|  |
| --- |
| Visualizzatore di features in HTML5 |
| Progetto di Computer Music 2016 |
|  |
| Relazione del progetto su di un visualizzatore di features musicali basato su tecnologie web, principalmente HTML5 e Javascript. |
|  |

Visualizzatore di features in HTML5

Progetto di Computer Music 2016

# Introduzione

L’idea di partenza del progetto è di realizzare un visualizzatore di features usando tecnologie web, ovvero basandosi su Javascript e HTML5, possibilmente evitando librerie di terze parti e usando le risorse messe a disposizione dagli standard web sopra citati. In particolare la richiesta comprendeva 4 realizzazioni:

1. Visualizzazione di un segnale monodimensionale: ricevendo un valore in ingresso alla volta costruisce una rappresentazione monodimensionale, come la forma d’onda. Quando lo spazio di rappresentazione termina, viene sovrascritta con un aggiornamento a scorrimento verso sinistra.
2. Visualizzazione di un segnale bidimensionale: da un ingresso vettoriale di valori, rappresenta lo spettro del segnale audio
3. Visualizzazione a istogramma: ricevendo un ingresso vettoriale visualizza un grafico a barre dello spettro.
4. Visualizzazione libera: scelta libera di rappresentazione con forme, colori e oggetti che si muovono seguendo dei parametri delle features.

La generazione delle features è offline, per cui bisogna usare Matlab e il toolbox “MirToolBox”, esportarle in JSON e importarle sulla pagina HTML. La dimostrazione del progetto comprende una pagina web con un database di canzoni tra cui scegliere e visualizzare le features.

Lo studio del problema è iniziato partendo dalla conoscenza dei metodi richiesti e vari tentativi di implementazione, studiando anche i problemi prestazionali riguardanti alcune realizzazioni e trovandone una abbastanza performante.

# Matlab – MirToolBox

La parte di codice su Matlab è molto essenziale e si commenta da sé, in buona parte. Ho costruito uno script che legge nella cartella di test tutti i file audio disponibili e ne fa l’analisi. Successivamente vengono creati dei file JSON per ogni audio con i dati delle analisi e del file.

La suite di metodi MirToolBox permette di operare in gran precisione sui segnali audio, semplificando l’analisi delle features. Sono inclusi anche dei metodi specifici per lo studio delle features in machine learning, ma sono caratteristiche troppo avanzate per quello che mi è stato richiesto. In particolare, il plug-in tratta i dati delle analisi come “miraudio object”, struttura complessa in cui salva le caratteristiche analizzate. Per iniziare l’analisi di una traccia, dobbiamo innanzitutto caricarla in memoria attraverso il metodo “miraudio”: qui possiamo già specificare alcune caratteristiche dell’analisi, come Frame, Frequenza di campionamento, Canali ma ciò non toglie che possano esser ri-elaborate anche in futuro, semplicemente applicando un’ulteriore volta “miraudio” all’oggetto.

Una volta estratte le features non è possibile lavorarci direttamente in Matlab, bisogna convertirle in variabili manipolabili, quindi vettori/matrici: usando il metodo “mirgetdata” o “get” è possibile convertire alcuni parametri dall’oggetto miraudio a variabile di Matlab. I dati non sempre sono normalizzati o in ordine, per questo è sempre bene sperimentare per visualizzare il “come” vengono passati i dati di interesse a Matlab. Per questo, dopo l’analisi ho introdotto una serie di operazioni, per la normalizzazione/estrazione dei dati in un formato compatibile con JavaScript e le rappresentazioni. Introduco anche un arrotondamento a 4 cifre significative dopo la virgola, dato che JavaScript su alcuni browser ha delle incompatibilità con la lunghezza dei numeri e in ogni caso la percezione umana non è così sensibile da notare un errore a quel livello di precisione.

## JSON – JavaScript Object Notation

Per poter trasferire i dati analizzati da Matlab all’applicazione web, ho usato il formato JSON, come da richiesta. Questo formato di dato puro formattato permette di trascrivere in file di testo puro strutture di oggetti Javascript. Matlab ha un plug-in molto comodo e preciso che costruisce un file Javascript a partire da qualunque (o quasi) oggetto Matlab.

Per caricare il file JSON bisogna fare una richiesta a-sincrona al server che carica il file da poi tradurre in oggetti attraverso il metodo nativo di Javascript : JSON.parse(), il quale riceve in ingresso una “stringa” e la trasforma in oggetti.

# JavaScript e HTML5

Con l’ultima versione di HTML, sono stati universalmente dichiarati dei tag specifici per elementi multimediali per elementi che potessero essere interpretati direttamente dal browser senza dover passare da plug-in specifici. In particolare fanno comodo il tag <audio> che definisce un elemento audio riproducibile dal browser, il problema del formato del file audio è legato alle capacità del browser ma attualmente wav e mp3 sono pienamente supportati, e il tag <canvas> che rappresenta un area in cui ci sarà una rappresentazione webGL: questa rappresentazione è gestita in Javascript nativo o con librerie di terze parti che possono semplificare la gestione di elementi quali SHADERS e il calcolo matriciale complesso che sta dietro alle manipolazioni in 3D.

Per le prime tre rappresentazioni ho usato quello che offre JavaScript puro, le rappresentazioni sono veloci e in 2D. Per la rappresentazione personalizzata ho chiesto e ricevuto il permesso dell’utilizzo di una libreria esterna, THREE.js, così da potermi permettere una rappresentazione 3D nelle tempistiche chieste per la realizzazione del progetto.

### Web Audio

In JavaScript è stata implementa anche una ricca libreria per l’analisi e gestione del flusso audio. È possibile gestire non solo la riproduzione di una traccia audio ma anche effettuarne l’analisi durante la riproduzione: in realtà è un “blocchetto” che si introduce tra la sorgente e il sistema di riproduzione. Grazie a questa libreria è semplificata l’analisi in tempo reale delle features ed è possibile in pochi passaggi implementare un visualizzatore. Per questo è stata tralasciata, o implementata con meno cura, la funzione .currentTime() che fornisce l’informazione a quanto tempo siamo di riproduzione. Il metodo che è associato a tutti gli oggetti di tipo “audio”, ovvero dichiarati tra i tag HTML5 <audio></audio>, viene implementato in modo diverso tra i browser, variandone anche la precisione, addirittura tra una misurazione e l’altra. Questa rimane comunque una realizzazione migliore rispetto a doversi creare un “timer” in javascript.

### WebGL

Le rappresentazioni grafiche, 2D o 3D seguono lo standard dettato da webGL, estensione di OpenGL ma in linguaggio web. Le rappresentazioni 2D utilizzano dei metodi standard per la creazione di elementi base: per le linee utilizzo .beginPath(), lineTo(), moveTo() e stroke() , per i rettangoli il metodo fillRect(). In 2D le coordinate sono in Screen Space, ovvero “ribaltate” rispetto a quello che ci si aspetta: l’origine degli assi x e y è in alto a sinistra e la x incrementa da sinistra verso destra, mentre la y incrementa dall’alto verso il basso. Questo può portare confusione alla lettura del codice, in quanto ci si aspetterebbe l’incremento di y e si legge un decremento. In 3D, la libreria THREE.js si preoccupa di realizzare una scena coerente con lo spazio di visualizzazione secondo i suoi criteri: in particolare la dichiarazione della struttura geometrica del piano è diversa rispetto a quello delle box rettangolari, risulta un piano verticale che ho dovuto ruotare di 90° per averlo nella posizione voluta.

## Struttura del programma

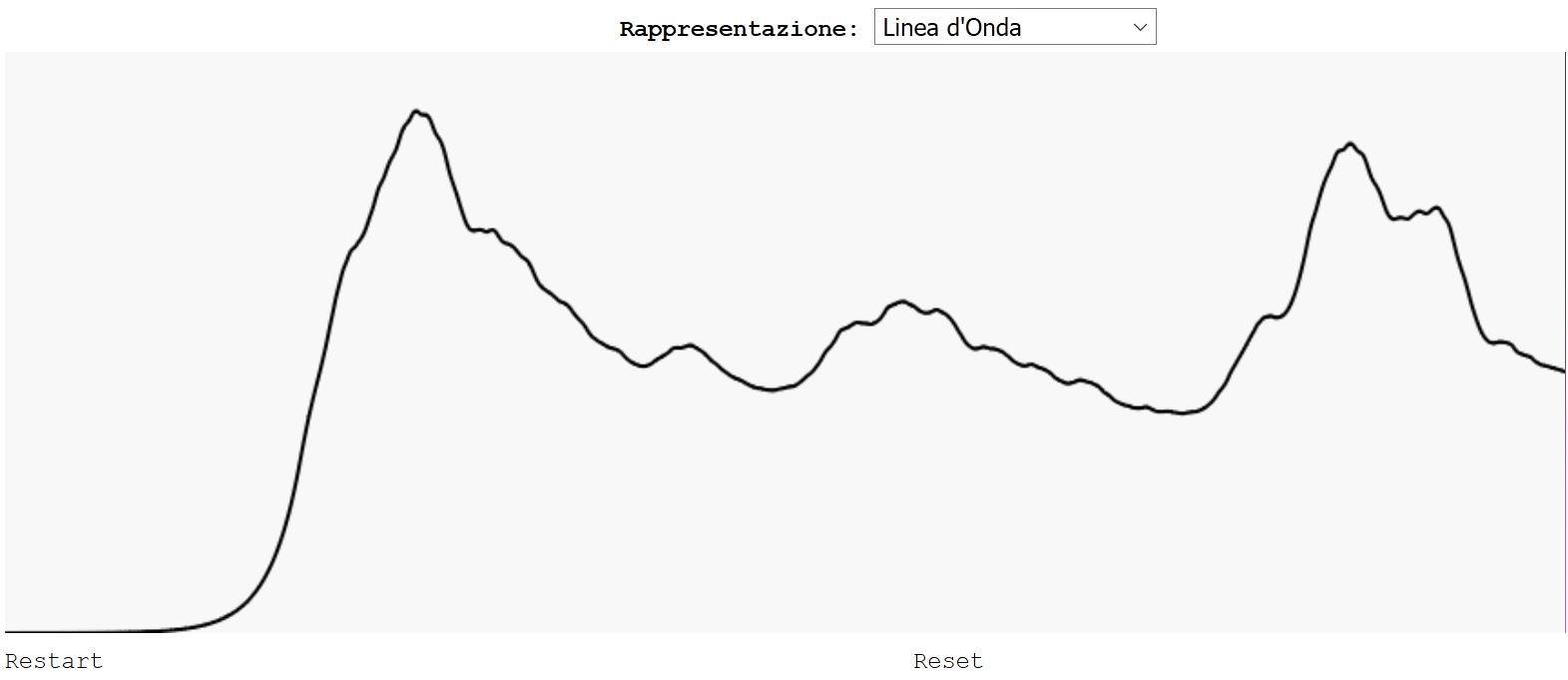
Lo script viene avviato attraverso un file js chiamato loader.js nel quale si fa il caricamento dei dati generali e la ricerca dei file JSON per ogni traccia audio. Una volta terminato il caricamento della struttura dati di analisi, viene lanciato lo script che crea le rappresentazioni.

Le rappresentazioni sono selezionabili da un menu a tendina, anche quando è in esecuzione un brano o un'altra rappresentazione. Sulla destra è possibile selezionare quale traccia eseguire tra quelle disponibili analizzate: alla selezione, viene interrotta l’esecuzione corrente e caricato il nuovo set di dati. È possibile mettere in pausa, riprendere e far ricominciare l’esecuzione dal campione iniziale. Al termine dell’esecuzione del brano viene ripristinata la visualizzazione alla condizione iniziale.

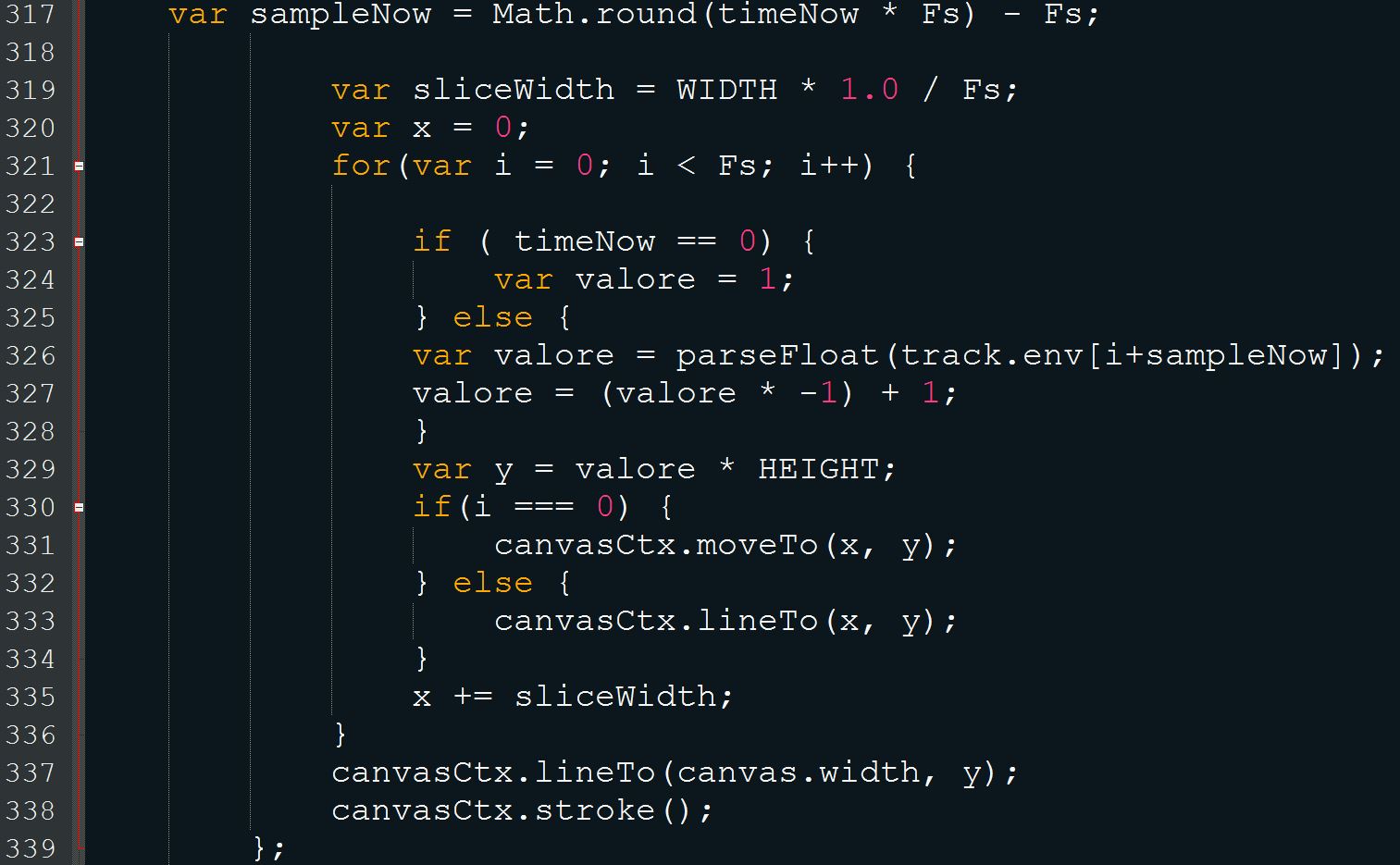
Sopra le rappresentazioni, vengono visualizzate le informazioni relative al file in riproduzione: Titolo, frequenze di campionamento, canali di filterbank e durata in secondi.

La visualizzazione di ogni rappresentazione viene determinata dalla funzione “draw()” che differisce come dichiarazione per ognuna delle diverse visualizzazioni. Nonostante possa essere più efficiente effettuare la selezione della rappresentazione all’interno della draw() ho preferito esplicitare la diversa dichiarazione per ogni rappresentazione, in modo che fosse più leggibile il codice. Questo porta a delle ripetizioni di alcuni elementi comuni di gestione, come il rilevamento del campione attuale e del momento temporale.

# Rappresentazione “Linea d’Onda”

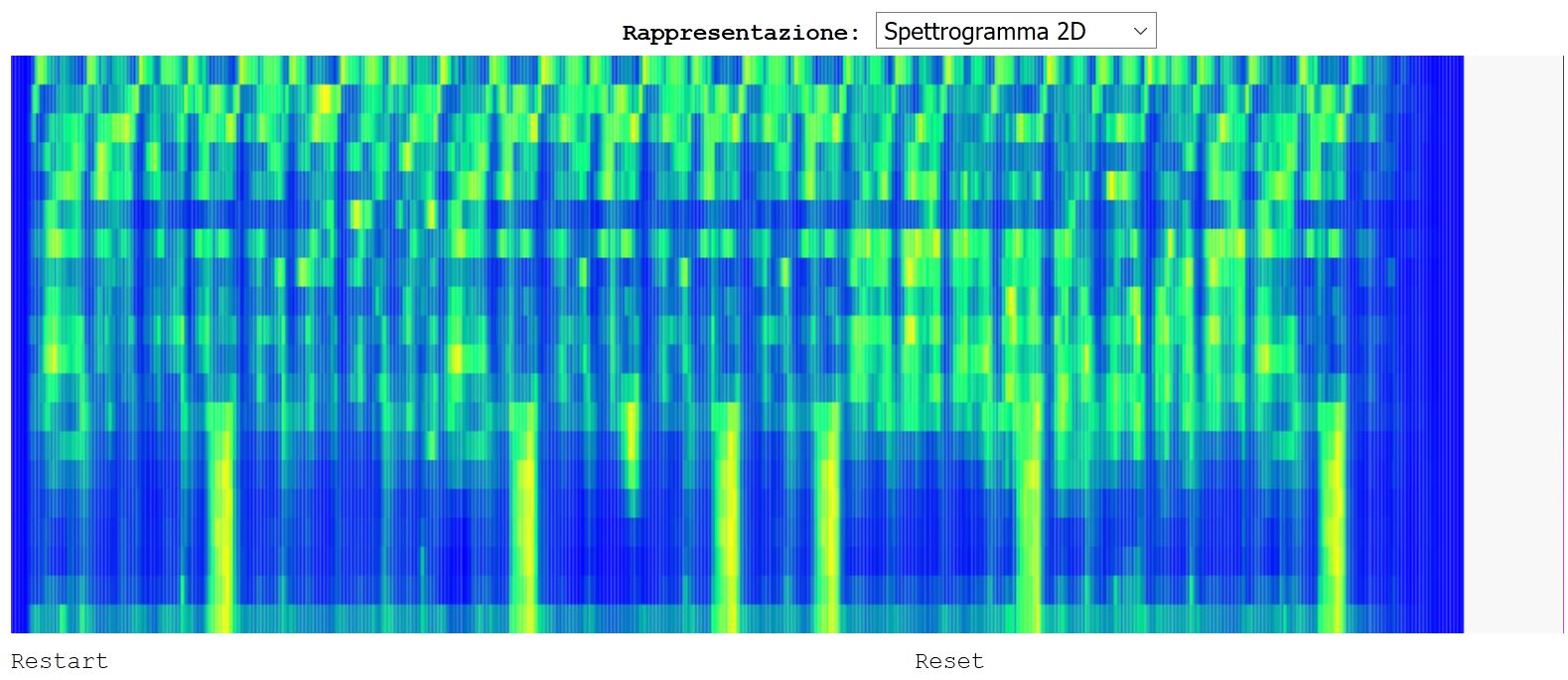


Per questa rappresentazione ho pensato di usare una linea tra i due estremi del “canvas” e spostarne i punti intermedi a seconda del valore di envelope attuale. I campioni rappresentati sono quelli letti all’interno di un “buffer” pari quanto la frequenza di campionamento, quindi rimane visualizzata l’onda fino a 1 secondo passato.

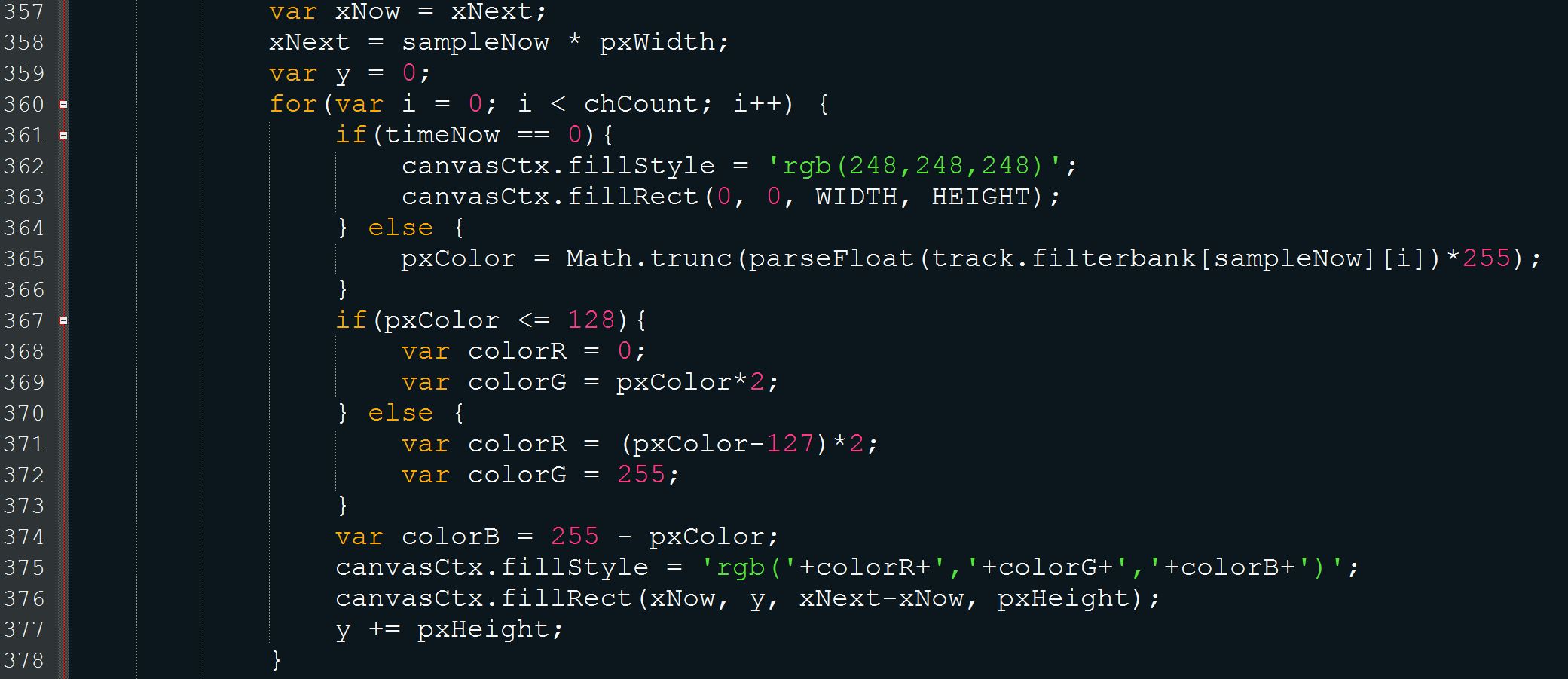


Il campione attuale viene “portato indietro” di Fs in modo sincronizzare la rappresentazione video con quella audio: in ogni refresh dello schermo rappresento un secondo (o Fs campioni) di segnale. Il valore viene parsato in Float per assicurarsi di leggere un numero e non una stringa. Da Matlab i valori vengono girati per poter avere una rappresentazione coerente con lo schermo (linea 327).

# Rappresentazione “Spettrogramma 2D”

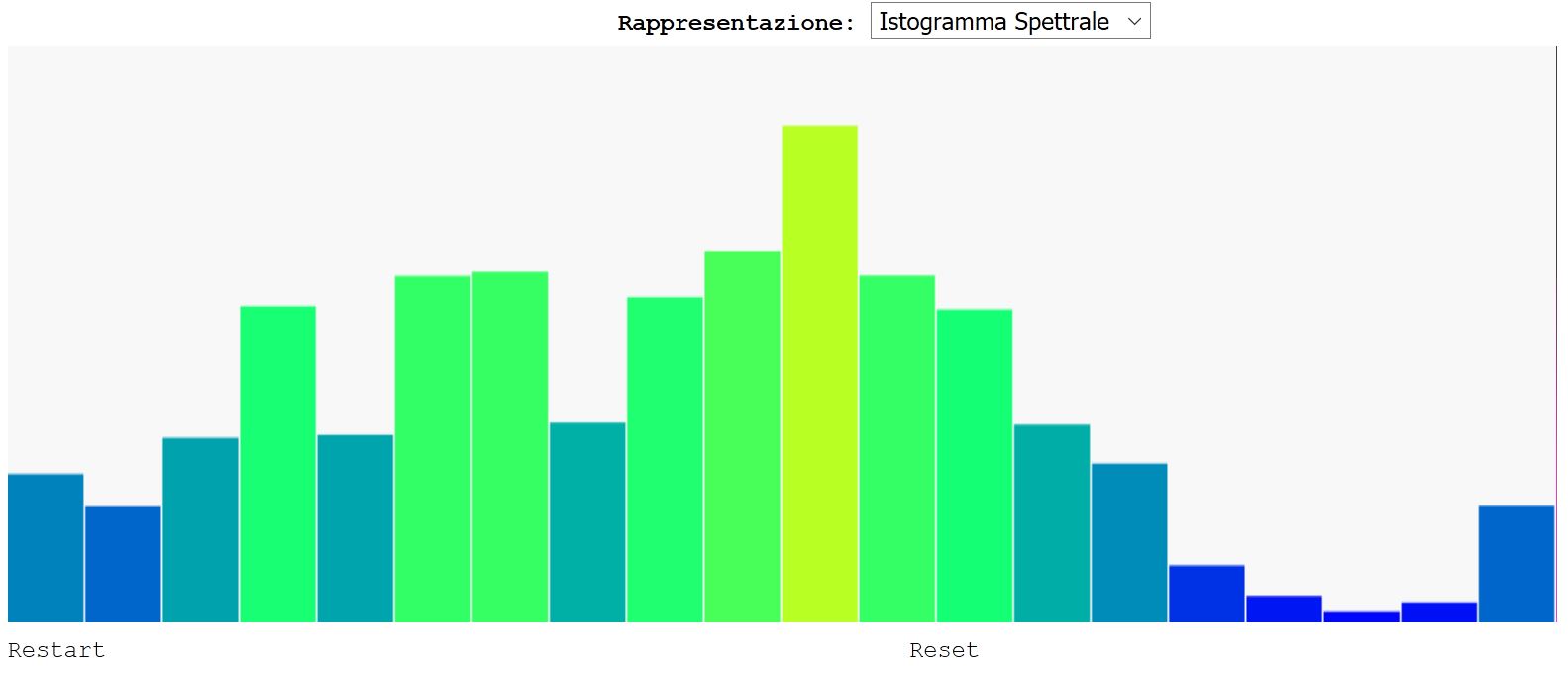


In questo caso prendiamo i valori in uscita dal filterbank, si tratta della envelope dei canali del filterbank, per creare uno spettrogramma dei valori ad ogni frequenza. Il colore varia a seconda dell’intensità registrata sul canale, passando dal blu in caso di bassi valori, al giallo per valori intensi. Il “canvas” viene sezionato verticalmente a seconda del numero dei canali, orizzontalmente a seconda del numero di campioni: per ogni campione in esecuzione viene colorato un rettangolo di sezione per ogni canale con i valori estratti dalla “matrice” di dati ricavati dal filterbank. I valori dei colori sono troncati, dato che il valore decimale non è visibile e porta instabilità in certe situazioni.

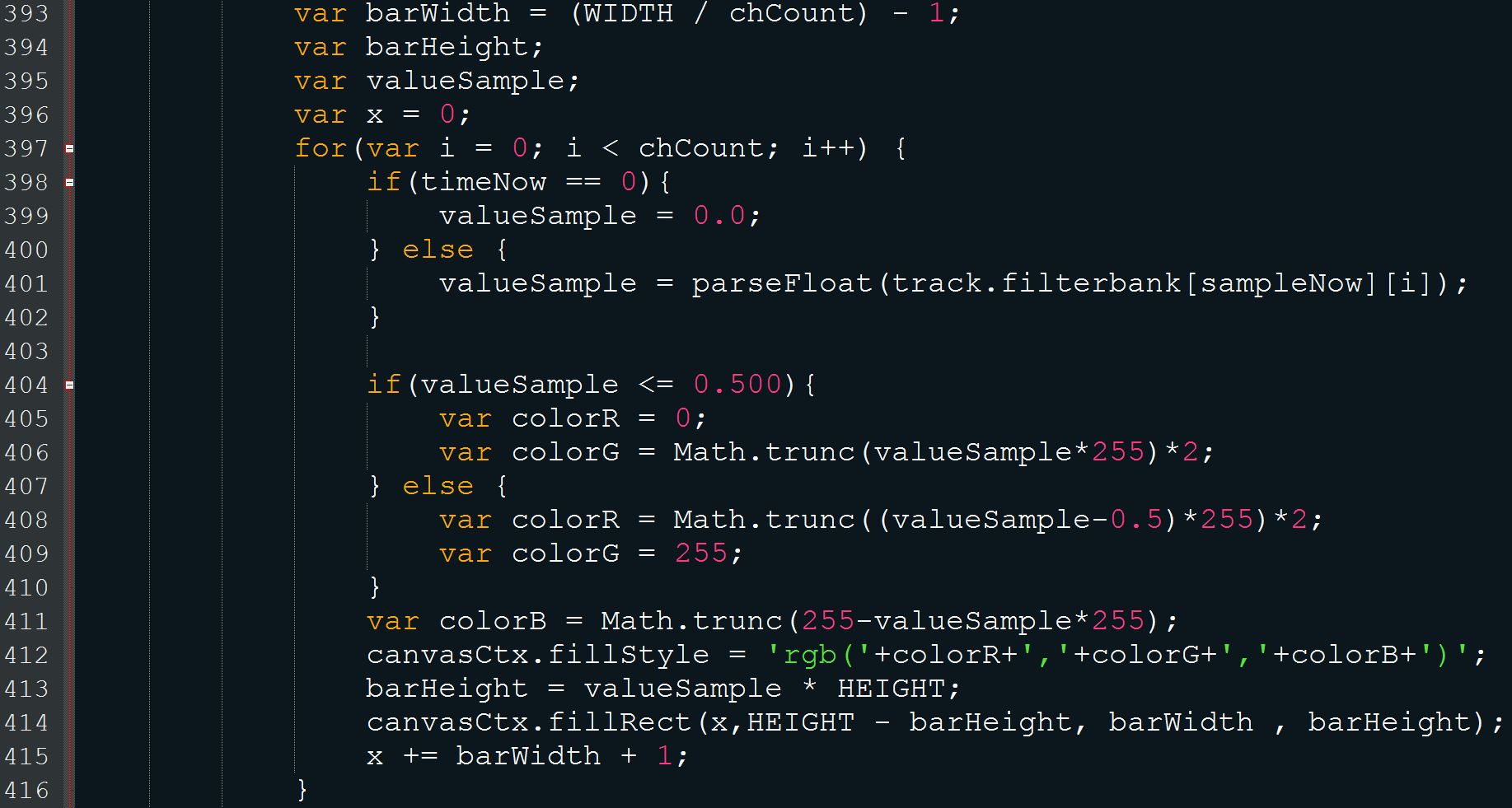


I rettangoli di “canvas” che vengono colorati sono tra xNext e xNow, non è stato implementato un modo per rappresentare i campioni prima di quando viene avviata la rappresentazione, per cui al cambio in tempo reale inizierà a rappresentare dal campione attuale.

# Rappresentazione “Istogramma Spettrale”



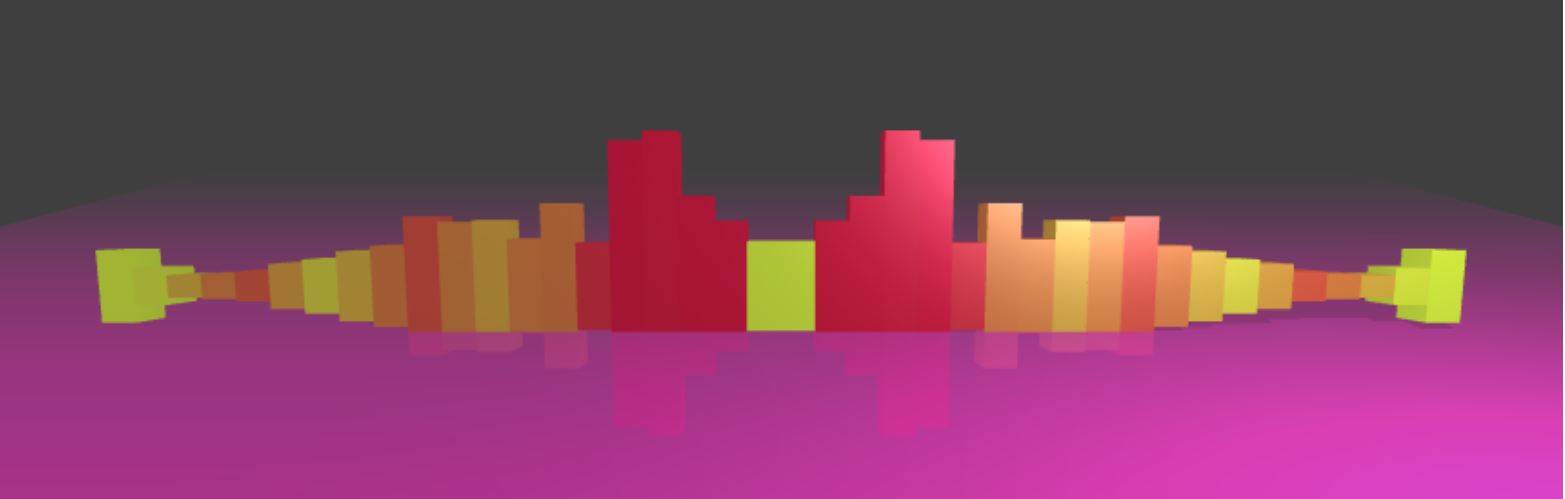
La terza rappresentazione richiesta segue il comportamento di un istogramma: rappresento ogni valore di ogni canale con un rettangolo, colorato a seconda dell’intensità. Anche in questo caso uso i valori estratti per la seconda rappresentazione, ovvero envelope dei filterbanks, e anche i colori sono coerenti con quelli usati nella rappresentazione precedente, blu per valori bassi, giallo per valori intensi.



La posizione e il numero di barre sono proporzionali al numero di canali di analisi e alla dimensione del “canvas”. In questo caso varia l’altezza del rettangolo in proporzione al campione attuale.

# Rappresentazione “Personalizzata”

Nella quarta rappresentazione avevo campo libero di creare un visualizzatore come meglio credevo e quindi ho scelto di provare a rappresentare in 3D alcune features caratteristiche. Ho scelto di rappresentare, innanzitutto dei dati che avevo già analizzato, come envelope e filterbank, più qualche caratteristica unica delle canzoni come il “pitch” e il “tempo”. Avrei voluto anche rappresentare i “peaks” e gli “onset” ma non ho trovato una rappresentazione che mi soddisfacesse e per lungo tempo mi sono anche dovuto scontrare con le limitazioni di JavaScript, come spiego successivamente.



Quindi, in questa rappresentazione abbiamo un piano colorato semi trasparente, di colore proporzionale al tono della canzone, il quale si muove come percorso dalla envelope della canzone: l’effetto è come quello di un telo che viene scosso e l’onda lo percorre tutto.   
Al centro della rappresentazione ci sono due set di parallelepipedi posizionati specularmente e di numero proporzionale ai canali del filterbank: questi seguono i valori dell’envelope dei filterbank per variare in altezza.  
Due luci puntiformi percorrono due orbite che si incrociano sopra ai parallelepipedi con velocità proporzionale al tempo della canzone.  
La rappresentazione è limitata visivamente in lontananza da un effetto nebbia che sfuma i bordi netti del piano inferiore.

# Problematiche di realizzazione

Lo sviluppo del progetto non è stato senza problemi e limiti dettati dal tipo di implementazione richiesta. Le difficoltà più grandi sono state create da un’incompatibilità, che mi ha portato via molto tempo nel tentativo di risolverla e ancora non risolta se non grazie a jQuery, e il fatto di usare un metodo “contro-corrente” per effettuare l’analisi audio rispetto a quello che è il “canone” in JavaScript e HTML5.

### Audio Wave Form vs Audio Envelope

Nella prima realizzazione, intendevo rappresentare la forma d’onda letteralmente come verrebbe rappresentata in un grafico ma ciò portava ad avere uno spropositato numero di campioni, una struttura dati pesantissima e una rappresentazione troppo veloce per la percezione visiva umana. La visualizzazione, specie nel grafico a barre e spettrale, risultava in uno sfarfallio troppo veloce per essere apprezzabile. Per questo ho scelto in seconda lettura l’Audio Envelope, rappresentando l’andamento generale, ha una curva più morbida e intuitiva, visivamente è gradevole e di facile lettura oltre ad essere effettivamente la feature che viene usata per rappresentare visivamente una traccia audio.

### JSON e i browser

Ho incontrato dei problemi dal punto di vista di interpretazione e caricamento da parte di alcuni browser Internet (in particolare Firefox) che non riuscivano a interpretare i file JSON correttamente attraverso il parser nativo, probabilmente a causa del gran numero di dati nel singolo file. Su internet, dove la maggior parte della gente usa jQuery per semplicità e comodità d’uso, è stato difficile trovare una soluzione alternativa o la causa del problema. Nonostante ciò, ho trovato varie documentazioni che parlano del problema del caricamento di file JSON di notevoli dimensioni in browser (consigliano tutti lo split ogni 40mega circa), ma benché i miei fossero di dimensioni minori il parser dava problemi comunque dichiarando che il file non fosse “ben formattato”. A scopo di ricerca della fonte del problema ho provato anche con file di minore dimensione e diversi elementi formattati JSON, ma tutti mi hanno dato lo stesso problema. Per questo, sotto il benestare del professore, ho usato il metodo jQuery parseJSON che, non so bene per qual motivo, ha funzionato a dovere con tutti i browser e i file JSON che gli ho “dato in pasto”, risolvendo il problema.

### JavaScript e la precisione

Il fatto di non poter avere un riferimento preciso riguardo al punto di esecuzione del brano è un problema grosso. Non solo porta ad andare “alla cieca” ma spesso può portare a falsi trigger di eventi: ho provato per un po’ a inserire gli onsets e i picchi nella realizzazione “personalizzata” ma non avendo una scansione temporale precisa abbastanza da trovare il campione in cui si realizzava il picco ho provato inserendo delle soglie intorno al campione corretto che, per poter essere visto da tutti i browser correttamente, rimanevano troppo larghe e portavano a dei falsi trigger di eventi.

Un altro problema legato alla precisione che ho riscontrato è stato nella gestione dei dati a virgola mobile: anche in questo caso a seconda del browser, l’interpreter di javascript mi ha creato problemi con l’approssimazione dei valori decimali dei numeri, spesso riempiendo di cifre inutili numeri arrotondati. Per risolvere questo problema ho limitato e troncato la precisione dei numeri a massimo 3 cifre significative dopo la virgola, a seconda del caso, sia in Matlab che nello script, dato che non sempre è bastato fornirgli dei dati precisi.

### JSON, JavaScript e i tipi di dato

La struttura dati JSON permette lo scambio semplificato di strutture dati complesse ma non la definizione di tipi di dato in modo univoco. Nel codice ho dovuto esplicitare dei cast di tipo di dato perché ogni tanto e a seconda del browser interpretava il dato come stringa invece che float, nonostante fosse sempre lo stesso dato con la stessa precisione: i metodi come parseFloat mi restituiscono il tipo di dato corretto.

### Asincronia e prestazioni

Il programma ideato si basa sul tempo corrente letto dal browser in esecuzione ma, a seconda di come viene implementato nel browser il riproduttore audio, questo può dare spazio a rappresentazioni poco coerenti o impulsive. Mi è capitato più di una volta che Firefox, in situazioni di alto carico generico del PC, non riuscisse a leggere bene consecutivamente il file audio, per cui la rappresentazione era scattosa, o altre volte in cui il browser dava priorità alla riproduzione dell’audio, rallentando la rappresentazione grafica. Quest’ultimo caso è evidente nel secondo metodo di rappresentazione se si formano dei rettangolini più grandi degli altri, dovuti al fatto che il programma javascript ha fatto la currentTime non regolarmente, ma asincronamente.

### CSS e impaginazione

L’interpretazione del CSS è da sempre stato uno dei problemi dei browser Internet: ognuno interpreta le dimensioni come preferisce e il web-designer deve sfruttare come può i sistemi per realizzare correttamente la visualizzazione richiesta. Nel mio caso la pagina web ha due “canvas”, uno per la rappresentazione 3D e uno per le rappresentazioni 2D, i quali sono sovrapposti ma a un valore di z-index differente, così da sovrapporsi precisamente. Purtroppo non tutti i browser gestiscono gli elementi in modo identico e ho dovuto inserire un accorgimento per far sì che fossero perfettamente sovrapposti sulla maggior parte dei browser.

# Conclusioni

Il sistema funziona, è personalizzabile e di facile implementazione. I limiti sono dettati dalle macchine su cui deve girare il programma e il browser che deve interpretarlo. Durante lo studio del progetto ho sentito più volte una sensazione di superato o di star sviluppando qualcosa che in realtà viene risolta in un modo più immediato e stabile: ricercando ho trovato web-audio. Le API ufficiali di JavaScript dedicate all’analisi e modulazione del suono in tempo reale su browser, sono un tool modulare che si interpone tra la sorgente audio e il diffusore, fanno l’analisi in tempo reale del campione in riproduzione dando accesso a dati di analisi già formattati per il web, in scale tra 0 e 255, pronti per essere utilizzati. Questo fa riflettere sull’obiettivo della rappresentazione in atto e su quale metodo utilizzare quando. Nel momento in cui voglia rappresentare per gusto estetico delle features audio, molto probabilmente propenderei per le API ufficiali, garantendo un’analisi in tempo reale anche di tracce dell’utente. Quando invece mi serve una rappresentazione precisa, il MirToolBox sembra essere la soluzione più comoda, ma la rappresentazione web sarà limitata, in risoluzione, in coerenza e dimensione dei dati accessibili.