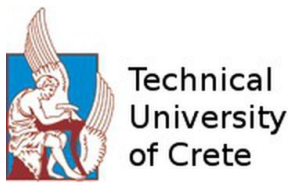
Department of Electronic and Computer Engineering
Technical University of Crete

Implementation of a web - based application for the creation of musical loops via interaction with 3D objects

Saringelos Athanasios



Dissertation Thesis Committee:
Katerina Mania, Associate Professor (Supervisor)
Stavros Christodoulakis, Professor
Georgios Chalkiadakis, Associate Professor

Acknowledgments

This thesis is a result of a collective effort. As a token of my appreciation I would like to mention some of the contributors.

Firstly and foremost I would like to thank my Supervisor, Associate Professor Katerina Mania, not only for her advice, guidance and supervision through the whole process but also for the real excitement she showed from the beginning of this journey. Her ideas were also precious to the project. She trusted me and I hope she will keep undertaking projects that involve imagination and creativity in the future because many students need to have this alternative.

I would also like to thank all the authors of the ThreeJS library and especially the main author Ricardo Cabello, who created and released for free a great Web Graphics Library such ThreeJS is. Also, I would like to thank all the authors of the new Web Audio API. Many other developers from around the world, who helped me in various ways should be mentioned here, but this is unfortunately impossible.

My friends played a great role in developing this thesis and I want to thank them for their support and their ideas which led to a great improvement of the project.

I would also like to thank Stefan for our discussions, Stavros for his advice, Sotiris for our cooperation, Chris Wilson for giving me enlightenment in many topics, Professor K. Ougrinis for his imaginative ideas.

Finally, I would like to thank my family for their support and infinite patience!

Abstract

Some years ago, the web was considered a useful medium for exchanging structured data or conduct other trivial activities. Today, the web is the main focus of many software developers, as its support capabilities are getting so strong that the functionality and appearance of the web applications are very similar to what the native offer. But its advantage lies on their flexibility : the code of a web app is able to reach instantly every device in the world that employs a browser. In contrast, the native apps we are used to run on PCs, smartphones or tablets, are tied to a specific operating system and device. Due to increasing focus on web development the World Wide Web Consortium (W3C), which is the main international standards organization for the Web, has put all of its efforts to refine the technologies the web is built upon by publishing new standards and APIs, to enable the web to be the main platform of choice for building applications.

This thesis (Looparound) exploits two of the latest APIs and aims to develop a web application that gives the user the capability to navigate in a 3D environment, interact with it and manipulate the sounds its objects can produce, create audio loops with great time precision in extreme speed, apply some effects on them and even produce sinusoidal sounds with a parametric oscillator. In a nutshell, this web application provides to the user a 3D environment that contains a number of 3D interactive objects. Every object is interactive, conducts a specific movement and has parameters the user can alter. Every object is also recordable via the 2D sequencer at the top of the screen. The user can either record an object's sound by interacting directly with it or by interacting with the 2D sequencer itself. The 2D sequencer has also its own parameters. There are also overall parameters which apply universally to the whole application. In addition, there is the ability to connect a motion controller (e.g. Leap Motion) and a HMD(as Oculus Rift) to navigate and interact with the scene. The user is also able to download his recording in the WAV format.

This thesis shows not only the great tools the new APIs are providing to us, but also the fact that web applications now can have really complex structures , many abilities and manipulate enormous data structures efficiently. The important part is that all of the above are achieved inside the browser environment, giving the ability to the user to access thousand of modern applications just by clicking a url (rather than install numerous plug-ins or extensions on their computer). A browser (preferably Google Chrome) and a fairly modern GPU and CPU are enough to run this application almost in any device.

Table of Contents

List of Figures

1 Introduction and objectives

1.1 Introduction

Web Applications have evolved much the last few years. A Web Application (which we will call “web app” for the rest of this document) used to be a mere tool for accessing, editing and storing data from a database remotely. A modern web application though, can do much more than that and that is going to be explained by the next paragraphs.

First of all, users demand from the applications to serve them in many and different ways. Not only do they want that, but they are always desirous of betterment of the app experience, appearance and “feel”. Until recently, their need for fulfillment of the above was satisfied by the desktop applications. There was a disturbing problem with that fact, though. Every user that wished to have a software product to do certain things, had to download (or buy it if it was not free of charge) the compatible version for his current Operational System and his current device characteristics. After that, he had to install it on his computer. It is a common secret that this was not always an easy task, especially for the new and inexperienced users of a computer. All of these, in the case that the developers of that piece of software had actually decided to release a version that was compatible with your Operational System and your device. Otherwise you were not able to have the functional software product in your hands at all.

On the other hand, internet speed was improving very quickly around the globe together with the performance of the various device components as CPU, GPU, RAM Storage and others. A new way of exploiting these assets came to the surface slowly in the beginning but surely. Web browsers (which are basically the host environment of the web app) started to integrate these assets into their artillery.

This fact led the developers around the world to start changing or even reinventing the way they used to create an app. They embraced this evolution and begun to develop, bearing in mind that most of the tools a desktop app developer can use, are available now also for the web apps. This gave the newly developed apps abilities that they did not have before.

In addition, the problem of the absence of universality we mentioned before, begun to fade after the rise of this enhanced web platform, which was called "*Web 2.0*". These apps were not much concerned about your operational system or your device, nor did they need installing. You just have to have a modern browser installed in your device and rarely a few plugins.

1.2 Aims and objectives

Some of our principal aims and objectives are to exploit the new dimensions Web 2.0 and HTML5 creates for the web applications, to learn the new tools and APIs recently released together with HTML5 and to document a full experience of the process of designing and developing an application of this kind. In addition, we did not aimed for perfection of deliverables but for a more experimental approach of constructing an app using these newly found tools of the web.

On the other hand, the passion me and my supervisor share for 3D Graphics, music and audio led us to choose to create an Audio/Visual application using the latest libraries, frameworks and web tools that are available. We finally came to the decision to build an interactive 3D environment in the browser which would enable the user to navigate inside it and interact with it in various ways. Not only that, but the 3D objects of the environment could produce sound clips after interaction and by this process we decided to enabled the user to build sound loops and compose little songs using a highly demanding (performance wise) graphic sequencer with which he could edit the loops even further.

We also aimed to become familiar with the process of analyzing which of the tools that are in offer could suit our needs, investigating their possible abilities and their potential drawbacks. A secondary aim was to dive into javascript architecture and programming, since this language seems to be a necessary tool for the future and it also has other uses too. A new browser environment was being born and we wanted to be a part of it and a part of the future it determined.

Finally, we aimed to create an interesting, entertaining application that is attractive to people, can show them a glimpse of how future applications will be and maybe let them create some nice audio loops and even download them on their pc.

1.3 Brief description

Looparound is a single page web application (SPA). A single-page application is a web application or website that fits on a single web page with the goal of providing a more fluid user experience similar to a desktop application. In a SPA, either all necessary code is retrieved with a single page load, or the appropriate resources are dynamically loaded and added to the page as necessary, usually in response to user actions [1].

After the page has loads, the user is having the 3D scene in front of him. He is informed about how to interact with it but there are also tips that pop up in every mouseover that are of help. Navigating is being done using the classic desktop gaming method WASD and in addition there is "space" and "shift" key that enable the camera to move up/down respectively. Also, using the dragging technique of the mouse the user is able to rotate the camera focus as it is moving with the keys. This is a new method of manipulating the camera.

Then he has to activate all objects that are invisible inside the scene by default. The hidden objects in the scene can become visible by interact with the only object (out of the interactive and audio-related objects) that is able to make the next appear. The user has to interact with every newly visible object until they all appear in the scene. This "chaining" is a gaming achievement concept. In addition, every time an object is clicked an optical soundwave is produced together with the sound that shows the radius of the object's reach of visibility.

After becoming familiar with the scene and its objects, the user is able to interact with the 2D sequencer that is responsible of visualizing, saving and editing the audio loops. The sequencer is built with a HUD display on our mind, meaning that is a 2D graphic sequencer which stays always in front of the user while navigating the scene. Each interactive object can be visible in the sequencer by clicking its setting icons near it. After it is visible there, the user can record its sound clips by setting slots to ON/OFF in the sequencer. The sequencer has many features built in such as **play, replay, reset, stop, mute, volume level, tempo and record.**

1.4 Structure of the thesis

In this chapter we introduced to the user the approach we followed to choose the technologies and tools as well as a brief description of the app's functionality and our goals.

Chapter 2 analyzes the research we did to examine our possible choices for development frameworks, libraries and APIs as well as their potential benefits or drawbacks.

Chapter 3 analyzes the system architecture as well as the implementation technologies. Architecture is a vital part of any modern scalable and maintainable web application.

Chapter 4 analyzes the requirements we set to satisfy in the beginning of this journey. We approached this process as if we had a client to work with and serve him from our position as development team.

Chapter 5 focuses on how the implementations was done in the end. Which tools we used, which tools we invented and how they were connected with each other.

Chapter 6 is a summary of the whole experience and a suggestion of potential future uses, needs and opportunities.

2 Research Overview

2.1 Introduction

In this chapter we are going to analyze our research we did on new technologies and tools but also on the behaviour a system like this could have. We will also take a look on other applications that share pieces of functionality with ours. We will begin the analysis of the research by making a description of the current state of the web 2.0 in general.

2.2 Modern web applications

Web 2.0

The term **Web 2.0** popularized by Tim O'Reily at the O'Reily Web Conference in late 2004. Web 2.0 describes World Wide Web sites that use technology beyond the static pages of earlier Web sites. Most important features of this new Web 2.0 are : improved interaction with the web app, “heavy” code blocks that run in the client side (the browser) usually written in user-friendly languages such as Javascript, offline capabilities, great manipulation of local storage and enormous communication between client and server side in real time.

These features enabled essential technologies to be exploited by the developers. Ajax requests for instant data interchange with the server, Cloud Computing, responsive User Interface, Web services, Graphics, Audio manipulation and many other amongst them. The web was now a place that big things could happen. The interaction between the users and the app, the users and other users, the client and the server was never as impactive as then before.

Some of the new types of web apps that arose are :

- **Social Networking Websites** - serve as platforms to build social networks or social relations among people who share similar interests, activities, backgrounds or real-life connections.
- **Online Marketing Websites** - featuring real time bidding etc.
- **Scientific Based Websites** - serving as tools for extreme scientific processing and not only.
- **File Storage/Sharing Website** - suitable for file sharing, hosting and storing.
- Multimedia Websites - featuring live video calls, video playbacks, photo shooting etc.
- **Blogging Websites**
- **Curating with RSS Websites** - which use a family of standard web feed formats to publish frequently updated information to the listeners.
- **Gaming Websites**

- **Audio/Graphic Websites** - interactive web apps with desktop capabilities in audio and graphics manipulation.

HTML5

One of the key features of Web 2.0 and a fundamental factor of its current form is **HTML5**. HTML5 is a core technology markup language of the Internet used for structuring and presenting content for the World Wide Web. As of October 2014 this is the final and complete fifth revision of the HTML standard of the **World Wide Web Consortium (W3C)**. The previous version, HTML 4, was standardised in 1997. HTML5 represents the biggest leap forward in Web Standards in almost a decade. Unlike the specifications that came before it, HTML5 is not merely intended to present content to a Web browser. Its goal is to bring the Web into maturity as a full-fledged application platform -a level playing field where video, sound, images, animations, and full interactivity with your computer are all standardized. And it may be a long way off still, but elements of HTML5 are already reshaping the way we use the Web.

Its core aims have been to improve the language with support for the latest multimedia while keeping it easily readable by humans and consistently understood by computers and devices (browsers, parsers etc.) .

In particular, HTML5 adds many new syntactic features. These include the new <video>, <audio> and <canvas> elements, as well as the integration of scalable vector graphics (SVG) content (replacing generic <object> tags), and MathML for mathematical formulas. These features are designed to make it easy to include and handle multimedia and graphical content on the web without having to resort to proprietary plugins.

In addition, HTML5 specifies scripting application programming interfaces (APIs) that can be used with JavaScript. Existing document object model (DOM) interfaces are extended and *de facto* features documented. There are also new APIs, such as:

- Timed media playback
- Offline Web Applications
- Document editing
- Drag and drop
- Cross-document messaging
- Browser history management
- MIME type and protocol handler registration
- Web Storage

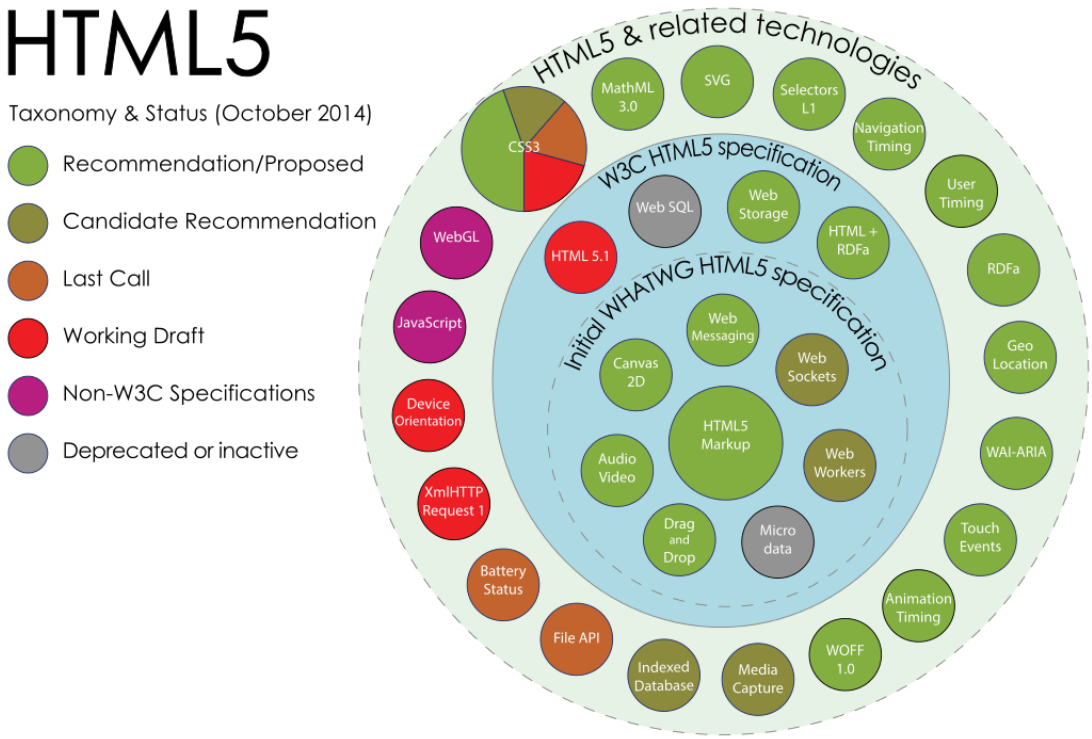


Figure 1 : HTML5 new APIs

Rich Media Applications

A rich Internet application (RIA) is a Web application that has many of the characteristics of desktop application software, typically delivered by way of a site-specific browser, a browser plug-in, an independent sandbox, extensive use of JavaScript, or a virtual machine. Adobe Flash, JavaFX, and Microsoft Silverlight are currently the three most common platforms, with desktop browser penetration rates around 96%, 76%, and 66%, respectively (as of August 2011).

Google trends shows (as of September 2012) that frameworks based on a plug-in are in the process of being replaced by HTML5/JavaScript-based alternatives :

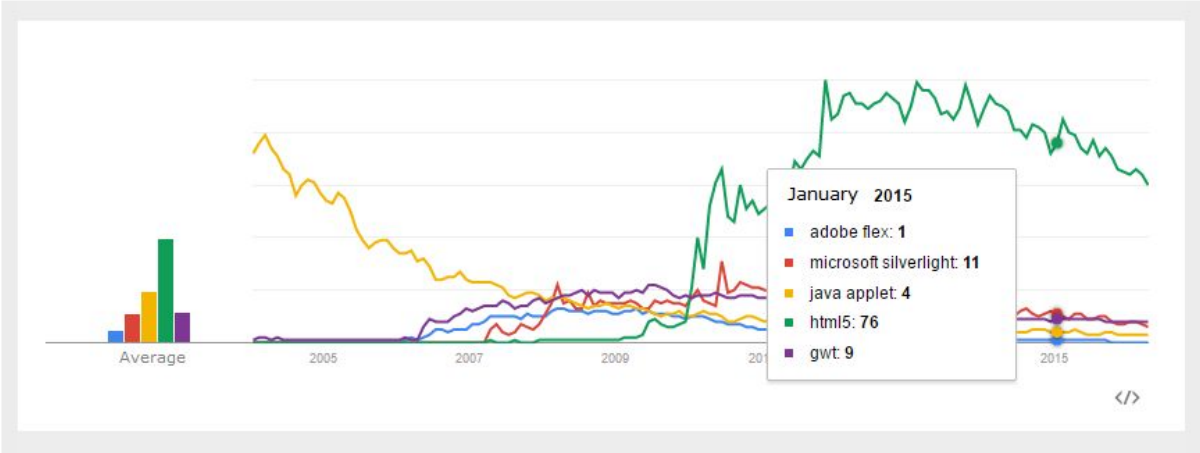


Figure 2 : Google trends plug in to HTML5 comparison

Users typically need to install a software framework using the computer's operating system before launching the application, which typically downloads, updates, verifies and executes the RIA. This is the main differentiator from HTML5/JavaScript-based alternatives like Ajax that use built-in browser functionality to implement comparable interfaces. As can be seen on the List of rich Internet application frameworks which includes even server-side frameworks, while some consider such interfaces to be RIAs, some consider them competitors to RIAs; and others, including Gartner, treat them as similar but separate technologies.

The future

In November 2011, there were a number of announcements that demonstrated a decline in demand for rich internet application architectures based on plug-ins in order to favor HTML5 alternatives. Adobe announced that Flash would no longer be produced for mobile or TV (refocusing its efforts on Adobe AIR). Pundits questioned its continued relevance even on the desktop and described it as **the beginning of the end**.Research In Motion (RIM) announced that it would continue to develop Flash for the PlayBook, a decision questioned by some commentators Rumors state that Microsoft is to abandon Silverlight after version 5 is released. The combination of these announcements had some proclaiming it **the end of the line for browser plug-ins.**

Looparound as a rich application

The app for which this thesis was written, is a Rich Media Application (RMA). However, after acknowledge the fact that a new architecture is emerging for modern applications with which they are independent of plugins and extensions, we decided to follow this path for Looparound too. Looparound uses exclusively HTML5/javascript APIs to achieve its results in Audio and Graphics representation. Next, we will analyze some of the APIs and libraries that were available to choose from and why did we choose them.

2.3 3D Graphics on the web

WebGL API

The OpenGL specification by the Khronos Group describes an abstract API for drawing 2D and 3D graphics on a device. It is a cross platform library that defines a set of functions that are available to the client program.
WebGL (Web Graphics Library) is based on OpenGL ES 2.0 and provides an API for 3D graphics. WebGL is a JavaScript API for rendering interactive 3D computer graphics and 2D graphics within any compatible web browser without the use of plug-ins. WebGL is integrated completely into all the web standards of the browser allowing GPU accelerated usage of physics and image processing and effects as part of the web page canvas.It uses the HTML5 canvas element and is accessed using Document Object Model interfaces. Automatic memory is provided as part of the JavaScript language.

In early 2009, the non-profit technology consortium Khronos Group started the WebGL Working Group, with initial participation from Apple, Google, Mozilla, Opera , and others. Version 1.0 of the WebGL specification was released March 2011. WebGL is widely supported in modern browsers. That includes Desktop, Mobile and Tablet browsers. Without any other tools/plugins but a modern browser and a fairly good GPU, every user is able to access instantly any application that contains 3D graphics.

Libraries for WebGL abstraction

Our research for how to implement fast and solid graphics for our application led to a many WebGL libraries that are free for the users. We will analyze the most popular amongst them.

Flash Player

Adobe Flash Player (labeled Shockwave Flash in Internet Explorer and Firefox) is freeware software for using content created on the Adobe Flash platform, including viewing multimedia, executing rich Internet applications, and streaming video and audio. Flash Player can run from a web browser as a browser plug-in or on supported mobile devices Flash Player was created by Macromedia and has been developed and distributed by Adobe Systems since Adobe acquired Macromedia. Flash Player supports vector and raster graphics, 3D graphics, an embedded scripting language called ActionScript, and streaming of video and audio. ActionScript is based on ECMAScript, and supports object-oriented code, and is similar to JavaScript.

Babylon.js

Babylon.JS is undoubtedly one of the best JavaScript 3D games engine out there in the wild as of now for creating professional grade games that you can sell. Babylon.JS is the outcome of David Catuhe's love for 3D gaming engines. He has experience in creating 3D games engines in DirectX, OpenGL, and Silverlight etc. and has finally created one in JavaScript. Some of the key features of Babylon.js framework include scene graphs with lights, cameras, materials and meshes, collisions engine, physics engine, audio engine and optimization engine at the core. Its primary disadvantage though is that is fairly young (2013) with a small community of users that it is still evolving. [2]

Three.js

Three.js is another comprehensive and powerful JavaScript 3D library for doing everything 3D, right from creating simple 3D animations to creating interactive 3D games. Three.js library brings much more than just supporting WebGL renderer, it comes packed with SVG, Canvas and CSS3D renderers as well. However, from games perspective you might want to focus just on WebGL renderer of three.js library. It has a huge support from a large community and even the creator (MrDoob) is available for

questions and advice through github, stackoverflow and his personal blog. Three.js has also many other libraries that can work on top of it and make it even more powerful. It is an open-source library that currently holds the highest popularity. In addition, it has a very small learning curve enabling even the most inexperienced user to start programming web graphics with it. [2]

Playcanvas.js

PlayCanvas is an enterprise grade open source JavaScript based WebGL game Engine that has got tons of developer tools to help you build 3D games within no time. PlayCanvas.js is built by a professional community and was not an open source initially but now you can fork it on GitHub and start using it for your next 3D game project, free of cost. It also comes with cloud based editor that runs in your browser, so getting started with PlayCanvas is as easy as navigating to the Editor URL. Editor supports collaboration among teams which essentially means many people can work on the same project in parallel. [2]

D3.js

D3.js is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG, and CSS. D3's emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation. [2]

Unity

Unity is a cross-platform game engine developed by Unity Technologies and used to develop video games for PC, consoles, mobile devices and websites. First announced only for OS X, at Apple's Worldwide Developers Conference in 2005, it has since been extended to target more than fifteen platforms. It is a cross-platform game engine with a built-in IDE developed by Unity Technologies. [2]

Our choice

Our choice was three.js library because of many reasons. First of all, it is a free open-source framework that has a huge supporting community to back it up and when someone is using a new technology this is really important. Secondly, it suited our needs for the specific application. We did not want to program in very high level, nor in a very low level. We did not aiming to develop a classic 3D game but on the other hand Looparound was not a data visualization app or a simple graphical representational app but it had more needs. Also, three.js is written in pure javascript and it abstracts webgl in a way that makes it very easy to learn, understand and integrate. Finally, the abundance of project we encounter in the web using three.js and on the three.js support website made us think that this is the way to go. [3]

2.4 Web Audio API

Audio on the web has been fairly primitive up to this point and until very recently has had to be delivered through plug-ins such as Flash and QuickTime. The introduction of the audio element in HTML5 is very important, allowing for basic streaming audio playback. But, it is not powerful enough to handle more complex audio applications. For sophisticated web-based games or interactive applications, another solution is required. The Web Audio API aims to include the capabilities found in modern game audio engines as well as some of the mixing, processing, and filtering tasks that are found in modern desktop audio production applications.

The APIs have been designed with a wide variety of use cases in mind. Ideally, it should be able to support any use case which could reasonably be implemented with an optimized C++ engine controlled via JavaScript and run in a browser. That said, modern desktop audio software can have very advanced capabilities, some of which would be difficult or impossible to build with this system. Apple's Logic Audio is one such application which has support for external MIDI controllers, arbitrary plugin audio effects and synthesizers, highly optimized direct-to-disk audio file reading/writing, tightly integrated time-stretching, and so on. Nevertheless, the proposed system will be quite capable of supporting a large range of reasonably complex games and interactive applications, including musical ones. And it can be a very good complement to the more

advanced graphics features offered by WebGL. This API has been designed so that more advanced capabilities can be added at a later time.

Using the Web Audio API was the only way to go if we did not want to import plugins or extensions. Which, as we have previously stated, we did not want to. Web Audio API became then a vital part of our development.

3 Requirement Analysis

3.1 Introduction

What does the analysis include?

Before we even begin to think what technologies we might need to build our app, we have to have an idea of what we wish to build. What our application is going to be about, what current needs it fulfills and what new abilities might give to the users. After having the initial, general idea for our web application and what we aim to do by creating it within the context of this thesis, it's time to begin mapping out our application. We have to create a visual model of various aspects before we ever write one line of code. In doing so, it will help eliminate any problems and ensure that functionality that we chose to be in the application doesn't get missed.

We broke down the Requirement Analysis into the following steps:

Requirements Gathering

this is the practice of collecting the requirements of the system from users, customers and other stakeholders. Because of the purpose of this application, the choice of the app's functionality was ours and it was based on the academic needs of this thesis.

Use Case diagrams

A use case [4] is a methodology used in system analysis to identify, clarify, and organize system requirements. In this context, the term "system" refers to our web application. Use case diagrams are employed in UML (Unified Modeling Language), a standard notation for the modeling of real-world objects and systems.

Storyboarding

A storyboard is a visual representation of the user interface of an application, showing screens of content and the connections between those screens. A storyboard is composed of a sequence of scenes, each of which represents the current state of our application. Scenes are connected by segue objects, which represent a transition between states.

3.2 Requirements gathering

Our application is going to be a tool for giving the user the opportunity to become familiar with 3D environments, be able to navigate, interact with them and create audio loops, play them, record them as well as edit them and update them.

General requirements

The application must be completely independent from plug-ins, add-ons etc.

The application must use pure WebGL/Javascript to create the graphic environment.

The application must use ThreeJS as a high-level library to manipulate WebGL more efficiently.

The application must run in any browser that supports WebGL, Web Audio API and Javascript.

The application must use the new Web Audio API to manipulate the sound clips efficiently.

There must be a 3D environment, with interactive 3D objects.

The whole 3D environment must be thematically related to the sounds the object can produce.

The 3D environment must have limits for navigation.

3D objects

The objects must be able to receive mouse events.

The objects must conduct a specific movement when clicked.

The objects must produce a specific sound when clicked.

The objects should graphically represent the time and the duration of the sound they produce.

The objects should be related thematically to the sound clip they are assigned to.

The objects must have graphically represented parameters of the sound they produce (such as reverb, volume)

The objects must have Recording Mode that can be switched off and on.

2D sequencer looper

There must be a 2D sequencer-looper that visualizes time in discrete steps.

The sequencer-looper must enable the user to create, edit and delete audio loops.

There must be classic sequencer parameters that the user can change (such as tempo, volume, mute).

There must be a play, pause and reset functionality in the sequencer-looper.

The user must be able to record the loops via the sequencer and download them in mp3 format.

There must be a specific number of time slots (possibly 16).

There will be a specific number of object tracks (possibly 5).

Each slot must be graphically represented by a 2D shape (possibly circle).

Each slot must be clickable.

Each slot must have 2 states : “on” and “off", which must be graphically represented and are changeable by clicking.

Each track will be assigned to a current 3D object.

Each track will be graphically recognized as an object's track.

Tracks must be from 0 to 5 at any moment.

The user must be able to assign/deassign an object to one of the 5 tracks (if they are not all currently assigned to other objects).

The assignment can happen by change the state of the object's “Recording Mode” (possibly in the object’s parameters)

Every track that has an object assigned to it must be visible and every track that has not, invisible.

Audio

Audio must be good quality in mp3 format.

Audio clips must be loaded synchronously in “application load” time.

There must be different “kits” of sound samples that are thematically organized and that are available for the user to select any of them.

Audio processing must be handled very strictly time-wise because of the need of scheduling the production of a large amount of sound clips in parallel and in specific time stamps.

Audio must be parametric and able to add effects on it.

There can be an ability of creating sound waves with specific parameters from the user (sound wave type, frequency etc.).

3.3 Personas

Personas are fictional users that correspond to the different profiles of real world users. By describing user personas, the designer understands user needs and can deduce the goals the users will be looking to achieve when using the application.

Thus, the purpose of personas is to create reliable and realistic representations of your key audience segments for reference. These representations should be based on qualitative and some quantitative user research and web analytics.

We have decided to include in our report the following personas :

Bill, 12 years old, student

Bill has been very interested in the modern applications that emerged in the web the last few years. He also likes listening to music, especially hip hop. Bill would be very excited to try to create sound loops interacting with the 3D environment, then download them and send them to his friends and invite them to do the same.

Archy, 18 years old

Archy likes videogames. He also likes surfing in the internet. Navigating in the 3D environment, interact with the objects, trying the effects and the various features of the application. He tries every different combination of the features of the application to

synthesize the best audio loop he can. He receives this application as a challenge and his mission is to fully understand it and exploit the full potential of the application.

Salomi, 25 years old web developer

Salomi has just graduated and wants to make a career as a web developer with her specialty being in Web Audio/Graphic Applications. He searches the web for applications that feature advanced web audio/graphic possibilities so she can learn the new abilities of the recent Graphic/Audio APIs of the internet. She uses extensively the application, learning how to create a WebGL scene, add objects, interact with them. She also learns how to synchronize with great precision and performance. She would like to work in a web developer team that aims to create games for the internet or "transfer" old desktop games to the web.

3.4 Use case diagrams

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

Below we will graphically represent the basic Use cases and the extensions for some of them. Next, we will describe each use case's steps with detail.

Diagram representation

Below are documented the principal use cases for a user. Not all the use cases are simple, thus we furtherly explain the complex ones in the next diagram.

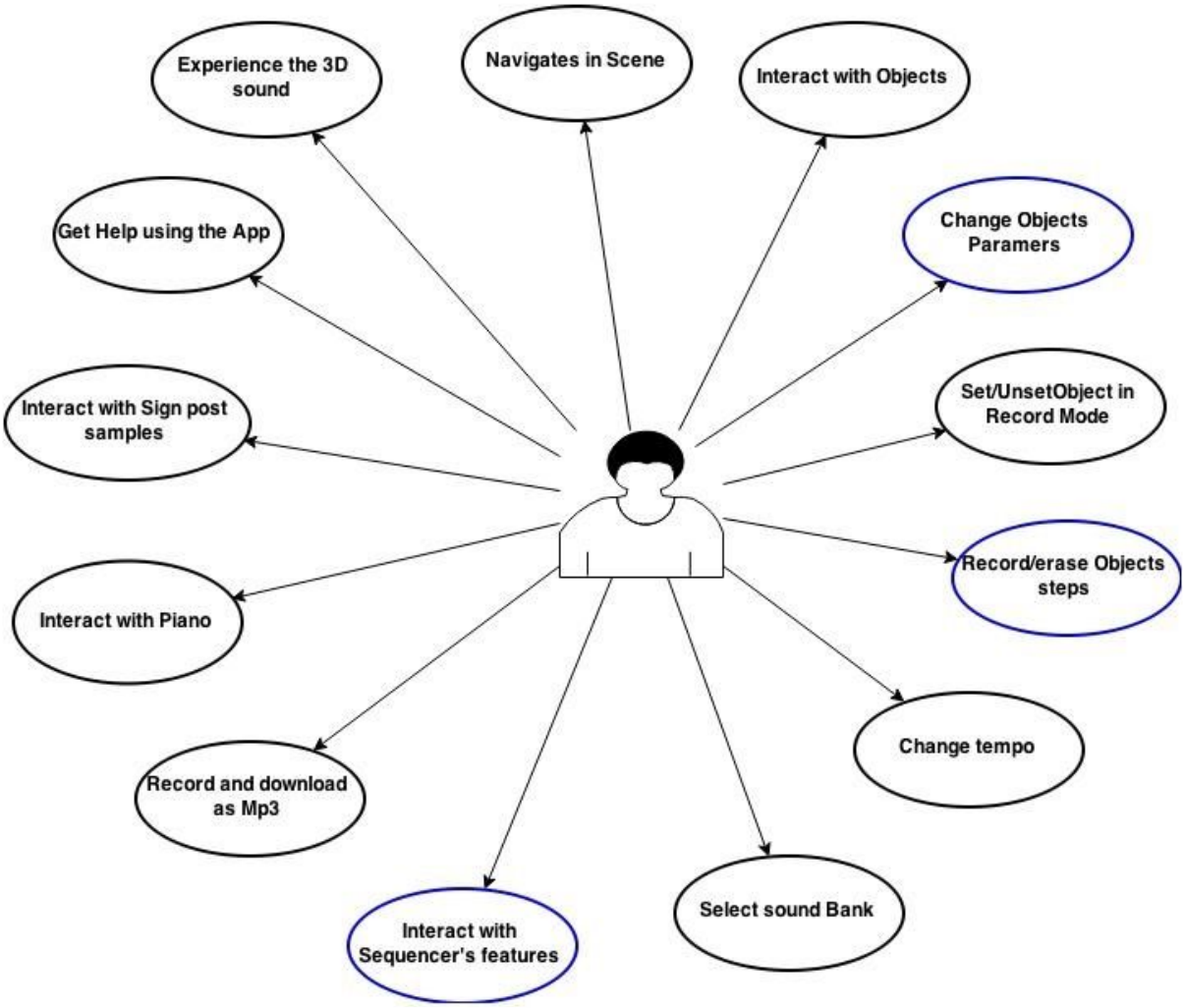


Figure 3 : Use Case Diagram for the User complex Use Cases

Below the complex use cases are deconstructed to the simple ones :

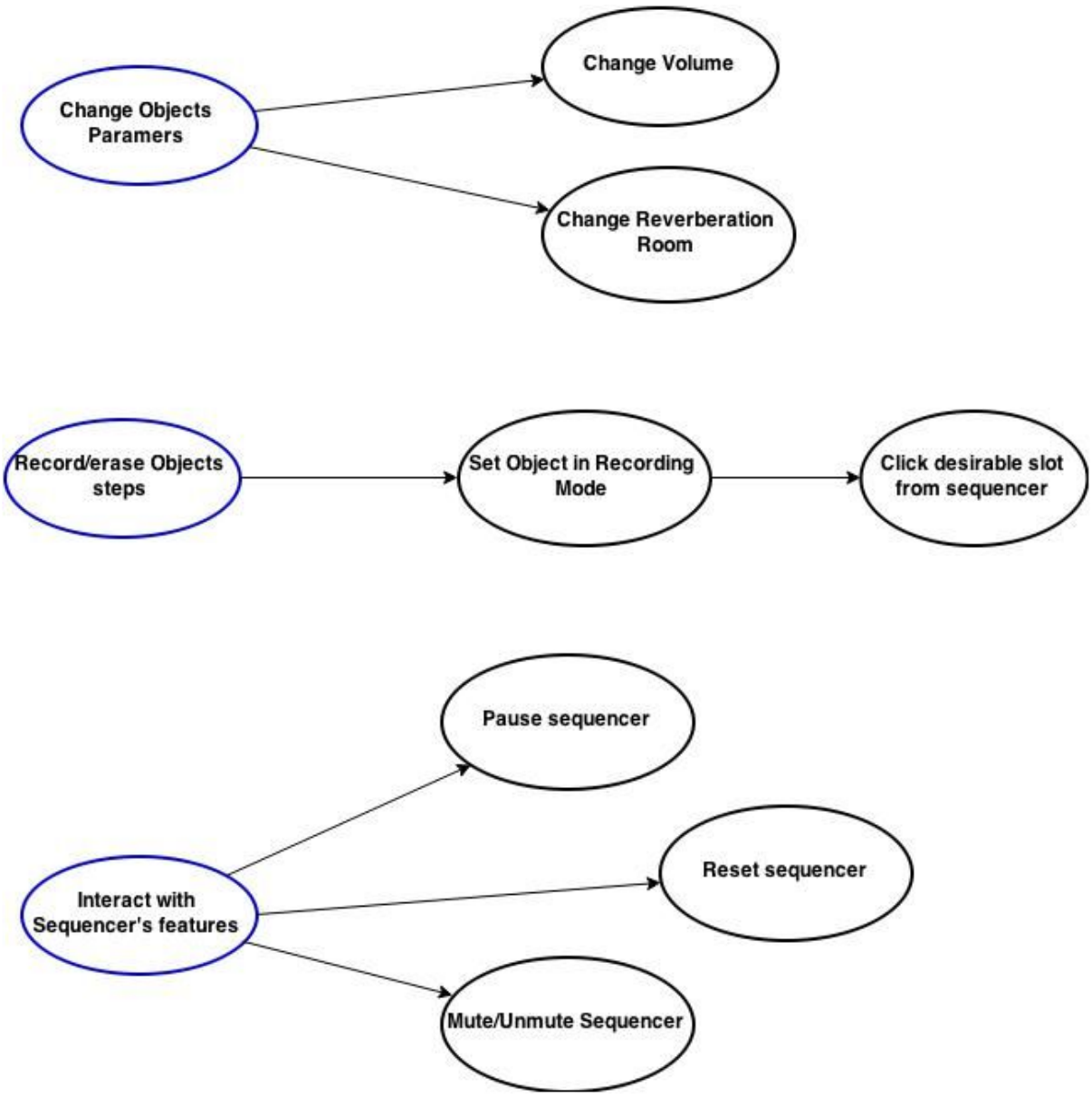


Figure 4 : Use Case Diagram for the complex Use Cases

Detailed analysis of use cases

The User wants to Navigate the Scene

The user uses the keyboard and the mouse events to navigate the 3D scene.

The User wants to interact with objects

The user mouse over an object. The cursor changes. Then the user clicks the object and the object moves and produces a sound and a graphical soundwave (colored sphere that has an increasing radius).

The User wants to change object's parameters

1. The user moves the camera towards an object. The user click on the small ball and drags it up or down and the volume of the specific object changes.
2. The user moves the camera towards an object. The user scrolls with the mouse above the small cylindrical object and that changes its color, size and the reverberation that the sample now has.

The User wants to activate or deactivate object's recording mode

The user moves the camera towards an object. The user click on the “recording mode” button and it strart to blink (if it was not in recording mode) or stops blinking (if it already was)

The User wants to record or erase object's slots.

The user approaches an object and click its “recording mode” button to put it in recording mode. If there are less than 6 tracks in the sequencer the track of the specific object appears in the sequencer. The user click in the desirable slot and record or erase the specific slot.

The User wants to change the tempo.

The user clicks on the upwards arrow in the upper left of the screen to increase the tempo or in the downwards arrow to decrease it. The new tempo value appears next to the buttons.

The User wants to select a sound bank.

The user clicks on one of the four different icons in the upper right on the screen. The sound sample that each of the object produce have changed.

The User wants to interact with sequencer features

1. The user clicks on the play/pause button and the sequencer stops or starts to play.
2. The user clicks on the reset button and all recorded slots of all tracks are erased.
3. The user clicks on the mute/unmute button and the sequencer changes to volume 0 or 1

The User wants to record and download his composition as mp3.

The user creates a loop. The user click on the “vinyl” icon in the sequencer. The application starts to record every sound that comes out of the system. The user clicks again on the “vinyl” icon and the recording stops. A message appears giving the user the ability to download the sound clip in .wav format. The user click on the link and the browser download the sound file.

The User wants to interact with the piano.

The user click a piano key and the application starts an oscillator that produces a sound with the specific frequency to which the piano key is assigned to. Also, a graphical representation of a sine with its size analogous to the frequency height appears.

The User wants to interact with the sign posts.

The user clicks on a sign post and it produces a sound sample. In addition, the clicked signpost starts to rotate by the Y axes till the user clicks it again.

The User wants to get help and info about the application use.

The user clicks on the question mark button in the bottom of the screen. A box appears. When the user mouses over an object or a control element of the application, some information appears in the box.

The User wants to experience the 3D sound of the application.

The user navigates the scene, moving around the object of the scene that produce sounds. When he clicks an object or the sequencer plays a recorded sound the sound volume and panning are calculated based on the position of the camera in the scene so the user (especially if he wears headphones) listens to the sound as if it was produce in front of him, next to him, behind him, far away from him etc.

3.5 Prototyping

By creating an application prototype, we gain insight on what the final app will actually look like. We can eliminate problems before we have gotten too deep into the programming to turn back or need to implement a “expensive” fix.

In developing this application, we created some rough sketches of what the application UI might look like. Then, we presented these sketches to other developers and simple users, potentially users of the application, and we got some feedback on the usability of the app.

Below, some of the initial rough sketches will be shown, as well as some comments and the feedback of the people we showed them.

First phase : initial rough prototypes

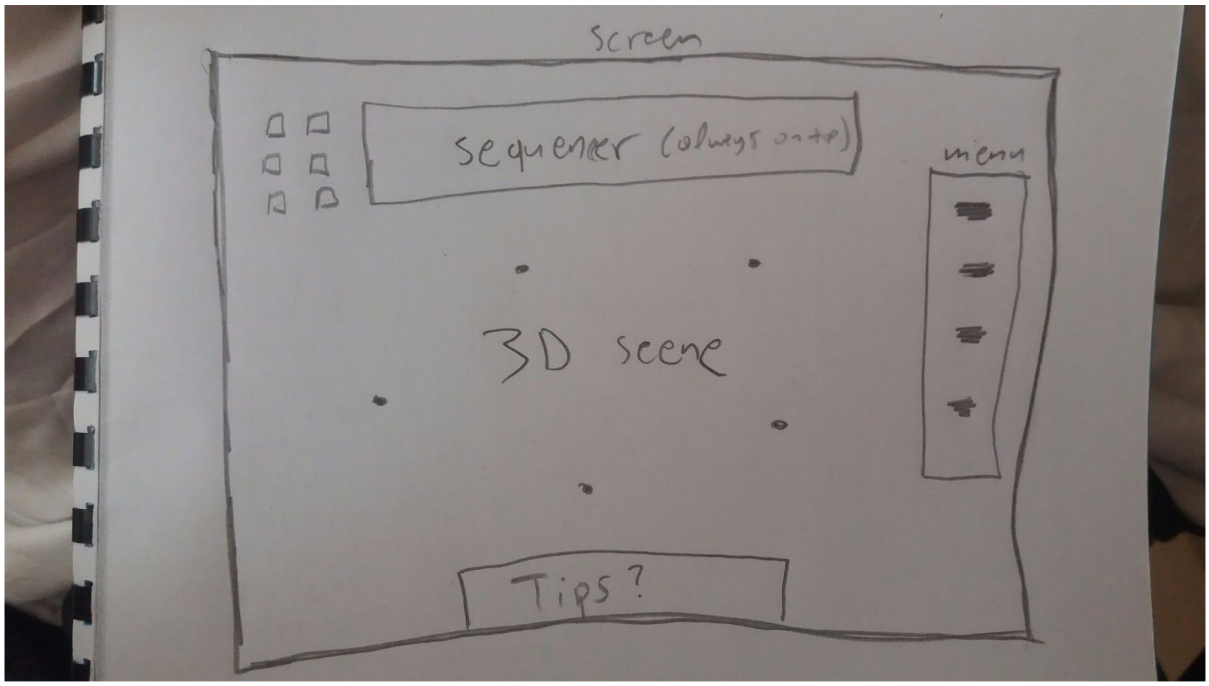


Figure 5 : Initial hand-written prototype of the App's UI

Here the basic idea is documented via a hand-written paper diagram that shows the basic components of the UI

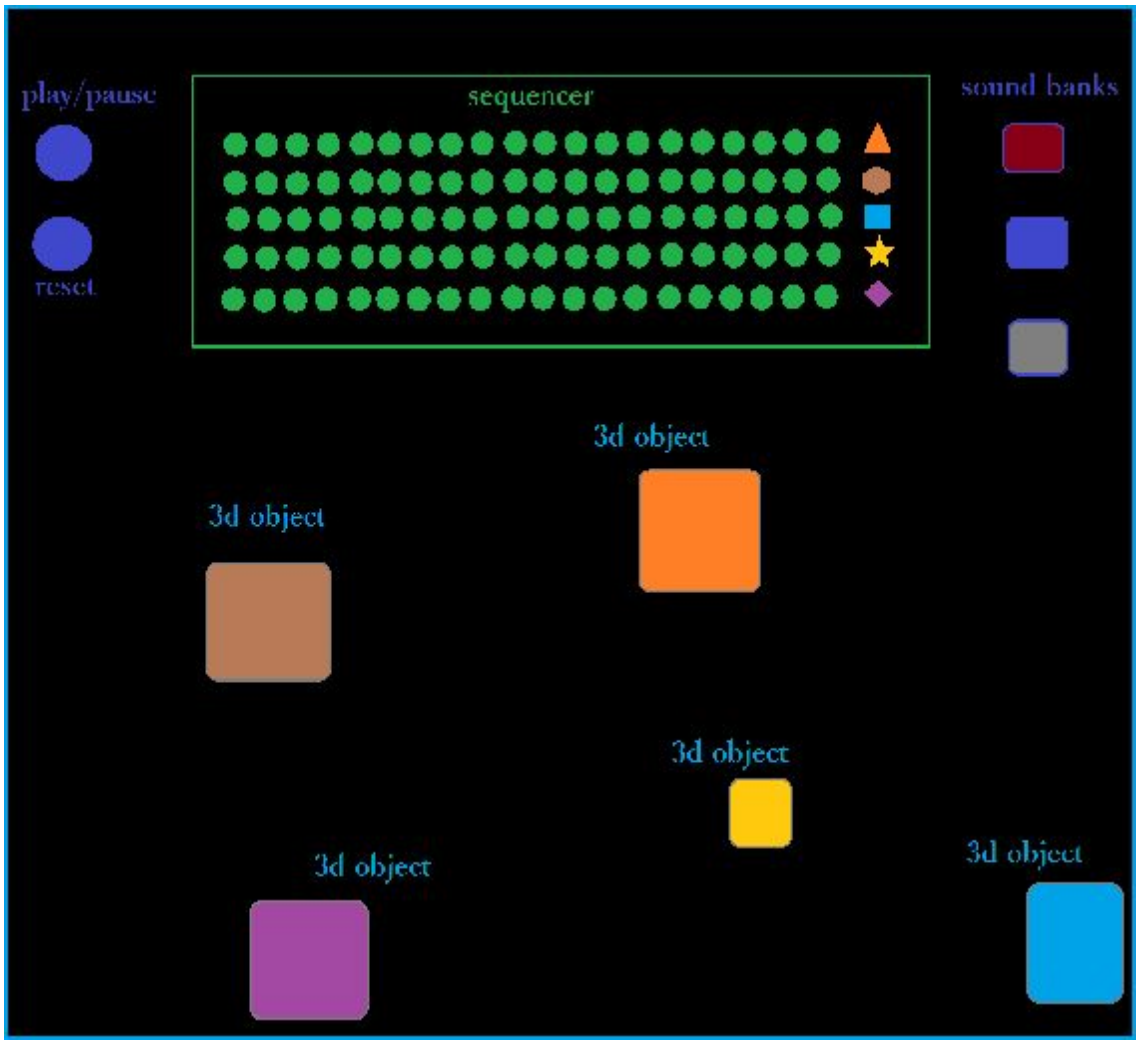


Figure 6 : Initial Visual prototype of the App's UI

This is one of the first UI prototypes using a computer. It represents in 2D the positions of the 3D objects and the sequencer at that time of the process. In the right of the sequencer are the sound banks with which the user is able to change sound samples and in the left are some basic buttons for controlling the sequencer.

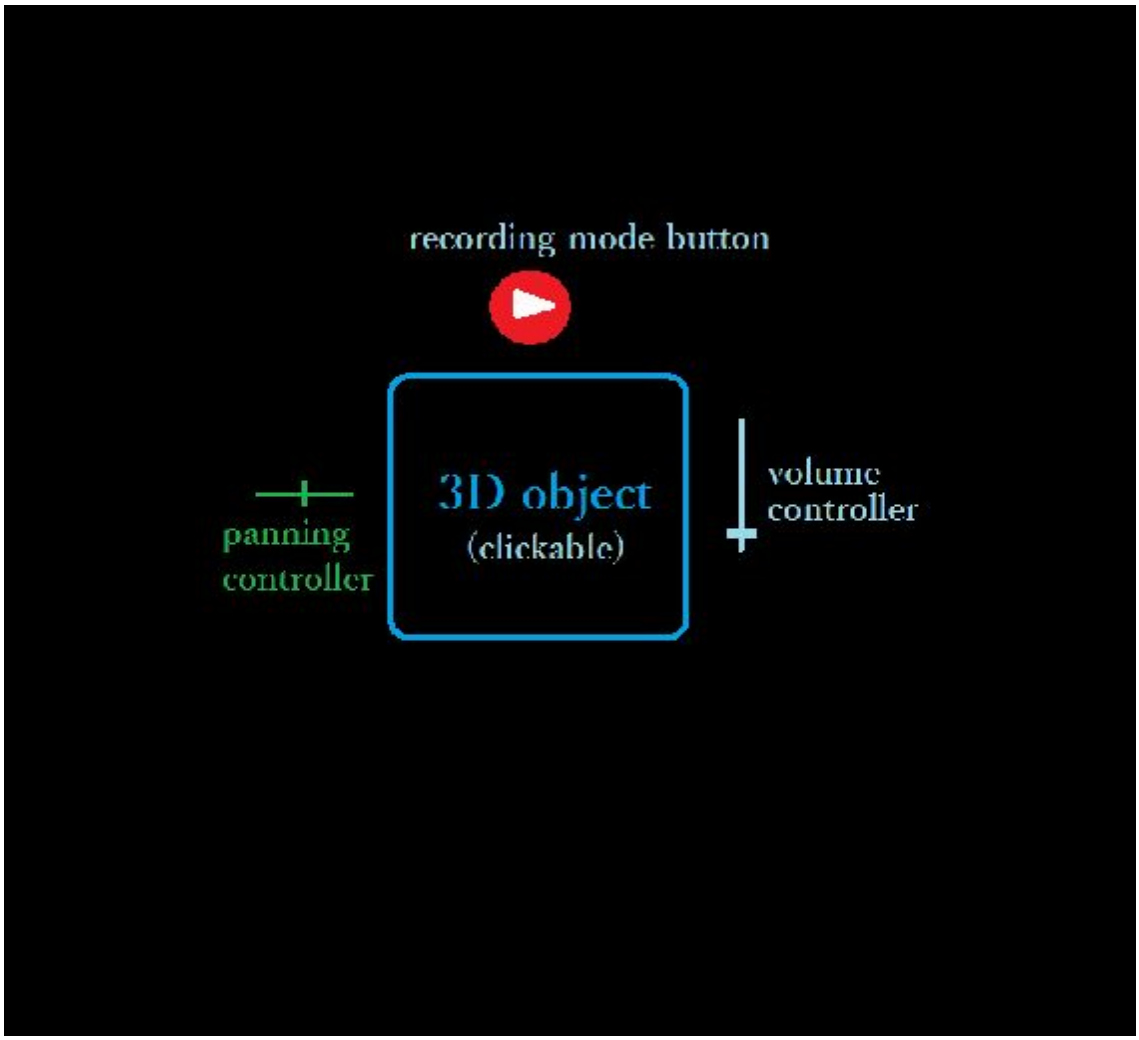


Figure 7 : Initial Visual prototype of the App's UI for an object

This is one of the first prototypes of a 3D object. At that time, our thoughts included 3 controllers for each object : panning controller, recording (rec) controller and volume controller.

Note : A storyboard was useless to our case because this is a SPA application and every action happens in the same domain.

Second phase : Evaluation and exploitation of the feedback

After having developed the above prototypes, the evaluation of them by us, 1 outside developer and 1 user helped us to achieve the betterment of the UI prototypes. Below

we will present the criteria upon which the evaluation and improvement was done, as well as some excerpts of the user, developer and our reports on the matter.

Visibility of System Status

Goal : The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

Developer : The UI prototype is not clear enough on which state is the application every moment. The user will need enough time to get used to the application to be able to feel comfortable with its every moment state. Though, the feedback of every action of the user seems to be direct and straight, thus understandable by the user.

Suggestion : Create a label that shows at every moment in what state is the application (recording, initial etc.)

User : The UI is organized well in space and its elements are well defined. I

Suggestion : I would prefer that the buttons and the various controllers to be represented by smart icons rather than just colored shapes.

Exploit the feedback : The evaluation of this first prototypes by the developer and the user led us to represent the controllers and the buttons with icons. The icons will have thematic relationship with the function of the controller they are assigned to. We considered but did not agreed to include a "state label" in our app because it would prevent the user of having an immersive experience.

Match between system and the real world

Goal : The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

Developer : The UI should use the appropriate colors to represent the analogous actions based on the standard rules for coloring UI elements when creating UIs for humans. Also the shapes of the buttons, slots and the controllers should be analogously appropriate.

Suggestion : Make the slots with double circles, the one inside the other and when the one is active it should be filled with an appropriate color.

User: The UI is organized well in space and its elements are well defined.

Suggestion : I would prefer that the buttons and the various controllers to be represented by smart icons rather than just colored shapes.

Exploit the feedback : The evaluation by the developer and the user of these first prototypes, led us to represent the controllers and the buttons with icons. The icons will have thematic relationship with the function they are related to. Also, the time slots will be consisted of 2 circles, (one inside the other) so the user can identify the state of the slot more easily. Below is shown the updated prototype (fig. 5).

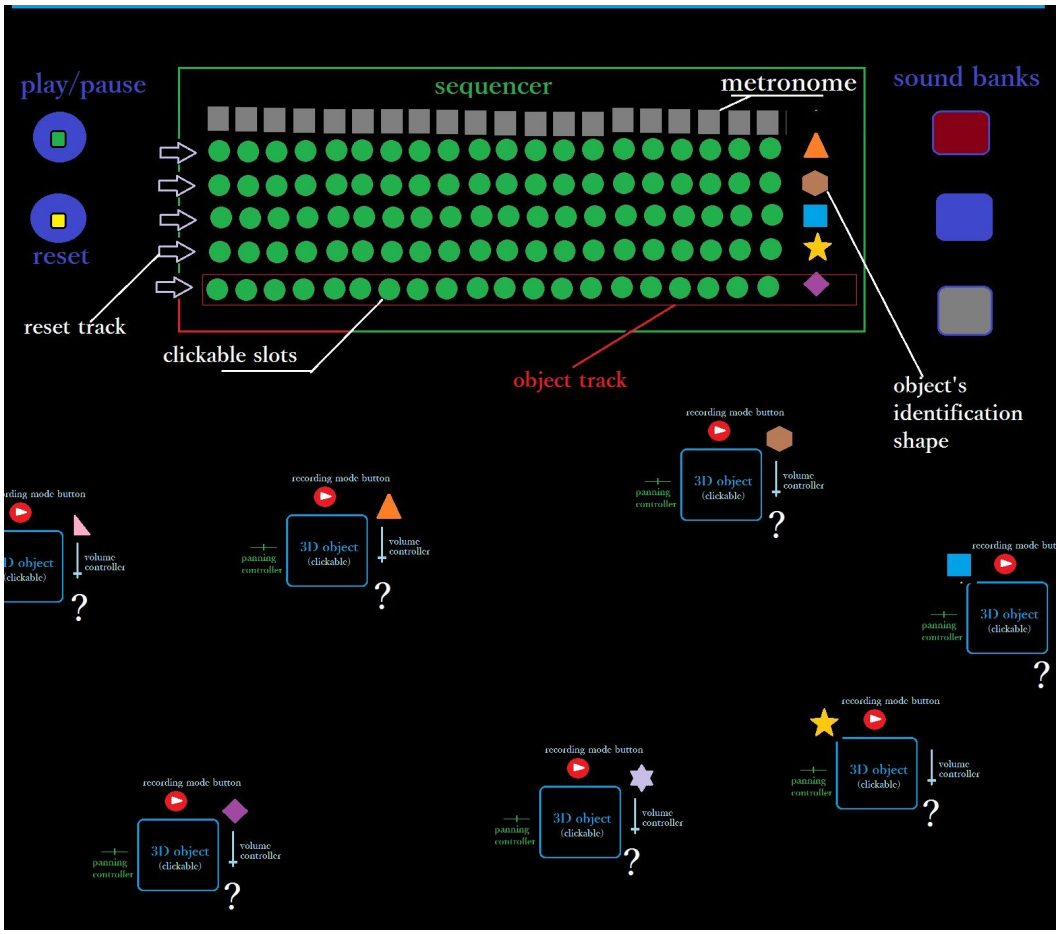


Figure 8 : Later Visual prototype of the App's UI

Figure 9 : Final result after development

4 System Architecture

4.1 Introduction

In this chapter we will analyze the architecture of our system, both theoretically and practically, as well as the dangers, opportunities and results of this exploration of creating a new or apply a known architectural model to an application such as this.

There are many different ways to build a web application, including a lot of different tool choices, and a lot of new theory to learn. A complex, modern web application needs much research before to decide what tools to use and it is easy to become frustrated by the sheer number of choices available. Even though there are many frameworks that can take care the architecture for you (e.g. Ember.js, React.js), due to lack of a database in our system and having in mind that we are dealing with a SPA app we decided to try to emulate the classic MVC pattern in the client side. The

4.2 Client side MVC

Model–view–controller (MVC) is a software architectural pattern mostly (but not exclusively) for implementing user interfaces. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user.

Although originally developed for desktop computing, model–view–controller has been widely adopted as an architecture for World Wide Web applications in major programming languages. Several commercial and noncommercial web application frameworks have been created that enforce the pattern. These frameworks vary in their interpretations, mainly in the way that the MVC responsibilities are divided between the client and server.

Description of the MVC pattern

As with other software patterns, MVC expresses the "core of the solution" to a problem while allowing it to be adapted for each system. Particular MVC architectures can vary significantly from the traditional description here.

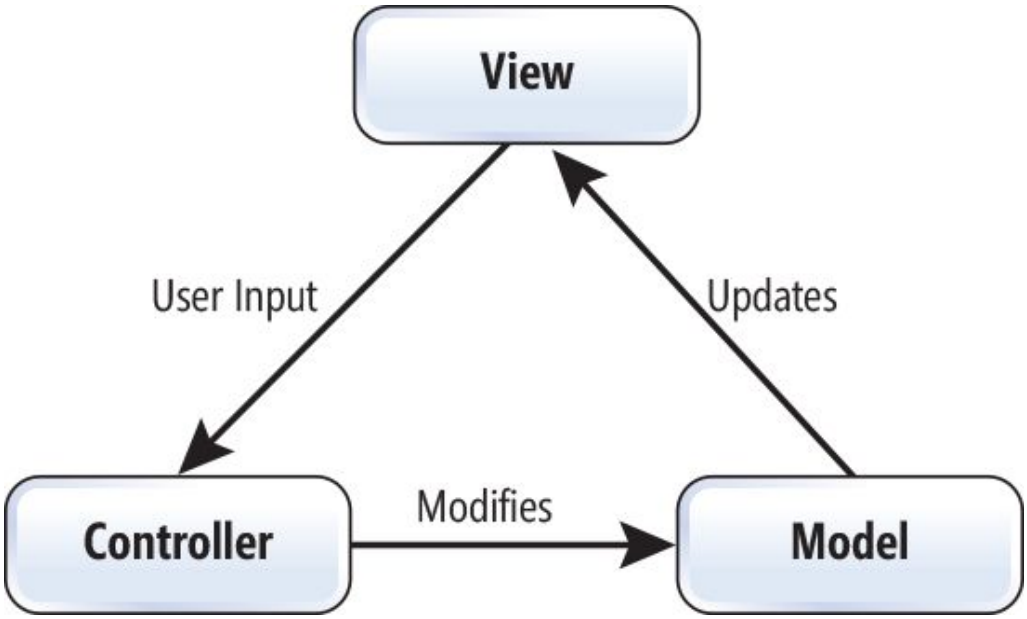


Figure 10 : Module Interaction in MVC pattern

Advantages of MVC pattern

Multiple simultaneous views of the same model.

The change propagation mechanism insures that all views simultaneously reflect the current state of the model.

Changes affecting just the user interface of the application become easier to make.

With MVC it can be easier to test the core of the application, as encapsulated by the model.

Disadvantages of MVC pattern

The view and the controller are closely coupled which makes modification to one affect the other.

Changes to the model interface will necessitate changes to the controller and the view.

When the model is active frequent changes to model can result in excessive updates of the corresponding views.

Components

The central component of MVC, the model, captures the behavior of the application in terms of its problem domain, independent of the user interface. The model directly manages the data, logic and rules of the application.A view can be any output representation of information, such as a chart or a diagram; multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants. The third part, the controller, accepts input and converts it to commands for the model or view.

A **controller** can send commands to the model to update the model's state (e.g., editing a document). It can also send commands to its associated view to change the view's presentation of the model (e.g., by scrolling through a document).

A **model** stores data that is retrieved according to commands from the controller and displayed in the view.

A **view** generates an output presentation to the user based on changes in the model.

MVC in Looparound

In our case, MVC pattern was a good choice to make due to the application's nature. Although, in Looparound we do not have any server side processes and tasks to be executed. Having all logic, UI elements and controlling in the client, implementing the pattern became not an obligatory task but a figurative one. So Looparound does not implement a genuine MVC pattern but rather an MVC-like one. Thus, MVC organization helped us configure the aspects of the functionality of the application and organize them as good as it was possible.

Model

The model holds the data and the process the application updates them so it is a very definitive component of our app. In looparound, the data is represented by all the information the user interacts with to create his loops or to just interact with the application.

These are :

Audio samples

Scene data

User settings

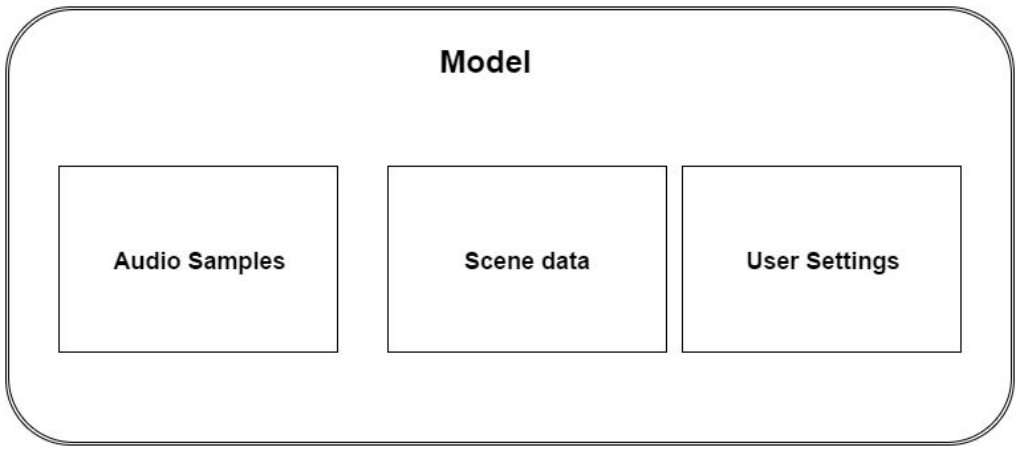


Figure 11 : Model in Looparound

View

The view is responsible for both the practical and aesthetic presentation of the application's data to the user. In our case, the view constructs the :

Scene (interactive 3D objects etc.)

User Menu

User sequencer

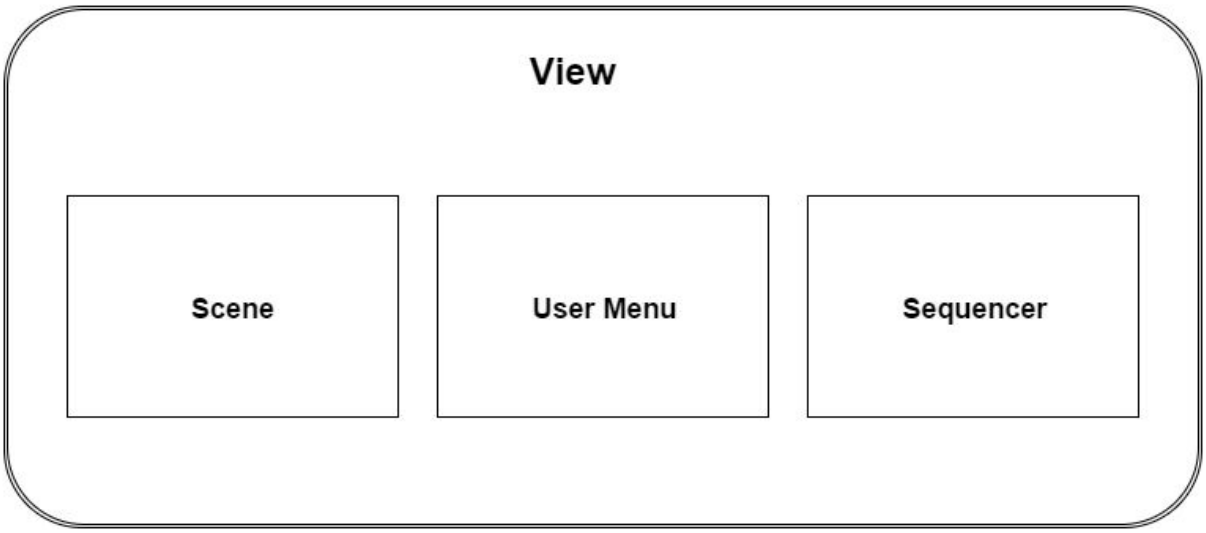


Figure 12 : View in Looparound

○

Controller

The controller is the glue between the view and the model. In our case, our controller receives the user events via the view (3D objects, user menu etc) updates the model (information about the loops etc) and vice versa. In our case the controller is all the event handlers that are bound with the 3D object interaction events and the user menu events .

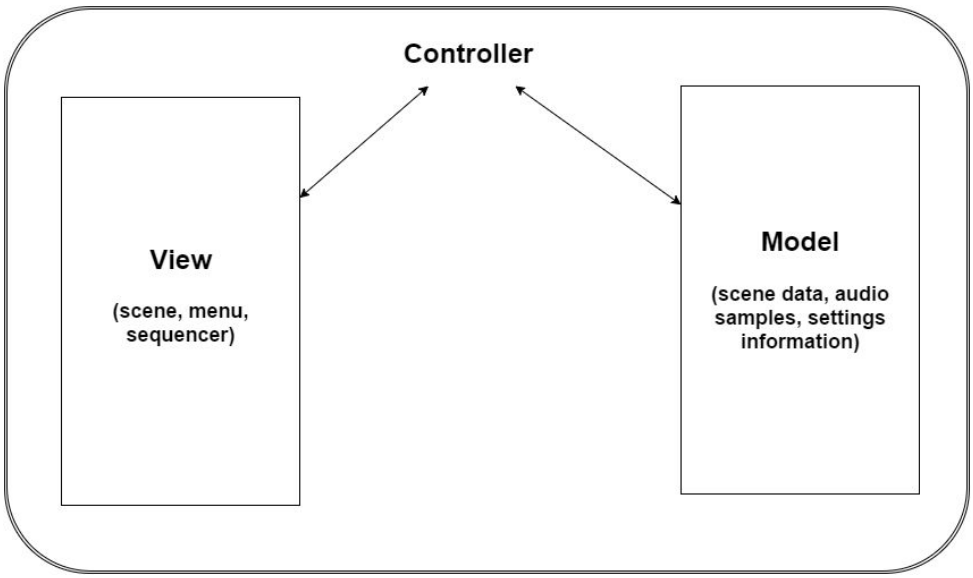


Figure 13 : Controller in looparound

4.3 Layers

Our application's architecture is based on an MVC architecture, as has been previously stated. In this chapter we will analyze furthermore the approach we took on the architectural aspect of the application. We will divide the whole app into 3 layers : the boundary layer, the control layer and the entity layer. Though, we will differentiate from the classic MVC pattern here and we will add a live-processing layer which will be explained in depth.

Boundary Layer

The Boundary Layer is responsible for the interaction of the application and the user. User actions through this Layer is bound to an Event Pattern. Events are sent to the Application Controller for further processing. The Boundary Layer is also responsible for updating the user interface after changes.

In our case, the boundary Layer is more complex than a classic UI implementation using HTML, CSS and Javascript/jQuery. Our interface is a combination of classic UI elements (HTML/CSS/Javascript) and the HTML5 canvas element on which the 3D scene is drawn and through which the user is able to interact with the app in various ways.

Below is presented the diagram that shows what is included in the boundary layer :

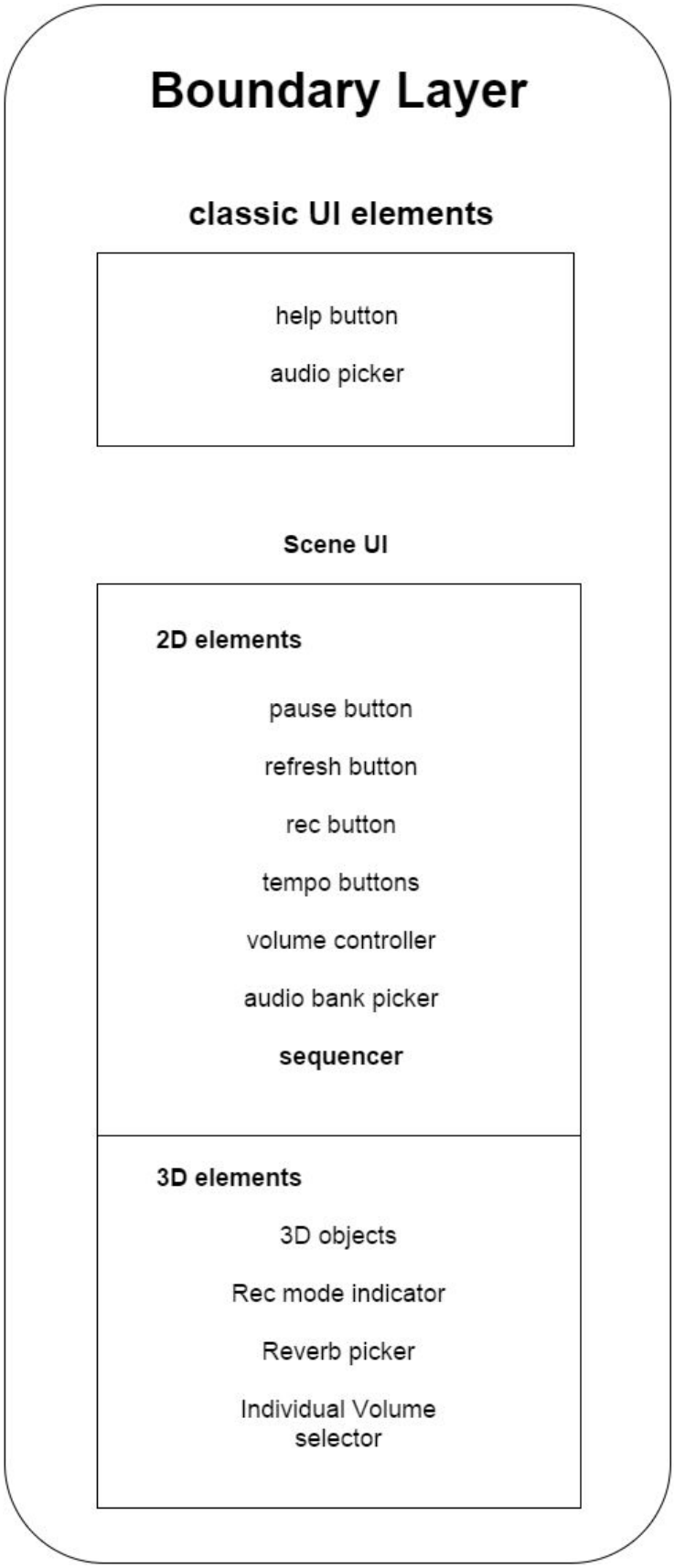


Figure 14 : Boundary Layer

Control Layer

The Control Layer is usually responsible for the flow of control, the authorization functionality of the application and basically giving access to the core of the application by receiving requests and execute everything that is need to serve them. Also, almost always the controller is on the server side of the application.

In our case, the control layer is on the client side. It communicates with the boundary layer, the entity layer and the live-processing layer. Its job is to receive the user inputs and do what is appropriate to serve them. It is basically composed of handlers that are responsible to process user changes.

Below is presented the control entity layer :

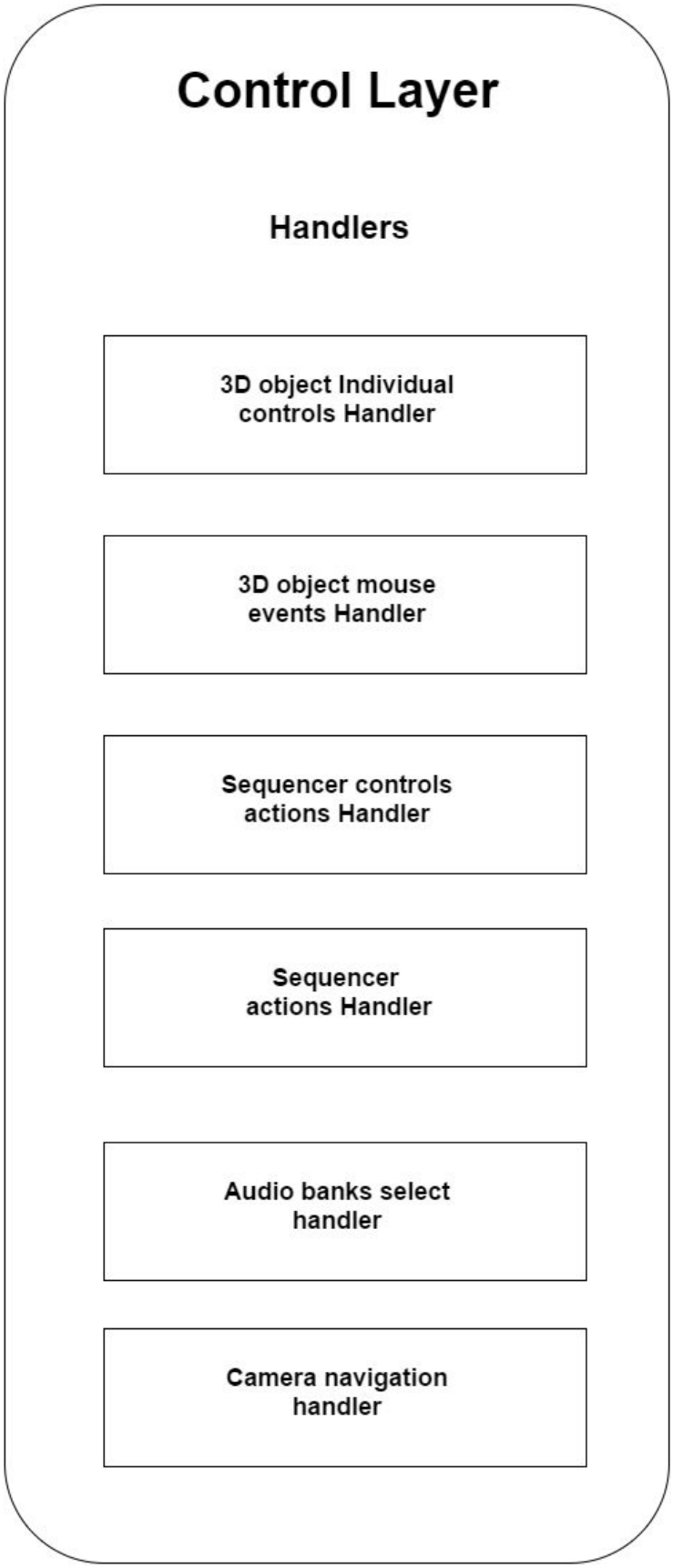


Figure 15 : Control Layer

Entity Layer

The Entity Layer contains the objects that have data that is needed for the application to function. In general, the entity layer lies in the server and there are many specific structures that support this functionality which are well designed.

In our case, the entity layer is on the client side has not so much importance in our implementation so we do not use a specific architecture, we just store any data we need in simple javascript structures.

Live Processing Layer

This layer is a custom layer that we implemented to support a live processing audio mechanism. In this application, there is an audio manipulation mechanism that is running constantly and handles the recurrent audio playbacks based on the user's choices.

We are going to examine the structure of this feature in depth in the implementation chapter which but we will have a quick look also here. It was inspired by Chris Wilson's article [5] on how to schedule audio playbacks using the new Web Audio API for the browsers which we have already analyzed.

Our live processing layer is connected to the control layer which “sees” the sequencer. The sequencer has 16 slots which have a time distance depended on the tempo set by the user. Each object can “enter” the sequencer by clicking on its rec icon and then 16 new slots are assigned to it from which the user can select which ones will play the sound of the specific object at which time. Basically, through the view the user updates the model and through the controller the live processing layer gets its information on the sequencer current state, being able to process the data needed to play the audio loops with great time precision. Further analysis will be made in the implementation chapter.

Combining all layers

Below is a diagram that shows how are all layers intercommunicate :

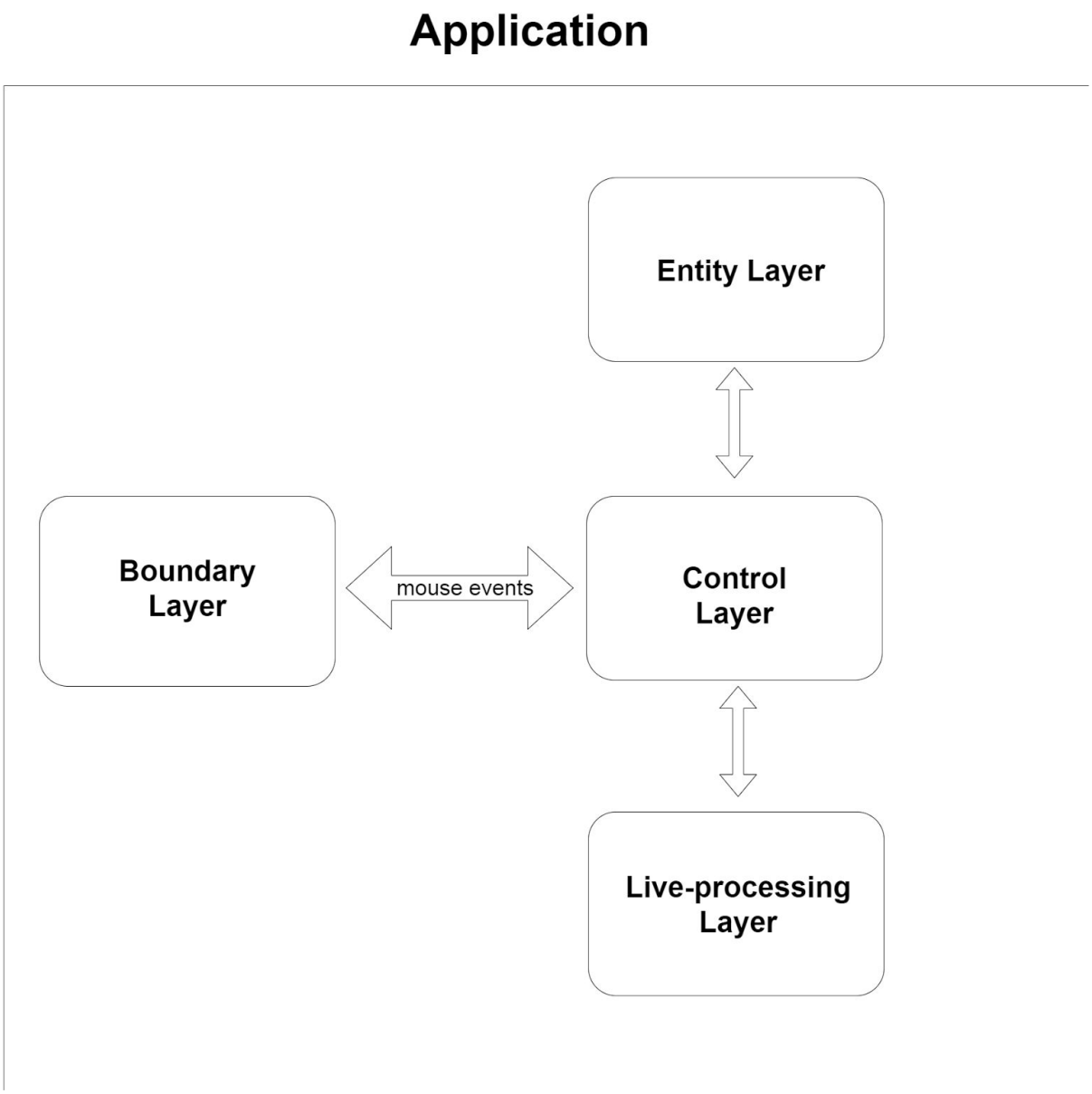


Figure 16 : Layer communication

5 Implementation

5.1 Introduction

In the previous chapter we analyzed the requirements and aims of our app, as well as the technologies we chose to fulfill them. In this chapter we will describe how we used these technologies to create some of the most vital parts of the app. We will also provide some pieces of code (mostly from our app but not only) to illuminate specific aspects of the implementation.

5.2 Audio manipulation

Chris Wilson, a known software engineer who works on the development of the web audio API has said : “One of the biggest challenges in building great audio and music software using the web platform is managing time” [5]. He also believes that the audio clock in the web is widely misunderstood. Web audio API has come to help us on this.

The sequencer

A sequencer is basically a tool with which you can record, play back and edit sound clips with finite duration. A sequencer usually has an option for a tempo. Using a tempo enables the user play/record sound clips in quantized time. That mean that the tempo value is used as a step with which we divide time so we can synchronize our sound clips with each other. Our application uses tempo always by default, with no deactivation enabled for it. The user is able to set its value through the UI though.

Figure 17 : screenshot of a commercial recording sequencer

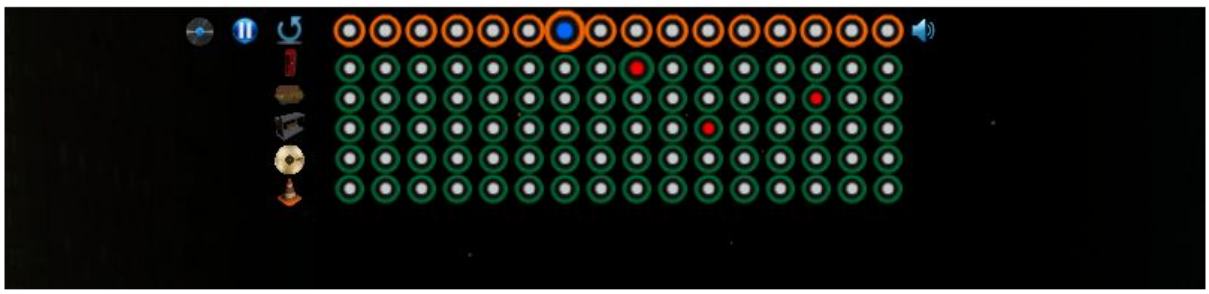


Figure 18 : screenshot of looparound's sequencer

In the above picture (figure 16) we can see a real, commercial sequencer in action. It is called Fruity Loops and as we can see it has visualizations of the time steps there are in a loop. A loop is called something that repeats itself until it is stopped. In the case of a sequencer, what we repeat is the playbacks of the recorded sound clips but given a specific tempo thus, time step (after some processing of its value) until the next quantum of time. In the next picture (figure 17) we can see our implementation of the ui of looparound's sequencer.

Rhythm plays also a role. In looparound we have hardcoded a 4/4 rhythm to simplify the use of the sequencer. That means that we have slots beats in a time step. A time step on the other hand, is calculated based on the tempo. So we can divide the number of seconds a minute has with our tempo (60/tempo) and we will find our time step. Then, we can divide the time step with our 4 slots so we can actually end up having the time distance between 2 slots in seconds, which is the fundamental time quantum our application uses to play/record/edit the sound clips. The picture below will help us realize more the timing divisions we just described.

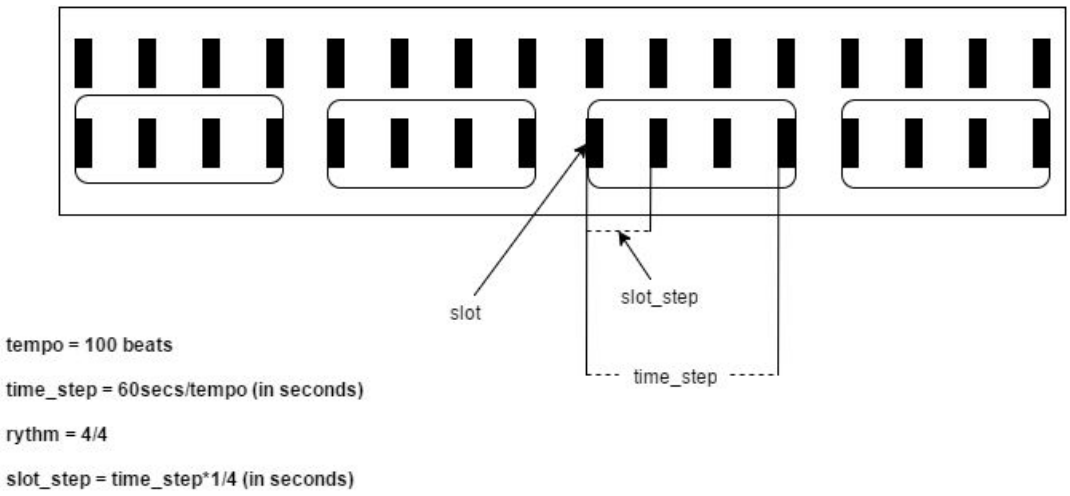


Figure 19 : time divisions in our sequencer

Every 3D object that is in Rec mode, the system appears and assigns to it 16 time slots (as in the above picture) that are clickable and have an On/Off state. If a slot of a 3D objects is On, that means the system loop will play that sounds when it reaches that time slot. The picture below will give the reader a better idea of the concept.

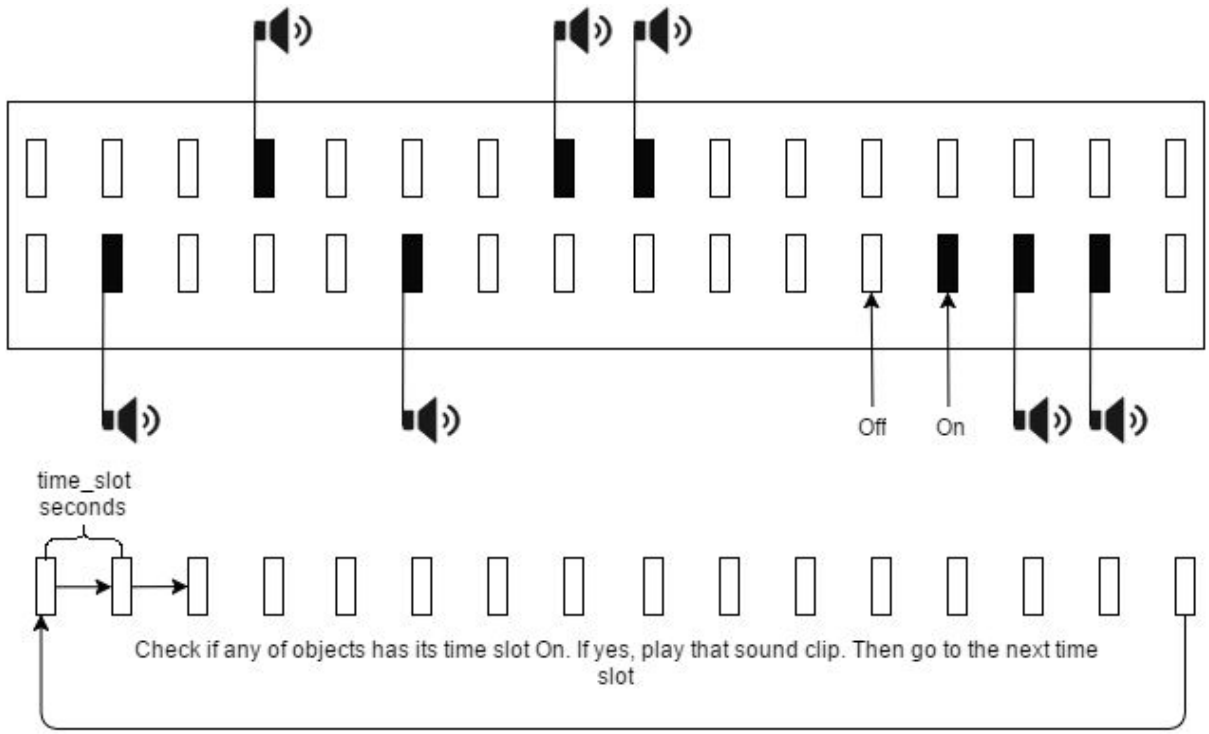


Figure 20 : how playbacks work in the sequencer

The above analysis documents both the position one is in when he has to deliver a software sequencer and our initial approach of implementing it. Although, even if in theory the sequencer seems simple, developing it in the web bears a huge problem (if someone aims to deliver a great one) : **time precision**.

I will now quote some thoughts Chris Wilson has expressed on the matter on his blog [5] :

We can have the much-beloved and much-maligned JavaScript clock, represented by Date.now() and setTimeout(). The good side of the JavaScript clock is that it has a couple of very useful call-me-back-later window.setTimeout() and window.setInterval() methods, which let us have the system call our code back at specific times. The bad side of the JavaScript clock is that it is not very precise. For starters, Date.now() returns a value in milliseconds - an integer number of milliseconds - so the best precision you could hope for is one millisecond. This isn't incredibly bad in some musical contexts - if your note started a millisecond early or late, you might not even notice - but even at a relatively low audio hardware rate of 44.1kHz, it's about 44.1 times too slow to use as an audio scheduling clock. Remember that dropping any samples at all can cause audio glitching - so if we're chaining samples together, we may need them to be precisely sequential. The up-and-coming High Resolution Time Specification actually does give us a much better precision current time through window.performance.now(); it's even implemented (albeit prefixed) in many current browsers. That can help in some situations, although it's not really relevant to the worst part of the JavaScript timing APIs.

The worst part of the JavaScript timing APIs are that although Date.now()'s millisecond precision doesn't sound too bad to live with, the actual callback of timer events in JavaScript (through window.setTimeout() or window.setInterval) can easily be skewed by tens of milliseconds or more by layout, rendering, garbage collection, and XMLHTTPRequest and other callbacks - in short, by any number of things happening on the main execution thread. Remember how I mentioned "audio events" that we could schedule using the Web Audio API? Well, those are all getting processed on a separate thread - so even if the main thread is temporarily stalled doing a complex layout or other long task, the audio will still happen at exactly the times they were told to happen - in fact, even if you're stopped at a breakpoint in the debugger, the audio thread will continue to play scheduled events!

Since the main thread can easily get stalled for multiple milliseconds at a time, it is a bad idea to use JavaScript's setTimeout to directly start playing audio events, because at best your notes will fire within a millisecond or so of when they really should, and at worst they will be delayed for even longer. Worst of all, for what should be rhythmic sequences, they won't fire at precise intervals as the timing will be sensitive to other things happening on the main JavaScript thread.

So what can we do? Well, the best way to handle timing is to set up a collaboration between JavaScript timers (setTimeout(), setInterval() or requestAnimationFrame() - more on that later) and the audio hardware scheduling.

Let us rephrase our problem, so we can examine how we solved it in the next paragraph : we need extreme time precision to make a sequencer work correctly, and classic javascript events can not help us due to their dependable in the main thread nature. Below, we will present our solution which is based on Chris Wilson article [5].

As mr. Wilson mentioned, the key is to look ahead time and schedule our playback events. Web Audio API gives us this opportunity by giving a number of milliseconds and an audio buffer that holds the audio sample. Of course that sounds much easy and one would wonder why don't we schedule all of our playback events from the start and why do we have to consider all these complicate things we described above. The answer is **flexibility and dynamicity.** We can not have an entirely pre-scheduled system where we are not able to change its parameters (especially the **tempo** parameter)! The best practise is to schedule the closest playback events so if some parameters change the next scheduled playbacks can be scheduled using the new parameters resulting the system to behave smartly, quickly and elegantly. An important question is how much do we look ahead and schedule playback events? Here is a quote from Chris Wilson on the matter :

The main point is that we want the amount of "scheduling ahead" that we're doing to be large enough to avoid any delays, but not so large as to be create noticeable delay when tweaking the tempo control. Even the case above has a very small overlap, so it won't be very resilient on a slow machine with a complex web application. A good place to start is probably 100ms of "lookahead" time, with intervals set to 25ms.

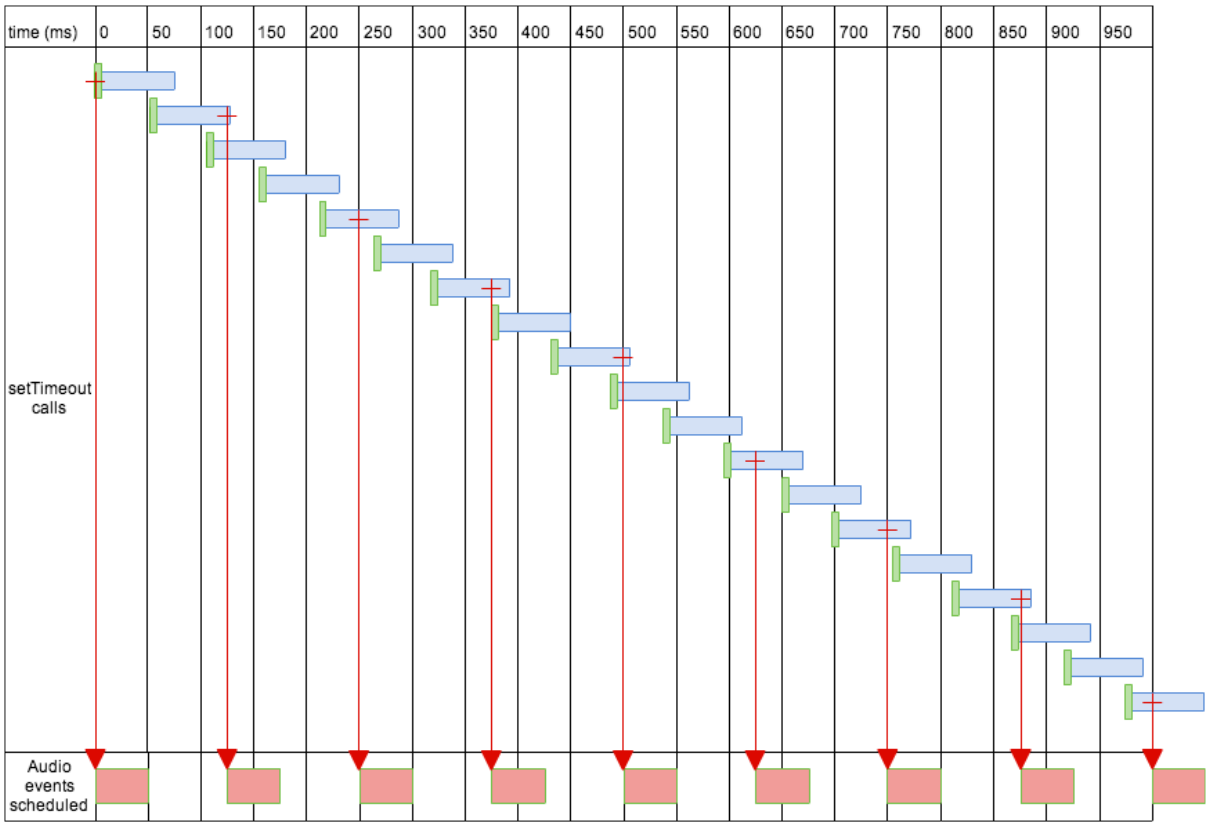


Figure 21 : scheduling playbacks with interval and lookahead times (1)

In the above image we can see an example of values for interval and lookahead times. The image shows that the scheduling function will run every 50 ms (appr. due to main thread drawback) and will look ahead to 75 ms and see if there are any playback events that must be scheduled and schedule them. We need the timing events to overlap because we have to predict a possible delay in calling the scheduling function. The problem is that in slower machines the main thread is busier than usually, which means that we have to make this overlap much bigger to avoid jittering even missing a playback.

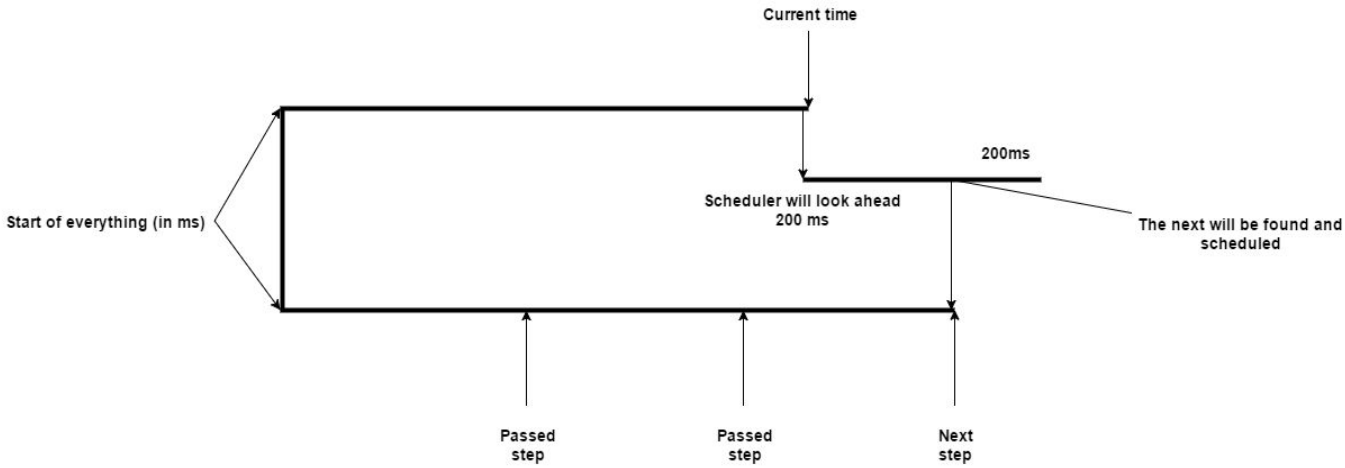
In our case in Looparound, we chose to have an interval of 0 ms so the scheduling function will run as often as possible by the main thread, and a value of 200 ms of look-ahead time. We did this to predict as good as possible the audio playbacks even in slower machines or when our main thread is really busy. Although, even with these slightly extreme values for look ahead and interval, in some case jittering and missing an event can not be avoided (e.g. when we giving focus to another tab in our browser). Let us show the piece of code that implements all of the above :

```
var tempo = 100 ;
var step = 0 ;
```

```
var startTime ;
var timeoutId ;
var timeNext ;
var contextPlayTime_ ;
var JamtimeStamp = 0 ;

function startScheduler(){

   noteTime = 0.0;
   startTime = window.context.currentTime ;
   ultimateScheduler();
}

function ultimateScheduler() {

  var currentTime = window.context.currentTime;
  currentTime -= startTime;

  while (noteTime < currentTime + 0.200) {

     var contextPlayTime = noteTime + startTime;

     for (var i = 0; i <= objectInfo.length; i++) {

       play(currentKit.returnBuffer(i), contextPlayTime,i);

       objectFunctions[i].vibrate(i);

       setTimeout( scaleMeForAWhile(PlayingSlotMatrix[j][step]),180);

     }

     updateVars();

  }

  timeoutId = setTimeout("ultimateScheduler()", 0);

}
```

```
function stopScheduler(bool){

    clearTimeout(timeoutId);

}

function updateVars(val){

    step++;

    if (step == 16) {
        step = 0;
    }

    noteTime = noteTime + (60 / tempo) * 0.25;

}
```

Let us give a diagram that will clear the above code out a bit :



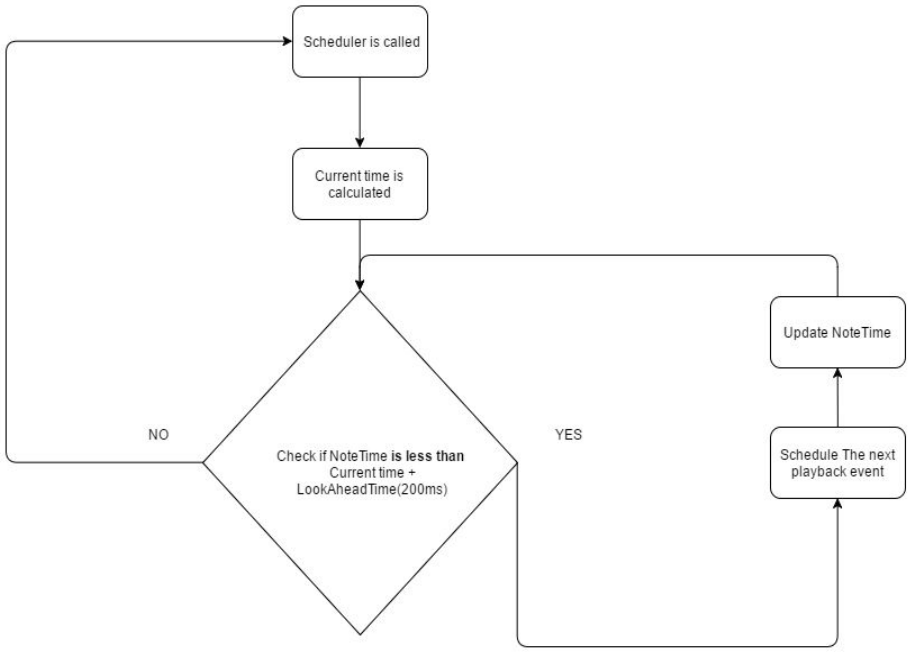


Figure 22 : scheduling playbacks with interval and lookahead times (2)

Using this timing mechanism, we are able to have the best precision in time for our sequencer steps which are calculated based on our tempo. By this, we can succesfully schedule our audio and animations events.

The playback Audio mechanism

A simple, typical workflow for web audio would look something like this [6]:

1. Create audio context
2. Inside the context, create sources — such as <audio>, oscillator, stream
3. Create effects nodes, such as reverb, biquad filter, panner, compressor
4. Choose final destination of audio, for example your system speakers
5. Connect the sources up to the effects, and the effects to the destination.

For example :

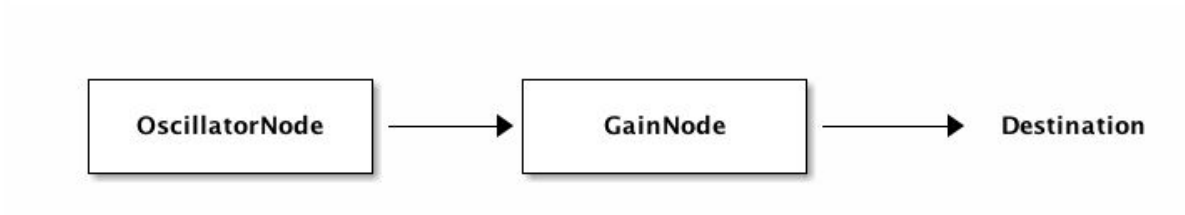


Figure 23 : simple example of routing audio nodes

In our case, we load our samples as mp3s and store them to audio buffers ready to be used. After our research we ended up with many opinions of computer scientist that support the recreation of a new source audio node and no reusing the same :

```
function play(buffer, time) {
    var source = context.createBufferSource();
    source.buffer = buffer;
    source.connect(context.destination);
```

```
    source.start(time);
}
```

Based on the specific object's data that is stored, every time an object track's slot is active which means that a playback event has to happen, the system creates an AudioBufferSourceNode, applies to it some parameters and then the sound is played. The diagram below will illuminate more the structure of this mechanism :

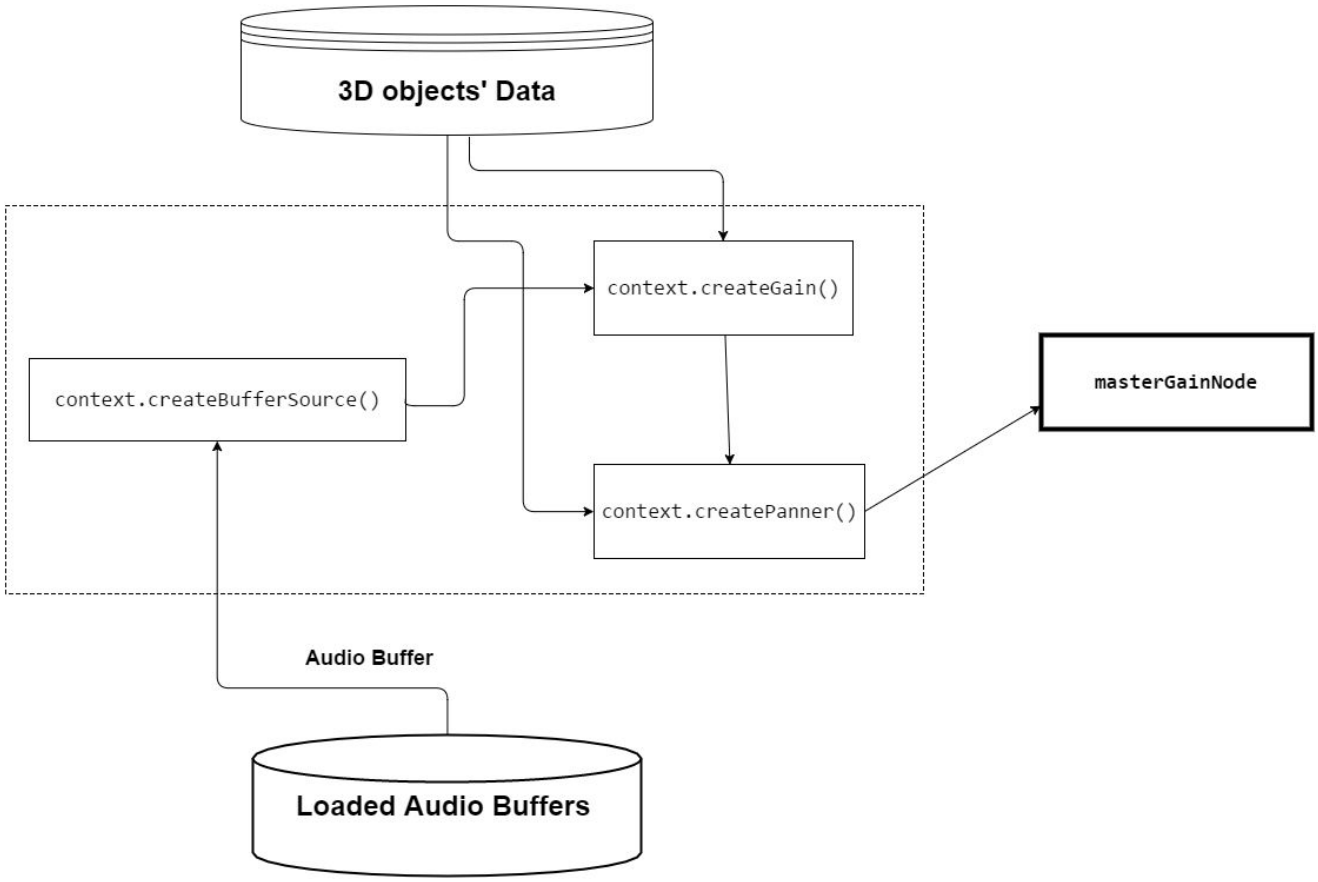


Figure 24: playing an audio samples based on specific parameters

Here is a sample of code that implements the above mechanism :

```
function play(buffer, noteTime, i) {

    var voice = context.createBufferSource();
    voice.buffer = buffer;

    var dryGainNode = context.createGain();
    dryGainNode.gain.value = objectsData[i].gain;

    var panner = context.createPanner();


    panner.setPosition(objectsData[i].panX,objectsData[i].panY, objectsData[i].panZ);
    voice.connect(panner);
    panner.connect(dryGainNode);

    dryGainNode.connect(masterGainNode);

    voice.start(noteTime);

}
```

5.3 3D Graphics Implementation

Introduction

3D graphics in the web is a fairly new addition, compared with the age of the web itself. WebGL API is based on OpenGL API, as we have stated before, and the library we have chosen to use to abstract WebGL, which is rather low level, is threeJS. In this chapter we will present how we implemented the features that are related to 3D graphics in our application using this library.

3D Objects

Our 3D objects are divided in 2 categories : **Objects with preloaded geometry** and **Objects whose geometry is constructed dynamically.** Let us examine how we fetch and manipulate the preloaded objects. These objects are loaded via the THREE.OBJMTLLoader giving the urls of the files. Then, we use the geometry and the material to construct a THREE.Mesh which we add to the scene. Below there is a significant case of loading a geometry/material of a brick which we replicate many times using the same geometry and material to better the performance :

```
var BrickLoader = new THREE.OBJMTLLoader();
var firstTime = true;

BrickLoader.addEventListener('load', function (event) {

 obj = event.content;

 obj.traverse( function ( child ) {

  if ( child.geometry !== undefined ) {

   for (var i = 0; i <104; i++) {

     if(firstTime){
```

```
        materialArray.push(child.material); // 17
        firstTime = false;
      }

      // main road bricks

      if( i <= 55 || i >= 62)
      {
       var BrickMeshR = new THREE.Mesh( child.geometry, child.material );
       BrickMeshR.receiveShadow = true;
       BrickMeshR.position.set(355, 0, (52-i)*100.5);
       THREE.GeometryUtils.merge(mergedGeometry, BrickMeshR,15);
      }

      // Side Road bricks

      if( i < 49) {
       var BrickMeshLS = new THREE.Mesh( child.geometry, child.material );
       BrickMeshLS.receiveShadow = true;

       BrickMeshLS.position.set((52-i)*100.5 - 27,0,-305);
       BrickMeshLS.rotation.y = Math.PI / 2;
       BrickMeshLS.scale.set(1,1,1);

       THREE.GeometryUtils.merge(mergedGeometry, BrickMeshLS,15);

       var BrickMeshRS= new THREE.Mesh( child.geometry, child.material );
       BrickMeshRS.position.set((52-i)*100.5 - 27,0,-884);
       BrickMeshRS.rotation.y = Math.PI / 2;
       BrickMeshRS.scale.set(1,1,1);

       THREE.GeometryUtils.merge(mergedGeometry, BrickMeshRS,15);
      }

    }

   }

} );
```

```
});

BrickLoader.load("3D/SidewalkBrick/SidewalkBrick.obj",
"3D/SidewalkBrick/SidewalkBrick.mtl" );
```

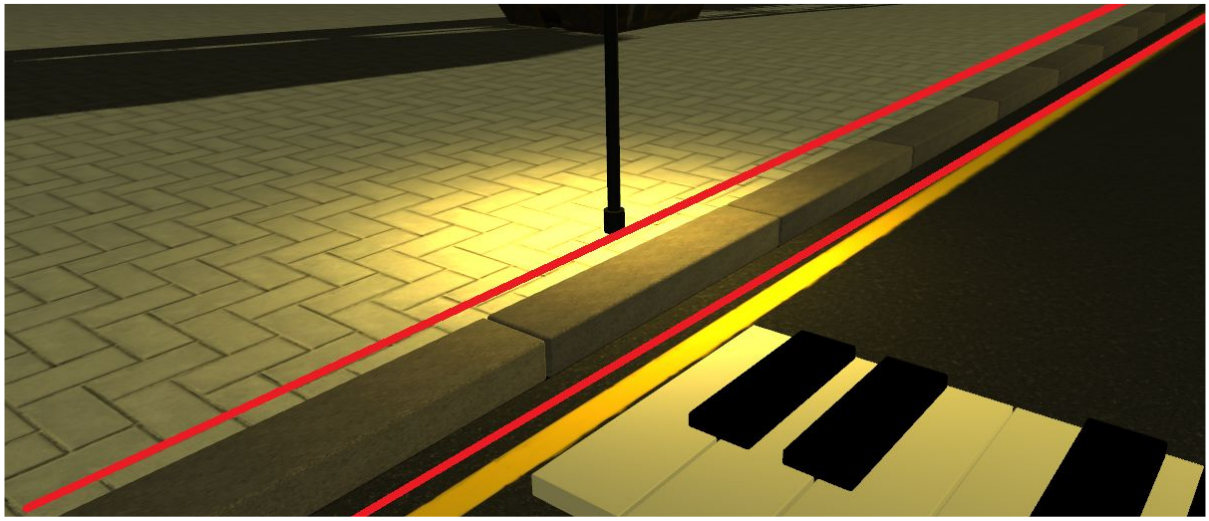


Figure 25: static 3D objects (bricks)

The above bricks are static objects which means that we do not alter them or assign to them any events during the lifecycle of the app. The important part is the 3D objects that are dynamic and interactive which are loaded with the same process. Though, we have to keep track of them because we have to animate them or assign specific mouse events or other to them.

After all dynamic objects have loaded and the meshes have been created and added to the scene, we create their individual 3D UI that has the following features : Rec Button, Volume Slider, Reverb Room. Also, they are identified by a unique id with which we manipulate the mouse events for animations and affecting sequencer slots.

In the below diagram we will indicate how the model (the stored information about how the view will be constructed) interact with the controller to create the view and assign to it events :

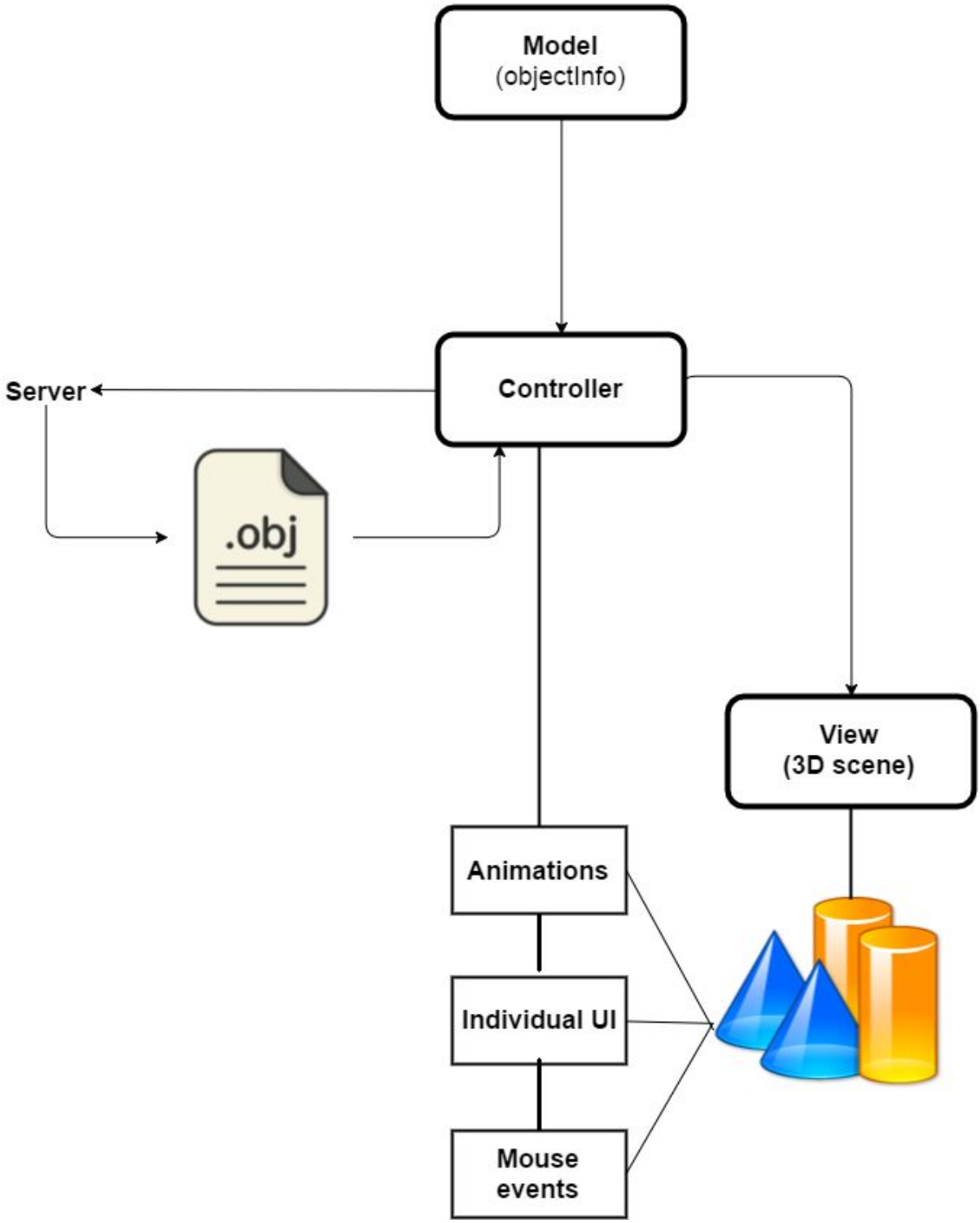


Figure 26 : 3D event creation and event binding

Animations

As we stated above, the controller assigns to the dynamic 3D objects animations. There are three kinds of animations that can be triggered : object movement, scaling sound -sphere and smoke simulations.

Object Movement

Every time an object is clicked or the sequencer triggers its activation an animation is also triggered for this object which is usually a simple positioning parameter animation. That means that its THREE.mesh.position or THREE.mesh.rotation variables are changed through time for a specific time amount and then they return to their default state.

Figure 27 : Example of object movement animation

Scaling Sound Sphere

Every time an object is clicked, a 3D sphere with a random color is created using as origin the position of the object. Then it scales and becomes transparent through time until it is big enough and transparent enough to disappear. This is a visual aid for the user to understand better which and when each object produces a sound.

Figure 28 : Example of sound sphere animation

Smoke Simulation

In some specific objects, we have assigned not only the above animations but also an air simulation that is more interesting and entertaining for the user. This is done using a particelEngine.js which is loaded as an independent module. Given a specific id for the object we wish to produce smoke for and the time we want the smoke to "live" the engine produces particles using a THREE.ParticleSystem which basically creates many particles with the same material and change their position through time with the above math equation for y and other equations for x,z which are defined by the current value of y :

*particle.y += deltaSmoke * this.forTime *5 * 50/(particle.y+8) ;*

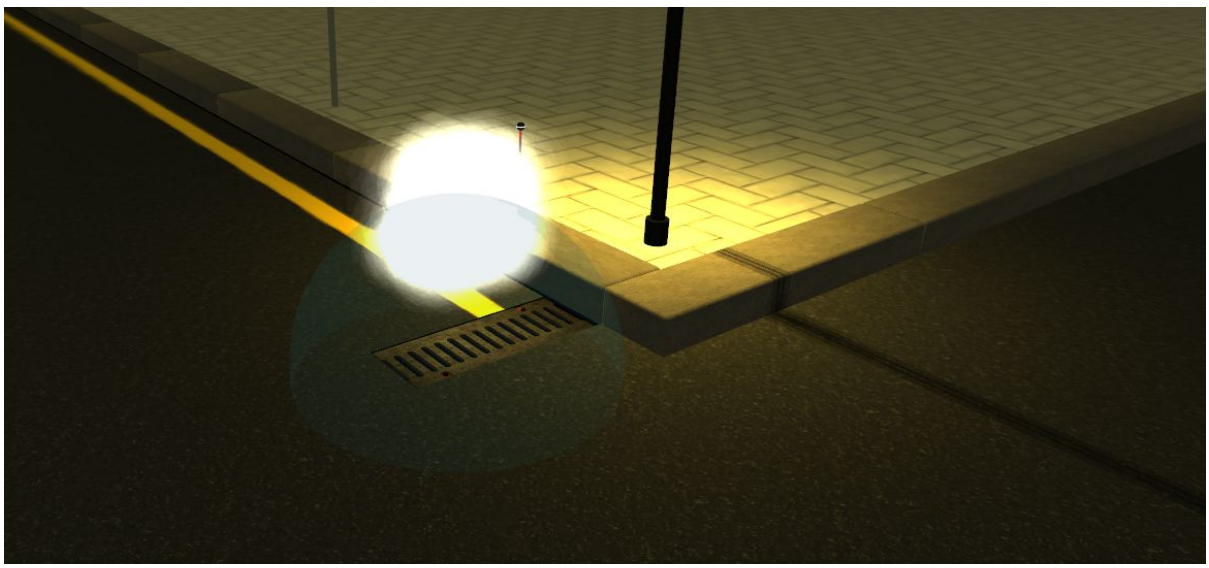


Figure 29 : Example of smoke animation

Camera Manipulation

In this paragraph we will analyse how camera is operated by the user. We have implemented a hybrid custom method which is based on classic WASD for

front,back,left,right and mouse events to rotate the camera. Usually if someone desires to implement a WASD feature in a graphic app on the browser, he has to use THREE.PointerLockControls which has a significant drawback : it hides the mouse. In our case it is very important to have the pointer of the mouse available for interacting with the app so we could not use this solution.

Our approach was to add to the scene an invisible cube (any other object would also do the job) which stay always in front of the camera. The camera is, inside every render() execution, set to camera.lookAt(cube.position.clone()) which enables us to move the cube instead of the camera which results to the camera moving accordingly. That handles successfully the from, back, right, left movements. We also use Shift key and space key for up and down directions. What happens with the camera rotations then?

Camera rotations are handled by the mouse x,y coordinates. When the user clicks on the scene and moves the mouse the controller gets the x,y coordinates and translates them into camera rotation values. Note that the camera rotation by point (and axis) is defined also by the cube. That leaves the mouse available to any other interaction inside the application.

Below the diagram shows how the camera manipulations are handled.

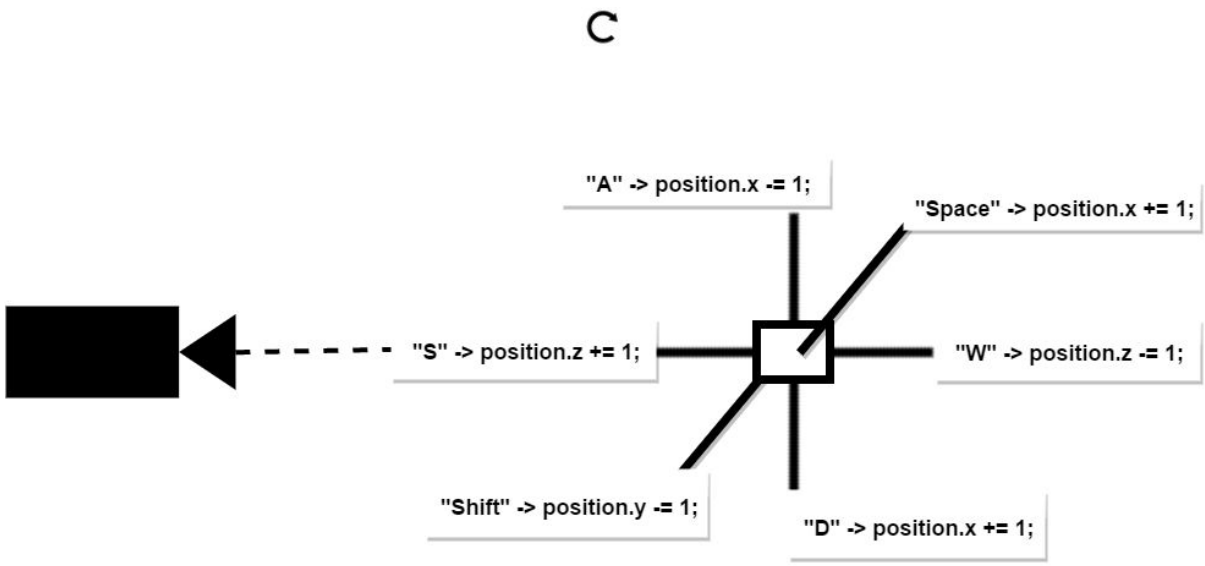


Figure 30 : camera positioning with key events

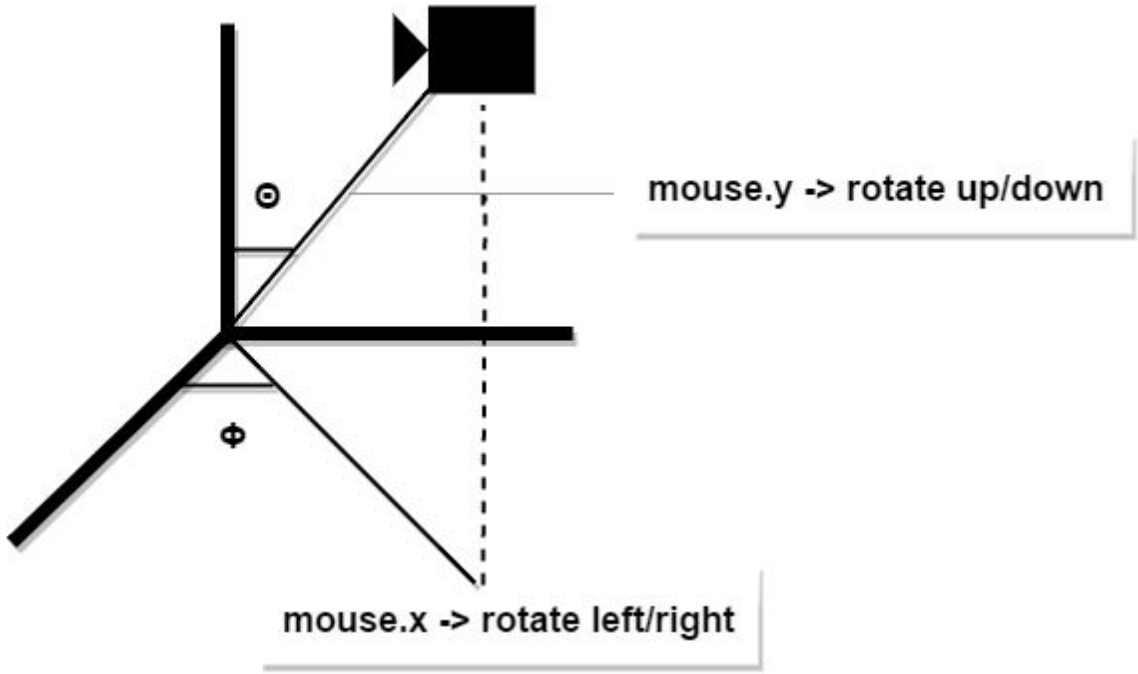


Figure 31 : Camera rotations with mouse events

3D Spatial Sound

The application's object are interactive as we have previously mentioned and that interaction includes an mp3 sample being played every time they are clicked or activated by the sequencer. Aiming to enhance the user 3D experience we implemented a 3D spatial sound effect.

As is well know, the audio our devices play is usually stereophonic which means that we have 2 audio signals blended in one file. When the device plays that file the two signals are divided and the one is sent to the left speaker and the other to the right. Having two separate audio sources enables us to create a rather immersive sound environment for the user especially when he is using headphones.

Stereophonic sound *or, more commonly, stereo, is a method of sound reproduction that creates an illusion of multi-directional audible perspective. This is usually achieved by using two or more independent audio channels through a configuration of two or more loudspeakers (or stereo headphones) in such a way as to create the impression of sound heard from various directions, as in natural hearing [7].*

In our case, we have a 3D space with various 3D objects that produce sounds. Having in mind the stereophonic sound effect we implemented a 3D spatial sound feature. We achieved that by a customly designed effect which is called panning. *Essentially, a panning control, or pan pot, adjusts the levels of a particular source that are being played through the left and right speakers, allowing the user to 'place' it within a stereo panorama [8]*.

Fortunately, web audio api provides to us a panner node which is very helpful to achieve the above effect :

The StereoPannerNode interface of the Web Audio API represents a simple stereo panner node that can be used to pan an audio stream left or right. It is an AudioNode audio-processing module that positions an incoming audio stream in a stereo image using a low-cost equal-power panning algorithm [9].

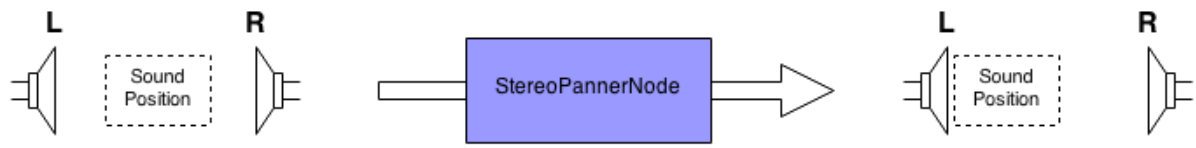


Figure 32 : Panner node

There is a pan property that takes a unitless value between -1 (full left pan) and 1 (full right pan) so we can inform the panner node where do we want to position our audio. But how do we know how much left or right do we want the audio to be heard given a 3D scene and a camera through which the user is looking at? In addition, we want to know how far away is our object from the camera position which will lead us to also modify the audio gain to simulate a near-far feeling. Below we will examine our custom solution to this problem.

First of all, our camera has always a point in space that is looking at. Technically there is no such point in the THREE.camera object but there is just a frustum. Although, the user gets the impression that he is looking at the center of our scene at any time through the camera. That means that every sound that is coming from a source positioned closely to the vector, which originates at the center of the screen and its direction is towards the back of the scene as it shown the specific moment, has to have pan parameter 0. Every other object that is far from this vector has a pan parameter that needs to be calculated manually given its position relatively to the vector. We will also have to calculate the distance from the camera position to adjust the gain parameter as we mentioned.

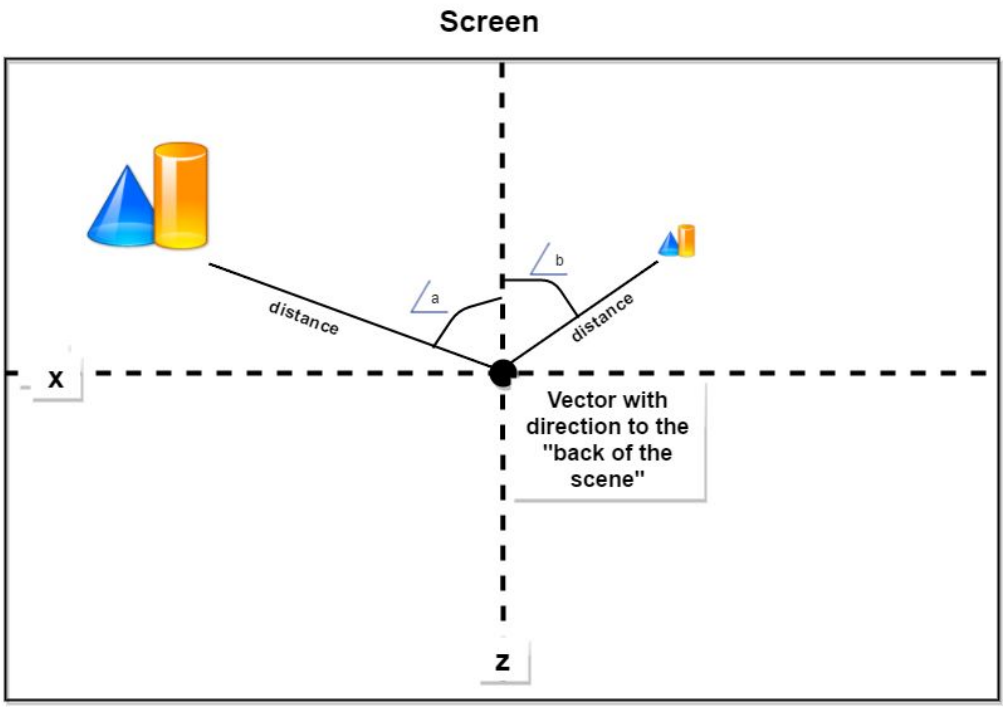


Figure 33 : Calculation of panning using angles

Figure 34 : Demonstration of panning in looparound

In the above image we demonstrate how we calculate the pan parameter using the x,z position parameters of an object and the camera to scene vector. If the angle a is high the pan parameter will get a high negative value close to -1 and vice versa. The gain parameter of the specific object will also be adjusted given the distance from the camera at the current moment. Note that after a specific limit we do not reduce the gain parameter further because we aim to always hear an object even a little.

Another significant aid that we use to enhance the spatial audio experience for the user is the reverb effect. This is handled only by the user through the individual UI controller of each object. Scrolling above the *animated room* results to increase the reverberation

effect and that is shown also visually by the change in the size and color of the 3D animated room.

Performance Issues

There are many challenges in developing an application such as this as far as performance is concerned. FPS drop is the most significant challenge and we tried to handle this in the best possible way. Here we will mention some of our approaches we took to handle this issue :

Geometry Merging : all static geometries are merged to one which extremely reduces the WebGL calls.

Shared geometry : objects that have the same geometry share all a specific instance of THREE.Geometry.

Shared Material : objects that have the same material share all a specific instance of THREE.Material.

Use least number of vertices : In some cases that the objects had a great level of detail we chose to either remove them at all or replace them with simpler geometries.

5.4 Hybrid UI

2D HUD

HUD has been popularized by the game industry. *A Head-Up Display is the method by which information is visually relayed to the player as part of a game's user interface.The HUD is frequently used to simultaneously display several pieces of information including the main character's health, items, and an indication of game progression (such as score or level) [10].* The HUD stays always visible during navigation in the game which is very helpful to the user if he wants immediate access to it.

Our HUD is positioned at the top of our screen :

Figure 35 : looparound's HUD

Our HUD is implemented using a THREE.Orthographic camera and a second THREE.Scene . HUD includes both our sequencer's slots which are created with a number of THREE.Mesh(THREE.PlaneGeometry. THREE.MeshBasicMaterial) which are clickable and for which we store an ON/OFF state and some other html elements which are also clickable and with ON/OFF state.

Both the sequencer slots and the html elements are connected to the controller which inform when a user action is taken.

Individual object's ui

Each interactive object has a private ui that is made of three 3D small object near it. This individual UI sets 3 parameters : Recording mode, Reverberation Level, Volume Slider :

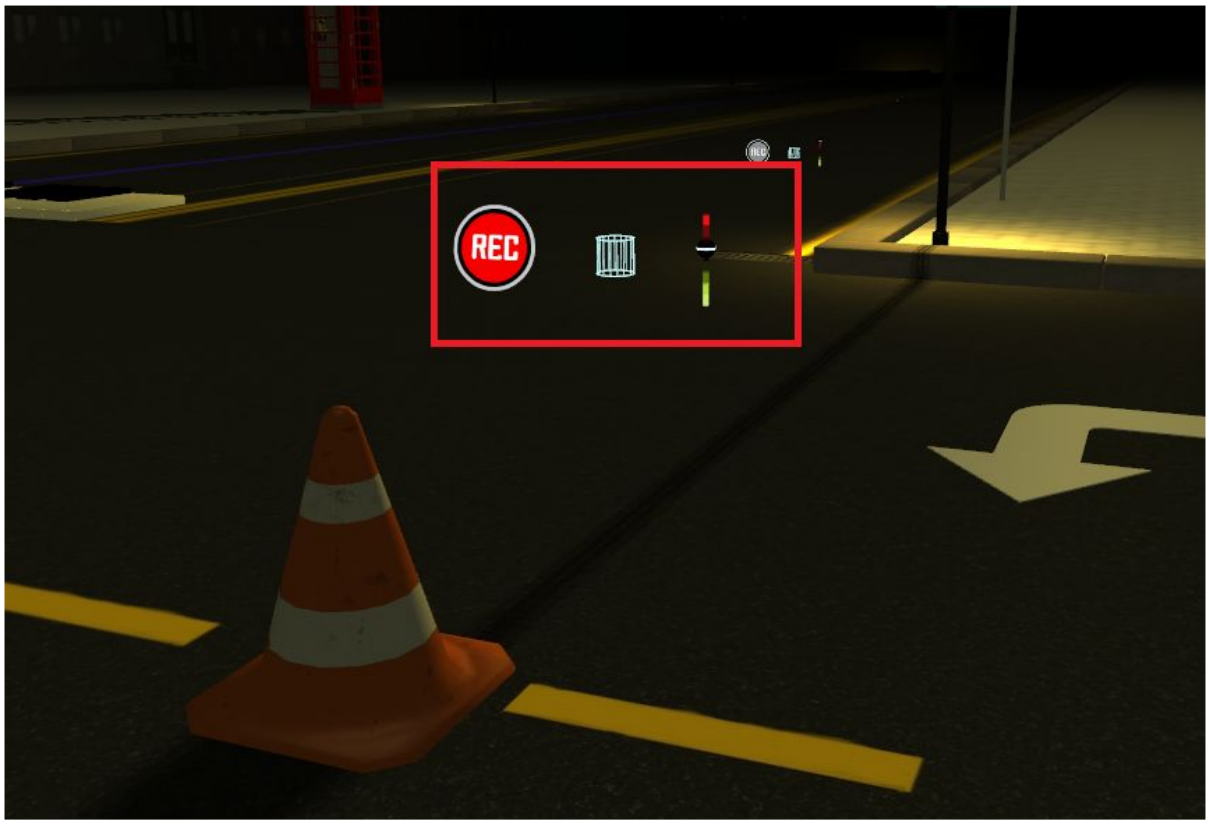


Figure 36 : Individual object's 3D UI

Rec Mode

Rec Mode is constructed by a THREE.Mesh that is clickable and has a state that is indicated by its material (On/Off). When Rec mode is clicked the object enters the sequencer and it is ready to record its sound.

Reverberation Level

Reverberation Level is constructed using a THREE.geometry which looks like a room. It is a standard way to indicate reverberation levels in commercial sequencers also. The level of reverberation is manipulated by scaling the room using the scroll of the mouse. The room also changes its color when scaled.

Volume Slider

Volume slider is constructed using a THREE.Object3d with a THREE.TubeGeometry and a THREE.SphereGeometry. The tube represents the slider and it is colored to indicate the level of volume intensity and the sphere is draggable by the user along the tube (up/down).

6 Summary and conclusions

6.1 Introduction

In this chapter we will summarize our thoughts on the development of the thesis as well as mention some of the future improvements and possible extensions that the applications could benefit from.

6.2 Aims and results

Our aim was to develop a web application using the latest tools and technologies related to web graphic and web audio as well as some of the most valuable frameworks of modern web application. Other important aim and result was to create an entertaining, education and bug free application with great user experience with which a person is able to become familiar with an interactive 3D environment and audio synthesis.

Through this process one of the most valuable lessons that we benefit from was the unexpected problems and obstacles someone can face and how he can manage to overcome them. Nothing was taken for granted and there were not easy paths to follow. We did much research beforehand, during the requirements gathering phase, but also during development.

One of the bad aspects of the resulted product is the scalability and maintenance of the application. As far as architecture is concerned, this application is not an example nor a generic model to follow. This happened because of us giving focu to the nature of the app and the features and not in the architecture itself. But after having developed this

app we can note that architecture is the most important thing of a real commercial product. Scalability and maintainability is the most vital aspect of an application that it is out there as. It is a very often phaenomenon for an application to be developed by teams of persons so the above is totally vital for this reason as well.

6.3 Future Improvements

Future improvements might include :

Virtual reality

VR is totally rising both in the web and the desktop applications. There is an abundance of Head Mounted Displays on sale in a wide range of prices nowadays in the market. Oculus Rift, HTC Vive and Samsung Gear VR are some of the best for pc applications and Google Cardboard is one of the cheapest solution for mobile. Especially with the THREE.OculusRiftEffect extension and the upcoming WebVR API [11] for the browser the virtual reality could be a very accessible feature for the web applications. For looparound a HMD could enhance the UX and by presenting him the scene in real 3D and also letting him manipulate the camera with its tracking system.

Motion Controllers

Motion sensors are another possible feature our application could integrate. There could be a feature with which the user could enable interaction via a connected motion controller such as Leap Motion [12]. The user then would be able to interact with the object with his hand movements or even manipulate the camera with his hand position to navigate the scene.

Social Media Character

Another suitable update for this application could be giving it a social media character. The user then could create a profile, log in and create their loops. After creating a loop they could publish it in the public pool where there are published loops from other

users giving them the opportunity to also comment on a specific loop, like another, download on their pc etc. This would increase their interest to the site and would add an extra motivation for its use.

Connect External Instrument

A very interesting extension would be to enable the user connect a microphone, a guitar or any other source and let him record his loops together with the instrument/mic audio.

Record on loaded song

A possible future feature would be to enable the user to load any song he likes and record his beats on it. The system would have to detect its tempo and configure all of the related parameters based on this.

Camera and video

It would be great for a user to see himself singing and record that together with the audio loops. This feature is related to the Connect External Instrument feature. Then, the user could export a video with the audio loops he created and him singing on it, a unique videoclip!

Store/restore project

A store/restore feature would be a very handy feature for the user. He could create a audio loop with specific parameters and save it to a Database and even share it with his friends and welcome them alter it to what they wish.

6.4 Conclusion

Developing this application made me realize so many important things for each phase of the process. Planning, searching, researching, documentation and implementation were all aspects of the same thing : providing a great experience for the user using the best tools I was able to find. I was taught how to handle problems, make libraries work together and use tools to achieve the final desired result. Although, the two greatest lessons I was taught during this process were **a. planning beforehand and using the best architecture is the most important thing** and **b. wen is becoming a totally different and advanced thing than it was in the past, and programmers now can really make a difference inside it.**

References

[1] https://en.wikipedia.org/wiki/Single-page_application
[2] https://en.wikipedia.org/wiki/List_of_WebGL_frameworks
[3] http://threejs.org/examples/
[4] http://searchsoftwarequality.techtarget.com/definition/use-case
[5] http://www.html5rocks.com/en/tutorials/audio/scheduling/
[6] https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API
[7] https://en.wikipedia.org/wiki/Stereophonic_sound
[8] http://www.dawsons.co.uk/blog/what-is-panning
[9] https://developer.mozilla.org/el/docs/Web/API/StereoPannerNode
[10] https://en.wikipedia.org/wiki/HUD_(video_gaming)
[11] https://developer.mozilla.org/en-US/docs/Web/API/WebVR_API
[12] https://www.leapmotion.com/