual Reality

Virtual reality usually refers to a computer technology that uses several devices and tries to evoke a sensation of reality within virtual scene using realistic graphics and sounds. The main device is virtual reality headset 2.4.

Although at the beginning it was thought exclusively for bringing more immersive scenarios and experiences in video-games quickly other fields such as medicine, asset management industry, relaxing therapies or tourism adapt this technology. As a conclusion we can say that is useful in several fields because we can make a realistic reality with high benefits in education or training.

In recent years with the launch of devices such as Google CardBoard or Samsung VR with very economical prices which allow bringing virtual reality experiences to our Smartphones. With the standardization of the first versions of WebVR, the technology in web environments is further consolidated. Initiatives exist to show contents through immersive experiences to the users for commercial

(a) Google Cardboard (b) HTC Vive

Figure 2.4: VR Headsets

purposes since as we have commented with only a Smart-phone we can have a really good Virtual reality experience. Some examples of this are: Volvo – XC90 Test Drive[2] where we drive this car in different scenarios or New York Times - Displaced [3] is a documentary video that tells the story of children away from home. Both of this requires the installation of a mobile application to run it. Finally https://vr.with.in/ an example of virtual reality on the web that shows an interface for navigation between 360º videos.

We conclude that Virtual Reality comes to bring us inmersive scenarios and it is gaining wider acceptance between users.

2.5 WebVR

WebVR[5] is an experimental JavaScript API that provides support for virtual reality devices such as HTC Vive, Oculus Rift or Google Cardboard in a web browser. The main objectives of the API are:

- Detect virtual reality devices available.
- Gather devices capabilities.

[2] https://www.youtube.com/watch?v=Wuln2bJkp1k
[3] http://dragons.org/creators/imraan-ismail/work/the-displaced/

- Show images on the device with a stable frame rate.

The biggest drawback of that technology was the lack of support for the browsers (mainly Mozilla Nightly, and special Chromium builds are the unique that support WebVR) but with the recent introduction of WebVR 1.0 and 1.1 on March and April, 2016 respectively, this shows that WebVR is a maturing platform that is on its way to gaining wider acceptance.

2.6 Three.js

Three.js[6] is one of the most popular WebGL based library to build scenes in a easy form. It provides a higher level of abstraction than WebGL and simplifies tasks of rendering, creating meshes, data loading and export and import models. It brings debug utilities and a high-set of math functions to work with frustum, matrices, quaternion and more. There are a lot of plug-ins made by the community also. On his web there are more than 150 examples of using the library. At the figure 2.5 below we can see examples of the library.
It is specially important in this project because the main technology used through this project is A-Frame and it is built on top of three.js. So basic concepts of the library are needed when we extend A-Frame like in our solution.

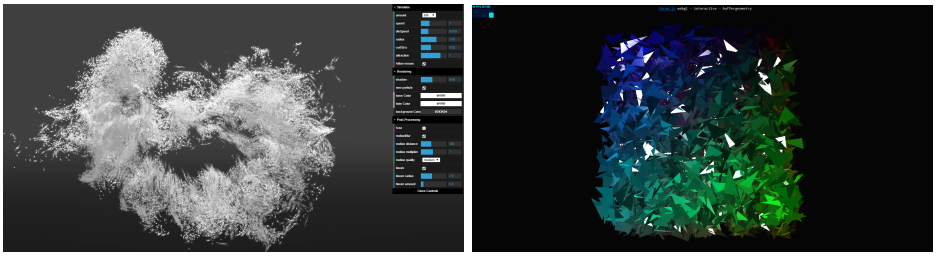


Figure 2.5: Three.js examples

2.7 A-Frame

A-frame[7] is a web framework for creating virtual reality experiences in a quick and simple manner you don't have anything to install to begin with. It is built on

the top of the popular 3d framework three.js framework. We can create scenes just only with HTML code and we abstract a lot of initial settings of the 3d scene for example the render loop, the vertices definition or handle texture loading. This framework has had a great acceptance by the VR community who improve and maintain the docs, create new issues and develop new components and submit them later to the A-frame registry. Another nice feature that is compatible with most libraries, frameworks and tools on JavaScript like angular, d3.js or jQuery. It also provides a handy built-in visual 3D inspector that allows us drag, rotate and scale the entities, copy and paste objects and see the results immediately. We can save those changes too.

A-frame uses entity-component-system pattern for code reuse and compose new entities. The use of this pattern is widely taken by video-games development the main idea is that every object of our scene is an entity which is a empty container by itself. Later we add different components which defines its appearance, behaviour and/or functionality and the system bring to us a global scope within a kind of component for example in a scene with several cameras we can detect the active camera through system of component camera.

For example, imagine we want to build a bicycle entity by assembling components so we define:

- geometry component that has the definition of appearance.
- material component that has the definition of bicycle color.
- position component that has information of bicycle's position.
- velocity component that has information about the direction, acceleration and velocity vectors in addition it has methods to start and stop.

At the picture 2.6 we can see a graphical example:

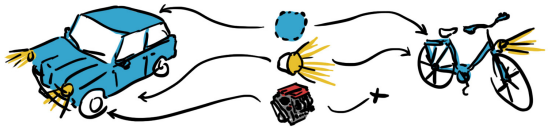


Figure 2.6: Abstract representation of Entity-Component Pattern

2.8 Crossfilter

Crossfilter[8] is a JavaScript library for exploring large multivariate dataset in the web browser. It supports extremely fast (less than 30ms) interaction with coordinated view, even with huge datasets containing a million or more records. Actually is not under active development but the author of the library considers that it is essentially complete. Even so we have chosen this technology to carry out the project because of its quickness in interactions and a great amount of data that can be used. Next we will explain more in detail the characteristics of the library and a basic example.

Before discussing a basic example to extract data from crossfilter we must know some basic concepts that will summarize these are: facts, dimensions and measures. The facts are each and every one of our records, for example if we have data from a source code repository, the commits table is a fact table. This table is the one we uploaded using JSON to crossfilter. A dimension in this context is a property of the data either existing or derived from several properties. The measure is ultimately an aggregated value from a dimension, in crossfilter these measures are achieved with the group and reduce methods.

The filtering feature in crossfilter works somewhat differently because when one dimension is filtered, all other data dimensions are affected in their calculations and groupings except the dimension which the filtering is performed on. In addition crossfilter filtering is stateful so future filters we apply are added to existing ones.

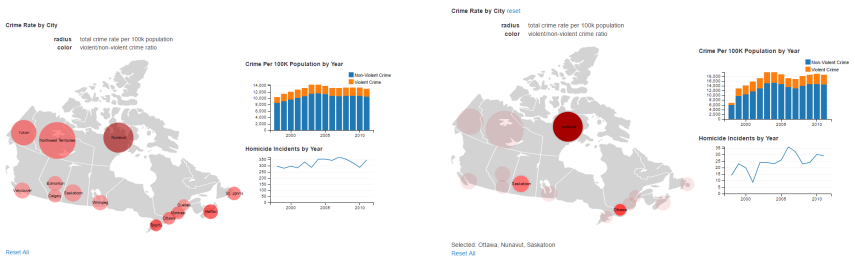
```
//we load the data with a Payments Table
var payments = crossfilter([
{ qty: 2, total: 190, tip: 100, type: "tab"},
{ qty: 2, total: 190, tip: 100, type: "tab"},
```

```
{ qty: 1, total: 300, tip: 200, type: "visa"},
{ qty: 2, total: 90, tip: 0, type: "tab"},
{ qty: 2, total: 90, tip: 0, type: "tab"},
{ qty: 2, total: 90, tip: 0, type: "tab"},
{ qty: 1, total: 100, tip: 0, type: "cash"},
{ qty: 2, total: 90, tip: 0, type: "tab"},
{ qty: 2, total: 90, tip: 0, type: "tab"},
{ qty: 2, total: 90, tip: 0, type: "tab"},
{ qty: 2, total: 200, tip: 0, type: "cash"},
{ qty: 1, total: 200, tip: 100, type: "visa"}
]);
//a single property dimension
var paymentsByTotal = payments.dimension(function(d) { return d.
    total; });
var paymentsByType = payments.dimension(function(d) { return d.
    type; });
//a dimension built with a derived property
var paymentsByTotalType= payments.dimension(function(d) { return
     d.total +" " + d.type; });
//we group by dimension's data.
var groupPaymentsByType = paymentsByType.group();
var groupPaymentsByTotal = paymentsByTotal.group();
// array of key value that counts  rows.
var countPaymentsByType = groupPaymentsByType.all() ;
console.log(countPaymentsByType);
//cash 2 , tab 8 , visa 2
//array of key value that sums total of each type and aggregates
     it
var volumePaymentsByType = paymentsByType.group().reduceSum(
    function(d) { return d.total}).all();
//cash 300 , tab 920 , visa 500
//we filter paymentsByTotal dimension
// selects payments whose total is 100
paymentsByTotal.filter(100);
//this grouping it is not affected by the filter
var countPaymentsByTotal = groupPaymentsByTotal.all();
//affected by the filter:
console.log(countPaymentsByType);
```

```
//cash 1, tab 0, visa 0
```

2.9 dc.js

dc.js[9] is a JavaScript charting library with native Crossfilter support, allowing highly efficient exploration on large multi-dimensional datasets. Charts rendered using dc.js are data driven and reactive and therefore provide instant feedback to user interaction. dc.js is an easy yet powerful JavaScript library for data visualization and analysis in the browser and on mobile devices. We can see the library in action in figure 2.7



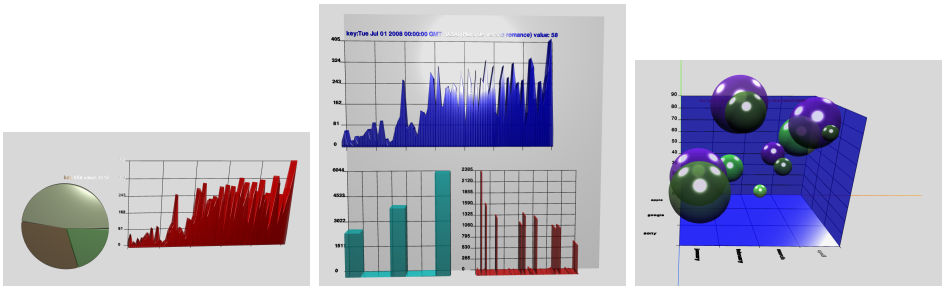
(a) initial status (b) after interaction

Figure 2.7: dc.js example with interaction

2.10 THREEDC

THREEDC[10] is a JavaScript library for building 3D charts passing to the objects the data, it has native integration with Crossfilter and we can do different interactions with the charts as tool tips, filters by a single value or a range of values that affects the other charts and drag and drop feature.We can do piecharts, barchart (on 2d and 3d) smooth curve chart and bubbles charts. We can arrange them into higher entities called panels. It is built on the top of three.js library. THREEDC was developed during 2016 by a URJC student and its integration with kibana library was done by another URJC student providing that library a good way for feeding chart's data and take representative ones. In 2017 the

student who developed the library is still doing more improvements for it. We can see in figure 2.8 an example output of the library.



(a) Basic scene (b) Charts on a panel (c) A Bubble chart

Figure 2.8: THREEDC Scenes

2.11 VBoard-UI

VBoard-UI[11] is a platform to create 3D dashboard of elasticsearch data in the browser. In the user interface of the platform we have different tabs that guide us step by step for the construction of our dashboard. In the visualization tab we create the visualization according to the chosen data, the type of graph selected, the metrics and buckets. The buckets in the context of ElasticSearch is simply a categorization of the data. In the panels tab we can add different charts created in the previous step to a panel. And finally in the tab dashboard we can create our dashboard from the charts and panels saved until now. This software is still under development but already has the features mentioned. It is being done by a master student of the URJC. He has made the integration of our library and can visualize different types of A-Frame charts in VBoard-UI. In the figure 2.9 we can see a example of the library.

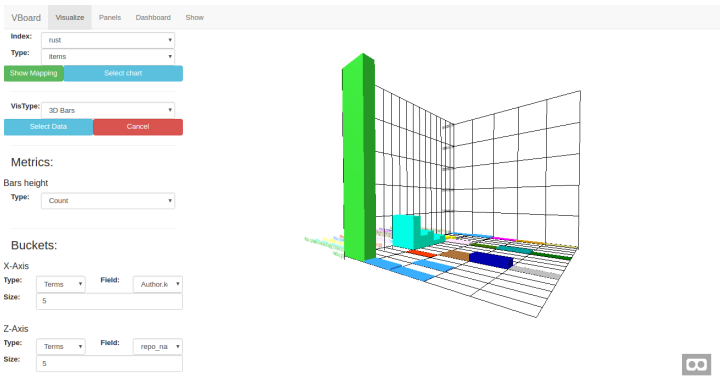


Figure 2.9: Creating a chart in VBoard-UI

Chapter 3

Development

This is the main chapter of memory in which we will explain the process carried out to achieve our objectives. We will analyze the use and development of the application from an incremental point of view, guiding the reader through each stage of the process. We will also comment on the development methodology carried out throughout this project.

3.1 SCRUM Methodology

SCRUM is an iterative and incremental agile software development framework for managing product development. It defines "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal", challenges assumptions of the "traditional, sequential approach" to product development, and enables teams to self-organize by encouraging physical co-location or close online collaboration of all team members, as well as daily face-to-face communication among all team members and disciplines in the project.

A key principle of SCRUM is its recognition that during production processes, the customers can change their minds about what they want and need (often called requirements volatility), and that unpredicted challenges cannot be easily addressed in a traditional predictive or planned manner. As such, SCRUM adopts

an empirical approach,accepting that the problem cannot be fully understood or defined, focusing instead on maximizing the team's ability to deliver quickly, to respond to emerging requirements and to adapt to evolving technologies and changes in market conditions.

In SCRUM there are three main roles defined:

1. **Product owner:** The person responsible for maintaining the product backlog by representing the interest of the stakeholders, and ensuring the value of the work the development team does.

2. **SCRUM master:** The person responsible for the SCRUM process, making suer it is used correctly and maximizing its benefits.

3. **Development team:** A cross-functional group of people responsible for delivering potentially shippable increments of product at the end of every sprint.

However, in our case the product owner and the scrum master are represented by the project tutor. Apart from that, we follow the SCRUM methodology faithfully and I was a part of a real development team because my project is a part of a bigger project inside my tutor's department.

A sprint (or iteration) is the basic unit of development in scrum. The sprint is a time-boxed effort; that is, it is restricted to a specific duration. The duration is fixed in advance for each sprint and is normally between one week and one month, with two weeks being the most common.

Each sprint starts with a sprint planning event that aims to define a sprint backlog, identify the work for the sprint, and make an estimated commitment for the sprint goal. Each sprint ends wit a sprint review and sprint retrospective, that reviews progress to show to stakeholders and identify lessons and improvements for the next sprints.

SCRUM emphasizes working product at the end of the sprint that is really done, In the case of software, this likely includes that the software has been integrated, fully tested and end-user documented.

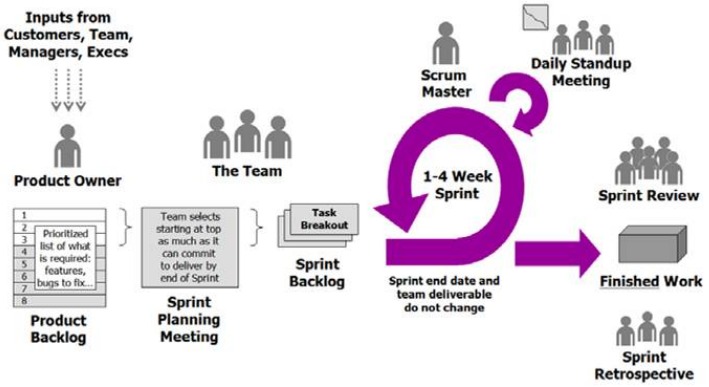


Figure 3.1: SCRUM Methodology

The sprints followed on this project:

1. **Iteration 0:** Investigation and preliminary study
2. **Iteration 1:** First demo (chart)
3. **Iteration 2:** Interactivity with the user and filtering
4. **Iteration 3:** 3D visualizations
5. **Iteration 4:** Library architecture
6. **Iteration 5:** Integration with a data dashboard.

3.2 Iteration 0: Investigation and preliminary study

Due to the specific characteristics of the project, there are currently few libraries that fit our requirements. A webGL-based JavaScript library, which also allows you to transform that scene into a virtual reality one via webVR. We have tested three.js supported from several plugins implemented by the community that allow the requirements which we described before. We have also tested the a-frame library which includes controls and default initialization to convert 3D scenes to VR scenes.

The goal was therefore to create a basic visualization that allows us to move through a 3D environment and toggle to a VR environment.

For the creation of this basic scene of Three.js was necessary, the creation of a scene, a camera, the geometry in order to draw it, define a div to contain after a canvas, add a canvas in which to render our content and define our scene's repainting function. In addition to adding the necessary control to allow the transformation of our scene to VR. We can see the results in figure 3.2.

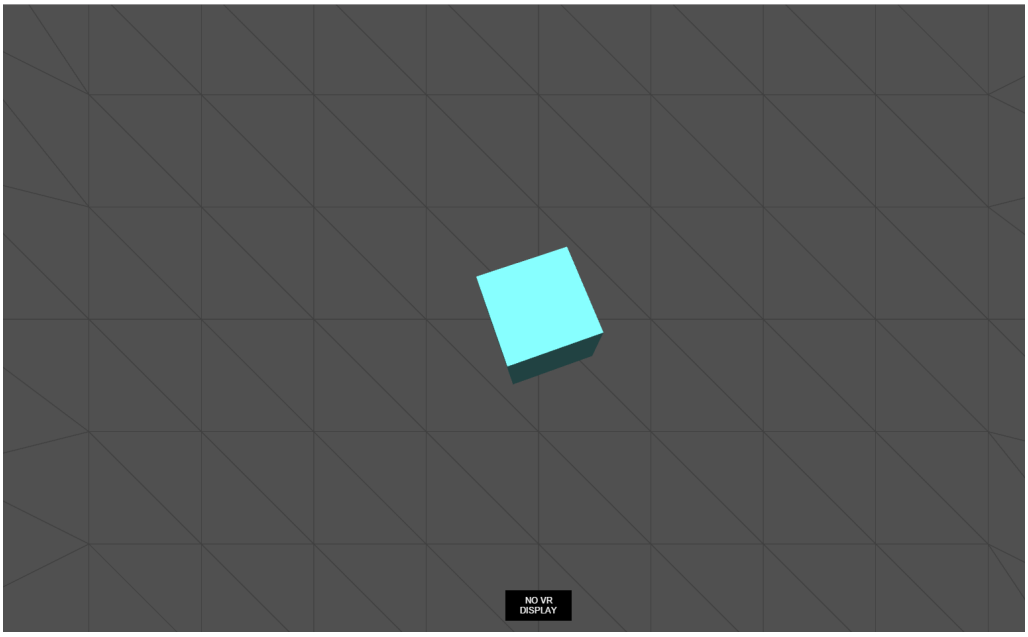
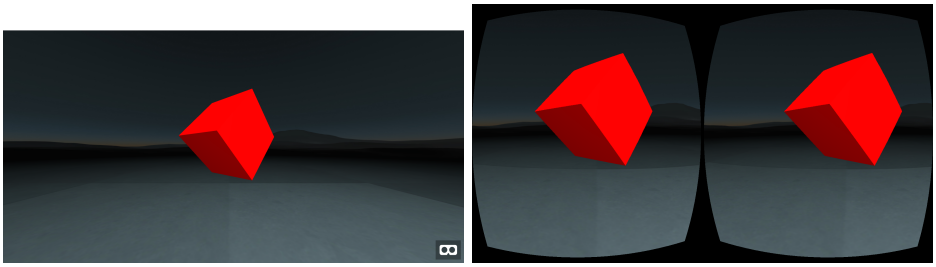


Figure 3.2: Three.js WebVR example



(a) Demo in PC (b) Entering VR in mobile

Figure 3.3: A-frame WebVR example

In a frame to get the same scene just use the HTML initialization code of the scene and use the geometry of a drawing. We can see results in figure 3.3.

In all cases it was not necessary to install any additional software.We needed to launch a server to display the content on pc and phone.

Because of its ease of development and also it fit so well to the required characteristics, A-frame was chosen as the base library for the development of our 3D dashboards.

3.3 Iteration 1: First demo (chart)

Once chosen the main technology for our project we proceed to create some charts from basic entities of A-frame. These charts will be: pie chart and bar chat. Our diagrams represent a data entry through a JavaScript object. We use A-Frame primitives to draw these diagrams which are used inside the A-Frame components that will generate these charts.

For the creation of the bar diagram the height of each bar will be varied according to the data passed by parameter. The attributes of depth, width and color will be the same for all bars that make up the chart. On the other hand the attributes of the bar diagram are included within the HTML markup inside bar chart's attribute.

To do this we use as the base entity the a-box primitive included in A-Frame. On this cube we can modify the different properties mentioned above. We create an A-Frame component that is in charge of managing the creation of these cubes with the height we need. This height will depend on the data entered and are scaled to the maximum height value of the chart. The properties to be modified are: width, height, depth, color and data. The last property was not included within the component legibility of the HTML code .It must be added within the JavaScript code. We see a result and the necessary HTML code in the figure 3.4.

```
<html>
 <head>
  <script src="https://aframe.io/releases/0.5.0/aframe.min.js"><
      /script>
  <script src="js/barchart.js"></script>
  <title>Demo barchart</title>
```

```
 </head>
 <body>
 <a-scene>
  <a-entity id="bars" barchart="width:14;height:3;depth:5;color:
      red;gridson:false"></a-entity>
 </a-scene>
<script>
 //ensure window loaded
 window.onload = function () {
  //inyecting data to an existing chart..
  var mypieChart = document.querySelector('#bars'); //<a-entity>
  mypieChart._data = [{ key: 'bla', value: 85 }...];
  mypieChart.emit('data-loaded');
}
</script>

</body>
</html>
```

Figure 3.4: Barchart built with A-Frame

As we can see the source code is very compact. For this visualization we have only needed to load the A-Frame library, register our component and include the portion of HTML that defines our diagram.

The strategy carried out for the creation of the pie chart is similar. First we create an A-Frame component responsible for managing our basic entities depending on the data. In this case the A-Frame primitive used is a-cylinder. The a-cylinder is a geometry in which its circular section lies on the plane XZ.So it is necessary to rotate the geometry and set the section to be drawn. That

set of geometries that we draw are part of our pie chart. For the colors used in the diagram we have generated a set of several colors. These are not generated randomly. In this first iteration we did not want to worry about implementing an algorithm that will generate colors within suitable hue and saturation values. The modifiable properties in this chart are: radius, depth and data. We see a result and the necessary HTML code in the figure 3.5.

```
<html>
<head>
<script src="https://aframe.io/releases/0.5.0/aframe.min.js"></
    script>
<script src="js/piechart.js"></script>
<title>Pie Chart in A-Frame</title>
</head>
<body>
  <a-scene>
   <a-entity camera wasd-controls look-controls mouse-cursor></a
       -entity>
   <a-entity id="pie" piechart="radius:4"></a-entity>
  </a-scene>
<script>
//ensure window loaded
window.onload = function () {
  //inyecting data to an existing chart..
  var mypieChart = document.querySelector('#pie') ; //<a-entity>
  mypieChart._data = [{ key: 'bla', value: 85 }, ...];
  mypieChart.emit('data-loaded');
}
</script>
</body>
</html>
```

As a conclusion to this first iteration we already have A-Frame components to work on and add more functionality either on improvements in this component or with other A-Frame components.

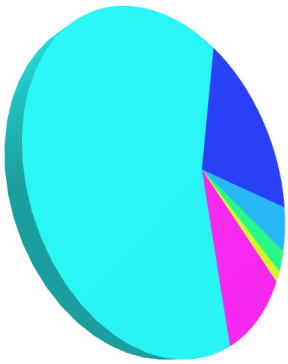


Figure 3.5: Piechart built with A-Frame

3.4 Iteration 2: Interactivity with the user and filtering

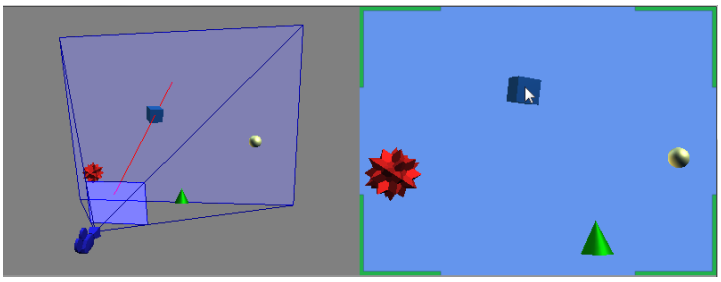
Once we have several charts we want to increase its functionality marking the next objectives:

- Show information about each element by moving the mouse over.
- Add crossfilter as data source.
- Update the charts with new data when clicking on an element.

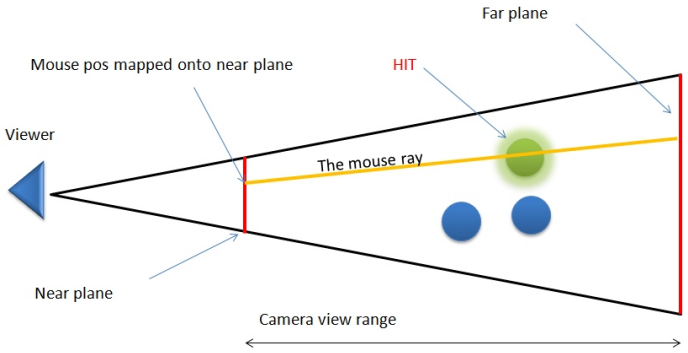
3.4.1 Mouse interactions

To handle the position of the mouse in our 3d scene and determine what element is interacting we use the ray-picking technique. To calculate the line we use the point pointing the mouse and the direction vector of our camera. We see a graphical explanation on the figure 3.6.

We can then know which is the first element that has been intersected by that line. This is the theory that is implemented in the cursor component of A-Frame. This library's component is focused on VR scenes only. In order to be compatible with 3D scenes without VR we have added the mouse-cursor component that treats our cursor as the A-Frame cursor. This component is



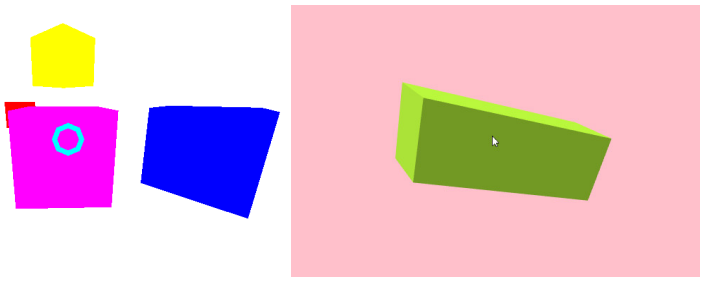
(a) Ray-picking example



(b) Ray-picking schema

Figure 3.6: Ray-picking technique

developed and maintained by the A-Frame community. This way you can execute mouseenter and mouseleave events from each of our A-Frame elements. In figure 3.7 we can see interactions with both cursors.



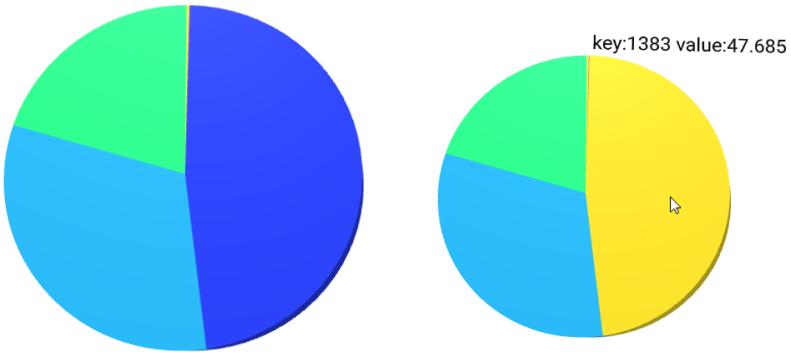
(a) A-Frame's cursor component (b) a-frame-mouse-cursor

Figure 3.7: Different Cursor component behavior comparison: A-Frame's built-in cursor and mouse-cursor

To handle the mouseenter and mouseleave events we also had two options: use the A-frame animation API or handle these events manually. After some test we handled the events manually. The reasons have been:

- More direct: We manage the color change the creation of the title etc. In just one site of our code.
- No new elements of the DOM are needed. To create an animation we had to create a new DOM element to specify its characteristics and wait for it to load the animation since being a new element.That is not immediate.

As for the information to display is a text that shows the key and value of that element. To display texts within a 3D scene is not trivial[12] either because multiple performance problems arise, to get geometries that adapt well to the distance the letter is displayed, not to generate rounded edges, quality of edges and kerning. These problems have been largely solved by the three.js community and we have a viable solution for displaying A-Frame text using the text component. We see a result of this first milestone in the figure 3.8.



(a) Initial state (b) mouse over a part of the chart

Figure 3.8: Piechart with mouse over interaction

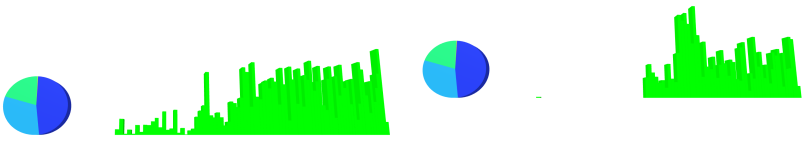
3.4.2 Add crossfilter as data source

As we saw in section 2.8 we used crossfilter which is a library that allows us to filter add and group large amounts of data very quickly. We choose it to be a data

source for our project. To be able to extract data from crossfilter it is necessary for our chart to host information of the dimension and the group. Once we pass this data to our component, inside it is called the corresponding API to extract the data and draw them. We also have the logic to determine if we have a group of crossfilter or data entered directly. The crossfilter data prevails over the data entered directly.

3.4.3 filtering in the charts

To perform the filtering in the charts we must handle a user interaction that tells us about what value to filter and then update the other graphs.The filtering of the graphs has been designed similar to the dc.js model. So the graph used for the filtering is not modified. To obtain all the existing graphics in a scene we have carried out a simple method: to collect all the elements of the scene and to check if they have a group of crossfilter. This method would be improved in the following iterations. We see in the figure 3.9 the result:



(a) Initial state

(b) After clicking a part of the piechart

Figure 3.9: Piechart is filtering the barchart

In conclusion we have for our charts different data access strategies. And different iterations with the user that allow to filter data.

3.5 Iteration 3: 3D visualizations

In this sprint we are going to make two new charts, in this case they are 3D that are: 3D barchart and bubble chart. These charts will include all the improvements made in the previous sprint so they accept both crossfilter data and data entered

directly. Interactions with the user are also included and we can filter by an element.

First of all, as we did in the 2D bar diagram, we are based on generating all the parts of our chart in the A-Frame primitive a-box. In this case we start generating the data source of crossfilter since to generate two-dimensional data is not a simple task. We had to carry out a reduction of data that collected with a dimension aggregated data of different type. The data structure thus differs from that of the 2D charts created earlier. Now we had to pass functions to our chart to know the properties that determine the position of the element on the Z axis and X axis and the height of the element. The properties that are allowed to modify of this chart are: height, width and depth. The color changes along the Z axis and is maintained in each element along the X axis. We need to pass those colors by argument too and the chart do not generate it explicitly. We see images of the result in the figure 3.10.

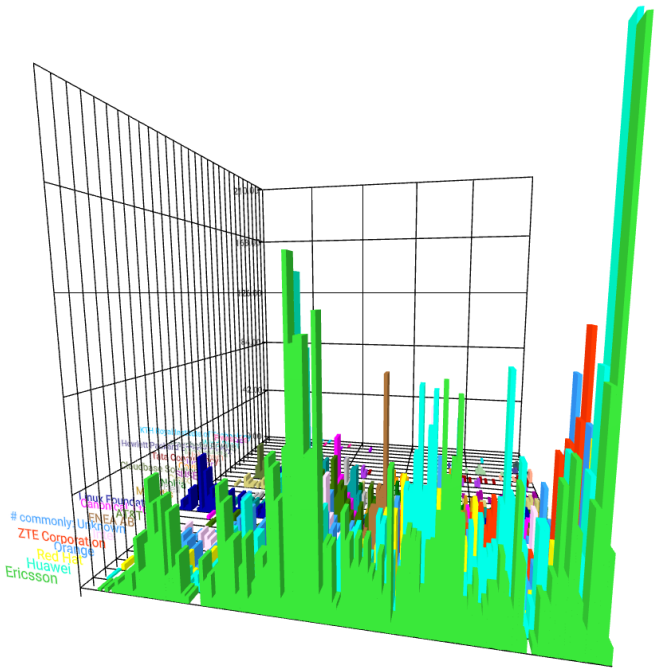


Figure 3.10: 3D Barchart

For the bubble chart we have based on the primitive of A-Frame a-sphere. The strategy carried out is the same and we first collect the data from crossfilter.

In this case we need another additional measure for the radius of each bubble. In bubble charts the choice of the maximum and minimum radius is important. If we do not choose correctly we will not be able to compare data and the visualization loses quality. The maximum radius in our case is determined by the following formula:

$$\frac{\min width, height, depth}{\min n^{o}casesXaxis, n^{o}casesZaxis} \tag{3.1}$$

As the previous chart we need to pass the colors an their relationship with elements in different position on Z axis. On this chart we can modify: height, width and depth. In the figure 3.11 we see the result.

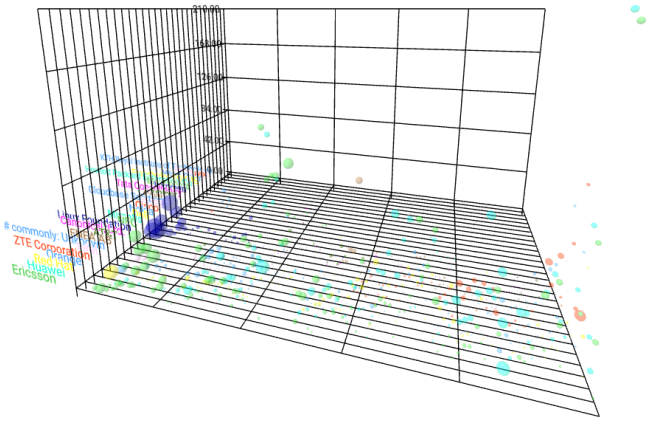


Figure 3.11: Bubble chart

As a conclusion to this sprint we have the realization of new graphics that provide greater visualization options.

3.6 Iteration 4: Library architecture

In this chapter we have two objectives: The first and most important is to encapsulate the functionality performed so far in a library. On the other hand the characteristic of collecting a set of charts in a superior entity that we will call panel.

One of the requirements of our library is that it is interchangeable with THREEDC. With this we mean that we will follow the same interface in our

API. This is important since our idea in the next sprint is to adapt our library with external software. THREEDC has made an integration with kibana so adapting our functions we will get that integration in a simple way. We also have the requirement to use the components made so far from A-Frame in the HTML code without using methods of the library. We define three different object types:

- **DashBoard:** It is an object which contain charts and panels. We have methods to list: the added charts and the added panels. You can add an empty dashboard to an existing scene and create a base scene with an empty dashboard.
- **Panel:** It is an object which contain charts. We also have methods to add and list the charts associated with the panel.
- **Chart:** This is an object that will be displayed on the screen. It has methods to specify its dimensions, color, data source, etc.

Our library exports a single global object on which we have the following methods:

- **addDashBoard (AFrameScene):** Allows us to associate a dashboard with an existing A-Frame scene. To do this, we must pass through this scene. Returns the dashboard object.
- **Dashboard (containerdiv):** Allows us to create a new scene from scratch. It initialize a default configuration by creating the scene and a default camera. Returns the dashboard object.
- **Panel:** creates a new panel to add charts to it.
- **barChart:** Creates an A-Frame entity with the associated barchart component.
- **pieChart:** Creates an A-Frame entity with the associated piechart component.

- **bubbleChart:** Creates an A-Frame entity with the associated bubblechart component.
- **barChart3d:** Creates an A-Frame entity with the associated barChart3d component.

In the creation of our library we use the **mixin** technique to avoid repeating common code throughout our objects. Creating a base object that will contain various methods and attributes that will be shared for the rest of objects. In our case we have used it for the creation of chart object. On the other hand we are also used **chaining** which is a technique in which each function returns the object itself. So you can make calls to functions in a row to reduce the code required.

Let's look an example of starting a scene in a div and creating a chart in it with our library. We simplify some code in order to be more shorter an legible.

```
<html>
<head>
<script src="aframe.min.js"></script>
...
<script src="js/demopiechartscrath.js"></script>
<title>Simple dashboard with a pie chart, from scratch</title>
</head>
<body>

<div id="myscene"></div>
</body>
</html>

//file: demopiechartscrath.js
// Example (assuming there is "myscene" in HTML, to place the
    dashboard)
window.onload = function () {
  var scenediv = document.getElementById("myscene");
  // 1
  myDashboard = aframedc.dashboard(scenediv);
  // 2
```

```
    var myPieChart = aframedc.pieChart();
    // Common
    var data = [{ key: 'bla', value: 85 }...];
    myPieChart = myPieChart.data(data).depth(3);
    myDashboard.addChart(myPieChart);
}
```

Figure 3.12: PieChart starting a dashboard with our library aframedc

For panel creation we create an A-Frame component that is constantly listening to its new aggregated immediate children elements. We have created a small algorithm that reallocates the aggregated charts inside the panel. Once a chart in panel is added or deleted it is detected by associated events and the remaining charts are reallocated. This algorithm is consulting the dimensions of the geometry,so it would also be valid for non-chart elements generated by our components. We see an example of code an its result associated in the figure 3.13. We do not put code refering to crossfilter initialization and creation for this example in order to be more shorter and legible. The HTML code is the same as the previous example, it only changes in script section.

```
window.onload = function () {
    // initialization
    //getJSON call, draw meshes with data
    $.getJSON("../data/scm-commits.json", function (data) {
            var json_data = data;
            init(json_data);
    });

    var init = function (json_data) {
```

```
        var scenediv = document.getElementById("myscene");
        // 1
        myDashboard = aframedc.dashboard(scenediv);
        // 2
        var mypiechart = aframedc.pieChart();
        var mybarchart = aframedc.barChart();
        var myPanel = aframedc.Panel();
        // init crossfilter dimensions and groups
        var objCF= initCFData(json_data);

        mypiechart.dimension(objCF.dimByOrg)
    .group(objCF.groupByOrg).radius(2.5)
    .setTitle("commits per company");
        mybarchart.dimension(objCF.dimByMonth)
    .group(objCF.groupByMonth).width(30)
    .setTitle("commits per month");
        myPanel.width(40).height(15)
    .nrows(1).ncolumns(2)
    .setTitle("commits per company and month");
        myPanel.addChart(mypiechart);
        myPanel.addChart(mybarchart);
        myDashboard.addPanel(myPanel);
  }
}
```

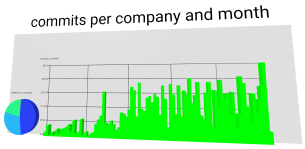


Figure 3.13: Two Charts inside a Panel with our library aframedc

As a conclusion including a library in our project, it increases the ease of creating our charts through HTML and also in a compact way in JavaScript.

3.7 Iteration 5: Integration with a data dashboard.

In this last sprint we have the following objectives, listed below:

- Creation of the Stacked Barchart chart.
- Generate a unified and minified js file for use by third-party developers.
- Make the latest adaptations for integration with VBoard-UI.
- Include the src option in our charts when we initialize the scene in HTML.

3.7.1 VBoard-UI Integration

The first step to achieving the goals of this sprint was to make the latest adaptations for integration with VBoard-UI. This integration was done by the developer of VBoard-UI who is a URJC master student. For this, it was necessary that in all our graphs the SetId method was exposed. We achieve this goal easily through our base object created in the previous milestone. On the other hand we had to adapt the input data of our 3D bar chart and bubble chart to accept a particular format, the format that the THREEDC library has for these graphics. About this library has already made a project to transform data from Elastik search and be able to pass them as input data to those books. This has been the main reason for adapting our data source. The changes produced in the charts have been in their geometric creation loops. It has also been necessary to create functions within these charts to assign the colors of each geometry without explicitly pass them. We agreed cases of the final format of the data and we will explain them in section 4.2. In the figure 3.14 we see a chart created from VBoard-UI:

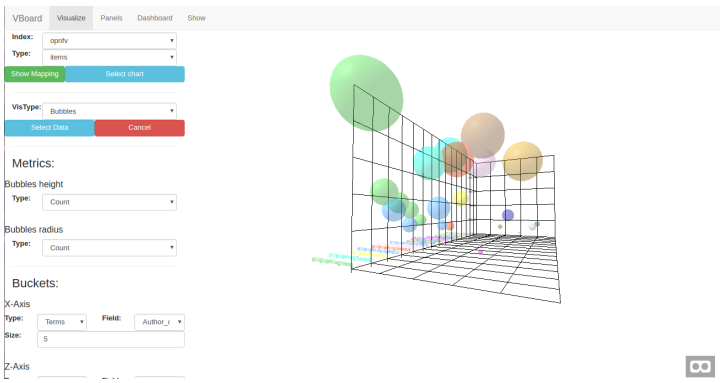


Figure 3.14: Creating an A-Frame chart with VBoard-UI

3.7.2 Stacked Barchart

The stacked barchart graphs are graphs of 2D bars that accumulate different segments in a specific bar in order to differentiate the parts that constitute a whole. For this example we do not start from zero since we have different graphs that have things in common with this graph.According to what is shown in the previous section of this sprint this is a 2 dimensional graph and we aim to accept the corresponding input data. This chart is very similar to the 3D Barchart in its geometry creation loop but each element is drawn on top of the previous element (increasing position on the Y axis) and not in depth. It changes the way to scale the maximum height since it is the maximum contribution of the set of elements in the X axis and not the maximum value of a specific element. Once these modifications have been made as we have already mentioned, we have reused parts of other charts such as upper title management and grids. In this chart we have added a color legend that is shown on the right side of the chart. We see a result of the chart in the figure 3.15.

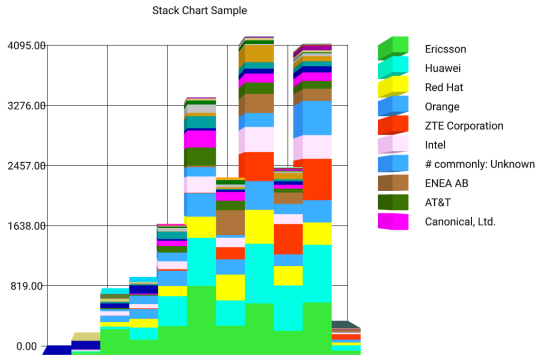


Figure 3.15: Stacked Barchart

3.7.3 Bundling js file

In this step we make the versioning and packaging of our software. We have decided to create a new repository in github[1] for making a better file structure of the project. This structure is more common and therefore readable for other

[1] https://github.com/fran-aguilar/a-framedc

users of the A-Frame community. For the bundling and minification tasks of our JavaScript library we have supported webpack 2[2]. This npm's[3] module is a bundler which processes all the JavaScript files of our application from an entry point and recursively creates our dependency graph.To begin we create an index.js file that will be our starting point of the application and in which we encapsulate all the created functionality, that are: A-Frame components and dashboard creation library that needs those components. In our exported file we do not include external references such as A-Frame, or components of other users used in our library. In the figure 3.16 we see the generated file structure for this sprint.

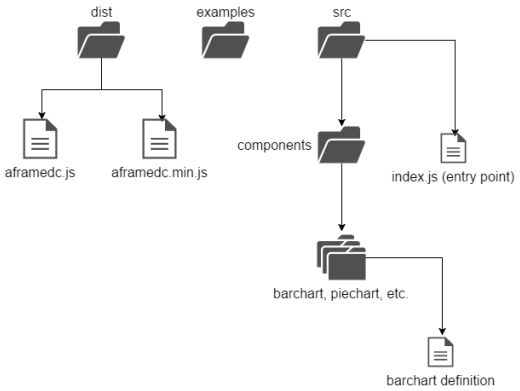


Figure 3.16: Current file structure

3.7.4 Including src option in charts

About the src option in our charts is the definition of a new HTML attribute that will receive the JSON object with the data to be displayed. With this we achieve that our charts are defined exclusively with HTML. Recall that this milestone is important since in A-Frame all the components and entities can be configured from HTML code. To achieve this we have modified the schema of all the charts created and we have added to our scheme that new src property. We have also modified the initialization of our graphics to observe this property. Here is an example of what is mentioned.

[2] https://webpack.js.org/
[3] https://www.npmjs.com/

```
<html>
<head>
<script src="aframe.min.js"></script>
</head>
<body>
<a-scene>
 <a-assets>
  <a-asset-item id="barsdata" src="scm-commits.json"></a-asset-
      item>
 </a-assets>
 <a-entity id="bars" barchart="width:14;gridson:true;title:
     example barchart;src:#barsdata"></a-entity>
</a-scene>
</body>
</html>
```

Chapter 4

Design and results

In this chapter we will make a technical description of the software developed, a user manual and lastly a demo that show the possibilities of the library.

4.1 Software Description

A-framedc is a library for building 3D charts in the browser. These charts can be visualized in 3D scenes that allow to activate stereoscopic mode to be visualized virtual reality headset. Interactions between different charts are allowed with the use of crossfilter library. The library allows you to initialize scenes from scratch or using an existing A-Frame scene.

As base technology we use the A-Frame library. In this technology we create our 3D scenes and extend this library to create the components that create our charts. This configuration allows us draw our graphs with the declaration of HTML code exclusively. Although the main use is in conjunction with the aframedc library. Both for the creation of the aframe components and for the creation of chart objects we rely on base objects that contain basic methods and properties. To extend the aframe library with a new component we must construct an object with the schema property and the init, update and remove methods.We can see in figure 4.1 piechart class diagram. All aframe components trigger a set of events. These events are used to notify and receive notifications.

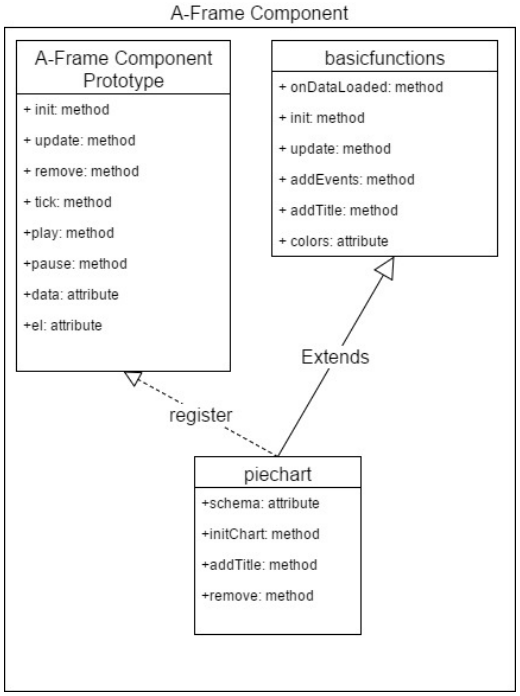


Figure 4.1: Piechart component: class structure

For our needs of updating chart components we add a new event that is released when we add properties that are not defined in this component and allow its updating. There are three ways to add the data information:

1. **using the dimension and group methods:** the dimension and group methods allow us to set a dimension and a group of crossfilter respectively to our chart. The chart accesses data which varies depending on the dimension's status. When these properties are defined in a chart, a handler function is added to each geometry of the graph so that it can be filtered by clicking on it.

2. **using the data method:** This method allows to establish an array of data to our chart. The library does not offer any behavior in the graph when clicking on it. Although you can include a behavior by setting a handler function with the customEvents method.

3. **Using the src attribute:** This attribute is part of the value of the A-Frame component and can be initialized from html or via JavaScript using the setAttribute method.

On the format of the data these will always be received in JSON format or in JavaScript object. About how these data have to be composed we determine three possible cases:

1. **Array with one key and one value:** The format of the data will be and array of objects with the properties key and value.All the value properties of the array must not be empty and it must be of the Number type.For example

```
[{key:"visa",value:5},{key:"visa",value:10}]
```

   is an example of valid JavaScript Object. This format is used by piechart,barchart and smoothcurvechart.

2. **Array with two keys and one value:** this format is slightly different than the previous one but we have two key properties. The restrictions for the value property are the same than the previous data format. A valid example of the data is:

```
[{key1: 20144,key2: "Ericsson",value: 0},{key1: 20144,key2: "
    Huawei","value": 0}]
```

   This format is used by barchart3d and stacked barchart.

3. **Array with two keys and two value:** A valid example of the data is:

```
[{key1: 20144,key2: "Ericsson",value: 0,value:  2},{key1:
    20144,key2: "Huawei",value: 0,value2:1 0}]
```

   This format is used exclusively on bubble charts.

4.2 Use of the library

We define three different object types in our library:

- **DashBoard:** It is an object which contain charts and panels. We have methods to list: the added charts and the added panels. You can add an empty dashboard to an existing scene and create a base scene with an empty dashboard.

- **Panel:** It is an object which contain charts. We also have methods to add and list the charts associated with the panel.

- **Chart:** This is an object that will be displayed on the screen. It has methods to specify its dimensions, color, data source, etc.

Our library exports a single global object on which we have the following methods:

- **addDashBoard(AFrameScene):** Allows us to associate a dashboard with an existing A-Frame scene. To do this, we must pass through this scene. Returns the dashboard object.

- **Dashboard(containerdiv):** Allows us to create a new scene from scratch. It initialize a default configuration by creating the scene and a default camera. Returns the dashboard object.

- **Panel:** creates a new panel to add charts to it.

- **barChart:** Creates an A-Frame entity associated with the barchart component.This is a chart which visualizes the evolution of a value or attribute along an axis, commonly time, with bars of different heights.Barchart creation on JavaScript:

```
aframedc.barChart().width(30).height(30).depth(30)
.gridsOn(false).data(...).setTitle("example")
```

An example with HTML markup:

```
<a-scene>
 <a-assets>
  <a-asset-item id="barsdata" src="scm-commits.json"></a-asset
      -item>
 </a-assets>
```

```
 <a-entity  barchart="width:30;depth:30; height:30; gridson:
     true; title:example; src:#barsdata"></a-entity>
</a-scene>
```

- **pieChart:** Creates an A-Frame entity associated with the piechart component.This is a chart which composes a circle that represents the different parts of a whole in a set of segments of that circle until completing it.Piechart creation on JavaScript:

```
aframedc.pieChart().radius(3).depth(30)
.gridsOn(false).data(...).setTitle("example")
```

An example with HTML markup:

```
<a-scene>
 <a-assets>
  <a-asset-item id="piedata" src="scm-commits.json"></a-asset-
      item>
 </a-assets>
 <a-entity piechart="depth:30; radius:3; gridson:true; title:
     example; src:#piedata"></a-entity>
</a-scene>
```

- **bubbleChart:** Creates an A-Frame entity associated with the bubblechart component.This is a chart that represents different spheres of different position and size depending on different categories. In the context of our library we play with the following parameters: width, height, depth and radius of the sphere.BubbleChart creation on JavaScript:

```
aframedc.bubbleChart()
.width(10)
.depth(10)
.height(10)
.keyaccessor(keyfunction)
.setTitle("contribution by company");
```

An example with HTML markup:

```
<a-scene>
 <a-assets>
  <a-asset-item id="bubbledata" src="scm-commits2k2v.json"></a
      -asset-item>
 </a-assets>
 <a-entity bubblechart="depth:30; gridson:true; title:example;
      src:#bubbledata"></a-entity>
</a-scene>
```

- **barChart3d:** Creates an A-Frame entity associated with the barChart3d component. It is a chart which has bars of different heights like barchart, but the height of each bar is determined by two values of two different categories. These bars are drawn along two axis.3D Barchart creation on JavaScript:

```
aframedc.barChart3d()
.width(10)
.depth(10)
.height(10)
.keyaccessor(keyfunction)
.setTitle("contribution by company");
```

An example with HTML markup:

```
<a-scene>
 <a-assets>
  <a-asset-item id="bartddata" src="scm-commits2k1v.json"></a-
      asset-item>
 </a-assets>
 <a-entity barchart3d="depth:30; gridson:true; title:example;
     src:#bartddata"></a-entity>
</a-scene>
```

- **barChartstack:** Creates an A-Frame entity associated with the barchart-stack component.This is also a bar chart, in this case each bar is composed of parts of different size that make up the total of the bar in that specific value. Stacked barchart creation on JavaScript:

```
aframedc.barChartstack()
.width(10)
.depth(10)
.height(10)
.setTitle("contribution by company");
```

An example with HTML markup:

```
<a-scene>
 <a-assets>
  <a-asset-item id="barstackdata" src="scm-commits2k1v.json"><
      /a-asset-item>
 </a-assets>
 <a-entity barchartstack="width:10; depth:10; gridson:true;
     title:example; src:#barstackdata"></a-entity>
</a-scene>
```

- **smoothCurveChart:** Creates an A-Frame entity associated with the smoothcurvechart component.This is a graph with a line representing the evolution of a value or attribute along an axis. This graph does not respond directly to user actions, mouse events etc. Your data is updated if your group / dimension of crossfilter is affected. Smooth curve chart creation on JavaScript:

```
aframedc.smoothCurveChart()
.width(10)
.depth(10)
.height(10)
.setTitle("contribution by company");
```

An example with HTML markup:

```
<a-scene>
 <a-assets>
  <a-asset-item id="curvedata" src="scm-commits.json"></a-
      asset-item>
 </a-assets>
 <a-entity smoothcurvechart="width:10; depth:10; gridson:true;
      title:example; src:#curvedata"></a-entity>
```

```
</a-scene>
```

We can see in figure 4.2 all the charts visualized.

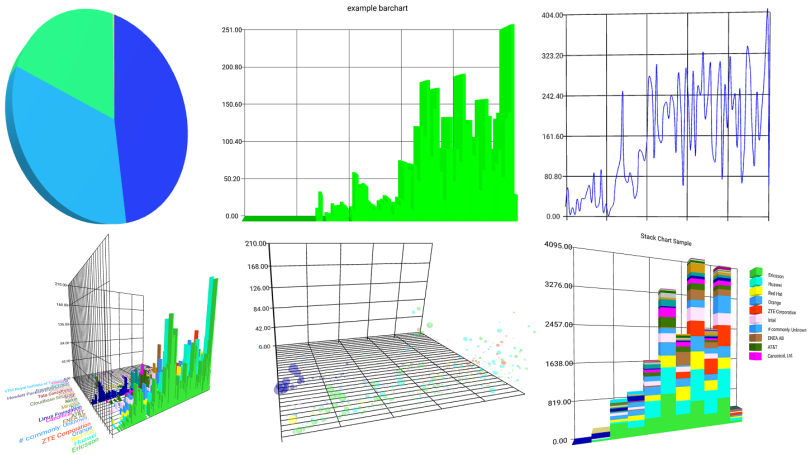


Figure 4.2: Different charts in a-framedc

Now we show a step by step use of the library with the following example. In this example we start from scratch a scene and we add a barchart.First we create a div which will be our scene's placeholder.

```
<html>
<head>
<script src="aframe.min.js"></script>
<script src="startfromscratch.js"></script>
</head>
<body>
<div id="myscene"></div>
</body>
</html>
```

Then on *startfromscratch.js* file we create our dashboard. We divide it in 5 steps. The step 1 and 2 changes when we already have an A-Frame scene created.

```
// 1 placeholder retrieving
var scenediv = document.getElementById("myscene");
// 2 DashBoard creation, and a-scene setup
myDashboard = aframedc.dashboard(scenediv);
// 3 piechart object creation
```

```
var mypiechart = aframedc.pieChart();
// 4 configurating our chart
mypiechart.data([{key:"visa",value:5},...])
.radius(2.5)
.setTitle("First pieChart");
// 5 adding it the scene
myDashboard.addChart(mypiechart);
```

Thanks to the A-Frame components ship with our A-Frame libary we can create the charts just using HTML code, like in the following example:

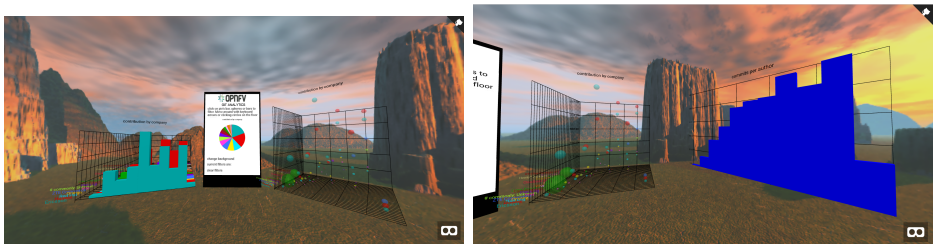
```
<html>
<head>
<script src="aframe.min.js"></script>
</head>
<body>
<a-scene>
<a-assets>
<a-asset-item id="barsdata" src="http://path/to/your/file/scm-
    commits.json"></a-asset-item>
</a-assets>
<a-entity id="bars" barchart="width:14;gridson:true;title:
    example barchart;src:#barsdata"></a-entity>
</a-scene>
</body>
</html>
```

4.3 A complex example

In the following example we see almost all the characteristics of the library. It is a 3D scene already initialized with different elements such as: the company logo or a 360 degrees background and different circles on the ground to interact with them. The texts within the central geometry at the start of the scene perform different actions (background change, cleaning of existing filters, etc.) working as a user interface. On the other hand the library is used to create the charts.Thanks to A-Frame these entities returned by the library can be rotated, displaced or

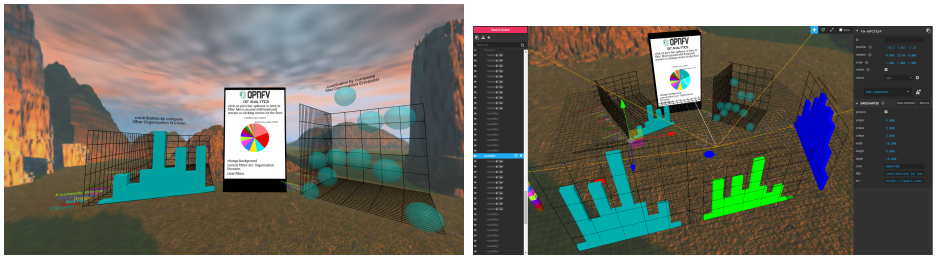
include more behaviors. The chart's data have been created with the crossfilter library so we can interact with mouse click. That interaction perform a filter on crossfilter's dimension with this value.

Thanks to A-Frame we can also see this demo in Virtual Reality devices with stereoscopic vision. It is also possible to reallocate the scene to our choice thanks to the A-Frame scene inspector that allows us to change the position, rotation, textures, etc. of all our entities.



(a) Initial charts (b) Right side of scene

Figure 4.3: Initial state of the demo with OPNFV git repository statistics



(a) Interacting with the charts (b) Editing the scene

Figure 4.4: Interacting with the charts and editing the scene with A-Frame inspector

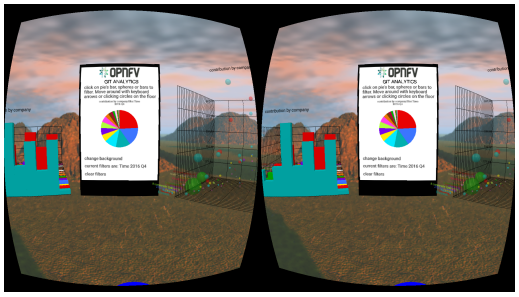


Figure 4.5: The OPNFV demo when we enter on VR

Chapter 5

Conclusions

We can say that the objectives proposed for our library have been fulfilled for the most part. We have been able to recreate part of the basic functionality of other chart creation libraries in which we have inspired such as: dc.js, highcharts and ultimately THREEDC as an example of 3D data graphics library. Support on both PCs and mobile devices is high enough at least to be able to view the 3D environment. Important aspects such as performance (frame rate per second), page load time and the number of downloaded items are within normal parameters. On the other hand its compatibility with A-Frame allows us to start from scenes already created and to modify different attributes and behaviors of the charts. Adapting the syntax of THREEDC has been achieved but there are still aspects such as the exchange dimensions between both libraries that have not been achieved.

On the other hand an external software of creation of data dashboard VBoard-UI has realized an integration of our library increasing its possibilities. The library has been advertised in the corresponding A-Frame slack, and examples of the library have been published. It has been published on npm in the second week of June reaching hundreds of downloads.

5.1 Results

I started this on last days of January 2017 and ended at the last days of May 2017 so I estimate around 350 hours of work. As a result of this work we have this library. It's open source and all the source code is on github[1]. This library is also a npm's package and uploaded to it[2]. All the demos I have realized are on the web too[3].All this information is on the specific web of the project[4].

5.2 Application of lessons earned

Although the degree that I have made is Telecommunications Engineering around a third of the subjects had a programming component which has been able to establish a good programming bases knowing their main programming paradigms as object oriented programming. In other subject I learned the different Web technologies and even from a general knowledge about OpenGL and computer graphics.

5.3 Lessons learned

The main lesson learned has been the deeper knowledge of the JavaScript language and in general of web technologies. Although A-Frame is a library that is mainly based on HTML, to take full advantage of all its possibilities and to extend it you need a good knowledge of JavaScript. Throughout the project I have also learned quite a few concepts about creating computer graphics and the most used techniques within the creation of 3D scenes. An important aspect of this project is that it has been inspired by other existing open source JavaScript libraries and I use of them as a step to create my solution. Examples of this are: the crossfilter library, dc.js or existing components of the community of A-Frame. It has improved my ability to read and write in English as it is the first time I

[1] http://github.com/fran-aguilar/a-framedc
[2] https://www.npmjs.com/package/a-framedc
[3] https://fran-aguilar.github.io/a-framedc/
[4] https://fran-aguilar.github.io/

have undertaken a english text of so many words. I have also had to read a lot of information from the internet and books in English.

5.4 Future work

Although we have undertaken several important steps with the creation of this library by combining representation of 3D data graphics and an immersive environment. There is still much to explore. Objectives such as different virtual environments or a better user interface. Accept more devices like gamepads or the use of more advanced virtual reality devices like HTC Vive. On the enlargement of the library we would have several fields of improvement. These are:

- Representation of new charts.
- Support for animations in charts.
- Selection of a set of data in the charts

Appendix A

README.md on GitHub

a-framedc 3D charts built with A-frame. A-framedc ships with a set of A-Frame components and a library to use them in easier way.It provides the following features:

- Create 3D chart visualizations of data.
- No additional software must we installed for executing our software thanks to A-Frame.
- Similar functionality as dc.js library with filter possibilities and different types of charts.
- A fast filter response thanks to crossfilter.
- The ability to change between a 3D scene and a Virtual reality scenario.
- Render our charts on a existing scenario, or create a scene from scratch.

Our library exports a single global object (a-framedc) on which we have the following methods:

- **addDashBoard(AFrameScene):** Allows us to associate a dashboard with an existing A-Frame scene. To do this, we must pass through this scene. Returns the dashboard object.

- **Dashboard(containerdiv):** Allows us to create a new scene from scratch. It initialize a default configuration by creating the scene and a default camera. Returns the dashboard object.
- **Panel:** creates a new panel to add charts to it.
- **barChart:** Creates an A-Frame entity associated with the barchart component. This is a chart which visualizes the evolution of a value or attribute along an axis, commonly time, with bars of different heights.
- **pieChart:** Creates an A-Frame entity associated with the piechart component. This is a chart which composes a circle that represents the different parts of a whole in a set of segments of that circle until completing it.
- **bubbleChart:** Creates an A-Frame entity associated with the bubblechart component.This is a chart that represents different spheres of different position and size depending on different categories. In the context of our library we play with the following parameters: width, height, depth and radius of the sphere.
- **barChart3d:** Creates an A-Frame entity associated with the barChart3d component. It is a chart which has bars of different heights like barchart, but the height of each bar is determined by two values of two different categories. These bars are drawn along two axis.
- **barChartstack:** Creates an A-Frame entity associated with the barchartstack component.This is also a bar chart, in this case each bar is composed of parts of different size that make up the total of the bar in that specific value.
- **smoothCurveChart:** Creates an A-Frame entity associated with the smoothcurvechart component.This is a graph with a line representing the evolution of a value or attribute along an axis. This graph does not respond directly to user actions, mouse events etc. Your data is updated if your group / dimension of crossfilter is affected.

Browser Installation Install and use by directly including the browser files:

```
<head>
<title>My A-Frame Scene</title>
<script src="https://aframe.io/releases/0.5.0/aframe.min.js"></script>
<script src="https://unpkg.com/a-framedc@1.0.7/dist/aframedc.min.js"></script>
</head>

<body>
<a-scene>
<a-assets>
<a-asset-item id="barsdata" src="http://path/to/your/file.json"></a-asset-item>
</a-assets>
<a-entity id="bars" barchart="width:14;gridson:true;src:#barsdata"></a-entity>
</a-scene>
</body>
```

NPM Installation Install via NPM:

```
npm install a-framedc
```

Then register and use.

```
require('aframe');
var aframedc = require('a-framedc');
```

Contact This project is under active development if you have any issue please let me know. Every help is much appreciated.

Appendix B

Demo with Controller

In this appendix I'll talk about several changes I made on the section 4.3's demo to support input from gamepad. After several tests made with a smartphone running android 6.0 OS and PS4 wireless controller,Chrome and Firefox browsers couldn't recognize my gamepad.

As an alternative solution we use WebRTC[1] to stablish a connection between our PC(with the gamepad connected) and our phone. The A-Frame community already use this implementation and the created an WebRTC server[2] which serves gamepad and keyboard controls. On that server we put a pair-code generated on client side to binding that page to the client. On the other hand there is an A-Frame component[3] which we put on our A-Frame scene. This component randomly generate a pair code and it is listening on the socket created. This connection waits for the host.Once the connection is stablished the gamepad controls and their events are retrasnmitted to the other pair. On the following pictures we see the proccess:

[1] https://webrtc.org/
[2] https://github.com/donmccurdy/proxy-controls-server
[3] https://github.com/donmccurdy/aframe-proxy-controls

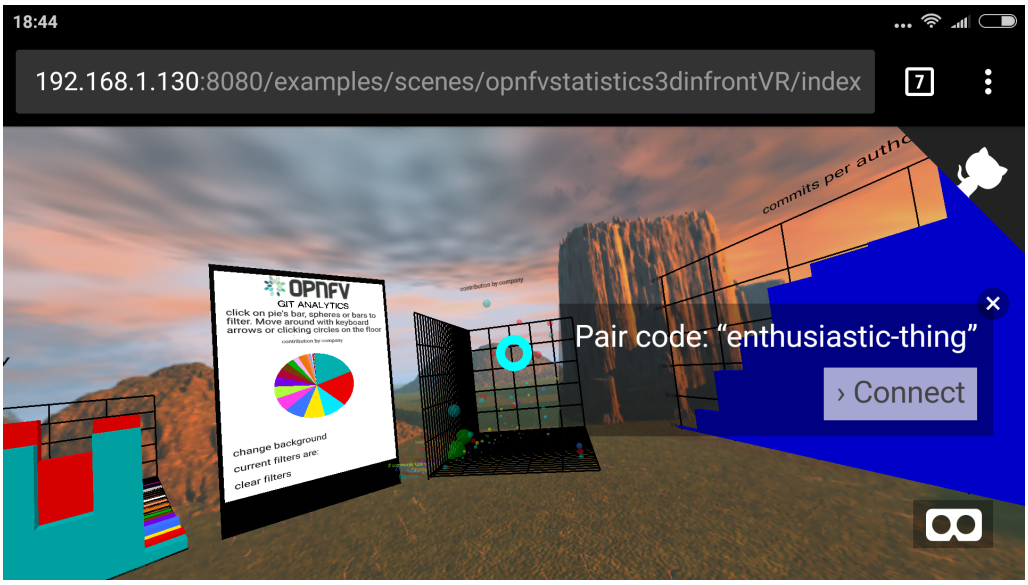
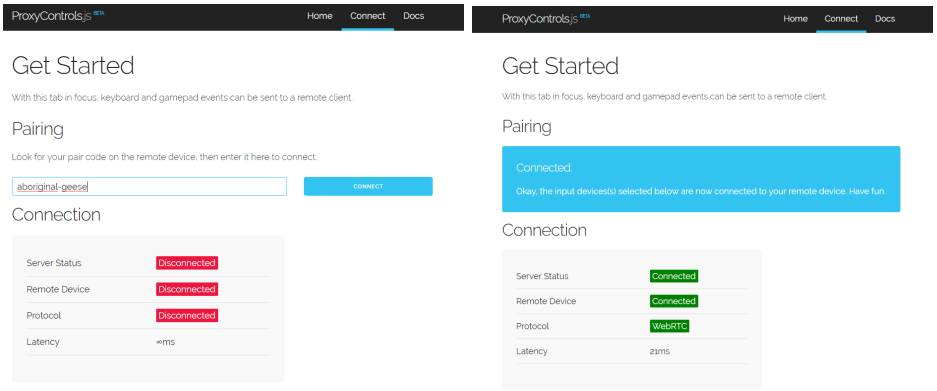


Figure B.1: Step 1: accessing the scene on the phone



(a) Accesing server through our PC (b) Entering code and connecting

Figure B.2: Step 2: Server Side Connection

Bibliography

[1] C. Musciano and B. Kennedy, *HTML & XHTML: The Definitive Guide*. O'Reilly, 2000.

[2] M. Pilgrim, *HTML5: Up and Running*. O'Reilly, 2010.

[3] D. Crockford, *JavaScript:The Good Parts*. O'Reilly, 2008.

[4] T. Parisi, *WebGL:Up and Running*. O'Reilly, 2012.

[5] Webvr website. [Online]. Available: https://webvr.info/

[6] three.js website. [Online]. Available: https://threejs.org/

[7] A-frame web page. [Online]. Available: https://aframe.io/

[8] crossfilter api reference. [Online]. Available: https://github.com/square/crossfilter/wiki

[9] dc.js web. [Online]. Available: https://dc-js.github.io/dc.js/

[10] A. Alonso. Threedc git repository. [Online]. Available: https://github.com/adrianalonsoba/THREEDC/

[11] D. M. Lumbreras. Vboard-ui git repository. [Online]. Available: https://github.com/VBoard/VBoard-UI

[12] P. Khachi. It's 2015 and drawing text is still hard (webgl, threejs). [Online]. Available: https://www.eventbrite.com/engineering/its-2015-and-drawing-text-is-still-hard-webgl-threejs/

[13] N. C. Zakas, *Maintainable JavaScript*. O'Reilly, 2012.

[14] A-frame source code. [Online]. Available: https://github.com/aframevr/aframe/

[15] A-frame documentation. [Online]. Available: https://aframe.io/docs/

[16] R. Dudler. git - the simple guide. [Online]. Available: http://rogerdudler.github.io/git-guide/index.es.html

[17] three.js documentation. [Online]. Available: https://threejs.org/docs/

[18] dc.js git repository. [Online]. Available: https://github.com/dc-js/dc.js

[19] crossfilter git repository. [Online]. Available: https://github.com/square/crossfilter

[20] LaTeX:Wikibook in English. [Online]. Available: https://en.wikibooks.org/wiki/LaTeX/

[21] can i use? website. [Online]. Available: http://caniuse.com/#search=webgl

[22] Deloitte. Tech trends 2017. [Online]. Available: https://www2.deloitte.com/uk/en/pages/technology/articles/tech-trends.html

[23] S. Penadés. Object pickin. [Online]. Available: https://soledadpenades.com/articles/three-js-tutorials/object-picking/

[24] D. Lyons. Rendering text in webvr. [Online]. Available: https://developers.google.com/web/showcase/2017/within

[25] 10 best uses of virtual reality in marketing. [Online]. Available: http://mbryonic.com/best-vr/

[26] Webvr api — mdn. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebVR_API