UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in
Ingegneria Informatica

SOUNDRISE 2.0: SVILUPPO DI UN'INTERFACCIA GRAFICA INTERATTIVA IN THREE.JS PER SUPPORTARE PERSONE CON DISABILITÀ UDITIVE

Relatore: Prof. Sergio Canazza Targon

Correlatore: Dott. Alessandro Fiordelmondo

Laureando: Gabriele Turetta

Anno Accademico 2022-2023
Data di Laurea 21/03/2023

Sommario

Nel 2012 Stefano Giusto e Marco Randon svilupparono come progetto di tesi di laurea magistrale *Soundrise*, un'applicazione stand-alone a scopo didattico-terapeutico rivolta a un pubblico di giovane età come strumento di supporto per la terapia vocale o per esercitare la modulazione della voce nel canto.
Il funzionamento dell'applicazione si basa sull'analisi di quattro parametri vocali principali: pitch, intensità, durata e timbro. Questi parametri vengono rilevati dall'applicazione e utilizzati per animare il modello di un sole che si modifica nell'aspetto in maniera differente a seconda della feature vocale analizzata.
L'applicazione quindi utilizza l'input vocale dell'utente per creare un'esperienza visiva interattiva, fornendo al bambino un feedback visivo sul suono emesso, aiutandolo a correggersi, con l'obiettivo di educarlo nell'utilizzo della propria voce.
Oggi il progetto originario di Soundrise non è più funzionante, a causa del mancato mantenimento del codice negli ultimi 10 anni.
Il presente lavoro si propone di ricreare ex novo Soundrise come applicazione web, mantenendo l'idea di fondo e l'intento del lavoro originario ma aggiornando e dando più importanza all'interfaccia utente, che nel progetto iniziale è stata trascurata e implementata in modo minimale. Oggi infatti il giovane pubblico è diventato più esigente, essendo quotidianamente a contatto con dispositivi multimediali, sia per la didattica che per l'intrattenimento.
Nella presente tesi viene trattata inizialmente l'interazione tra l'uomo e la macchina, con un focus sui sistemi informatici sviluppati come supporto alla didattica e alle disabilità. Si passa poi a esporre il progetto originario di Giusto e Randon, analizzandone l'idea di fondo e le specifiche base, per poi analizzare il passaggio alla versione 2.0 di Soundrise, pensata per il web e migliorata nell'interfaccia.
Quindi vengono presentate le tecnologie utilizzate per lo sviluppo del nuovo Soundrise, da Web Audio API a Three.js, per proseguire nell'analisi approfondita dell'effettiva implementazione dell'interfaccia utente.
Una discussione su possibili sviluppi futuri del progetto, anche dal punto di vista grafico, conclude questo lavoro di tesi.

Indice

Elenco delle figure

Capitolo 1

Interazione uomo macchina: supporti alla didattica e alle disabilità

L'avanzamento esponenziale dei supporti informatici nella società moderna ha reso necessaria l'elaborazione di un sistema di comunicazione adatto allo scambio di informazioni tra l'uomo e la macchina. Il livello di complessità raggiunta ha portato infatti alla costruzione di un dialogo affascinante in cui sia l'utente che lo strumento partecipano attivamente, scambiandosi informazioni e ottenendo feedback in tempo reale. Il processo di implementazione, cosciente di questa necessità, richiede quindi un approccio più umanistico al mondo asettico delle macchine, poiché il supporto tecnico non deve più risolvere solo un problema specifico, ma deve tenere anche presente l'utilizzatore finale a cui si rivolge, cioè è necessario "portare l'usabilità all'interno del processo di progettazione" [8].
Con il termine usabilità si intende l'ottimizzazione delle interazioni che l'utente ha con il prodotto, per garantire un livello di soddisfacimento adeguato alle esigenze specifiche umane. Tale soddisfacimento può essere definito come il raggiungimento dei seguenti obiettivi [8]:

- efficacia: la capacità del sistema di risolvere il problema per cui è stato progettato
- efficienza: il modo in cui il sistema aiuta l'utente a portare a termine i compiti per cui il sistema stesso è stato progettato
- sicurezza: la protezione dell'utente da situazioni pericolose o indesiderabili

- facilità di apprendimento e di ricordo: il livello di semplicità con cui l'utente impara a utilizzare il sistema e ricorda come utilizzarlo nelle sessioni successive.

Un ambito in cui l'interattività del software è fondamentale e che è di interesse per il lavoro presente è quello della didattica, in particolar modo quella rivolta a persone di giovane età, nativi digitali che necessitano di essere costantemente stimolati nell'apprendimento con forme sempre nuove. Non adeguarsi al progresso e rimanere fedele a un sistema analogico come unico strumento per l'istruzione significa accettare la morte della curiosità culturale dei giovani. Evitare l'innovazione dei sistemi didattici per pigrizia o timore significa accettare la sedimentazione nei ragazzi di un sentimento di avversione nei confronti di tutto ciò che è cultura, limitandoli nel pensiero e tarpando le ali al loro percorso di studi. Adattarsi e innovarsi è invece possibile, affiancando ai classici libri di testo laboratori virtuali, contenuti multimediali del web e software interattivi, per creare una struttura stimolante che le tradizionali forme di supporto all'istruzione a volte non riescono a garantire.
Esiste ad esempio *JigSpace* [2], applicazione che consente di esplorare in diversi ambiti disciplinari l'interno di oggetti in modo interattivo e tridimensionale (vedi figura 1.1), dalle grandi invenzioni di Da Vinci alla pressa in legno con caratteri mobili di Gutenberg. Permette inoltre di ricreare ambienti storici per visualizzare da spettatore scenari di battaglie accadute, il tutto utilizzando la Realtà Aumentata, la tecnica che aggiunge il virtuale all'ambiente reale, conferendo a quest'ultimo un ampliamento di visione e una maggiore possibilità di interazione.
AR Makr [2] è un'altra applicazione che sfrutta la realtà aumentata e mette a disposizione semplici strumenti per importare nell'ambiente virtuale disegni e foto, permettendo ai bambini di creare le proprie gallerie d'arte visitabili virtualmente. C'è poi *Flashcards Maker* che permette di creare kit di flashcards per facilitare l'apprendimento di vocaboli stranieri, *Kahoot!* [2] che permette di creare quiz in maniera giocosa, o *Flippity* [2], una risorsa gratuita per gli insegnanti integrabile con Google Sheets, che consente la creazione di quiz, schede didattiche, presentazioni, giochi di memoria, parole crociate e molto altro, rendendo più giocoso e sereno l'apprendimento.
Un altro ambito che è di interesse per il lavoro presente e in cui i sistemi informatici interattivi dimostrano il loro contributo essenziale è quello del supporto alle disabilità, in tutte le sue forme. Le applicazioni pensate per persone affette da disabilità rappresentano una testimonianza della potenza della tecnologia quando viene impiegata per migliorare la vita delle persone. Esse non sono solo un efficace supporto tecnologico, ma rappresentano anche un segno di solidarietà e

di inclusione sociale, testimoniando come la società si stia rendendo sempre più consapevole della necessità di sostenere e valorizzare le differenze, garantendo a tutti gli individui la possibilità di sviluppare al meglio le proprie capacità e talenti. Ogni individuo, indipendentemente dalla sua condizione fisica o mentale, dovrebbe poter godere di tutte le opportunità che la società offre. Tuttavia, questo diritto viene spesso ostacolato da barriere architettoniche, sociali e tecnologiche. Le applicazioni pensate per i disabili cercano di abbattere queste barriere, fornendo soluzioni personalizzate che consentono loro di superare le difficoltà quotidiane e di esprimere il loro pieno potenziale.
Tra le tante soluzioni proposte per supportare le disabilità, è interessante citare ad esempio *Seeing Ai* [6], applicazione sviluppata da Microsoft Corporation che sfrutta la potenza dell'intelligenza artificiale per aiutare la vita quotidiana di non vedenti e ipovedenti. Tra le varie funzionalità offerte, *Seeing Ai* permette di pronunciare del testo scritto a mano non appena viene visualizzato davanti alla fotocamera o salvare i visi delle persone in modo da poterle riconoscere e ottenere una stima della loro età, sesso ed espressione. O ancora, l'applicazione genera un segnale acustico corrispondente alla luminosità nell'ambiente circostante, e permette di ascoltare le descrizioni delle foto salvate sul dispositivo.
Esiste poi *Wheel Mate*, pensata per persone con disabilità motorie, che fornisce una panoramica dinamica dei servizi pubblici più vicini e accessibili ai disabili. Anche in questo caso l'idea del software nasce dalla necessità di agevolare la quotidianità in tutti i suoi momenti, soprattutto in quelli che noi affrontiamo con naturale semplicità ma possono risultare più impegnativi per gli invalidi.
E ancora, sempre per ipovedenti e non vedenti, esiste Be My Eyes [1], un ambizioso progetto che mette in contatto utenti con disabilità visive e volontari normovedenti pronti a dare aiuto nei piccoli problemi quotidiani. All'utente che ha bisogno di aiuto per controllare ad esempio la scadenza di un prodotto alimentare o leggere delle istruzioni, basterà rivolgersi all'interfaccia di Be My Eyes, che notificherà prontamente i volontari normovedenti chiedendo di intervenire attraverso una videochiamata con i richiedenti aiuto.
Sono poi numerose le applicazioni che utilizzano la AAC (Augmentative and Alternative Communication), che comprende tutti i metodi di comunicazione utilizzati per integrare o sostituire il parlato e la scrittura per coloro che hanno difficoltà nella produzione o comprensione della lingua parlata o scritta. Applicazioni di questo tipo forniscono un sistema di comunicazione basato su immagini offrendo un set espandibile di icone, che reagiscono al tap dell'utente riproducendo una parola o un concetto specifico. Esempi di sistemi di questo tipo sono *Voice4u AAC*, *Proloquo2Go*, *CoughDrop* e *Speak for Yourself*.

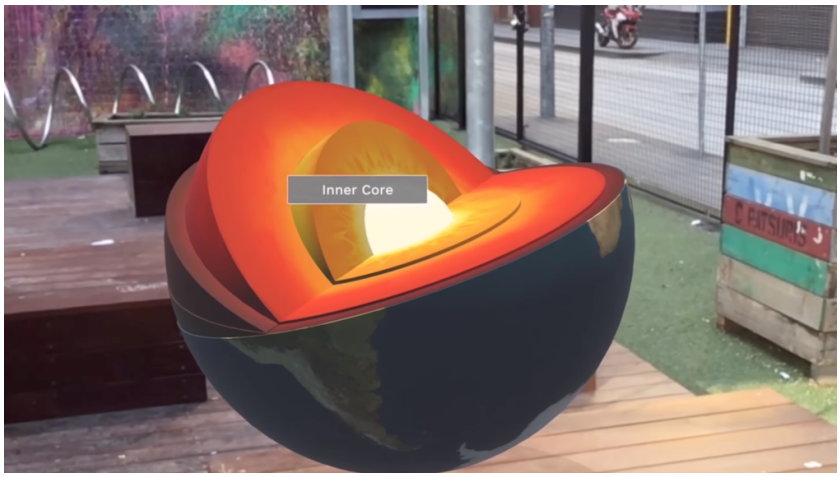

Figura 1.1: *schermata di JigSpace, con la visualizzazione della struttura interna della Terra*

È importante notare inoltre che esigenze altamente specifiche e personalizzate possono portare a prodotti altamente sofisticati, e diventa quindi fondamentale rendere queste applicazioni il più possibile user friendly e capaci di mantenere la capacità interattiva e la creatività degli utenti, specie per quelli di giovane età. Solo in questo modo questi strumenti possono diventare la chiave del successo del progetto educativo di inclusione dei ragazzi disabili nella società di tutti, come fondamento di una piena cittadinanza e realizzazione individuale.
Sono stati analizzati due ambiti in cui i supporti informatici svolgono un ruolo d'aiuto fondamentale, vale a dire la didattica e le disabilità. Soundrise 2.0 si va a collocare a cavallo tra questi due campi, presentandosi come uno strumento a scopo didattico-terapeutico, rivolto a utenti di giovane età con disabilità uditive e quindi con difficoltà nella lingua parlata, e al contempo si costituisce come strumento di supporto all'allenamento vocale di giovani ragazzi nell'esercizio del canto.

Capitolo 2

Soundrise, la nascita del progetto

2.1 Soundrise 1.0

Il progetto iniziale di Soundrise nasce come risultato di una tesi di laurea magistrale del 2012 di Stefano Giusto e Marco Randon, laureati magistrali in Ingegneria Informatica all'Università di Padova, in collaborazione con la dott.ssa Serena Zanolla dell'Università degli Studi di Udine e dei prof. Federico Avanzini, Antonio Rodà e Sergio Canazza del Dipartimento di Ingegneria dell'Informazione dell'Università di Padova.
L'applicazione originaria si presenta come un prodotto stand-alone, realizzato con Pure Data (Pd), un ambiente di programmazione grafica real-time per l'elaborazione audio e video, sviluppato in C e scritto per essere multipiattaforma, permettendo l'esecuzione del programma in sistemi operativi diversi e anche su dispositivi mobili. L'idea di partenza è poter rappresentare tramite dei tratti grafici ben distinti quelle che sono le feature più significative della voce, con il fine di farle comprendere ai bambini. L'applicazione riceve infatti input vocali dall'utente e risponde visualizzando a schermo il modello di un sole che si anima in maniera differente sulla base dei parametri principali della voce (pitch, intensità, durata e timbro). In questo modo il bambino ottiene un feedback visivo sul suono emesso, aiutandolo a correggersi, con l'obiettivo di educarlo nell'utilizzo della propria voce.
Il progetto infatti, è stato proprio pensato per scopi logopedistici, considerando le difficoltà che alcuni bambini con problemi di udito hanno nel pronunciare certi tipi di suoni e nel conoscere i tratti distintivi della voce, ma anche per educare i ragazzi nell'esercizio del canto, permettendogli di esercitarsi in autonomia a modulare la propria voce.

Figura 2.1: *Una stanza logo-motoria. A terra un tappeto verde su cui l'utente può muoversi, a sinistra un terminale con Soundrise in esecuzione*

Il progetto iniziale di Soundrise prevede anche la *Stanza Logo-Motoria*, un'estensione dell'applicazione visibile in figura 2.1, che permette di aumentare l'interattività multimodale del sistema Soundrise. Si tratta di un sistema real-time che attraverso una webcam analizza il movimento del corpo e la mimica dell'utente all'interno di un ambiente sensorizzato, selezionando la feature vocale da estrarre e rappresentare graficamente in base alla zona prefissata della stanza in cui l'utente decide di spostarsi fisicamente ed esercitare la propria voce.
Per realizzare questa estensione gli autori hanno optato per *EyesWeb*, una piattaforma per la progettazione e lo sviluppo di interfacce e sistemi realtime multimodali, che comunica con Pure Data tramite il protocollo di comunicazione OSC (Open Sound Control). EyesWeb legge l'ingresso proveniente da una webcam e manda in uscita il segnale di controllo per Pure Data [5].
La stanza multimodale non è stata implementata per Soundrise 2.0.

2.2 Specifiche base

Le feature del segnale vocale che Soundrise analizza e rappresenta graficamente in tempo reale, animando il sole, sono le seguenti, visibili anche nello schema in figura 2.2:

- **intensità**: caratteristica del suono che dipende dall'ampiezza dell'onda e permette di ordinarlo su una scala che si estende dal piano al forte; Questo

parametro viene mappato nella variazione di dimensione del Sole.

- **altezza (pitch)**: caratteristica del suono che dipende dalla frequenza dell'onda sonora che lo ha generato. Permette di distinguere se un suono è acuto o grave; viene mappato nella variazione di altezza del sole

- **durata**: il periodo di tempo occupato dall'emissione di un suono; Nell'interfaccia grafica, il sole manterrà un sorriso per l'intera durata del segnale vocale.

- **timbro**: caratteristica del suono determinata dagli armonici che accompagnano il suono fondamentale. Permette di distinguere due suoni identici per intensità e altezza, ma emessi da due sorgenti diverse. Per la voce umana, il riconoscimento di questa proprietà è relativa all'identificazione delle cinque vocali della lingua italiana; Nell'interfaccia grafica a timbri diversi sono fatti corrispondere colori diversi del sole.

L'estrazione e l'elaborazione del segnale vocale viene effettuata attraverso oggetti messi a disposizione da Pure Data. L'oggetto `[env]` per esempio prende un segnale audio in ingresso e analizza 1024 campioni all'interno della finestra di analisi per un periodo pari alla metà della dimensione della finestra. Quindi, l'oggetto manda in uscita il corrispondente valore efficace dell'ampiezza in dB. Il valore ritornato viene quindi utilizzato per l'analisi di durata e intensità del segnale vocale. Per il primo viene considerato un valore di fondoscala, cioè una soglia di rumore sotto il quale il segnale vocale si può considerare come rumore di fondo e di conseguenza il sole non reagisce all'input vocale. Per l'analisi dell'intensità invece il valore in dB della potenza del segnale in ingresso è mappato su una scala di valori reali che va da 0 a 4 che regolano le dimensioni del sole.
Per quanto riguarda invece l'analisi dell'altezza della voce, gli autori hanno scelto l'oggetto `[fiddle]`, che stima il pitch di un segnale che giunge al suo ingresso basandosi sull'algoritmo per il calcolo della FFT (Fast Fourier Transform). L'analisi avviene su finestre di 2048 campioni, con periodi di campionamento pari a metà della dimensione della finestra. Come per l'analisi di durata e intensità è presente un lower bound, al di sotto del quale il pitch viene considerato nullo. É anche presente un limite superiore, che indica il pitch massimo accettato.
L'analisi del timbro vocale risulta invece più complessa degli altri parametri. In questo contesto assume importanza il concetto di centroide spettrale, definibile come la media ponderata delle frequenze presenti nello spettro, dove il peso di

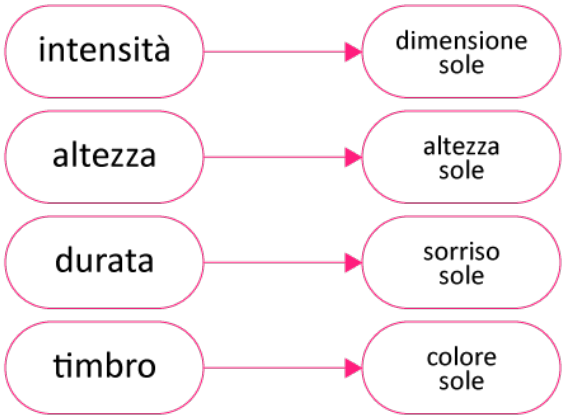

Figura 2.2: *mappatura delle feature vocali nell'aspetto del sole*

ogni frequenza è dato dall'ampiezza corrispondente. Esso costituisce un ottimo indicatore automatico del timbro musicale. Ottenuto il centroide spettrale si prosegue nell'identificazione delle vocali passando per lo studio delle formanti, che sono le frequenze di risonanza di un tratto vocale quando viene pronunciata una vocale, vale a dire le frequenze che generano la massima ampiezza di oscillazione del sistema. Lo spettro di fonemi è costituito da diverse formanti ma le prime tre sono sufficienti perchè contengono abbastanza informazioni affinché possa essere identificato e classificato un timbro vocalico:

- **F1**: la frequenza della prima formante dipende dall'apertura della bocca;
- **F2**: la frequenza della seconda formante dipende dalle diverse posizioni della lingua;
- **F3**: la frequenza della terza formante dipende dalla qualità e la chiarezza del fonema pronunciato;

2.3 Soundrise 2.0, per il web

Il progetto originario di Soundrise oggi non è più funzionante, a causa della mancata manutenzione del codice negli ultimi dieci anni, che impedisce ora di essere

compilato ed eseguito sui nuovi dispositivi.
Questa tesi, insieme a quella dei colleghi laureandi Andrea Zanetti e Riccardo Fila, si propone come obiettivo quello di sviluppare ex novo una versione di Soundrise per il web, adottando nuove tecnologie per il suo sviluppo e aggiornando l'interfaccia grafica per renderla più fruibile al giovane pubblico, che, avendo quotidianamente contatti con dispositivi multimediali, è diventato più esigente rispetto a dieci anni fa.
Come si può notare in figura 2.3, L'interfaccia grafica del progetto iniziale è stata trascurata rispetto alle altre componenti del progetto, nonostante si tratti di un elemento fondamentale, soprattutto se si considera che l'applicazione è destinata all'uso didattico per bambini piccoli. L'obiettivo di un'interfaccia curata e ben progettata è proprio quello di rendere piacevole l'esperienza dell'esercizio del bambino, trasformando così l'uso didattico-terapeutico in un momento di gioco.
Nel nuovo Soundrise, si è scelto di sviluppare un'interfaccia grafica in 3D, caratterizzata da modelli che ricordano la plastilina, ispirati allo stile dei film d'animazione in stop motion che da sempre affascinano i bambini in tenera età (figura 2.4).
La scelta poi di sviluppare il programma per il web, nasce dall'esigenza di rendere l'applicazione democratica, accessibile da chiunque e ovunque sia disponibile una connessione a internet, incentivando i giovani utenti che ne hanno bisogno a utilizzarla in qualunque momento, con il mezzo che più preferiscono. La nuova interfaccia, infatti, è stata progettata per adattarsi alle diverse dimensioni dello schermo, garantendo così una fruizione ottimale in ogni situazione.
Tuttavia, eccetto questi cambiamenti volti a modernizzare l'applicazione originaria, l'idea di fondo e lo scopo didattico introdotte all'inizio del capitolo sono state mantenute. Le feature del segnale vocale analizzate sono le stesse, quindi intensità, pitch, durata e timbro, le quali sono mappate graficamente nelle caratteristiche del sole definite in precedenza e schematizzate in figura 2.2.
Questa tesi in particolare, oltre a trattare in modo approfondito lo sviluppo dell'interfaccia utente del nuovo Soundrise, analizzerà solamente intensità e pitch. Le altre feature verranno implementate successivamente e analizzate nelle tesi dei colleghi Andrea Zanetti e Riccardo Fila.

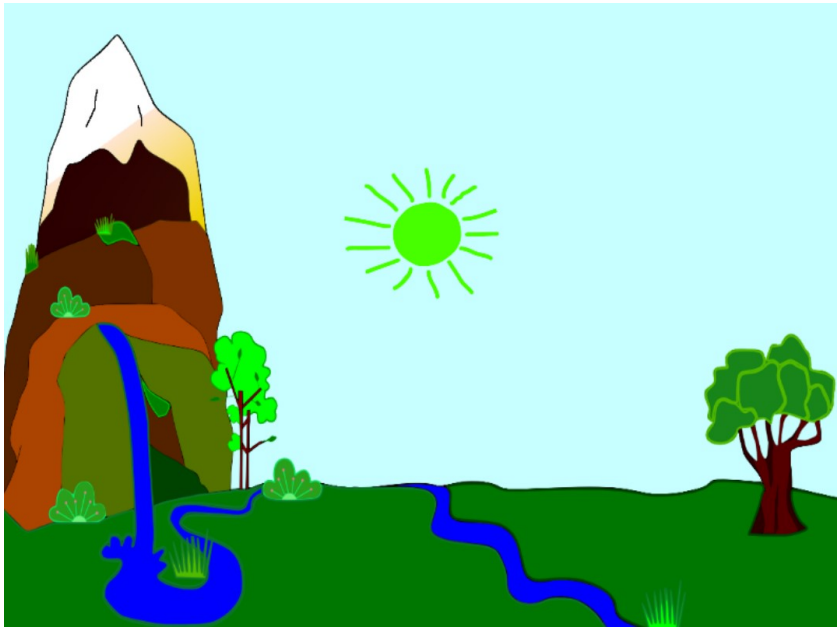

Figura 2.3: *l'interfaccia utente originaria di Soundrise*

Figura 2.4: *la nuova interfaccia utente di Soundrise*

Capitolo 3

Tecnologie Utilizzate

3.1 Web Audio API

La Web Audio API [10] è un API (Application Programming Interface) che fornisce un sistema potente e versatile per controllare l'audio su Internet, consentendo agli sviluppatori di creare strumenti interattivi, aggiungere effetti all'audio e creare visualizzazioni grafiche.
L'API prevede la gestione di operazioni audio con l'utilizzo di `audioContext`, una struttura all'interno della quale dei nodi collegati tra loro compiono le operazioni audio desiderate, formando una struttura modulare che offre la flessibilità necessaria per creare funzioni audio complesse.
Le uscite di questi nodi possono essere collegate agli ingressi di altri, che mescolano o modificano questi flussi di campioni sonori. Una volta che il suono è stato sufficientemente elaborato ed è stato ottenuto l'effetto desiderato, può essere collegato all'ingresso di un nodo di output (destinazione), che salva il risultato o invia il suono agli altoparlanti o alle cuffie.
Un semplice e tipico workflow per lavorare con l'elaborazione audio è il seguente:

1. creare la struttura `audioContext`. La sua instanziazione ci dà accesso a tutte le funzionalità dell'interfaccia

2. creare sorgenti audio all'interno del contesto (nodi sorgente)

3. aggiungere nodi che modificano e analizzano l'audio secondo i parametri desiderati (nodi di effetto)

4. scegliere la destinazione finale dell'audio, ad esempio le casse audio del terminale dell'utente (nodo destinazione)

5. connettere i nodi sorgente a zero o più nodi effetto e poi al nodo destinazione scelto

3.1.1 Algoritmi di estrazione di pitch e RMS

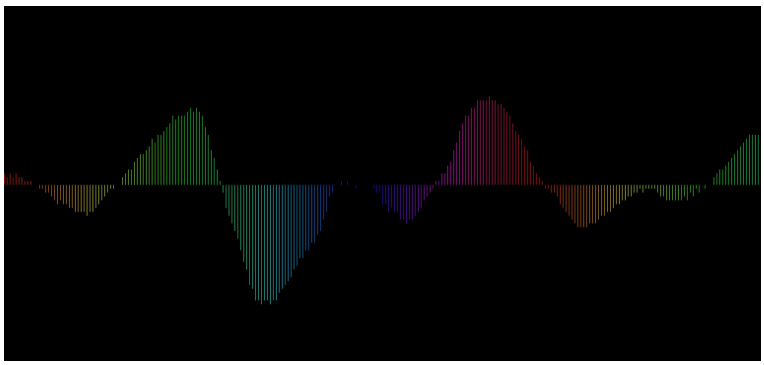

Figura 3.1: *visualizzazione in tempo reale dello spettro audio*

Il collega laureando Andrea Zanetti ha usato ampiamente Web Audio API nel suo lavoro di tesi per creare delle componenti utili ad analizzare gli input vocali, per permettere una mappatura fedele delle feature vocali sull'aspetto del sole. La classe `Microphone` da lui sviluppata rappresenta un microfono virtuale che acquisisce l'audio dal dispositivo dell'utente attraverso il browser. È in grado di fornire una rappresentazione numerica dell'onda sonora in tempo reale, nonché di analizzare la sua intensità e la sua frequenza. Più nel dettaglio:

- `getSamples()`: è la funzione che restituisce un array di campioni normalizzati del segnale audio corrente. Fornisce cioè una rappresentazione numerica dell'onda sonora che può essere poi ulteriormente elaborata attraverso l'analisi del volume o la rilevazione del pitch.

- `getVolume()`: restituisce il valore medio dell'ampiezza dei campioni audio raccolti dal microfono. Rappresenta una stima del volume del suono registrato.

- `getRMS()`: calcola il RMS (Root Mean Square) del segnale audio, cioè l'indicatore dell'intensità vocale. Un'analisi più approfondita della funzione è consultabile al capitolo 4.4.

- `getPitch()`: prende in input un array di ampiezze rappresentante un frame di un'onda sonora e restituisce mediante autocorrelazione la frequenza in Hertz (pitch) del suono. Un'analisi più approfondita della funzione è consultabile al capitolo 4.4

In figura 3.1 è possibile vedere la visualizzazione grafica della forma d'onda di un suono ricevuto in input con il microfono. Lo screenshot è stato acquisito da un'interfaccia sviluppata da Andrea Zanetti per testare il funzionamento della classe `Microphone`.

3.2 Three.js

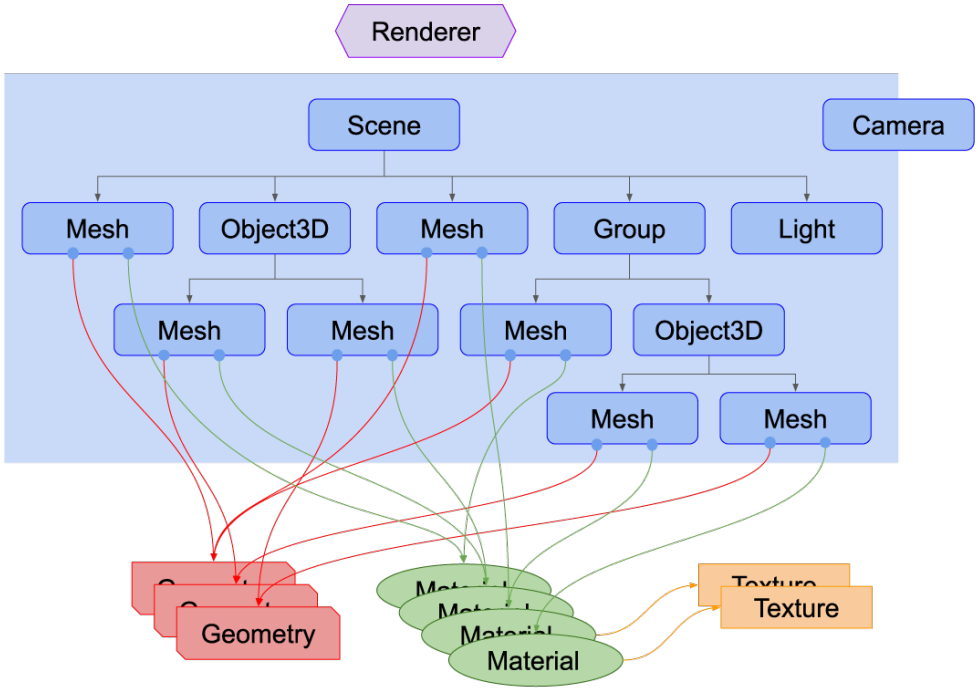

Figura 3.2: *gli elementi fondamentali di un progetto Three.js*

Three.js è la libreria JavaScript più importante per la gestione di elementi 3D nel web e il presente lavoro ne ha fatto ampio uso. Three.js nasce con l'esigenza di gestire in maniera più semplice e sintetica la renderizzazione di elementi grafici in una pagina web. È costruito infatti su WebGL (Web-based Graphics Library), una libreria grafica per il web che agisce a un livello di astrazione inferiore rispetto a Three.js e richiede quindi molte più operazioni per eseguire anche semplici istruzioni.
Ogni progetto Three.js richiede degli elementi fondamentali che sono sempre presenti. Uno schema di come questi elementi dialogano tra loro per creare un ambiente 3D è visibile in figura 3.2 [9]:

- **Renderer**: è l'elemento più importante della scena. Passandogli come parametro l'oggetto `Scene` e `Camera`, restituisce il render, vale a dire il disegno, della porzione di scena 3D visibile dalla camera.

- **Scene**: è definibile come la scenografia del progetto. Costituisce la radice dell'albero formato da tutti gli elementi della scena ed è modificabile rispetto a vari parametri come il colore del background, la sua intensità, la nebbia ecc. Ogni oggetto creato successivamente è figlio dell'oggetto scena e padre di possibili altri oggetti, creando una struttura gerarchica ad albero in cui l'aspetto di ciascun oggetto figlio risponde all'aspetto del padre.

- **Geometry**: è l'insieme di vertici che costituisce una delle geometrie della scena. La libreria offre modelli built-in facilmente implementabili come sfere, cubi, piani. Permette anche di creare geometrie personalizzate e di importare elementi 3D in diversi formati. Quest'ultima modalità è stata ampiamente utilizzata per lo sviluppo dell'interfaccia del nuovo Soundrise.

- **Material**: rappresenta le proprietà di superficie usate per definire il colore della geometria, la sua lucentezza, metallicità ecc.

- **Texture**: come `Material` costituisce l'aspetto di superficie del modello. Una texture tuttavia indica generalmente un file immagine caricato nella scena.

- **Mesh**: è l'oggetto inteso come `Geometry` a cui è stato applicato un materiale di superficie.

- **Light**: le luci della scena che rendono visibili le mesh e le loro ombre. Ne esistono diversi tipi a seconda dell'esigenza dello sviluppatore e sono schematizzate in figura 3.3 [7]:

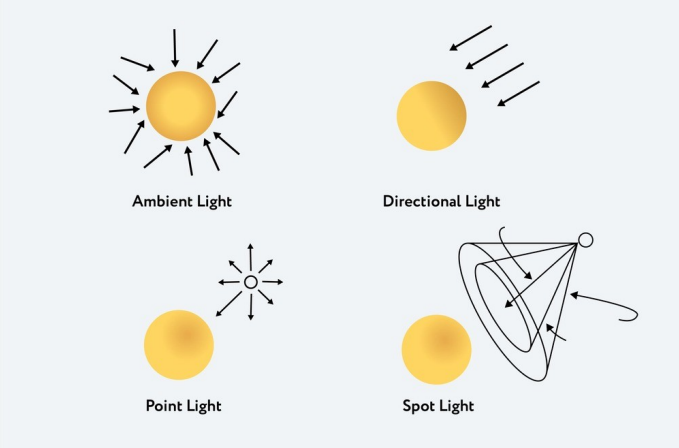

Figura 3.3: *le principali tipologie di luce in Three.js*

- **Ambient Light**: luce di background usata per illuminare tutti gli oggetti della scena allo stesso modo. Non avendo direzione non può essere utilizzata per creare ombre. Nel presente progetto non è stata utilizzata, poiché per creare la luce di ambiente e rendere quindi visibili le mesh si è preferito utilizzare una mappa HDR (High Dynamic Range) [3], un formato immagine che contiene una gamma dinamica estesa rispetto alle immagini tradizionali, ossia contiene un maggior numero di informazioni riguardo la varietà di toni di luce presenti nell'immagine. Questi metadati relativi all'illuminazione possono essere utilizzati nella modellazione 3D per ottenere un'illuminazione più naturale e simulare meglio l'ambiente esterno o le condizioni di luce interne.
  Per comprendere meglio il concetto, in figura 3.4 è riportata l'interfaccia di Soundrise con lo sfondo HDR reso visibile.
- **Directional Light**: viene emessa in una direzione specifica e i suoi raggi sono paralleli tra loro, simulando una sorgente posta infinitamente lontano dall'oggetto illuminato. A differenza della luce d'ambiente permette la visualizzazione di ombre, essendo una luce dotata di direzione specifica.
- **Point Light**: luce emessa da un singolo punto in tutte le direzioni, simulando ad esempio una sorgente luminosa di un ambiente interno.
- **Spot Light**: luce emessa da un punto verso una specifica direzione su una forma a cono, il cui vertice risiede nel punto che è la sorgente di luce e la base è posta a distanza infinita.

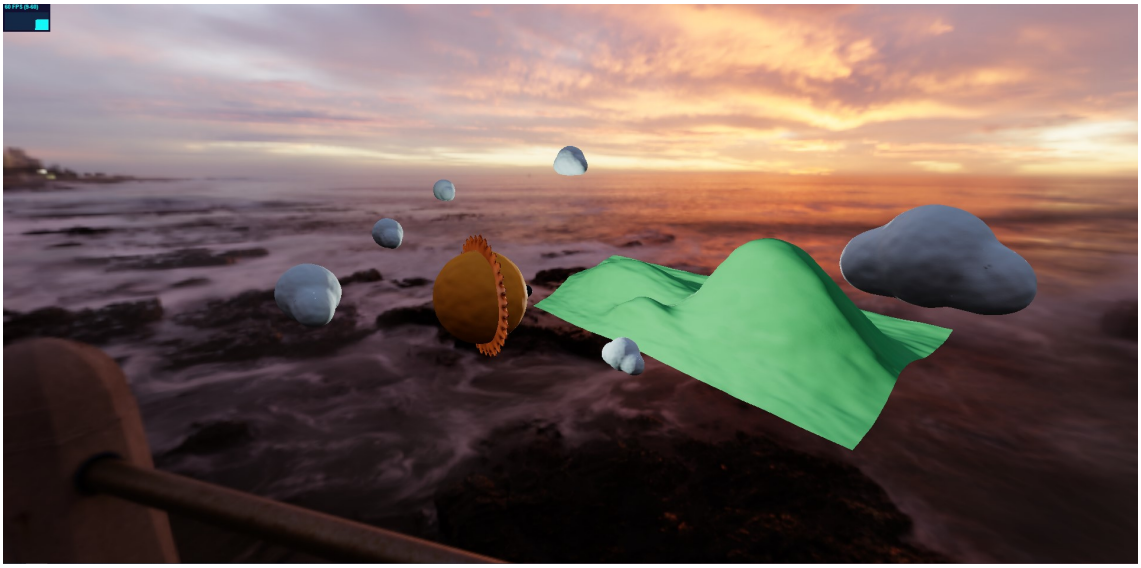

Figura 3.4: *Soundrise da un'altra prospettiva, con lo sfondo HDR reso visibile*

La Spot Light è stata l'unica sorgente di luce built-in di Three.js usata per il progetto. Una è stata implementata per essere ancorata al sole e simulare la luce proveniente dal corpo celeste, una seconda è stata aggiunta per illuminare la scena frontalmente, affiancandosi alla luce di ambiente già fornita dal background HDR e rendendo più brillanti i colori degli elementi in camera.

- **Camera**: A differenza degli altri elementi principali della scenografia, la `Camera` non è un oggetto figlio di `Scene` (figura 3.2), ma agisce in maniera autonoma. Un tipo di camera implementabile è quella prospettica (utilizzata in Soundrise), che rappresenta lo spazio tenendo conto della percezione dell'osservatore utilizzando tecniche di riduzione prospettica per creare l'illusione di profondità e di distanza. L'altro tipo di camera disponibile è quella ortografica, che invece non visualizza la scena in prospettiva, ma gli elementi mantengono le loro dimensioni indipendentemente dalla loro posizione rispetto alla macchina da presa.
  Due rappresentazioni schematiche della camera prospettica e ortografica sono visibili in figura 3.5 [4]

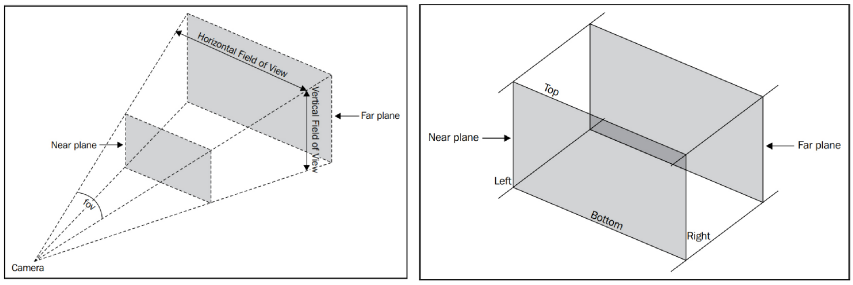

Figura 3.5: *a sinistra una camera prospettica, a destra una ortografica*

3.3 Blender Software

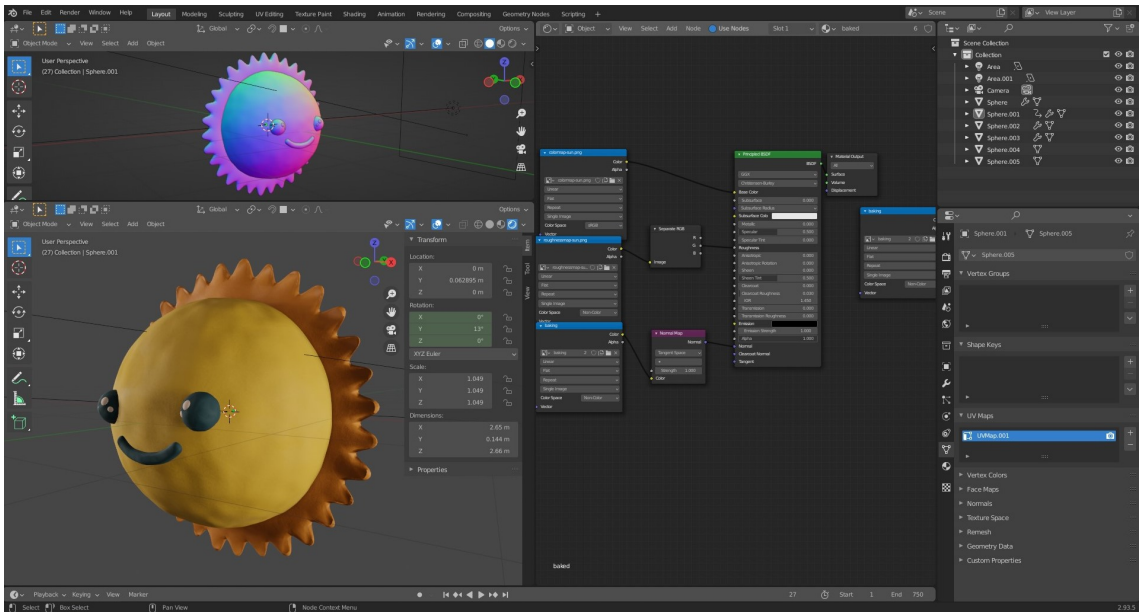

Figura 3.6: *uno screenshot dell'interfaccia di Blender*

Blender è un software di grafica tridimensionale open-source utilizzato per la creazione di animazioni, modelli e video e grazie alla sua vasta gamma di strumenti e funzionalità, si rende molto versatile per applicazioni in diversi ambiti come l'architettura, il design, l'animazione e il cinema.
La forza di Blender risiede nell'essere gratuito e open source. L'enorme comunità di appassionati che opera dietro le quinte di Blender ha permesso la creazione di

una vasta gamma di plug-in e script nel corso degli anni, permettendo al software un'evoluzione che continua a procedere senza sosta e lo ha reso un perfetto competitor degli standard industriali, che rispetto a Blender non sono gratuiti e non offrono lo stesso supporto didattico consultabile gratuitamente sul web. In figura 3.6 è riportato uno screenshot dell'interfaccia di Blender durante la modellazione del sole, protagonista del progetto.

3.3.1 Lo Shader Editor

Lo Shader Editor di Blender, visibile nella sezione destra in figura 3.6, è uno strumento potente che consente di creare materiali complessi e dettagliati per gli oggetti 3D. Questo editor è particolarmente utile per gli artisti e i designer che desiderano creare effetti visivi avanzati, come la simulazione di superfici metalliche o la creazione di materiali organici. Grazie alla sua interfaccia intuitiva, lo Shader Editor consente di lavorare con diversi nodi che rappresentano diverse proprietà dei materiali, come il colore, la luminosità e la trasparenza.
Per questo progetto lo shader editor è stato utilizzato per simulare un materiale simile alla plastilina, irregolare e con impronte digitali sparse sulla superficie.

3.3.2 Texture Baking

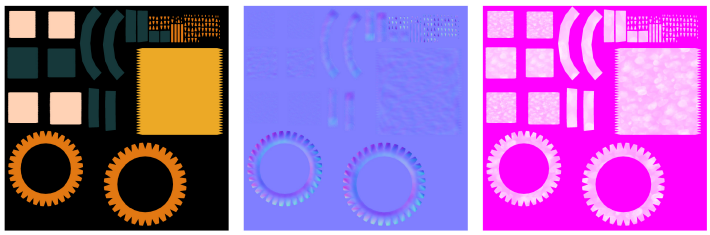

Figura 3.7: *da sinistra a destra la color map, la normal map e la roughness map*

Lo Shader Editor appena introdotto permette la creazione di materiali molto complessi e personalizzabili, ma essendo uno strumento interno di Blender, rende complessa l'esportazione del materiale creato su altri software di modellazione 3D, su motori grafici per videogiochi, e anche su Three.js.
Per sopperire a questo problema è necessario introdurre il concetto di *texture baking*, un processo che crea immagini 2D della superficie renderizzata di una mesh.

Queste immagini possono essere poi rimappate sull'oggetto utilizzando le coordinate UV dell'oggetto, che sono le coordinate bidimensionali (U e V) di una texture. In altre parole il baking produce delle immagini bidimensionali che replicano il materiale creato e, combinate tra loro, vengono poi utilizzate per "incartare" l'oggetto 3D. In figura 3.7 sono visibili le mappe 2D risultate dal baking della mesh del sole, che verranno poi ricombinate per ricreare il materiale originario implementato nello Shader Editor.

Capitolo 4

Sviluppo dell'interfaccia

La struttura gerarchia del progetto del nuovo Soundrise è molto semplice ed è stata implementata seguendo le linee guida consigliate dalla documentazione ufficiale di Three.js [9]:

- `Assets`
  - `background`: cartella contenente sfondi per la scena
  - `cloud`: modello 3D della nuvola
  - `hdr`: cartella contenente mappe HDR
  - `hills`: modello 3D delle colline
  - `sun`: modello 3D del sole
- `index.html`
- `app.js`

Il file index.html è una semplice pagina HTML in cui vengono definite le metainformazioni necessarie e viene applicato uno stile CSS attraverso un foglio incorporato tramite il tag `<style>`. Non si è optato per collegare un foglio di stile CSS esterno perché gli elementi di stile sono praticamente assenti essendo l'interfaccia completamente occupata dagli elementi visivi definiti in `app.js`. Nel `<body>` di `index.html` è presente anche l'importazione di Three.js, che può essere effettuata installando la libreria in locale con un gestore di pacchetti come NPM, o accedendo ai pacchetti necessari da remoto, utilizzando una CDN (Content Delivery Network). Si è optato per questa seconda soluzione, inserendo nel foglio HTML un tag `<script>` di tipo `importmap`:

```
  <script type="importmap">
{
  "imports": {
    "three": "https://unpkg.com/three@0.147/build/three.module.js",
    "three/addons/": "https://unpkg.com/three@0.147/examples/jsm/"
  }
}
</script>
```

App.js invece, contiene il codice javascript che effettivamente implementa l'interfaccia. Si può suddividere in quattro macrosezioni che verranno analizzate nei capitoletti successivi:

- l'inizializzazione degli elementi fondamentali della scena di Three.js con la funzione `init()`.
- l'importazione dei modelli 3D con la funzione `load()` e la loro gestione.
- il loop di animazione e la renderizzazione della scena frame per frame, con la funzione `animate()`.
- L'analisi di intensità e pitch degli input vocali con le funzioni `sourceMicrophone()`, `getRMS()`, `getPitch()`.

4.1 Inizializzazione della scena 3D

La funzione init() è responsabile di inizializzare gli elementi fondamentali dell'ambiente 3D, già introdotti in precedenza al capitolo 3.2. Il primo elemento creato è `scene`, la scenografia contenitrice di tutti gli elementi in scena:

```
scene = new THREE.Scene();
```

Si procede poi a inizializzare il motore di rendering e i suoi campi:

```
//creazione del renderer
renderer = new THREE.WebGLRenderer({alpha: true});

//imposta il rapporto tra i pixel del renderer
//e i pixel fisici del dispositivo su cui viene eseguito il codice
renderer.setPixelRatio( window.devicePixelRatio );
```

```
//imposta la dimensione del rendering sulla larghezza
//e l'altezza della finestra del browser.
renderer.setSize(window.innerWidth, window.innerHeight);

//imposta l'encoding dei colori sulla modalità sRGB, che è lo standard
//utilizzato nella maggior parte dei display
renderer.outputEncoding = THREE.sRGBEncoding;

//abilita il calcolo delle ombre nella scena
renderer.shadowMap.enabled = true;

//gestione della tonemapping della scena
renderer.toneMapping = THREE.ACESFilmicToneMapping;
renderer.toneMappingExposure = 0.3;
```

Viene quindi definita la camera prospettica, responsabile dell'inquadratura. In ordine, i suoi parametri definiti sono il FOV (Field Of View), cioè l'angolo di apertura del campo visivo della camera, l'Aspect Ratio, la distanza del piano frontale e la distanza del piano posteriore. Per una comprensione maggiore si rimanda alla figura 3.5 [4]:

```
camera = new THREE.PerspectiveCamera(
60,
window.innerWidth / window.innerHeight,
0.1,
1000);
```

A proseguire è stato definito un oggetto di tipo `OrbitControls`, uno di tipo `GridHelper` e un terzo di tipo `AxesHelper`, utili in processo di sviluppo. Il primo perché permette al programmatore di muoversi nell'ambiente 3D con l'utilizzo del mouse, per studiare gli elementi della scena da prospettive diverse, gli altri due perché visualizzano un piano cartesiano e i suoi assi per permettere una comprensione maggiore della profondità della scena (figura 4.2).
Infine si effettua l'upload dell'immagine di background di un cielo diurno, e della mappa HDR, i cui dati relativi alla luce ambientale vengono utilizzati per creare una prima illuminazione della scena. In figura 4.1 si può osservare l'interfaccia di Soundrise quando solo le `Spotlight` sono attive e la mappa HDR è disattivata.

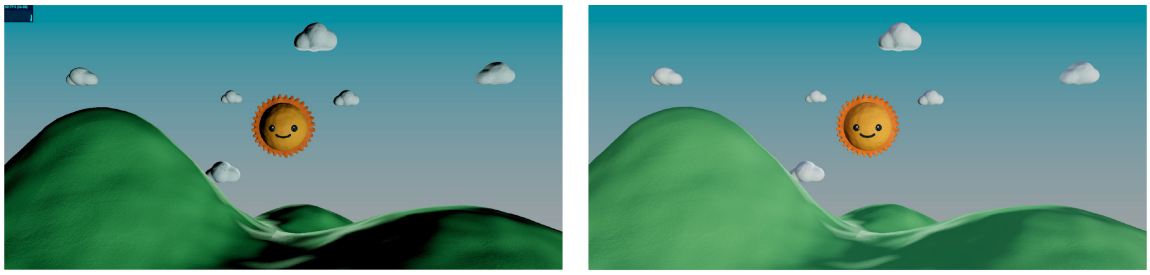

Figura 4.1: *a sinistra Soundrise con la mappa HDR disattivata a destra quando è attiva*

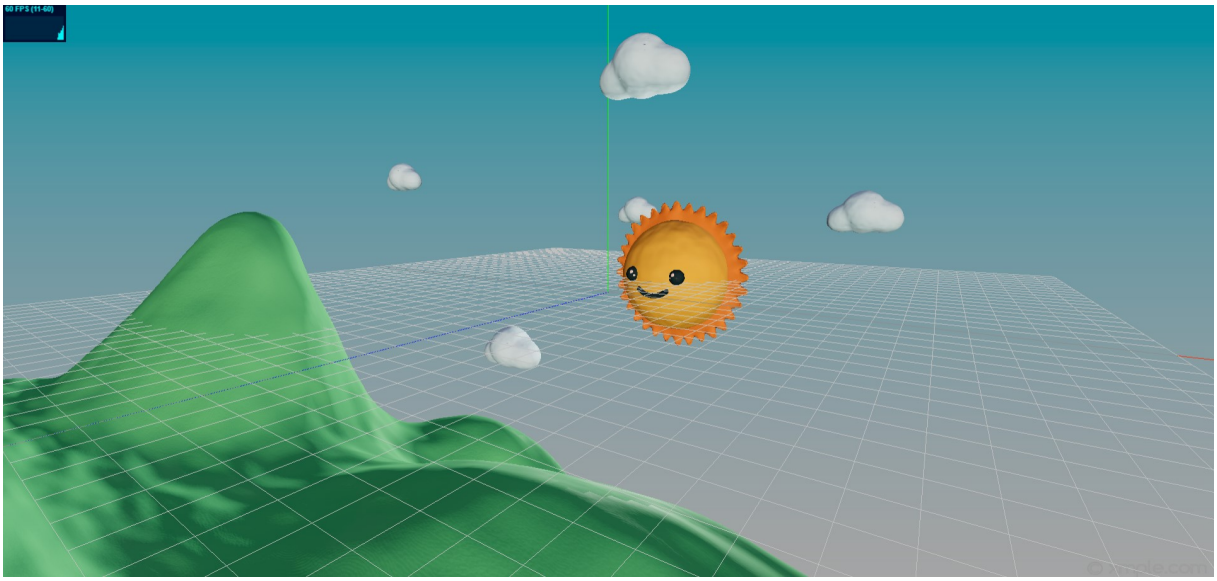

Figura 4.2: *il Grid Helper e l'Axes Helper*

4.2 L'importazione dei modelli 3D

Gli elementi 3D importati in Three.js che popolano la scena sono il sole, la nuvola e le colline. Per l'upload dei modelli si è scelto l'utilizzo di un oggetto di tipo `GLTFLoader`, passandogli come parametro il percorso del modello 3D creato in Blender ed esportato in formato .glTF. Oltre al file .glTF sono presenti nella repository del progetto una versione del modello in formato .bin e i .png risultati dal texture baking, illustrato al capitolo 3.3.2. Combinando questi elementi il loader di Three.js permette la renderizzazione nell'ambiente tridimensionale del modello importato.
Nella pratica, il caricamento dei diversi corpi 3D avviene con modalità analoghe, seguendo questa struttura:

```
let model; //dichiarazione variabile che si riferisce al modello
loader.load('./path/modello.gltf', function (gltf) {

        model = gltf.scene
        model.traverse(n => {
                if (n.isMesh) {
                //gestione ombre relative al modello
                n.castShadow = true;
                n.receiveShadow = true;
                }
        });

        //altre possibili operazioni sul modello
        model.position.z = -8; //modifica posizione lungo l'asse z
        model.scale.set(3.5, 3.5, 3.5); //modifica dimensioni

        //aggiunta del modello alla scena
        scene.add(model2);

}, undefined, function (error) {

        console.error(error);

});
```

Per il modello del sole in particolare, sono state compiute delle operazioni aggiuntive, ancorandogli una luce di tipo `Spotlight` e attivando l'animazione

dei raggi implementata in Blender durante la fase di modellazione.
Per agganciare il cono di luce al modello del sole in modo da seguirlo nei suoi movimenti è stato utilizzato un oggetto di tipo `Object3D`, che si comporta come contenitore del modello del sole e della `Spotlight`. L'ancoramento effettivo viene quindi implementato definendo il campo `target` della luce:

```
container = new THREE.Object3D();
container.add(sunmodel);
spotLight1.target = sunmodel;
container.add(spotLight1);
scene.add(container)
```

Per attivare invece l'animazione dei raggi creata in Blender si utilizza un mixer di animazione `AnimationMixer`, e lo si associa al modello del sole passandogli quest'ultimo come parametro. Quindi si recupera la lista delle animazioni (nel nostro caso solo una) dal modello .glTF e si gestisce la riproduzione di esse attraverso il mixer appena creato:

```
mixer = new THREE.AnimationMixer(model2); //inizializzazione mixer
const animations = gltf.animations; //recupero della lista di animazioni
const animation = mixer.clipAction(animations[0]); //scelta dell'animazione
//da riprodurre (in questo caso la prima e unica)

animation.play();
scene.add(model2);
```

Per permettere la riproduzione in loop dell'animazione è necessario poi aggiungere nella funzione `animate()`, responsabile di aggiornare la scena a ogni frame, la seguente riga di codice:

```
mixer.update(clock.getDelta());
```

Questa semplice riga permette di aggiornare la posizione e l'orientamento di tutti gli oggetti associati al mixer di animazione (nel nostro caso il sole), utilizzando il parametro `clock.getDelta()` per determinare quanto tempo è trascorso dall'ultima volta che è stato chiamato `mixer.update()`. Il parametro delta è solitamente calcolato come la differenza di tempo tra l'ultimo frame di rendering e il frame corrente.

4.3 Il loop di animazione

Il loop di animazione è un ciclo di rendering continuo che viene eseguito dall'engine grafico al fine di disegnare la scena 3D in modo fluido e continuo.
Durante ogni iterazione del loop, viene invocato il metodo `requestAnimationFrame()`, che indica al browser di eseguire il rendering della scena al successivo frame disponibile. Three.js aggiorna quindi la posizione e l'orientamento degli oggetti della scena in base alle trasformazioni impostate, applica le luci e le texture, e disegna i pixel della scena sullo schermo.
Nel presente lavoro il loop di animazione è stato utilizzato per visualizzare i cambiamenti dell'aspetto del sole in maniera fluida e senza interruzioni ed è stato implementato con la funzione `animate()`, che è così strutturata:

```
function animate() {
        //aggiorna scena e renderer ad ogni frame dell'animazione
        requestAnimationFrame(animate);

        if(sunModel){
            //le operazioni sul render del sole iniziano
            //quando il modello stesso è stato caricato nella scena

            //operazioni sul modello, descritte sotto.
        }

        //gestione dell'animazione dei raggi creata in Blender
        //vedi capitolo 4.2
        if(mixer) mixer.update(clock.getDelta());

        //renderizzazione del frame corrente
        render()
}
```

Più nel dettaglio, vengono prima utilizzate le funzioni `getRMS()` e `getPitch()`, descritte al capitolo 4.4, per ottenere gli indicatori di intensità vocale e pitch. Tali valori sono quindi limitati inferiormente e superiormente, rispetto a delle soglie ottenute in seguito a dei test effettuati, per poi essere mappati linearmente in un nuovo range di valori attraverso una semplice formula di proporzione:

$$\text{output} = \frac{\text{value} - \text{oldMinimo}}{\text{oldMassimo} - \text{oldMinimo}} \cdot (\text{newMassimo} - \text{newMinimo}) + \text{newMinimo}$$

In particolare il valore di `rms` viene normalizzato nel range $[0.01, 0.4]$ e quindi mappato nel range $[3, 10]$, mentre il valore di `pitch` viene normalizzato nel range $[40, 420]$ e mappato nel range $[-16, +16]$.
A ogni frame, il valore ottenuto dalla mappatura viene inserito rispettivamente nell'array `rmsArray` e `pitchArray`, i quali dopo il centesimo elemento avviano una procedura di calcolo della media di tutti gli elementi presenti al loro interno. In questo modo ogni 100 dati di pitch e rms, la media viene utilizzata per aggiornare rispettivamente la posizione del sole lungo l'asse delle altezze e le dimensioni del corpo celeste. Inoltre, dopo aver calcolato la media, l'elemento più vecchio in ciascuno dei due array viene rimosso per lasciare spazio ai nuovi valori calcolati al frame successivo.
Dopo aver effettuato l'aggiornamento dell'aspetto del modello, la funzione di animazione si occupa di gestire il cielo, che risponde ai movimenti del sole aumentando e riducendo la propria luminosità per simulare un cielo notturno.
In particolare, nel range di posizioni $y \in [-11, +16]$, una variabile di controllo dell'esposizione della scena si ottiene mappando linearmente la posizione del sole in un nuovo range di valori $[0, 0.5]$. La nuova variabile ottenuta viene assegnata al campo di controllo dell'esposizione generale, che regola la luminosità dell'immagine renderizzata. In tal modo quando il sole raggiunge la sua altezza massima ($y = +16$) la scena avrà luminosità massima (`renderer.toneMappingExposure = 0.5`), mentre quando il sole cala fino a $y = -11$, la scena avrà l'esposizione generale ridotta al minimo (`renderer.toneMappingExposure = 0`). Come si può vedere nella rappresentazione schematica in figura 4.3, per $y < -11$ solamente l'immagine di background che rappresenta il cielo si abbassa di luminosità, con modalità analoghe a quanto fatto con l'esposizione generale della scena. Si è scelto di adottare questa tecnica per mantenere visibili almeno parzialmente gli elementi 3D della scena. Infatti, variando solo l'esposizione generale lungo tutto l'asse di spostamento verticale del sole, si otterrebbe una scena quasi completamente buia nei valori più inferiori di altezza, quando cioè il sole inizia a nascondersi dietro le colline.
y = -11 inoltre, costituisce la soglia di attivazione del cielo stellato, costituito da qualche centinaia di oggetti puntiformi di colore bianco, disposti in maniera pesudorandomica all'interno dello spazio 3D (figura 4.4), e renderizzati sin dal caricamento della pagina web di Soundrise, ma non visibili grazie al parametro di opacità settato al minimo. Per y = -11, vengono invocate le funzioni `increaseOpacity()` e `decreaseOpacity()` che aumentano e diminuiscono la visibilità delle stelle rispettivamente quando il sole sale e scende oltre la soglia indicata (figura 4.5).

Figura 4.3: *rappresentazione schematica del comportamento del sole*

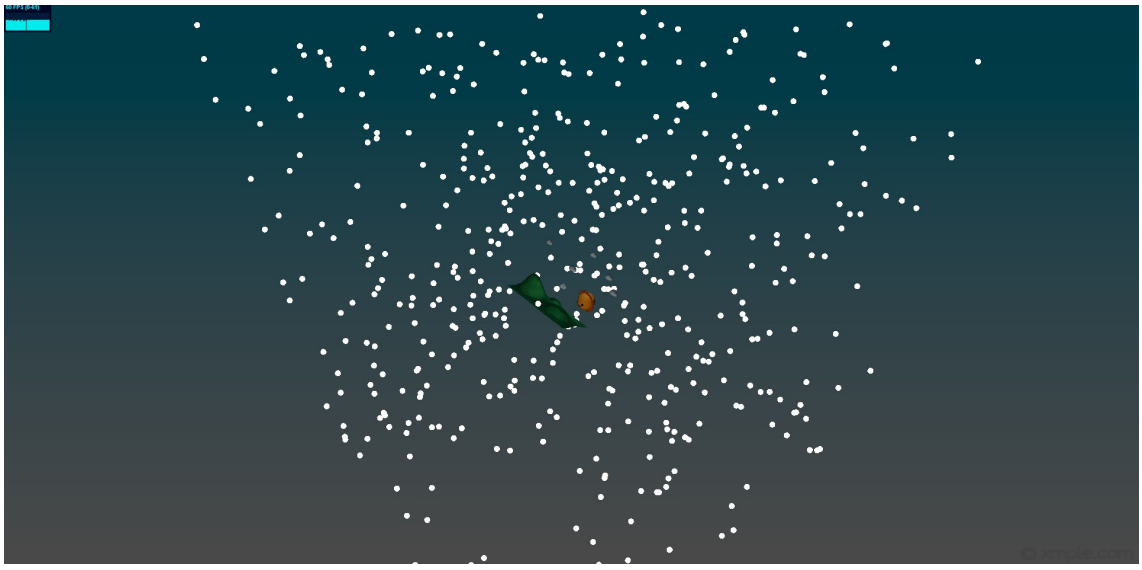

Figura 4.4: *la generazione randomica delle stelle nello spazio*

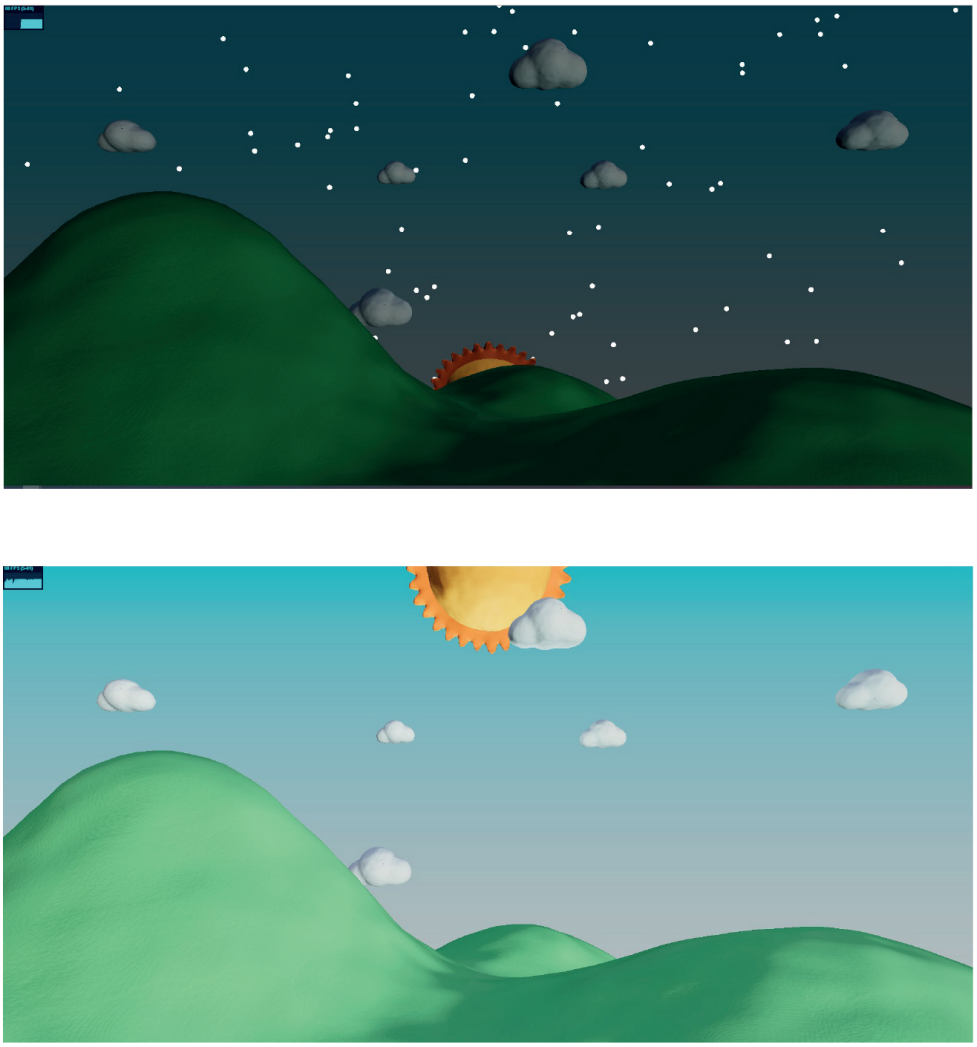

Figura 4.5: *sopra il sole all'altezza minima, con il cielo di stelle attivato, sotto il sole all'estremo superiore, con l'esposizione della scena al massimo*

4.4 Analisi degli input vocali

Tre funzioni del codice si occupano di ricevere gli input del microfono e analizzare pitch e intensità. Questo codice è stato oggetto di studio nella tesi del collega Andrea Zanetti. Si rimanda quindi al lavoro da lui effettuato per un'analisi più approfondita:

- `sourceMicrophone()`: avvia l'input audio del microfono del dispositivo dell'utente e lo connette ad un analizzatore audio. In particolare con un blocco `try-catch` tenta di accedere al microfono dell'utente chiamando il metodo `getUserMedia` sull'oggetto `mediaDevices`. La funzione `getUserMedia` restituisce una *Promise* che viene risolta con un oggetto `MediaStream` contenente l'input audio del microfono se la richiesta ha successo.

  Quando la promise viene risolta, la funzione `.then` viene chiamata con l'oggetto `MediaStream` come argomento. All'interno della funzione di callback, viene creato un nodo `MediaStreamSource` che rappresenta l'input audio del microfono e viene connesso a un oggetto `analyser` che analizza il segnale audio:

  ```
  try {
    await navigator.mediaDevices.getUserMedia(constraints)
    .then(stream => {
      var audioInput = audioContext.createMediaStreamSource(stream);
      audioInput.connect(analyser);
    });
  } catch(err) {
    console.log(err)
  }
  ```

- `getRMS()`: prende in input un array di ampiezze del segnale audio, calcola il RMS (Root Mean Square) di tale segnale e lo restituisce. Il RMS costituisce l'indicatore dell'intensità vocale e viene utilizzato nel loop di animazione per regolare le dimensioni del sole. Si calcola come:

  $$RMS = \sqrt{\frac{1}{n}(x_1^2 + x_2^2 + ... + x_n^2)}$$

  dove n rappresenta il numero di valori di cui si calcola il RMS.
  In particolare la funzione cicla attraverso l'array `timebuf` (array di ampiezze

del segnale audio) e a ogni iterazione somma il quadrato dell'ampiezza dell'elemento corrente all'accumulatore `rms`. Quindi l'RMS del segnale audio viene calcolato dividendo `rms` per la dimensione dell'array, ed estraendo la radice quadrata del risultato.

- `getPitch()`: prende in input un array di ampiezze `timebuf` rappresentante un frame di un'onda sonora e restituisce la frequenza in Hertz (pitch) del suono. Il calcolo viene effettuato mediante autocorrelazione, uno strumento matematico che confronta il segnale all'istante t con un altro valore di sé stesso ritardato di una quantità τ.
  Più nel dettaglio, nella funzione l'array `timebuf` viene ridimensionato attraverso due cicli for successivi, eliminando tutti gli elementi con ampiezza inferiore a una certa soglia `thres` definita in precedenza. Questo viene fatto per rimuovere eventuali parti della traccia audio con ampiezza molto bassa, che potrebbero essere causate da rumore o silenzio, e che potrebbero influire negativamente sul calcolo del pitch della traccia.
  Successivamente, con due cicli for annidati vengono calcolati i valori dell'autocorrelazione della traccia e viene quindi cercato il massimo valore e la sua posizione. Su questo massimo e su questa posizione viene calcolato il periodo `T0` del segnale, che permette di restituire il valore di ritorno della funzione, cioè la frequenza fondamentale del segnale, calcolata come la frequenza di campionamento diviso per `T0`.

Capitolo 5

Conclusioni e sviluppi futuri

Il lavoro svolto con questa tesi ha permesso di ottenere una prima versione di Soundrise per il web, curando e modernizzando l'interfaccia utente rispetto alla versione del progetto del 2012. Il risultato sperato, che solo l'utilizzo pratico potrà confermare, è stato migliorare e rendere giocosa l'esperienza didattica dei giovani utenti a cui questo prodotto si rivolge.
Lo sviluppo dell'applicazione non si conclude con questa tesi, che anzi ne costituisce solo l'introduzione. Difatti l'analisi delle altre due feature vocali non trattate nel presente lavoro (durata e timbro) sarà consultabile nelle tesi dei colleghi Andrea Zanetti e Riccardo Fila, che studieranno in generale il progetto da altre prospettive e analizzeranno anche più in profondità i tecnicismi che si celano dietro l'analisi dei parametri di intensità vocale e pitch.
Una volta finalizzato Soundrise con tutte e 4 le feature vocali implementate, si potrà procedere a testare l'applicazione su gruppi di bambini e apportare tutte quelle migliorie che renderanno l'applicazione un prodotto completo ed evoluto, disponibile al pubblico per l'utilizzo.
Un primo obiettivo da poter raggiungere è ad esempio il miglioramento della modalità di rappresentazione delle feature vocali. In questa prima versione implementata, i parametri vocali vengono rappresentati graficamente tutti in contemporanea, quindi il sole si modifica nell'aspetto modificando in simultanea la sua posizione, la scala, il colore e il sorriso, rendendo più difficile il riconoscimento da parte dell'utente della feature vocale analizzata.
Una soluzione possibile per ovviare a questo problema è inserire nell'interfaccia utente un piccolo menù che permetta al bambino di selezionare il singolo parametro vocale da analizzare e visualizzare a schermo, in modo che l'esercizio vocale sia ottimizzato. A tal proposito non è da escludere che la mappatura delle

Figura 5.1: *render 3D di una casa di plastilina*

feature vocali nell'aspetto del sole possa essere modificata, qualora dovessero essere individuate delle animazioni che rappresentano meglio un parametro vocale e migliorano l'esperienza didattica dell'utente.
Dal punto di vista grafico poi, sono sicuramente possibili modifiche volte a migliorare l'interfaccia. La scena può essere arricchita con altri elementi 3D (e.g. in figura 5.1), specie nelle colline che occupano la sezione frontale della scena, o del codice può essere implementato per creare delle piccole animazioni che rendano la scena meno statica, come del fumo proveniente dalla casa (figura 5.1), o delle lente traiettorie pseudocircolari per le nuvole in cielo.
Come per il progetto del 2012 inoltre, si potrebbe permettere all'utente di scegliere l'ambiente in cui giocare con il sole di Soundrise, rendendo disponibile, oltre al paesaggio collinare, un ambiente cittadino o marittimo. Diventerà però necessario prestare maggiore attenzione a ottimizzare il codice e i modelli 3D per permettere al motore grafico di renderizzare le scene in velocità senza ritardi fastidiosi per l'utente.
Concludendo, sarebbe interessante implementare anche un semplice menù di benvenuto all'apertura dell'applicazione web, con una sezione dedicata alla guida all'uso di Soundrise, un bottone per la selezione dell'ambiente visivo in cui giocare e un bottone per avviare la sessione di allenamento vocale.

Bibliografia

[1] M. AVILA, K. WOLF, A. BROCK, AND N. HENZE, *Remote assistance for blind users in daily life: A survey about be my eyes*, (2016).

[2] L. BIANCATO AND D. TONIOLI, *101 idee per una didattica digitale integrata*, Erickson, 2021.

[3] BLÉ, *Cosa sono le mappe hdr o file hdri*. https://xn–bl-8ia.com/rendering/cosa-sono-le-mappe-hdr-o-file-hdri/.

[4] J. DIRKSEN, *Learning Three.js, the JavaScript 3D Library for WebGL*, PACKT Publishing, 2013.

[5] S. GIUSTO, *Soundrise: studio e progettazione di un'applicazione multimodale interattiva per la didattica basata sull'analisi di feature vocali*, Master's thesis, Università di Padova, 2012.

[6] C. GRANQUIST, *Evaluation and comparison of artificial intelligence vision aids: Orcam myeye 1 and seeing ai. journal of visual impairment blindness*, (2021).

[7] INTEXSOFT, *Introduction to 3d: Three.js basics*. https://intexsoft.com/blog/introduction-to-3d-three-js-basics/.

[8] J. PREECE, Y. ROGERS, AND H. SHARP, *Interaction Design*, Apogeo, 2004.

[9] THREE.JS, *Fundamentals*. https://threejs.org/manual/.

[10] W. TURNER, *JavaScript for Sound Artists*, CRC Press, Taylor Francis Group, 2017.