Una tecnica basata sul Web per la renderizzazione di un volto parlante con coarticolazione facciale

Università degli Studi di Palermo

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

Tesina per l'insegnamento di Informatica Grafica

Titolare del corso: Prof. Roberto Pirrone

Tutor: Ing. Orazio Gambino

A cura di

Emanuele Cipolla (matricola 0578327)

Nicolò Monte (matricola 0584439)

A.A. 2012/2013

Sommario

1. Introduzione

Il presente lavoro affronta il problema della realizzazione di un volto tridimensionale, in grado di emulare in maniera fedele la coarticolazione facciale durante il parlato. Tale problema è stato affrontato mettendo a disposizione un'applicazione, nella fattispecie web-based, tramite la cui interazione l'utente può digitare del testo, che verrà pronunciato da una testa parlante.

Per rendere l'animazione più verosimile, è stata prevista la possibilità di tener conto dell'influsso di una particolare emozione nella deformazione del volto della testa parlante sia in maniera interattiva che programmatica, gestita tramite un file XML; quest'ultimo uso può consentire di sottoporre il testo di input ad un software esterno che provi a determinarne la componente emozionale.

Ciò che viene qui presentato ricalca quanto realizzato in [1], tuttavia non da intendere come una banale trasposizione in tecnologia WebGL di quanto citato, ma configurandosi come un profondo riadattamento della soluzione, secondo una chiave meno convenzionale e aperta a metodiche che sempre più si stanno affermando come standard *de facto* dell'informatica di largo consumo. Le principali motivazioni per cui si è scelto di proporre un'applicazione basata sul web sono due:

a) Oggi sempre più applicativi vengono proposti come fruibili dal web, per cui si è voluto seguire questo trend esaltandone i peculiari vantaggi (tra cui portabilità e compatibilità indipendente dalla macchina utilizzata)

b) Orma il web 2.0 impone standard grafici sempre più spinti e che molto probabilmente non potranno ancora per molto affidarsi a tecnologie farragginose come Flash. Per tali ragioni gli autori hanno ritenuto opportuno esplorare la moderna e ancora in fase di sviluppo, tecnologia WebGl, abbandonando così la libreria Java3D per quanto concerne la soluzione proposta.

2. WebGL

WebGL è una libreria grafica per grafica sia 3D che 2D che si basa su OpenGL ES 2.0. Il codice che utilizza la libreria può essere inglobato in pagine HTML oppure scritto su file separati: pertanto programmare un'applicazione in WebGL equivale a sviluppare una comunissima applicazione basata sul web, in cui il server manda gli elementi grafici e il client li processa grazie alla libreria e all'hardware a bordo della macchina ospitante il browser; in quanto applicazione web, si mantiene la possibilità di utilizzare le ulteriori tecnologie e librerie client-side disponibili al fine di arricchire l'esperienza utente. Poiché, come si accennava inizialmente WebGL è basato su OpenGL ES 2.0, il mercato ha già a diposizione utenti potenzialmente esperti in WebGL: il fatto di scrivere tutto sotto forma di applicazione web, rende il mantenimento delle applicazioni più agevole, in quanto non si deve tener conto dell'hardware su cui effettivamente sarà eseguita, e trasparente all'utente (non dovrà preoccuparsi di installare patch e quant'altro).

Infine WebGL è uno standard aperto, nel senso che chiunque può importare quanto rilasciato per utilizzarlo e modificarlo senza alcun costo aggiuntivo.

WebGL utilizza per la renderizzazionea l'elemento Canvas di HTML5, un'area rettangolare della tua pagina web in cui tu puoi disegnare usando JavaScript; l'interazione da parte dello sviluppatore avviene tramite accesso al DOM dello stesso HTML: questo è un grande vantaggio a favore dell'utente il quale potrà

2.1. Requisiti

Il consorzio Khronos, che si occupa dello sviluppo di WebGL, mantiene aggiornata una lista che soddisfa tutta la casistica dei browser, distinguendo a parità di questi i sistemi operativi su cui essi girano e le versioni meno recenti delle schede video che forniscono supporto. Di seguito una breve carrellata distinguendo la prima volta per browser e la seconda volta per hardware. Per informazioni più dettagliate si rimanda a http://www.khronos.org/webgl/wiki/BlacklistsAndWhitelists.

2.1.1 Supporto ai browser

In breve la situazione per i browser è la seguente:

- **Mozilla:** dalla versione 4.0 in poi sia per Windows che per Linux
- **Chrome:** supporta su qualsiasi S.O.
- **Safari:** come Chrome solo che di default è disabilitato
- **Opera:** come Safari e supporta solo dalla versione 11
- **Internet Explorer:** ancora non supporta WebGL in quanto non ritenuta una tecnologia sicura e uno standard dai tecnici della Microsoft; si può comunque usare WebGL emulando Chrome tramite il plugin denominato Google Chrome Frame

2.1.2 Supporto hardware

Per quanto riguarda i chip grafici la situazione è piuttosto rosea in quanto le GPU che risultano problematiche sono in una quantità abbastanza esile:

- Nvidia GeForce FX Go5200 (solo su Chrome ma su qualsiasi s.o.)
- Nvidia GeForce 7300GT (solo su Chrome su Mac OS X)
- ATI Radeon X1900 (Chrome su Mac OX)
- Tutte le GPU della ATI su Linux
- Tutte le GPU con chip Intel Mobile 945 Express

Per tutte le restanti GPU basta che siano presenti sulla macchina ospitante driver recenti con una buona tolleranza (dal 2009-2010 in poi a seconda del modello).

2.1.3 Supporto mobile

- **Safari:** supportato appieno
- **Browser Android:** supportato solo da dispositivi di alcune case
- **Firefox Mobile:** supportato appieno in dipendenza però dall'hw del dispositivo
- **Blackberry PlayBook 2.0 Browser:** supportato appieno

2.2. Renderizzare in una pagina HTML

Grazie alla quinta release del protocollo HTML e l'introduzione delle primitive WebGL, gli sviluppatori posso arricchire l'esperienza di navigazione, tramite interfacce che prima era pensabili solamente in ambito "desktop", riutilizzando le nozioni e, a meno di porting quasi immediati, il parco software che da più di un decennio costituisce l'esperienza grafica sui calcolatori.

Tuttavia, al fine di rendere più immediato lo sviluppo, e di raccogliere in maniera ordinata delle caratteristiche grafiche avanzate, come usualmente avviene nel mondo dello sviluppo software, anche nel contesto di WebGL, stanno via via affiorando librerie di alto livello al servizio degli sviluppatori.
Quello che verrà mostrato tra breve, è una descrizione a sommi capi, della librerie utilizzata in questo lavoro per riassumere da un punto di astrazione più alto, le primitive della libreria WebGL e le funzionalità fornite da Javascript, per definire il comportamento dei sistemi di puntamento.
L'utilizzo di librerie del genere è stato essenziale per affrontare una problematica così complessa come la coarticolazione facciale e l'integrazione delle emozioni: infatti Three.js, alla stregua di altre librerie comparabili, rende allo sviluppatore, sotto forma di "classi" Javascript, le funzionalità per renderizzare un modello, per gestire una scena e per scandire il tempo dell'animazione. Inoltre Three.js, come si vedrà, mette a disposizione una funzione per la interpolazione lineare tra i vertici di un modello geometrico e un altro.

2.2.1. Librerie di alto livello

2.2.1.1. Three.js

Three.js è una libreria JavaScript nata per semplificare la realizzazione di applicazioni basate sul Web che richiedono contributi 3D. Nasce nel 2009, in un contesto precedente a WebGL grazie al lavoro di Ricardo "mrdoob" Cabello, che decise di portare in JavaScript del codice già scritto per funzionare con Macromedia Flash.

Il supporto al rendering per WebGL è stato aggiunto nel 2011.

Alcune buone ragioni per scegliere Three.js piuttosto che WebGL a basso livello sono le seguenti:

1. Three.js astrae i dettagli della API WebGL, rappresentando la scena 3D in termini di mesh, materiali e luci.

2. Three.js contiene diversi oggetti già pronti utilizzabili per lo sviluppo di giochi, animazioni, presentazione, modelli ad alta risoluzione ed effetti speciali.

3. Three.js utilizza tutte le buone prassi della grafica 3D per mantenere alta la performance senza sacrificare l'usabilità.

4. WebGL non fornisce un supporto nativo al picking (ovvero non è possibile sapere quando il puntatore del mouse sovrasta un oggetto senza ricorrere a soluzioni esterne). Three.js invece possiede questa capacità.

5. Three.js possiede degli oggetti potenti e facili da usare per la matematica 3D, come matrici, proiezioni e vettori.

6. Si possono caricare file in formati testuali esportati da package di modellazione 3D, oltre che in formati JSON e binari specifici per Three.js

7. E' possibile estendere facilmente la libreria tramite plugin.

8. È possibile utilizzare un renderer HTML5 2D nel caso in cui si adoperi un browser su cui WebGL non è supportato.

Nel caso di questo lavoro si è utilizzata la più vetusta release r43 in quanto le versioni successive sono state oggetto di estese modifiche che rendono buona parte del materiale consultabile in rete (documentazione e codice di esempio) obsoleto.
E' possibile reperire uno *snapshot* del codice della libreria all'indirizzo https://github.com/mrdoob/three.js/.

2.2.1.2. Sim.js

Sim.js è un framework per la simulazione creato dal programmatore Tony Parisi per il suo libro "WebGL Up & Running" per semplificare alcuni compiti ripetitivi che restano da compiere nell'utilizzo di Three.js, per esempio la trafila creazione mesh-aggiunta all'albero degli oggetti della scena-impostazione delle texture-aggiunta di eventi DOM utente.

Il framework è diviso in tre classi principali:

1. *Sim.Publisher:* La classe base per ciascun oggetto che genera ("pubblica") eventi. Quando si verifica un evento, Sim.Publisher itera lungo la sua lista di callback registrati, chiamando ciascuno con I dati dell'evento e l'oggetto fornito ("subscriber"). Sim.Publisher è alla base di quasi tutti gli oggetti Sim.js.

2. *Sim.App:* Questa classe raccoglie tutto il codice necessario a predisporre Three.js per una pagina, per esempio la creazione del renderer, l'oggetto top-level della scena e la telecamera. Si occupa anche di aggiungere gli handler DOM al canvas utilizzato per il rendering da Three.js per gestire il ridimensionamento, l'input del mouse e altri eventi. Sim.App gestisce anche la lista di singoli oggetti dell'applicazioni ed implementa il loop di esecuzione.

3. *Sim.Object:* La classe base per la maggior parte degli oggetti di un applicazione: gestisce lo stato di un oggetto e si occupa di alcune operazioni di base Three.js, tra cui l'aggiunta/eliminazione di un oggetto dalla scena e dei figli dalla gerarchia degli oggetti, ed il recupero/impostazione di proprietà standard Three.js come la posizione, la scala e la rotazione.

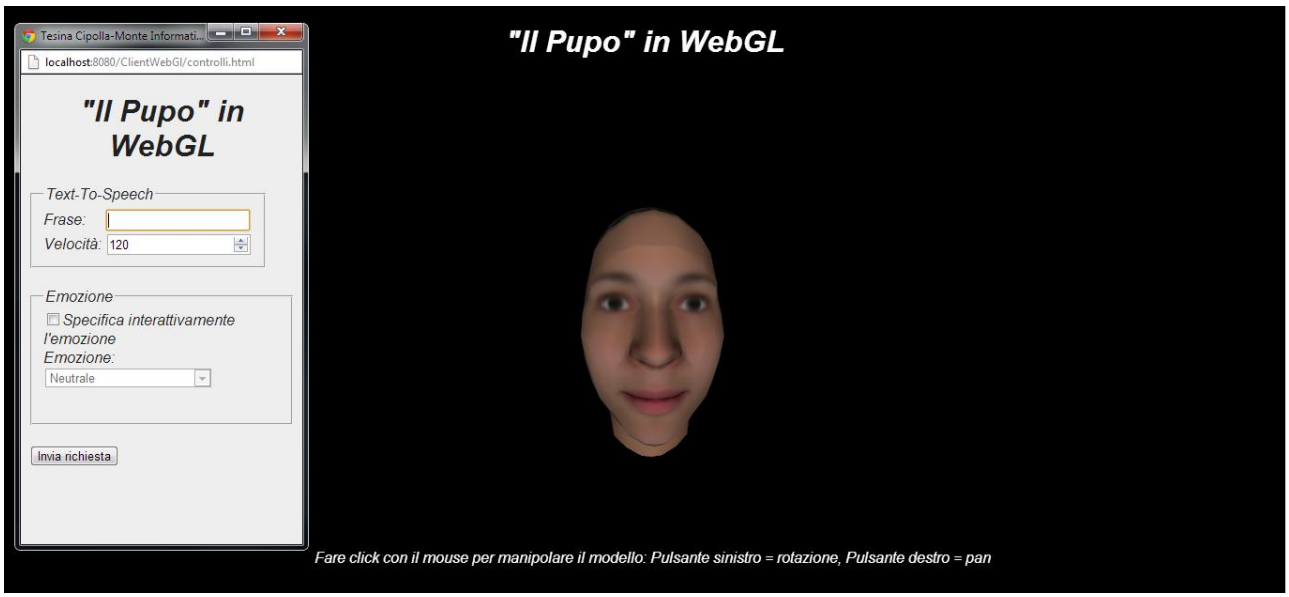
3. Il processo

Quando is apre il browser c'è la possibilità di scegliere tra due modalità di operazione.

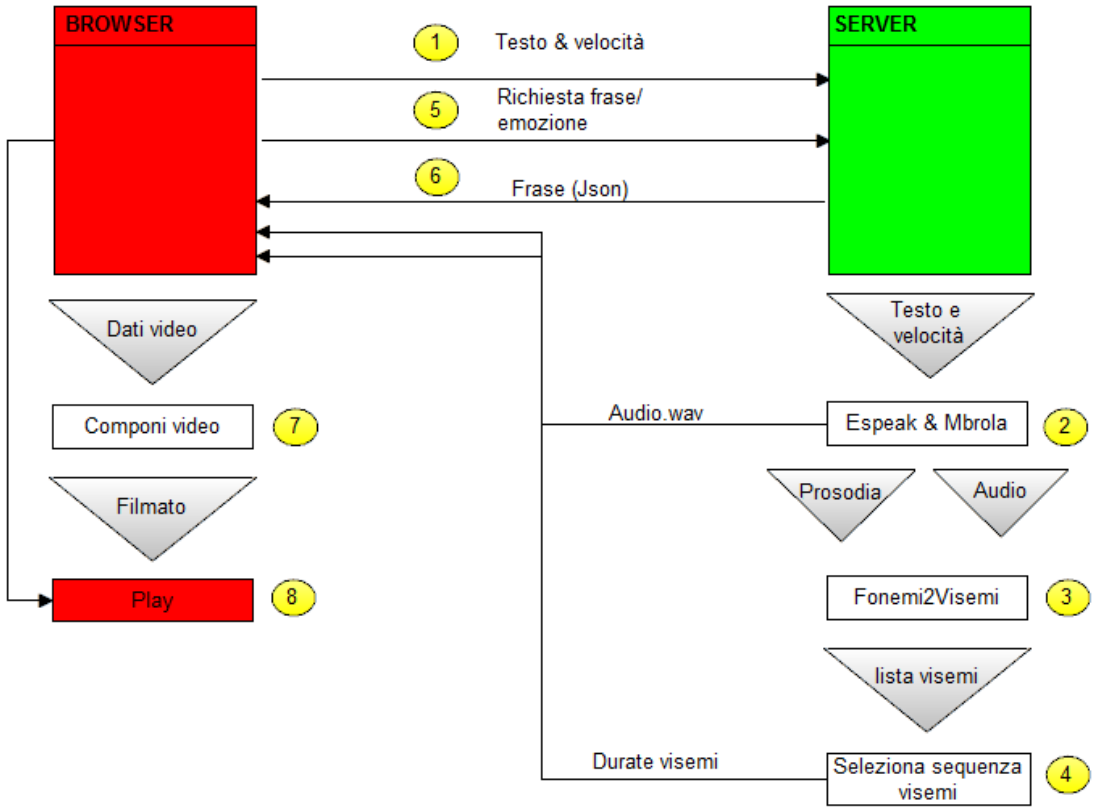
3.1. Modalità interazione con l'utente

In questo capitolo si vuole fornire una visione di alto livello del processo di esecuzione che porta al risultato finale dell'intera animazione.

L'applicazione è stata realizzata seguendo il paradigma client-server: non appena l'utente effettua l'accesso all'applicazione tramite il proprio browser, visualizzerà una schermata come da figura in basso: in essa viene visualizzato il volto in una posa neutrale, detta silenzio, ed in una finestra separata due campi in cui digitare il testo che si desidera sintetizzare e la velocità di pronuncia (e quindi anche dell'animazione), e altri due che permettono opzionalmente la scelta interattiva dell'emozione e la percentuale di influenza.

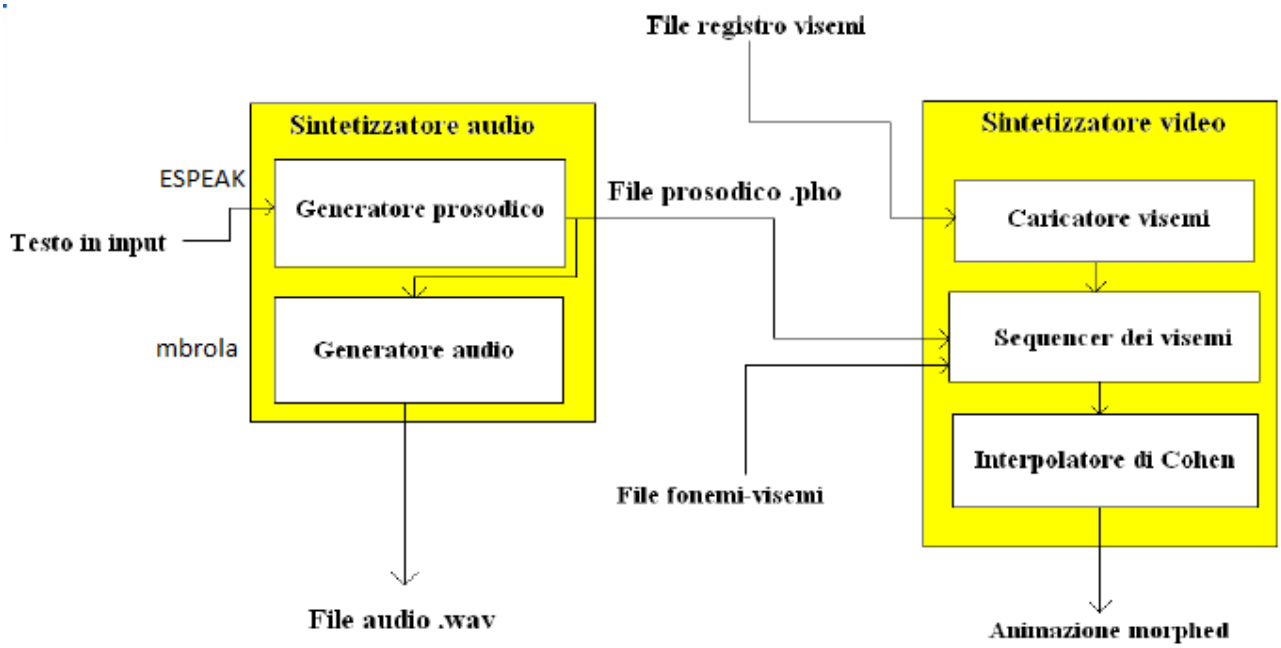


Da questo momento in poi si focalizzerà l'attenzione su ciò che avviene, in totale trasparenza all'utente, non appena si invia la richiesta di sintetizzazione: a tale scopo può venire in aiuto lo schema in basso che mostra le fasi salienti del processo:



Il processo consiste in un'interazione tra Client e Server per lo scambio dei dati testo, velocità, audio e video e di fasi di elaborazioni individuali, quest'ultime contrassegnate dai box rettangolari.
Non appena l'utente conferma la richiesta di sintetizzazione, il browser, cioè il Client, manda al server testo, velocità, ed opzionalmente valore percentuale e indicazione dell'emozione, che vengono ricevuti mediante una servlet che rimane in ascolto.
A questo punto il server estrae le informazioni e le converte nel formato apposito, per darle in input ai software Espeak, che genera la prosodia relativa al testo, ed Mbrola, che invocato successivamente genera il file audio.
Il server comincia a individuare quindi il set di visemi a partire dai fonemi contenuti nella prosodia: il server contiene infatti una copia di tutti i visemi che potenzialmente possono occorrere per un testo, quindi prepara una lista contenente nella giusta sequenza, i visemi relativi al testo digitato dall'utente, oltre ad un vettore con le corrispondente durate in millesimi di secondo: a partire da questa lista e dalla mesh relativa al volto in "silenzio" viene costruita l'apposito file di tipo "json" contenente l'insieme dei vertici delle mesh che occorrono nel testo e i vertici relativi all'emozione eventualmente scelta dall'utente; il client recupera queste informazioni tramite apposita richiesta.
Il Client dal canto suo, riceve il vettore delle durate e richiede il file "json" appositamente generato; grazie alle funzionalità di alto livello della libreria Three.js il client è in grado di importare i dati grafici e di elaborare l'animazione desiderata: non appena quest'ultima sarà pronta verrà eseguita in maniera sincronizzata all'audio ad essa associato (quest'ultimo gestito tramite le funzionalità di HTML 5).

Per completezza, si suggerisce un confronto con l'architettura del progetto [1], proponendo l'immagine sottostante, con l'immagine precedentemente mostrata.



3.2. Modalità file emozionale servito dal server

Come si è accennato in precedenza, la specifica di un'emozione in maniera interattiva mediante la finestra del browser è opzionale: infatti, per garantire la possibilità di integrazione con software esterni di terze parti, per impostazione predefinita l'emozione viene letta da un apposito file XML posto nella directory radice del server, denominato `emozioni.xml`.

```
<?xml version="1.0" encoding="UTF-8"?>

<emozione>

      <tipo>*h_su</tipo>

      <percentuale>80</percentuale>

</emozione>
```

Il file XML contiene all'interno del tag emozione:

- Il tag percentuale, che racchiude un valore compreso tra 0 e 100:
- Il tag tipo, che racchiude una stringa identificativa dell'emozione.

La servlet RiceviRichiestaUtente invocherà un'apposita funzione che costruirà una stringa identificativa del visema emozionale richiesto concatenando i valori contenuti all'interno dei tag tipo ed emozioni, ed effettuando il mapping al vero file contenente il visema emozionale, seguendo il contenuto del file

`fon2vis.txt` come se si trattasse dell'associazione fonemi-visemi. Il file sarà caricato ed i vertici in esso contenuti saranno aggiunti come ultimo morphing target a `frase.js`.

4. Implementazione

4.1. Il server

Il codice sorgente per le funzioni del server è articolato nei seguenti file:

4.1.1. Fonemi2VisemiIta.java

Contiene i metodi:

- leggiVisemi, che popola la tabella fonemi-visemi con il contenuto del file fon2vis.txt e la lista dei visemi disponibili con il contenuto del file nomi_visemi.txt;
- converti, che dato un fonema restituisce il corrispondente visema;
- getvisIndex, che restituisce l'indice relativo al determinato fonema nella tabella dei visemi

4.1.2. JSONWriter.java

Contiene l'implementazione di uno stream writer per oggetti String che inserisce opportunamente ritorni a capo, spaziature e tabulazioni. Utilizzato per il debug delle modifiche effettuate mediante la libreria json-simple al modello geometrico JSON del volto.

4.1.3. MandaJSON.java

Contiene l'implementazione di una servlet in grado di rendere disponibile un file JSON generato server-side senza rivelarne l'effettivo posizionamento nel file system: attualmente viene utilizzata per servire il file contenente le durate dei singoli visemi.

4.1.4. MandaWAV.java

Servlet analoga alla precedente che si occupa di rendere disponibile il file audio.

4.1.5. RiceviRichiestaUtente.java

La servlet basilare dell'intero server, si occupa di ricevere le informazioni riguardanti le parole da sintetizzare e la velocità del parlato, e demanda il compito dell'effettiva creazione delle risorse richieste ad un oggetto di tipo Sintetizzatore. Inoltre, crea il file JSON delle durate dei visemi e prepara il modello JSON del silenzio aumentato con le configurazioni dei vertici necessarie al morphing per la resa visuale delle parole richieste.

4.1.6. Sintetizzatore.java

La classe Sintetizzatore si occupa di invocare eSpeak ed MBROLA al momento opportuno per la produzione dei file prosodiaco ed audio. Inoltre, il file prosodiaco viene processato in modo tale che sia possibile in seguito effettuare l'associazione fonemi-visemi per il prosieguo dell'elaborazione.

4.1.7. TextPatcher.java

La classe TextPatcher si occupa di effettuare alcune semplici sostituzioni al testo in input a eSpeak per evitare situazioni di mancato funzionamento dovute a bug non risolti.

4.1.8. WavAppender.java

La classe WavAppender concatena un silenzio di durata prefissata all'audio generato da MBROLA: questo è stato necessario per compensare il lag di caricamento del modello JSON sul client, in quanto non è possibile sincronizzare pienamente la riproduzione dell'audio e il rendering del video.

4.2. Il client

Il client si compone di 6 file (escluse le librerie esterne):

4.2.1. controlli.html

Il file contiene il form che permette all'utente di inserire la frase, la velocità di sintetizzazione ed eventuale l'emozione con l'annessa percentuale.

Poiché questa pagina viene richiamata mediante apertura di finestra popup da parte di index.html al suo caricamento, possiede intrinsecamente la capacità di influenzare il comportamento della pagina chiamante, ed infatti, sebbene sia in questa pagina che vengono effettivamente compiute le richieste asincrone al server, la renderizzazione dell'animazione relativa alla frase viene comunque effettuata all'interno della finestra in cui è stato caricato index.html.

4.2.2. index.html

Il file contiene l'area in cui effettivamente verrà caricato il modello. Il "collante" tra tutti questi elementi è costituito dal codice JavaScript. In particolare, l'inizializzazione del modello e dell'area utile per il rendering viene effettuato utilizzando il metodo cross-browser di jQuery $(document).ready().

Vengono creati, nell'ordine:

- un oggetto canvas utilizzato per WebGL;
- un oggetto di tipo Sim.App;
- un oggetto di tipo Sim.Object che rappresenta il modello JSON da caricare, opportunamente espanso con le caratteristiche necessarie ai nostri scopi (ad esempio l'audio).

Tutti questi oggetti entrano a far parte dell'albero DOM che rappresenta index.html, in maniera non dissimile da quella di una qualunque applicazione web.

Viene quindi richiamato il metodo init() dell'oggetto Sim.Object che effettua il download del modello preliminare (in posa neutrale). A seguito dell'inizializzazione l'oggetto che rappresenta la mesh viene aggiunto come foglia DOM dell'oggeto che rappresenta l'applicazione (Sim.App).

La chiamata al metodo run() avvia il loop di esecuzione del rendering.

4.2.3. clientWebGl.css

Foglio di stile utilizzato da index.html.

4.2.4. clientWebGl.js

Il file contiene diverse funzioni JavaScript utilizzate per la comunicazione con il server e l'inizializzazione del processo di rendering. Si fa esteso uso della comunicazione asincrona AJAX.

La funzione validaParametri(frase, velocita) verifica che siano stati inseriti valori appropriati per quel che riguarda la frase da sintetizzare e la velocità di sintetizzazione

La funzione inviaRichiesta, se la validazione è andata a buon fine, contatta la servlet RiceviRichiestaUtente in maniera asincrona per trasmetterle i dati del form: non appena la ricezione viene confermata e la servlet comunica l'avvenuta produzione di quanto richiesto, viene chiamata la funzione ottieni_durate.

La funzione ottieni_durate() richiede, sempre in maniera asincrona, al server le durate dei singoli visemi, e le immagazzina in una struttura dati JavaScript per utilizzo successivo nel corso del loop di rendering. A comunicazione completata viene richiamata la funzione aggiorna_render.

La funzione aggiorna_render() richiede il modello completo JSON prodotto come già spiegato in precedenti paragrafi e ne avvia il rendering all'interno dell'area canvas predisposta in index.html.

4.2.5. pupoApp.js

Il file contiene l'oggetto JavaScript pupoApp, che estende Sim.App per le esigenze della nostra applicazione.

Il metodo init predispone telecamera (con relativo controllo tramite mouse/trackball, facendo uso della funzione ausiliaria createCameraControls) e luci ambientali per la scena.

Il metodo update aggiorna la posizione della telecamera e le luci in ogni iterazione del loop di esecuzione dell'applicazione.

Vengono inoltre impostate alcune costanti di uso comune per la telecamera.

4.2.6.. pupo.js

Il file contiene l'oggetto JavaScript pupo, che estende Sim.Object svolgendo alcuni compiti essenziali per la corretta animazione del volto.

Il metodo init preleva il modello mediante il loader JSON, memorizza i parametri ricevuti in ingresso per gli scopi dell'animazione e aggiunge l'oggetto JavaScript corrispondente al modello al DOM. Contestualmente viene creato un riferimento ad una time source interna di Three.js che viene utilizzata nel corso dell'animazione:

```
    this.clock = new THREE.Clock();
```

Il metodo handleLoaded viene richiamato, alla conclusione del prelievo del modello, ma prima della conclusione del metodo init, mediante apposito callback definito dal loader. handleLoaded non opera

direttamente sul modello, ma sul corrispondente oggetto THREE.Geometry. Viene inizializzata la mesh, viene calcolata la durata totale dell'animazione ed assegnata una funzione di animazione personalizzata per i nostri scopi, mesh.updateAnimation.

La funzione mesh.updateAnimation viene richiamata ricevendo in input il numero di secondi trascorsi da quando è avvenuta l'ultima iterazione del loop di rendering. Viene determinato l'istante corrente di riproduzione del frame corrente (frameTime): tale valore, inizialmente determinato a partire dalla durata totale del filmato, viene via via aggiornato tenendo conto del fatto che ogni visema ha una durata differente.

```
            var frameTime = this.duration / ( this.geometry.morphTargets.length
- 1 ); // Aggiornamento non basato sulle durate dei visemi (updateAnimation
viene chiamato di continuo, anche quando non riproduciamo)

            if ( this.durate_visemi )
                 frameTime = this.durate_visemi[this.indice_visema_corrente];

            this.time += delta;
```

Per prima cosa si verifica se il filmato è stato riprodotto per intero: in tal caso, si effettua una transizione smooth verso il visema del silenzio:

```
// Alla fine dell'animazione fermati

                    if ( this.time > this.duration) {
                    this.time = this.duration;

                    // WORKAROUND: in teoria basta impostare this.time a
this.duration per
                    // interrompere l'animazione, ma con alcune vocali lunghe
                    // ci sono problemi a riportare la mesh alla configurazione
originaria:

                    // Esempio: la parola "casa" pronunciata "c aaa saaa "
produce un artefatto
                    // alla fine della riproduzione

                    this.morphTargetInfluences[this.morphTargetInfluences.length
- 3]=0;

                    // tolgo dolcemente l'emozione
                    var decremento=0.020;
                    this.morphTargetInfluences[this.morphTargetInfluences.length
- 1]=Math.max(this.morphTargetInfluences[this.morphTargetInfluences.length - 1]-
decremento,0);
                    return;
                }
                if ( this.time < 0 ) this.time = 0;
```

L'animazione vera e propria viene effettuata interpolando linearmente i valori di influenza dei visemi per il morphing precedentemente allegati a frase.js, così da effettuare delle transizioni smooth:

```
var keyframe = THREE.Math.clamp( Math.floor( this.time / frameTime ), 0,
this.geometry.morphTargets.length - 1 );

            if ( keyframe != this.currentKeyframe ) { // attendi la transizione
di frame brusca appena l'if è soddisfatto
```

```
                this.morphTargetInfluences[ this.lastKeyframe ] = 0;
                this.morphTargetInfluences[ this.currentKeyframe ] = 1;

                this.morphTargetInfluences[ keyframe ] = 0;

                this.lastKeyframe = this.currentKeyframe;
                this.currentKeyframe = keyframe;

                if (this.durate_visemi && this.indice_visema_corrente !=
this.durate_visemi.length)
                    this.indice_visema_corrente++;
            }

            // emozione attiva
            this.morphTargetInfluences[this.morphTargetInfluences.length - 1]=1;

            var mix = ( this.time % frameTime ) / frameTime;

                this.morphTargetInfluences[ this.currentKeyframe ] = mix;
                this.morphTargetInfluences[ this.lastKeyframe ] = 1 - mix;
        };
```

Poiché è necessario anche riprodurre l'audio, nella stessa funzione viene aggiunto al DOM un oggetto Audio HTML 5 e precaricato contestualmente il file presente sul server.

Il metodo update essenzialmente richiama il corrispondente metodo di Sim.Object che disegna un nuovo frame, dopo aver verificato che il modello sia stato effettivamente caricato ed eventualmente avviando la riproduzione dell'audio.

5. Risultati e sbocchi futuri

Si ritiene che sia stata raggiunta la piena replica delle funzionalità del lavoro di partenza: la coarticolazione delle parole risulta piuttosto fluida e veritiera. La testa parlante in quanto tale viene renderizzata in maniera differente: sono state riscontrate delle proporzioni lievemente diverse. Inoltre, la differente posizione della fonte luminosa sembra attribuire tonalità diverse alla pelle – la libreria Three.js fornisce comunque ampia possibilità di intervenire in questo senso.

A scopo di test è stato richiesto al sistema di generare l'animazione e l'audio necessario a rendere le seguenti frasi. Inoltre, abbiamo confrontato l'efficacia della resa delle stesse sull'applicazione [1] e su questa.

-il cane guaisce: sembra funzionare bene su entrambe le applicazioni

-i guai non mancano mai: sembra funzionare bene su entrambe le applicazioni

-l'aiuola è sempre verde: in questa applicazione riesce ad articolare ma senza audio; in [1] si rileva un crash.

-il pizzaiuolo fa le pizze: in questa applicazione riesce ad articolare ma senza audio; in [1] si rileva un crash.

-lo stuoino per fare ginnastica: sembra funzionare bene su entrambe le applicazioni

-gioia infinita ed allegria abbondante: in questa applicazione riesce ad articolare ma senza audio; in [1] si rileva un crash.

Si rileva, infine, come all'avvio dell'applicazione sia già reso un modello della testa in posizione neutrale, in silenzio, rispetto allo schermo nero vuoto in [1].

Presentiamo una lista di possibili sviluppi futuri di questo lavoro, elencati senza alcuna pretesa di completezza o di maggiore importanza dell'uno rispetto all'altro.

- **Supporto a lingue diverse dall'italiano**: Il sistema prevede la possibilità di invocare eSpeak ed MBROLA con diverse voci e sistemi di generazione della prosodia semplicemente modificando la relativa riga di comando nella classe Sintetizzatore del server.

- **Chatbot**: Si potrebbe interfacciare il sistema utilizzato con un chatbot engine per fornire una risposta più verosimile alle interrogazioni dell'utente: si potrebbe utilizzare il software ProgramD in abbinamento ad un'opportuna base di conoscenza che permetta, data una stringa di input ricevuta dall'utente, la generazione di una risposta opportuna che sarà la frase effettivamente animata dalla servlet R*iceviRichiestaUtente*.

- **Inferenza delle emozioni in base al testo**: Si potrebbe sfruttare una o più tecniche di intelligenza artificiale per riconoscere il contenuto emozionale di un testo e verificare l'avvenuto riconoscimento mediante degli appositi visemi.

- **Sintetizzatore audio alternativo**: Il sistema è stato progettato in modo tale che sia possibile sostituire, agendo sulla classe Sintetizzatore del server, MBROLA con un altro software di pari funzione.

6. Appendice

Si fornisce prima di tutto una carrellata riguardante i software coinvolti, chiarendone le differenti modalità di utilizzo tra i sistemi Linux e quelli Microsoft Windows. Verrà presentato inoltre, materiale utile a chi intenda realizzare un'applicazione in WebGL utilizzando i software di base sfruttati per questo lavoro. Infine verranno delineati gli step necessari per eseguire il lavoro [1] utilizzato come riferimento, in quanto sono stati rilevati alcuni problemi che devono essere preventivamente risolti.

6.1. eSpeak

eSpeak è un sintetizzatore vocale open source multipiattaforma che utilizza un metodo basato sui formanti, che non utilizza campioni della voce umana ma ricrea la voce per elaborazione basandosi su un modello acustico generando forme d'onda di cui si modulano alcuni parametri acustici come la frequenza fondamentale, i toni e i livelli di rumore. Derivato da un analogo software per ACORN Risc OS, è stato notevolmente espanso negli ultimi anni: nel caso della lingua italiana, di notevole importanza è stata il contributo dell'Ing. Piero Cosi, ricercatore all'Istituto di Fonetica e Dialettologia del C.N.R. a Padova.

eSpeak può essere utilizzato come applicazione a se stante (con o senza interfaccia grafica) o come libreria.

Se utilizziamo eSpeak da interfaccia grafica, si utilizza contemporaneamente Mbrola generando così il file .wav, contenente il parlato del testo digitato; se lo si utilizza, come nel nostro caso, come applicazione da riga di comando, l'invocazione di Mbrola deve essere effettuata manualmente in un secondo tempo.

Pertanto l'utilizzo di Espeak sarà finalizzato alla sola generazione delle informazioni prosodiache tramite una riga di comando del tipo:

```
espeak -v mb-it3 --phonout=prova.pho "ciao mondo"
```

Il file .pho, contenente la prosodia, avrà nome prefissato e sarà utilizzato con Mbrola.

6.1.1. Windows

Sul sistema operativo Microsoft Windows 7 purtroppo non funziona l'ultima versione di Espeak da riga di comando: si consiglia di scaricare quella utilizzata nel lavoro [1] cioè eSpeak 1.40.

6.1.2 Linux

Su sistemi Linux invece non è stato riscontrato alcun problema con la versione più aggiornata (Espeak 1.46)

6.2. Mbrola

Mbrola è un software di sintesi vocale, distribuito gratuitamente ma non open source.

Tale programma per funzionare ha bisogno di un database contenente i difoni per la sintesi vocale contenente i difoni (messo a disposizione sul sito del software stesso) e di un file contenente le informazioni prosodiache (.pho).

Il software MBROLA non è un sistema *text-to-speech* completo (generazione del parlato di sintesi a partire da un testo); MBROLA infatti fornisce principalmente i database dei fonemi e dei difoni specifici per una determinata lingua, ma il testo da sintetizzare deve essere già convertito in precedenza in fonemi e in informazioni prosodiche nel formato richiesto dall'algoritmo. Quest'ultima caratteristica richiede a monte, l'utilizzo di un programma, tra cui Espeak, in grado di generare informazioni prosodiache a partire dal testo.

Come accennato al paragrafo precedente, si utilizza Mbrola da riga di comando, per generare il file wav.

6.3. Cygwin

Per utilizzare al meglio Mbrola, è stato necessario sfruttare la versione da riga di comando: per gli utenti Linux tale versione è supportata nativamente, mentre per quelli Windows si ha necessità della libreria cygwin1.dll, copiata nella cartella in cui si trova Mbrola.

6.4. Facegen

Facegen è un middleware per la generazione di volti tridimensionali prodotto dalla Singular Inversions che utilizza un approccio parametrizzato per definire le proprietà di un volto. Mediante un insieme prefissato di parametri è in grado di effettuare il morphing e di modificare un volto indipendentemente dalla risoluzione dell'output. FaceGen permette all'utente di delinerare volti in maniera casuale, interpolarli e normalizzarli, oltre a poterne esagerare alcuni dettagli, ed include degli algoritmi per impostarne l'età apparente, l'etnia di appartenenza ed il sesso: è anche possibile effettuare un controllo limitato delle espressioni facciali. È disponibile anche un set di espressioni fonetiche per l'animazione di personaggi che "parlano".

FaceGen può generare modelli 3D da fotografie frontali e laterali, oppure analizzando una singola fotografia.

I modelli generati possono essere esportati nel formato di interscambio Wavefront .OBJ: per ogni volto viene genrrata una coppia di file: i file MTL sono relativi contengono informazioni su come la luce interagirà con la tessitura, quest'ultima sotto forma di file JPEG. I file OBJ veri e propri contengono le coordinate dei vertici e altre informazioni di base della mesh.

6.5. Server

Un piccolo server si occupa di ricevere i dati necessari alla sintesi vocale da parte del browser e di predispore il modello da renderizzare, l'audio da riprodurre ed alcune informazioni aggiuntive, come la durata dei visemi.

Le informazioni strettamente riguardanti il modello da renderizzare vengono trasmesse in formato JSON, che offre la possibilità di parsing immediato in corrispondenti oggetti JavaScript. In particolare, è stata utilizzata

la libreria Java json-simple che permette una traduzione immediata degli oggetti Java in corrispondenti JSON.

L'audio viene invece reso semplicemente disponibile come file pronto al download da parte del server.

6.5.1 Creazione delle servlet

Sono state create tre servlet:

- **RiceviRichiestaUtente:** si occupa di ricevere il modulo inviato tramite metodo POST, contenente la frase da pronunciare e la velocità di pronuncia in termini di parole al minuto. Una volta ricevute entrambe le informazioni, invoca la classe **Sintetizzatore** per far produrre l'audio associato al testo e la relativa prosodia.
  Restituisce "OK" in caso di avvenuta generazione dei file .pho e .wav, i messaggi di errore relativi ad Espeak e/o Mbrola in caso contrario.
  Dopo ciò la servlet si occupa di scrivere su due file JSON distinti:
  a) le durate di ogni singolo visema;
  b) un modello JSON della testa a partire dal visema associato al silenzio opportunamente modificato con tante configurazioni di vertici target per ogni transizione da un visema all'altro quante ne servono per pronunciare la parola.

A monte tutti i possibili visemi, che si presentavano sotto forma di file *.obj*, sono stati convertiti in oggetti JSON tramite lo script python, denominato "*convert_obj_three.py*", facente parte della libreria Three.js.

Un esempio del modello JSON generato al punto b è il seguente:


```
{

    "metadata" :
    {
        "formatVersion" : 3,
        "sourceFile"    : "silence.obj",
        "generatedBy"   : "OBJConverter",
        "vertices"      : 2161,
        "faces"         : 2536,
        "normals"       : 0,
        "colors"        : 0,
        "uvs"           : 3829,
        "materials"     : 4
    },

    "scale" : 1.000000,

    "materials": [  {
    "DbgColor" : 15658734,
    "DbgIndex" : 0,
    "DbgName" : "Texture0",
    "colorAmbient" : [0.0, 0.0, 0.0],
    "colorDiffuse" : [0.69999999999999996, 0.69999999999999996,
0.69999999999999996],
    "colorSpecular" : [0.0, 0.0, 0.0],
    "illumination" : 2,
    "mapDiffuse" : "silence_HairBase.jpg",
    "specularCoef" : 20.0,
    "transparency" : 1.0
    },
```

```
    {
    "DbgColor" : 15597568,
    "DbgIndex" : 1,
    "DbgName" : "Texture1",
    "colorAmbient" : [0.0, 0.0, 0.0],
    "colorDiffuse" : [0.69999999999999996, 0.69999999999999996,
0.69999999999999996],
    "colorSpecular" : [0.0, 0.0, 0.0],
    "illumination" : 2,
    "mapDiffuse" : "silence_c1000Face.jpg",
    "specularCoef" : 20.0,
    "transparency" : 1.0
    },

    {
    "DbgColor" : 60928,
    "DbgIndex" : 2,
    "DbgName" : "Texture2",
    "colorAmbient" : [0.0, 0.0, 0.0],
    "colorDiffuse" : [0.69999999999999996, 0.69999999999999996,
0.69999999999999996],
    "colorSpecular" : [0.0, 0.0, 0.0],
    "illumination" : 2,
    "mapDiffuse" : "silence_c1000Head.jpg",
    "specularCoef" : 20.0,
    "transparency" : 1.0
    },

    {
    "DbgColor" : 238,
    "DbgIndex" : 3,
    "DbgName" : "Texture3",
    "colorAmbient" : [0.0, 0.0, 0.0],
    "colorDiffuse" : [0.69999999999999996, 0.69999999999999996,
0.69999999999999996],
    "colorSpecular" : [0.0, 0.0, 0.0],
    "illumination" : 2,
    "mapDiffuse" : "silence_c1000TTS.jpg",
    "specularCoef" : 20.0,
    "transparency" : 1.0
    }],

    "vertices":
[58.746600,66.791900,13.804800,64.189900,54.583600,10.850000,57.195400,78.700900
,9.563430,52.968200,88.094600,4.089560,47.307300,95.499500,4.276750,26.178900,10
1.018000,18.921700,82.033400,-19.205800,-57.626100,82.785600,-12.063400,-
59.040800,81.830000,5.752150,-58.145000,82.092400,11.425500 ...],

    "morphTargets": [],

    "morphColors": [],

    "normals": [],

    "colors": [],

    "uvs":
[[0.72456,0.17654,0.53249,0.21786,0.07116,0.56177,0.42079,0.86203,0.07116,0.5617
7,0.42079,0.86203,0.23924,0.06812,0.49129,0.05362,0.78839,0.50131,0.78839,0.5013
1,0.49129,0.05362,0.78839,0.50131,0.45772,0.68519,0.42079,0.86203,0.35025,0.8422
,0.45772,0.68519,0.03745,0.25299,0.59626,0.91161,0.59626,0.91161,0.03745,0.25299
```

```
,0.03745,0.25299,0.59626,0.91161,0.59626,0.91161,0.03745,0.25299,0.72462,0.10513
,0.20887,0.31698,0.41557,0.24265,0.20887,0.31698,0.41557,0.24265,0.25593,0.00211
,0.26048,0.2757,0.41557,0.24265,0.94181,0.49785,0.49322,0.67486,0.49322,0.67486,
0.94181,0.49785,0.3731,0.02787,0.76041,0.34352,0.3731,0.02787,0.76041,0.34352,0.
27983,0.73667,0.60985,0.20133,0.16983,0.6287,0.16983,0.6287,0.60985,0.20133,0.23
924,0.06812,0.22482,0.19194,0.26048,0.2757,0.45457,0.23439,0.20067,0.37955,0.454
57,0.23439,0.15676,0.75171,0.68651,0.18481,0.68651,0.18481,0.41248,0.03648,0.769
69,0.39609,0.41248,0.03648,0.70132,0.94136,0.456,0.87195,0.45772,0.68519,0.27989
,0.82237,0.18466,0.50458,0.53249,0.21786,0.18466,0.50458,0.53249,0.21786,0.25593
,0.00211,
    ... ]],

    "faces":
[11,141,140,143,142,0,1605,1606,1603,1604,11,140,144,145,143,0,1606,1608,1607,16
03,11,147,146,144,140,0,1609,1610,1608,1606,11,148,147,140,141,0,1611,1609,1606,
1605,11,150,149,152,151,0,1614,1615,1612,1613,11,151,152,154,153,0,1613,1612,161
6,1617,11,152,155,156,154,0,1612,1619,1618,1616,11,149,157,155,152,0,1615,1620,1
619,1612,11,155,158,159,156,0,1619,1622,1621,1618,11,158,143,145,159,0,1622,1603
,1607,1621,11,160,142,143,158,0,1623,1604,1603,1622,11,157,160,158,155,0,1620,16
23,1622,1619,11,162,161,164,163,0,1626,1627,1624,1625,11,161,151,153,164,0,1627,
1613,1617,1624,11,165,150,151,161,0,1628,1614,1613,1627,11,166,165,161,162,0,162
9,1628,1627,1626,11,168,167,170,169,0,1632,1633,1630,1631,11,171,168,169,172,0,1
635,1632,1631,1634,11,172,169,174,173,0,... ]

}
```


- **MandaWAV:** si occupa di inviare al browser il file audio tramite metodo GET, generato dall'ultima chiamata alla servlet **RiceviRichiestaUtente,** per l'uso con il tag <audio>.

- **MandaJSON:** si occupa di inviare al browser il file a) tramite risposta ad un'opportuna richiesta POST.

I due file, audio e prosodia, giacciono sul server e verranno sovrascritti ogni qualvolta l'utente farà, tramite browser nuove richieste.

6.6. Client

Consiste in una pagina HTML e diversi file JavaScript utili sia alla renderizzazione della scena, che per lo scambio della informazioni tra client e server.

Il testo digitato tramite un form HTML, unitamente alla velocità del TTS espressa in parole al minuto, viene inviato alla servlet RiceviRichiestaUtente, la quale, in caso di assenza di errori di Espeak ed Mbrola, restituisce al browser la stringa "REQ_OK": solo in questo caso, il browser, richiede il caricamento del modello (nella funzione "aggiorna render" la riga "handleVoce.load()") per evitare problemi di caching, nonchè l'avvio della riproduzione dell'audio.

Per scopi di debugging è possibile accedere ai morphing target precaricati nel modello da animare facendo riferimento all'oggetto `document.model.object3D.children[0]`.

6.6.1. Ritardo nella riproduzione del video

Poiché il file del modello JSON da prelevare è di dimensioni molto più elevate rispetto all'audio ed il prelievo stesso deve necessariamente avvenire in maniera del tutto asincrona, bisogna tenere conto di un

certo ritardo dovuto al fatto che non è possibile sincronizzare perfettamente l'avvio della riproduzione dell'audio e l'inizio dell'animazione. Per mitigare questo problema è stato generato mediante editor audio un silenzio di meno di un secondo che viene concatenato all'audio generato da MBROLA direttamente dal server. La durata del ritardo è sufficiente a mitigare il problema nel caso in cui client e server si trovino sulla stessa macchina: se si vuole realizzare un'infrastruttura più complessa (per esempio con un server remoto) potrebbe essere necessario modificare la durata del silenzio.

6.7. Applet originaria: esecuzione e riutilizzo di alcune sue parti nel nuovo lavoro

Prima di cominciare questo lavoro è stata necessaria una revisione della versione masterizzata su CD-ROM del lavoro precedente, al fine di visualizzarne qualche sessione di esecuzione. Chi fosse interessato a rieseguire il codice di [1] deve risolvere alcuni bug dovuti principalmente ad una masterizzazione delle cartelle con una gerarchia errata rispetto alla versione originale.
I bug riscontrati sono:

1) Mancano le librerie Java 3D, si veda il paragrafo apposito ("Come importare in Eclipse le librerie Java3d")
2) Errore nel file Talking.java, nel costruttore della classe si richiama il costruttore della classe MorphingBehaviour: venivano omessi i due input **fonemi e durata**
3) E' stata consegnata nel backup del lavoro [1], una classe che non veniva di fatto utilizzata: Talking.java. Presumibilmente si trattava di una versione ancora non funzionante della classe HeadApplication.java.

   Fatto questo è possibile eseguire il codice del lavoro [1], lanciando Eclipse, e cliccando col tasto destro il file HeadApplication.java e selezionando "RunAs" e poi "Applet".

4) Spostare Visemi e Sintetizzatore in \bin
5) Aggiunta del controllo se il sistema operativo è Linux o Windows per risolvere il divario tra la gestione dei path tra i due sistemi operativi: ciò è stato necessario per la corretta localizzazione dei visemi e dei programmi esterni da lanciare (Mbrola ed Espeak).
6) Non vengono pronunciate parole contenenti le sequenti successioni di lettere:
   a. "oio",
   b. "scia"
7) Nella classe TextPatcher, probabilmente per un errore di codifica, le previste sostituzioni necessarie per evitare problemi di parsing con Mbrola, non venivano effettuate.

Le classi che sono state importate dal lavoro [1] sono:

- Fonemi2VisemiIta:
- Sintetizzatore:
- TextPatcher:

Sono state mantenute invariate anche le liste di visemi e di mapping fonemi-visemi. Alcuni piccoli interventi sono stati necessari:

Nei file di testo contenente le informazioni necessarie al mapping fonema-visema è stato completato il path aggiungendo in testa "/ServerWebGl", posizione in cui vengono serviti i visemi.

1. La classe Fonemi2VisemiIta non è più una classe statica, in quanto è risultato opportuno memorizzare al suo interno il path relative alla directory assegnata da Tomcat all'applicazione ServerWebGl, per potere operare in maniera rapida sui file.

6.8. Deployment dell'applicazione

Per effettuare il deployment dell'applicazione (con questa dicitura si intende il bundle client+server), è necessario svolgere quantomeno i seguenti passi:

1. Installare eSpeak ed Mbrola sul sistema. Su Linux è stata utilizzata espeak 1.46 ed mbrola 1.2. Su Windows, invece la versione di espeak è precedente (1.40).
2. Importare i WAR forniti (ClientWebGl.war e ServerWebGl.war) all'interno di un application server (per lo sviluppo è stato utilizzato Tomcat 7.0).
3. Dotarsi di un browser che supporti WebGL (per lo sviluppo è stato utilizzato Chrome 24.0).

7. Codice

7.1. Codice Server

Di seguito il codice scritto interamente per questo progetto: i file non inclusi sono stati prelevati da [1] ed, eccezion fatta per alcuni bug che sono stati risolti come già precedentemente esposto, inseriti così come sono.

7.1.1. RiceviRichiestaUtente.java

```
package server_web_gl;

import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.io.StringWriter;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;
import java.nio.charset.Charset;
import java.util.ArrayList;
import java.util.LinkedList;
```

```
import java.util.ListIterator;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.json.simple.*;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

/**
 * Servlet implementation class ServerWebGl
 */
@WebServlet("/RiceviRichiestaUtente")
public class RiceviRichiestaUtente extends HttpServlet {
    private static final long serialVersionUID = 1L;
     private Sintetizzatore sintetizzatore;
     private Fonemi2VisemiIta fonemi_visemi;
     private String serverRoot;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public RiceviRichiestaUtente() {
        super();
        // per potere richiamare programmi esterni abbiamo bisogno di costruire
un path assoluto
        // ma la visibilit\u00e0 \u00e8 ordinariamente limitata alla directory radice
della gerarchia del progetto
        // per questo utilizziamo la funzione gi\u00e0 disponibile per farci
restituire il path assoluto della radice stessa
        // da concatenare a tutto il resto.
        sintetizzatore=new Sintetizzatore();
        fonemi_visemi=new Fonemi2VisemiIta();

    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
response)
     */


    public Boolean onWindows() {
        if (System.getProperty("os.name").startsWith("Windows")) {
            return true;
        }

        return false;
    }

    private String leggi_emozione_da_file(String nomeFile)
    {
         String emozione = null;
```

```
        try {
              File fileEmozione = new File(nomeFile);
              DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
              DocumentBuilder db = dbf.newDocumentBuilder();
              Document doc = db.parse(fileEmozione);
              doc.getDocumentElement().normalize();
              NodeList nodeLst = doc.getElementsByTagName("emozione");


              Node fstNode = nodeLst.item(0);


              if (fstNode.getNodeType() == Node.ELEMENT_NODE) {

                  Element fstElmnt = (Element) fstNode;
                  NodeList fstNmElmntLst =
fstElmnt.getElementsByTagName("tipo");
                  Element fstNmElmnt = (Element) fstNmElmntLst.item(0);
                  NodeList fstNm = fstNmElmnt.getChildNodes();
                  NodeList lstNmElmntLst =
fstElmnt.getElementsByTagName("percentuale");
                  Element lstNmElmnt = (Element) lstNmElmntLst.item(0);
                  NodeList lstNm = lstNmElmnt.getChildNodes();
                  emozione = ((Node) fstNm.item(0)).getNodeValue() + ((Node)
lstNm.item(0)).getNodeValue();
                  System.out.println(emozione);

                }

              } catch (Exception e) {
                e.printStackTrace();
              }

        return emozione;
    }

    private Boolean scrivi_file_json_visemi(String durate)
    {
        String nome_file_durate_visemi_utilizzati=serverRoot +
"Visemi/durate_visemi_utilizzati.json";

        File file_durate_visemi_utilizzati = new
File(nome_file_durate_visemi_utilizzati);

        file_durate_visemi_utilizzati.delete();

        scrivi_file_json(nome_file_durate_visemi_utilizzati, durate);

        if (file_durate_visemi_utilizzati.exists())
            return true;

        return false;
    }

    private void scrivi_file_json(String nomeFile, String contenuto)
    {
        try {
            PrintWriter scrittore_file = new PrintWriter(new
FileWriter(nomeFile));

            scrittore_file.print(contenuto);
```

```
            scrittore_file.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

    private JSONObject leggi_file_json(String nomeFile)
    {
        JSONParser parser = new JSONParser();
        JSONObject json_parsato = null;

            try {

                    json_parsato = (JSONObject)parser.parse(new
FileReader(nomeFile));
                } catch (ParseException | IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }



             return json_parsato;
    }

    void genera_file_json_fonemi_utilizzati(String[] fonemi)
    {
        JSONArray fonemi_utilizzati = new JSONArray();

        for(int i=0;i<fonemi.length;i++)
            fonemi_utilizzati.add(fonemi[i]);


scrivi_file_json(getServletContext().getRealPath("Visemi/fonemi_utilizzati.json"
),fonemi_utilizzati.toJSONString());
    }

    void genera_file_json_con_morphTargets(LinkedList<String>
lista_visemi_utilizzati) // String emozione, String percentuale
    {
        String nome_file_visema_silenzio = "";

        int i = 1;

        JSONWriter scrittore_json_ritorno_a_capo = new JSONWriter();
        JSONObject JSON_visema_silenzio = null;

        if (this.onWindows()) nome_file_visema_silenzio =
getServletContext().getRealPath("Visemi\\silence.js");
        else
nome_file_visema_silenzio=getServletContext().getRealPath("Visemi/silence.js");



          JSON_visema_silenzio = leggi_file_json(nome_file_visema_silenzio);

          ListIterator<String> itr = lista_visemi_utilizzati.listIterator();

          JSONArray morphTargets = new JSONArray();
```

```
            while (itr.hasNext()) // considero tutti i visemi
            {
                String nome_file_visema_corrente = itr.next();
                nome_file_visema_corrente =
getServletContext().getRealPath(nome_file_visema_corrente);

                JSONObject JSON_visema_corrente =
leggi_file_json(nome_file_visema_corrente);

                JSONArray vertici = (JSONArray)
JSON_visema_corrente.get("vertices");

                JSONObject morphTarget = new JSONObject();

                String targetName = "target"+String.valueOf(i);

                morphTarget.put("vertices", (Object)vertici);
                morphTarget.put("name", (Object)targetName);

                morphTargets.add(morphTarget);

                i++;
            }



            JSON_visema_silenzio.remove("morphTargets");
            JSON_visema_silenzio.put("morphTargets", morphTargets);
            try {
               JSON_visema_silenzio.writeJSONString(scrittore_json_ritorno_a_capo);
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

            if (this.onWindows())
scrivi_file_json(getServletContext().getRealPath("Visemi\\frase.js"),
scrittore_json_ritorno_a_capo.toString());
            else
scrivi_file_json(getServletContext().getRealPath("Visemi/frase.js"),
scrittore_json_ritorno_a_capo.toString());

            try {
               scrittore_json_ritorno_a_capo.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

        private Boolean aggiungiSilenzioWebGl(File output_wav)
        {
            File silenzio_wav = new File(serverRoot +
"Sintetizzatore/silenzio.wav");
            File temp_wav = new File(serverRoot + "Sintetizzatore/temp.wav");

            temp_wav.delete();

            WavAppender.fondiWav(silenzio_wav, output_wav, temp_wav);

            output_wav.delete();
            temp_wav.renameTo(output_wav);
```

```
        return output_wav.exists();
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        this.serverRoot=getServletContext().getRealPath("/WEB-INF");

        if (this.onWindows()) serverRoot = serverRoot + "\\";
        else serverRoot=serverRoot + "/";

        String frase = request.getParameter("frase");
        String velocita = request.getParameter("velocita");
        String emozione = null;
        String emozioneInterattiva =
request.getParameter("emozioneInterattiva");

        File output_pho = new File(serverRoot + "Sintetizzatore/output.pho");
        File output_wav = new File(serverRoot + "Sintetizzatore/output.wav");

        if (!output_pho.exists())
            output_pho.createNewFile();

        if (!output_wav.exists())
            output_wav.createNewFile();

        LinkedList<String> lista_visemi_utilizzati = new LinkedList<String>();

        JSONArray durate_visemi_utilizzati = new JSONArray();

        response.setContentType(""); // scegliere content type
        PrintWriter out = response.getWriter(); // serve per scrivere
materialmente la risposta per la chiamante

        String s = "";

        if (emozioneInterattiva!=null)
        {
            emozione=request.getParameter("emozione");
            if (emozione.compareTo("neutral") != 0) {
                String percentuale = request.getParameter("percentuale");
                emozione = emozione+percentuale;
            }
        }
        else

emozione=leggi_emozione_da_file(getServletContext().getRealPath("emozioni.xml"))
;

        if (frase!=null && velocita != null)
        {
            sintetizzatore.setRootDirectory(serverRoot);
            fonemi_visemi.setRootDirectory(serverRoot);

            sintetizzatore.sintetizza(frase, Integer.parseInt(velocita));

            int[] durate = sintetizzatore.getDurate();
            String[] fonemi = sintetizzatore.getFonemi();

            for (int i = 0; i < durate.length; i++) {

                // Converte il fonema in visema (path del visema)
```

```
                String visema = fonemi_visemi.converti(fonemi[i]);

                // cerca il path del visema e restituisce l'indice relativo
                // all'array di caricamento costruito nel costruttore
                //int temp = Fonemi2VisemiIta.getvisIndex(visema);

                lista_visemi_utilizzati.addLast(visema);
                durate_visemi_utilizzati.add(durate[i]); // aggiunge in coda
            }

            // l'emozione è l'ultimo morphtarget

            if (emozione!=null && emozione.compareTo("neutral")!=0)
            {
                String visema_emozione=fonemi_visemi.converti(emozione);
                lista_visemi_utilizzati.addLast(visema_emozione);
            }

            genera_file_json_con_morphTargets(lista_visemi_utilizzati);
            genera_file_json_fonemi_utilizzati(fonemi);

            if (output_pho.exists() && output_wav.exists() &&
scrivi_file_json_visemi(durate_visemi_utilizzati.toJSONString())
) // Se i file sono stati generati  andato
            {
                if (aggiungiSilenzioWebGl(output_wav))
                {
                    out.print("REQ_OK");
                    return;
                }
            }
            else
            {
                BufferedReader espeakStandardErrorStreamReader = new
BufferedReader(new

InputStreamReader(sintetizzatore.getEspeakStandardErrorStream()));

                BufferedReader espeakStandardOutputStreamReader = new
BufferedReader(new

InputStreamReader(sintetizzatore.getEspeakStandardOutputStream()));

                BufferedReader mbrolaStandardErrorStreamReader = new
BufferedReader(new

InputStreamReader(sintetizzatore.getMbrolaStandardErrorStream()));

                BufferedReader mbrolaStandardOutputStreamReader = new
BufferedReader(new

InputStreamReader(sintetizzatore.getMbrolaStandardOutputStream()));

                while ( (s = espeakStandardOutputStreamReader.readLine()) !=
null )
                {
                    out.println(s);
                }

                while ( (s = espeakStandardErrorStreamReader.readLine()) != null
)
                {
```

```
                    out.println(s);
                }

                while ( (s = mbrolaStandardOutputStreamReader.readLine()) !=
null )
                {
                    out.println(s);
                }

                while ( (s = mbrolaStandardErrorStreamReader.readLine()) != null
)
                {
                    out.println(s);
                }
            }
        }
        else
            out.println("Errore nella richiesta POST");

    }

}
```

7.1.2. MandaWAV.java

```
package server_web_gl;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.OutputStream;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class MandaWAV
 */
@WebServlet("/MandaWAV")
public class MandaWAV extends HttpServlet {
    private static final long serialVersionUID = 1L;


    public MandaWAV() {
        super();
        // TODO Auto-generated constructor stub
    }


    public Boolean onWindows() {
        if (System.getProperty("os.name").startsWith("Windows")) {
            return true;
        }

        return false;
    }
```

```
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        // TODO Auto-generated method stub
         // Get the absolute path of the image
        String serverRoot=getServletContext().getRealPath("/WEB-INF");  //serve
per far funzionare i path anche se si trasporta l'applicazione in un altro file
system
        String filename = "";

        if (this.onWindows())
        {
            serverRoot = serverRoot + "\\";
            filename = serverRoot + "Sintetizzatore\\output.wav";

        }
        else
        {
            serverRoot=serverRoot + "/";
            filename = serverRoot + "Sintetizzatore/output.wav";

        }



        // Get the MIME type of the image
        String mimeType = getServletContext().getMimeType(filename); //la prima
funzione mi restituisce un oggetto di servletContext che possiede le proprietà
questa servlet. La seconda funzione determina il tipo del file che gli sto
passando alla doGet
        if (mimeType == null) {
            getServletContext().log("Could not get MIME type of "+filename);
            response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
            return;
        }

        // Set content type
        response.setContentType(mimeType);

        // Set content size
        File file = new File(filename);
        response.setContentLength((int)file.length());

        // Open the file and output streams
        FileInputStream in = new FileInputStream(file);
        OutputStream out = response.getOutputStream();

        // Copy the contents of the file to the output stream
        byte[] buf = new byte[1024];
        int count = 0;
        while ((count = in.read(buf)) >= 0) {
            out.write(buf, 0, count);
        }
        in.close();
        out.close();


    }
```

```
}
```

7.1.3. MandaJSON.java

```
package server_web_gl;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class MandaJSON
 */
@WebServlet(description = "Invia al browser gli URL delle mesh OPPURE le
durate", urlPatterns = { "/MandaJSON" })
public class MandaJSON extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public MandaJSON() {
        super();
        // TODO Auto-generated constructor stub
    }

    public Boolean onWindows() {
        if (System.getProperty("os.name").startsWith("Windows")) {
            return true;
        }

        return false;
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        // TODO Auto-generated method stub

        String filename = "";
        String fileRichiesto = request.getParameter("fileRichiesto");

         PrintWriter pw = response.getWriter();  //var che serve per scrivere
l'output

        if (fileRichiesto != null)
        {
            if (fileRichiesto.compareTo("durate")==0) {
```

```
                filename = "durate_visemi_utilizzati.json";
            }
            else if (fileRichiesto.compareTo("fonemi")==0) {
                filename = "fonemi_utilizzati.json";
            }
            else
            {
                pw.print("REQ_ERR");
                return;
            }


        if (this.onWindows())
        {
            filename = "WEB-INF\\Visemi\\" + filename;

        }
        else
        {
            filename = "WEB-INF/Visemi/" + filename;

        }



        String mimeType = "application/json"; // settato manualmente
        String text = "";

        // Set content type
        response.setContentType(mimeType);

        // Open the file and output streams
        InputStreamReader isr = new
InputStreamReader(getServletContext().getResourceAsStream(filename));
        BufferedReader reader = new BufferedReader(isr);


        while ( (text=reader.readLine()) != null )
            pw.println(text);
        }
    }



}
```

7.2. Codice Client

7.2.1. pupo.js

```
//Custom JSON model class
Pupo = function()
{
    Sim.Object.call(this);
}

Pupo.prototype = new Sim.Object();
```

```
Pupo.prototype.init = function(param)
{
    var group = new THREE.Object3D;

    var that = this;

    var MAX_INT=9007199254740992; // anti-cache
    var url = param.url+'?'+THREE.Math.randInt(-MAX_INT, MAX_INT) || "";
    if (!url)
        return;

    var scale = param.scale || 1;

    this.scale = new THREE.Vector3(scale, scale, scale);


    // per l'audio
    this.audio_url = param.audio_url;
    this.durate_visemi = param.durate_visemi;

    var loader = new THREE.JSONLoader();

// PORT DA THREE.JS R46 A R55: GEOMETRY E MATERIAL ORA SONO DISTINTI

    //loader.load( url, function( data ) {
//      that.handleLoaded(data); } );


    loader.load( url, function( geometry, materials ) {
        that.handleLoaded(geometry, materials); } );

    // Tell the framework about our object
    this.setObject3D(group);


    // per l'animazione
    this.animating = false;
    this.clock = new THREE.Clock();
    this.animating = param.animating;

}


Pupo.prototype.handleLoaded = function(loadedGeometry, loadedMaterials)
{
    if (loadedGeometry instanceof THREE.Geometry)
    {
        //PORT A THREE.JS R55: var geometry = data;
        var geometry = loadedGeometry;
        var materials = loadedMaterials;

        // Just in case model doesn't have normals
        geometry.computeVertexNormals();

        // for preparing animation
        // senza questo for non vengono attivati i morphTargetInfluences

// PORT A THREE.JS R55:
/*   for (var i = 0; i < geometry.materials.length; i++)
        geometry.materials[i].morphTargets = true; */

         for (var i = 0; i < materials.length; i++)
```

```
            materials[i].morphTargets = true;

        var material = new THREE.MeshFaceMaterial(materials);

        var mesh = new THREE.MorphAnimMesh( geometry, material  ); // Mesh nel
libro

        // SOMMA DELLE DURATE EFFETTUATA COL MAP REDUCE
        if (this.durate_visemi.length > 0)
        {
            mesh.duration = this.durate_visemi.reduce(function(previousValue,
currentValue, index, array){
              return previousValue + currentValue;
        });
        }



        // Funzione di animazione personalizzata, scritta in modo tale da
aggiungere la
        // possibilit\u00e0 di impedire il loop
        this.indice_visema_corrente = 0;

        mesh.updateAnimation = function ( delta )
        {
            var frameTime = this.duration / ( this.geometry.morphTargets.length
- 1 ); // Aggiornamento non basato sulle durate dei visemi (updateAnimation \u00e0
chiamato di continuo, anche quando non riproduciamo)

            if ( this.durate_visemi )
                 frameTime = this.durate_visemi[this.indice_visema_corrente];

            this.time += delta;

// Alla fine dell'animazione fermati

                    if ( this.time > this.duration) {
                    this.time = this.duration;

                    // WORKAROUND: in teoria basta impostare this.time a
this.duration per
                    // interrompere l'animazione, ma con alcune vocali lunghe
                    // ci sono problemi a riportare la mesh alla configurazione
originaria:

                    // Esempio: la parola "casa" pronunciata "c aaa saaa "
produce un artefatto
                    // alla fine della riproduzione

                    this.morphTargetInfluences[this.morphTargetInfluences.length
- 3]=0;

                    // tolgo dolcemente l'emozione
                    var decremento=0.020;
                    this.morphTargetInfluences[this.morphTargetInfluences.length
- 1]=Math.max(this.morphTargetInfluences[this.morphTargetInfluences.length - 1]-
decremento,0);
                    return;
                }
                if ( this.time < 0 ) this.time = 0;

            // LOOP DELL'ANIMAZIONE PER TEST:
```

```
            //this.time = this.time % this.duration;


        var keyframe = THREE.Math.clamp( Math.floor( this.time / frameTime ), 0,
this.geometry.morphTargets.length - 1 );

            if ( keyframe != this.currentKeyframe ) { // attendi la transizione
di frame brusca appena l'if  soddisfatto

                this.morphTargetInfluences[ this.lastKeyframe ] = 0;
                this.morphTargetInfluences[ this.currentKeyframe ] = 1;

                this.morphTargetInfluences[ keyframe ] = 0;

                this.lastKeyframe = this.currentKeyframe;
                this.currentKeyframe = keyframe;

                if (this.durate_visemi && this.indice_visema_corrente !=
this.durate_visemi.length)
                    this.indice_visema_corrente++;
            }

            // emozione attiva
            this.morphTargetInfluences[this.morphTargetInfluences.length - 1]=1;

            var mix = ( this.time % frameTime ) / frameTime;

                this.morphTargetInfluences[ this.currentKeyframe ] = mix;
                this.morphTargetInfluences[ this.lastKeyframe ] = 1 - mix;
        };

        this.faceMesh = mesh;

        //caricamento file audio

        this.audio = new Audio();
        this.audio.setAttribute('src', this.audio_url);
        this.audio.preload = 'auto';
        this.audio.load();

        mesh.scale.copy(this.scale);
        this.object3D.add( mesh );
    }

}

Pupo.prototype.update = function(data)
{
    if(this.animating && this.faceMesh)
    {

        var delta = this.clock.getDelta();

          if (this.audio.paused) {
            this.audio.play();
        }

        this.faceMesh.updateAnimation(1000*delta);

    }

    Sim.Object.prototype.update.call(this);
```

```
}
```

7.2.2. pupoApp.js

```
// Constructor
PupoApp = function()
{
    Sim.App.call(this);
};

// Subclass Sim.App
PupoApp.prototype = new Sim.App();

// Our custom initializer
PupoApp.prototype.init = function(param)
{
    // Call superclass init code to set up scene, renderer, default camera
    Sim.App.prototype.init.call(this, param);

    // Sostituiamo la telecamera di default

    this.scene.remove(camera);

    camera = new THREE.PerspectiveCamera(
            105,         // Field of view
            this.container.offsetWidth / this.container.offsetHeight,  // Aspect
ratio
            .1,          // Near
            100000000       // Far
        );

    camera.position.set( -2, -5, 50);
    this.camera = camera;
    this.scene.add(camera);

    // Create a headlight to show off the model
    this.headlight = new THREE.DirectionalLight( 0xffffff, 1);
    this.headlight.position.set(0, 0, 1);
    this.scene.add(this.headlight);

    var amb = new THREE.AmbientLight( 0xffffff );
    this.scene.add(amb);

    this.createCameraControls();
};

PupoApp.prototype.addModel = function(model)
{
    this.addObject(model);
};

PupoApp.prototype.removeModel = function(model)
{
    this.removeObject(model);
};


PupoApp.prototype.createCameraControls = function()
{
```

```
// PORT DA THREE.JS R46 A R55: TrackballControls è stato reimplementato
esternamente (script incluso in index.html)
    var controls = new THREE.TrackballControls( this.camera,
this.renderer.domElement );
    var radius = PupoApp.CAMERA_RADIUS;

    controls.rotateSpeed = PupoApp.ROTATE_SPEED;
    controls.zoomSpeed = PupoApp.ZOOM_SPEED;
    controls.panSpeed = PupoApp.PAN_SPEED;
    controls.dynamicDampingFactor = PupoApp.DAMPING_FACTOR;
    controls.noZoom = false;
    controls.noPan = false;
    controls.staticMoving = false;

    controls.minDistance = radius * PupoApp.MIN_DISTANCE_FACTOR;
    controls.maxDistance = radius * PupoApp.MAX_DISTANCE_FACTOR;

    this.controls = controls;
};

PupoApp.prototype.update = function()
{
    // Update the camera controls
    if (this.controls)
    {
        this.controls.update();
    }

    // Update the headlight to point at the model
    var normcamerapos = this.camera.position.clone().normalize();
    this.headlight.position.copy(normcamerapos);

    Sim.App.prototype.update.call(this);
};

PupoApp.CAMERA_RADIUS = 5;
PupoApp.MIN_DISTANCE_FACTOR = 1.1;
PupoApp.MAX_DISTANCE_FACTOR = 10;
PupoApp.ROTATE_SPEED = 1.0;
PupoApp.ZOOM_SPEED = 3;
PupoApp.PAN_SPEED = 0.2;
PupoApp.DAMPING_FACTOR = 0.3;
```

7.2.3. index.html

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Tesina Cipolla-Monte Informatica grafica</title>

<link rel="stylesheet" href="clientWebGl.css" type="text/css">

<script src="libs/three.min.js" type="text/javascript"></script>
<script src="libs/TrackballControls.js" type="text/javascript"></script>

<script src="libs/jquery-1.6.4.js" type="text/javascript"></script>
<script src="libs/jquery.mousewheel.js" type="text/javascript"></script>
<script src="libs/RequestAnimationFrame.js" type="text/javascript"></script>
<script src="sim/sim.js" type="text/javascript"></script>

<script src="pupoApp.js" type="text/javascript"></script>
<script src="pupo.js" type="text/javascript"></script>
```

```
 <script src="clientWebGl.js" type="text/javascript"></script>


    <script>

    var renderer = null;
    var scene = null;
    var camera = null;
    var mesh = null;

    $(document).ready(
        function() {
            var container = document.getElementById("container");

            document.app=new PupoApp();
            document.app.init({ container: container });

            document.model= new Pupo();
            document.model.settings = {
                    url : "http://localhost:8080/ServerWebGl/Visemi/frase.js",
                    audio_url: "http://localhost:8080/ServerWebGl/MandaWAV",
                    durate_visemi: new Array(),
                    scale:0.23,
                    animating: false};

            document.model.init(document.model.settings);

            apriFinestraControlli();

            document.app.addModel(document.model); // aggiungiamo la mesh alla
foglia del DOM relativa a pupoApp
            document.app.run();
        }
    );
    </script>

</head>
<body class="finestraPrincipale">
    <h1 class="title">"Il Pupo" in WebGL</h1>
    <div id="container"></div>
    <div id="prompt">
    Fare click con il mouse per manipolare il modello: Pulsante sinistro =
rotazione, Pulsante destro = pan </div>
</body>
</html>
```

7.2.4. controlli.html

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Tesina Cipolla-Monte Informatica grafica</title>

<link rel="stylesheet" href="clientWebGl.css" type="text/css">
<script src="libs/three.min.js" type="text/javascript"></script>
<script src="libs/TrackballControls.js" type="text/javascript"></script>

<script src="libs/jquery-1.6.4.js" type="text/javascript"></script>
```

```
<script src="libs/jquery.mousewheel.js" type="text/javascript"></script>
<script src="libs/RequestAnimationFrame.js" type="text/javascript"></script>
<script src="sim/sim.js" type="text/javascript"></script>

<script src="pupoApp.js" type="text/javascript"></script>
<script src="pupo.js" type="text/javascript"></script>
 <script src="clientWebGl.js" type="text/javascript"></script>



</head>
<body class="finestraControlli" onload="inizializzazione();">
<h1 class="title">"Il Pupo" in WebGL</h1>
<form method="post" action="" name="moduloClientWebGl">
<div>
<fieldset class="fieldset-auto-width">
<legend>Text-To-Speech</legend>
<label for="frase" id="labelFrase">Frase:    </label> <input
type="text" name="frase" id="frase"/><br/>
<label for="velocita" id="labelVelocita">Velocit&agrave;:</label> <input
type="number" name="velocita" class="integer" id="velocita" value="120"
size="3"/>
</fieldset>
</div>
<br/><div>
<fieldset class="fieldset-auto-width">
<legend>Emozione</legend>
<input type="checkbox" id="emozioneInterattiva" name="emozioneInterattiva"
value="true"
onclick="specificaEmozioneInterattiva(document.getElementById('emozioneInteratti
va').checked)">Specifica interattivamente l'emozione<br/>
<label for="emozione"
id="labelEmozione">Emozione:    </label>
<select name="emozione" id="emozione" form="moduloClientWebGl"
onchange="controllaEmozione();">
  <option value="neutral" selected="selected">Neutrale</option>
  <optgroup label="Positive">
    <option value="*su">Sorpreso</option>
    <option value="*h">Felice</option>
    <option value="*f_sa">Felice e sorpreso</option>
    </optgroup>

  <optgroup label="Negative">
    <option value="*d">Disgustato</option>
    <option value="*sa">Triste</option>
    <option value="*a">Arrabbiato</option>
    <option value="*a_d">Arrabbiato e disgustato</option>
    <option value="*f">Impaurito</option>
    <option value="*f_sa">Impaurito e triste</option>
</optgroup>
</select><br/>
<label for="percentuale" id="labelPercentuale">Percentuale:</label>
<input type="number" name="percentuale" id="percentuale" class="integer" min="1"
max="100" value="100" size="3"/>
<span id="segnoPercentuale">%</span>
</fieldset>
</div><br/>
<input type="button" value="Invia richiesta" onclick="inviaRichiesta(); return
true;">

</form>
```

```
</body>
</html>
```

7.2.5. ClientWebGl.js

```
/* FUNZIONI DI SUPPORTO */
function apriFinestraControlli() {
  var left = 100;
  var width = 300;
  var height = 500;
  var top = (screen.height/2)-(height/2);
  return window.open('controlli.html', 'finestraControlli',
"menubar=no,toolbar=no,status=no,width="+width+",height="+height+",toolbar=no,le
ft="+left+",top="+top);
}

function creaConnessioneXMLHTTP_POST(url,lunghezzaParametri)
{
    xmlhttp="undefined";

    if(url!="" || lunghezzaParametri>=0)
    {

        if(window.ActiveXObject)
        {// code for IE6, IE5
            xmlhttp=new ActiveXObject("Microsoft.XMLHTTP"); //roba specifica di
IE
        }
        else // se esiste questo oggetto
        {// code for IE7+, Firefox, Chrome, Opera, Safari
            xmlhttp=new XMLHttpRequest();
        }

        xmlhttp.open("POST",url,true); //apriamo l'oggetto (true per richiesta
asincrona)
        xmlhttp.setRequestHeader("Content-type", "application/x-www-form-
urlencoded");  //specifichiamo che la richiesta contiene parametri codificati
        xmlhttp.setRequestHeader("Content-length", lunghezzaParametri);
//lunghezza dei parametri da codificare
        xmlhttp.setRequestHeader("Connection", "close");
  }

    return xmlhttp;
}

function inizializzazione()
{
    controllaEmozione();
    specificaEmozioneInterattiva(false);
    document.moduloClientWebGl.frase.focus();
}

function controllaEmozione()
{
    var handleEmozione = document.getElementById("emozione");
    var handlePercentuale = document.getElementById("percentuale");
    var handleLabelPercentuale = document.getElementById("labelPercentuale");
    var handleSegnoPercentuale = document.getElementById("segnoPercentuale");
    var emozione=handleEmozione.options[handleEmozione.selectedIndex].value;
```

```
    if ( emozione == "neutral" )
    {
        handleLabelPercentuale.style.visibility = "hidden";
        handleSegnoPercentuale.style.visibility = "hidden";
        handlePercentuale.style.visibility = "hidden";
    }
    else
    {
        handleLabelPercentuale.style.visibility = "visible";
        handleSegnoPercentuale.style.visibility = "visible";
        handlePercentuale.style.visibility = "visible";
    }

}
/* FUNZIONI DI CONVALIDA E INVIO DEL MODULO INPUT */

function validaParametri(handleFrase, handleVelocita)
{
    frase = handleFrase.value;
    velocita = handleVelocita.value;

    if (frase == "")
    {
        alert("Inserire almeno un carattere");
        handleFrase.focus();
        return false;
    }

    if (frase == "")
    {
        alert("Inserire almeno un carattere");
        handleFrase.focus();
        return false;
    }

    if (frase.match(/[\|%]/))  // NON FUNZIONA
    {
        alert("I caratteri \, | e % non sono ammessi");
        handleFrase.focus();
        return false();
    }

    if (isNaN(velocita) || velocita <= 0)
    {
        alert("La velocita' e' una quantita' intera positiva non nulla");
        handleVelocita.value="120";
        handleVelocita.focus();
        return false;
    }

    return true;
}

function specificaEmozioneInterattiva(booleano)
{
    var handleEmozione = document.getElementById("emozione");
    var handlePercentuale = document.getElementById("percentuale");
    var handleCheck = document.getElementById("emozioneInterattiva");

    handleCheck.checked = booleano;
    handleEmozione.disabled = !booleano;
```

```
    handlePercentuale.disabled = !booleano;
    controllaEmozione();
}

function inviaRichiesta()
{
    if
(validaParametri(document.moduloClientWebGl.frase,document.moduloClientWebGl.vel
ocita) == true)
    {
        var frase=document.moduloClientWebGl.frase.value;
        var velocita=document.moduloClientWebGl.velocita.value;
        var handleEmozione = document.getElementById("emozione");
        var handlePercentuale = document.getElementById("percentuale");
        var handleEmozioneInterattiva =
document.getElementById("emozioneInterattiva");
        var emozione=handleEmozione.options[handleEmozione.selectedIndex].value;
        var percentuale=handlePercentuale.value;

        var richiestaEmozioneInterattiva = handleEmozioneInterattiva.checked;

        var parametri =
"frase="+frase+"&velocita="+velocita+"&emozione="+emozione;

        if (emozione!="neutral")
            parametri=parametri+"&percentuale="+percentuale;

        if (richiestaEmozioneInterattiva==true)
            parametri=parametri+"&emozioneInterattiva=true";


xmlhttp=creaConnessioneXMLHTTP_POST("http://localhost:8080/ServerWebGl/RiceviRic
hiestaUtente",parametri.length);

        xmlhttp.onreadystatechange=function(){  //passiamo alla funzione
riempi campi la risp della servlet
            if(xmlhttp.readyState==4 && xmlhttp.status==200){

                risposta = xmlhttp.responseText;

                if(risposta == "REQ_OK")
                {
                    ottieni_durate();
                }
                else
                {
                    alert("Errore durante la connessione al server Text-to-
Speech");
                    document.moduloClientWebGl.frase.focus();
                }

            }  //4 significa richiesta completata - 200 significa richiesta http
andata a buon fine
        };

        if(parametri!="")
        {
            xmlhttp.send(parametri); //manda la richiesta HTTP e attendiamo che
si verifichi l'evento onreadystatechange
        }
    }
}
```

```
function aggiorna_render()
{
        nuovo_modello = new Pupo();

        nuovo_modello.settings =  {
                url : "http://localhost:8080/ServerWebGl/Visemi/frase.js",
                audio_url: "http://localhost:8080/ServerWebGl/MandaWAV",
                durate_visemi: window.durate_visemi,
                scale:0.23,
                animating: true};

        nuovo_modello.init(nuovo_modello.settings);

        document.app.removeModel(document.model);
        document.model = nuovo_modello;
        document.app.addModel(document.model);

 }

/* FUNZIONE PER OTTENERE DURATE VISEMI */

function ottieni_durate()
{
    var MAX_INT=9007199254740992;
    parametri="fileRichiesto=durate&randomizer="+THREE.Math.randInt(-MAX_INT,
MAX_INT);


xmlhttp_durate=creaConnessioneXMLHTTP_POST("http://localhost:8080/ServerWebGl/Ma
ndaJSON",parametri.length);

    xmlhttp_durate.onreadystatechange=function(){  //passiamo alla funzione
riempi_campi la risp della servlet
        if(xmlhttp_durate.readyState==4 && xmlhttp_durate.status==200){

            risposta = xmlhttp_durate.responseText;

            window.opener.durate_visemi=JSON.parse(risposta);

            window.opener.aggiorna_render();

        }  //4 significa richiesta completata - 200 significa richiesta http
andata a buon fine
    };

    if(parametri!="")
    {
        xmlhttp.send(parametri); //manda la richiesta HTTP e attendiamo che si
verifichi l'evento onreadystatechange
    }

}
```

7.2.6. ClientWebGl.css

```
canvas {
    width: 100%;
    height: 100%;
}
```

```
body.finestraPrincipale {
    background-color:#000000;
    color:#ffffff;
    font-family: Arial;
    font-style: italic;
}

body.finestraControlli {
    background-color:#eeeeee;
    color:#212121;
    font-family: Arial;
    font-style: italic;
}


h1.title {
    text-align:center;
}

#container {
    width:100%;
    height:80%;
    z-index:-1;
    position:absolute;
}

#prompt {
    width:100%;
     height:6%;
     bottom:10px;
     text-align:center;
     position:absolute;

}

fieldset.fieldset-auto-width {
          display: inline-block;
    }

input#integer {
    text-align: right;
}
```

8. Bibliografia

1. F. Milazzo, S. Romano, " Implementazione di un modello di coarticolazione facciale basato sul metodo di Cohen-Massaro", Tesina di Informatica Grafica A.A. 2008/2009.
2. A. Anyuru, "Professional WebGL Programming – Developing 3D Graphics for the Web", Wrox, 2012.
3. A. Watt, "3D Computer Graphics" 3rd Edition, Pearson – Addison Wesley, 2000.
4. D. Cantor, B. Jones, "WebGL Beginner's Guide", Packt Publishing, 2012.
5. T. Parisi, "WebGL Up&Running", O'Reilly, 2012.

6. D. La Porta, D. Terranova, “Una talking head 3D con espressioni per il web”, Tesina di informatica grafica A.A. 2009/2010.