Università degli Studi di Perugia
Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in

Informatica

Tesi di Laurea

Implementazione di un editor per mondi virtuali con tecnologia X3DOM

Laureando:

Federico Frenguelli

Relatori:

Prof. Osvaldo Gervasi

Anno Accademico 2010–2011

A Chiara, alla mia famiglia, ai miei amici

Indice

Elenco delle figure

Introduzione

Questo progetto si pone l'obiettivo di sviluppare un editor di scene virtuali come web application utilizzando le ultime tecnologie a disposizione per il 3d nel web, frutto di un processo evolutivo e di standardizzazione iniziato circa venti anni fa, per testarne le potenzialità e comprendere il grado di maturità raggiunto.

Si pensi ad un supermercato. Si immagini questo ambiente riprodotto in uno scenario virtuale. Due sono gli attori principali in gioco: il cliente, che si muove all'interno dell'ambiente, osserva e sceglie tra gli oggetti esposti cosa acquistare, e il gestore, che si occupa di predisporre ed organizzare gli scaffali con i prodotti a disposizione. Il gestore dunque configura la scena per il cliente. In uno spazio virtuale il gestore necessita di uno strumento, un editor che gli consenta di inserire, posizionare ed eliminare gli oggetti (in questo caso i prodotti del supermercato) e di salvare le proprie modifiche rendendole disponibili all'utente.

Il semplice caso d'uso descritto delinea un'applicazione di tipo gestionale, in cui diversi utenti accedono con vari livelli di autorizzazione ai servizi mes-

si a disposizione. È facile pensare da subito ad un software per il desktop come alla soluzione più naturale e semplice da implementare, vista anche la necessità di rappresentare una scena 3d con la quale l'utente possa interagire. Negli ultimi anni si è però assistito ad un fiorire di web applications sempre più complesse, dalle "office suite" fino ai gestionali per aziende e programmi per il photo ed il video editing.

Seguendo perciò questo trend di migrazione delle applicazioni dal desktop al web, si fa strada l'idea di avere un editor 3d come quello descritto interamente in un browser. Possibilità questa interessante per tutti i vantaggi che una web application porta con sé, quali la compatibilità cross-platform, nessuna installazione e update lato utente grazie all'utilizzo del browser come thin-client, facilità di integrazione con servizi web e possibilità di funzionalità avanzate (e.g. editing collaborativo). Sorgono però alcune domande. Quali tecnologie ci sono a disposizione per il 3d nel web oggi? Sono abbastanza mature per realizzare un applicazione affidabile e performante? Quali svantaggi presentano?

Il compito di trovare una risposta a queste domande è affidato alla prima parte di questo lavoro. Dapprima verranno introdotte le possibili soluzioni presenti nell'era ante web 2.0 quali i linguaggi VRML prima e X3D poi, integrati nei browser tramite plugins di terze parti. Si analizzeranno in seguito i motivi per i quali queste non hanno avuto il successo sperato. Da questo piccolo fallimento si vedrà poi come nasce l'idea di un web 3d senza plugin, "nativo", e quali tecnologie lo rendono possibile. Si parlerà dunque della specifica HTML5, ancora in via di definizione che grazie alle novità

introdotte pone le basi per il 3D nel web, di WebGL, un nuovo standard promosso dal Khronos Group che definisce un interfaccia di programmazione di basso livello per la creazione di contenuti 3D all'interno di un browser web, e di x3dom, un motore javascript che consente l'introduzione di elementi X3D come parte del DOM di una pagina HTML5 e che fa uso dell'API WebGL per visualizzarli.

L'unico modo di testare efficacemente questo sempre più fiorente intreccio di tecnologie è effettuare una prova sul campo. Il software Medea, fulcro di questo lavoro, nasce esattamente a questo scopo. Nella seconda parte si analizzeranno le specifiche funzionali e le caratteristiche tecniche del software. Si vedrà come sono state utilizzate ed interconnesse le tecnologie di cui sopra per raggiungere gli obiettivi prefissati. Si cercherà di valutare l'efficienza e l'usabilità della soluzione proposta, al fine di individuare e comprendere quali siano i pregi e quali i difetti e i limiti di questa implementazione di 3D nel web. Obiettivo finale è comprendere se queste tecnologie siano pronte e mature per supportare un'applicazione di produzione completa, performante, cross-platform e user-friendly. s

Capitolo 1

Tecnologie utilizzate

Portare il 3d nel web non è cosa di poco conto. In un continuo processo evolutivo, gli sforzi di molte aziende ed utenti si sono sommati fino a creare il complesso ecosistema di tecnologie che oggi ci consentono di avere un pizzico di virtualità all'interno del browser. Nel seguito vengono illustrati uno ad uno questi strumenti, in ordine di apparizione, per avere una visione d'insieme chiara e completa. Alcuni di questi sono stati creati e sono evoluti pensando al web. Altri sono stati riadattati per essere integrati nel browser e per interagire con il World Wide Web.

Le tecnologie trattate di seguito sono tutti standard aperti[1], le cui specifiche sono disponibili pubblicamente e gratuitamente e chiunque può contribuire al processo di decisione e di sviluppo. Questo lavoro non può e non

[1]Open Inventor. (2011, July 12). In *Wikipedia, The Free Encyclopedia*. Retrieved 09:05, August 25, 2011, from `http://en.wikipedia.org/w/index.php?title=Open_Inventor&oldid=439077866`

vuole considerare soluzioni chiuse e proprietarie.

1.1 Extensible 3D Markup Language

In principio era VRML. Il Virtual Reality Modeling Language nasce nel 1994 e viene ratificato come standard ISO nel 1997 con il nome VRML97[2]. Questo linguaggio consente di rappresentare una scena 3d in un semplice file di testo con estensione "wrl", detto anche "world". Vertici, spigoli, materiali, parametri per il texturing ed effetti, gestione luci, animazioni e suoni sono specificati in una struttura ad albero di tipo "scene graph" con una sintassi[3] semplice ed intuitiva. L'interazione con l'utente è gestita grazie a nodi sensore ed eventi. Si possono inoltre inserire degli script, codificati in java o javascript, in dei nodi appositi, garantendo la massima flessibilità ed adattabilità della scena disegnata.

Come in ogni nuovo standard vi erano in VRML delle evidenti lacune che andavano colmate. Una delle maggiori critiche sollevate da utenti e sviluppatori era la mancanza di una naturale integrazione con HTML. Per dirlo con le parole di Chris Phillips, sviluppatore per uno standard concorrente di VRML conosciuto come Chromeffects, *"VRML has no integration with HTML. When the VRML guys built VRML [...] it was all about 3D. They didn't work with the World Wide Web Consortium. They didn't even think about it. They were too busy getting the 3D to work"*. A questo si andavano ad aggiungere altre esigenze come, ad esempio, la necessità di trovare uno standard solido, e ampiamente diffuso tra gli sviluppatori, su

[2]ISO/IEC 14772-1:1997 – Per maggiori informazioni si veda `http://www.web3d.org/x3d/specifications/vrml/`

[3]La sintassi di un file VRML si basa sul formato standard Open Inventor, prodotto da SGI

cui basare il linguaggio. Questo avrebbe garantito facilità di integrazione in altre applicazioni (e.g. sistemi di authoring e di playback), ma anche un comportamento uniforme e consistente delle scene disegnate tra software di diversi produttori. Infine, vi erano numerose richieste di nuove features da parte degli utenti che andavano analizzate e selezionate per essere poi implementate.

A partire da questi requisiti nasce X3D come naturale evoluzione di VRML. L'Exensible 3D Markup Language mantiene così molte delle precedenti caratteristiche (garantendo retrocompatibilità) e allo stesso tempo amplia la specifica con nuove features[4] e capacità, prima fra tutte la possibilità di definire le scene utilizzando una sintassi XML[5]. La scelta di introdurre nella specifica l'Extensible Markup Language come nuovo formato va a colmare quel vuoto lasciato dalla mancata integrazione con HTML e il World Wide Web. XML è diventato la lingua franca del web ed, inoltre, è utilizzato da molte applicazioni per la rappresentazione e lo scambio dei dati, grazie anche alle numerose interfacce di programmazione per l'accesso ai dati, esistenti per ogni linguaggio, che ne rendono semplice la gestione, il controllo e la validazione.

[4]Tra quelle non citate, degne però di nota, vi sono:

1. divisione di X3D in componenti, che consente sia di utilizzare un sottoinsieme delle funzionalità offerte quanto più calzante con i requisiti, sia di introdurre in maniera graduale e pulita nuovi componenti garantendo un rapido supporto alle nuove tecnologie
2. miglioramento dell'interfaccia di scripting
3. miglioramento della specifica che definisce il comportamento runtime di X3D, garantisce che una scena X3D si comporti in maniera prevedibile e uniforme su diversi browser/player

[5]L'utilizzo di un formato XML-based è di fatto una possibilità in quanto la specifica, proprio per garantire la retrocompatibilità, ammette l'utilizzo del formato VRML-based

X3D viene ratificato come standard ISO nel 2004[6]. Sebbene abbia suscitato, insieme a VRML, l'interesse di un gran numero di utenti e di aziende e goda tutt'ora di una certa popolarità, non ha mai conosciuto il successo sperato. Secondo molti osservatori, il problema era l'assenza di una qualsiasi offerta al pubblico. In sintesi, al navigatore medio non interessava affatto volare in mondi 3D: ciò che gli interessava era semplicemente l'informazione[7]. Va aggiunto poi che gli strumenti di navigazione quali i player X3D erano spesso distribuiti come plugin per i browsers o in forma standalone, essendo queste le uniche possibilità. Mancava dunque quella parte di "vera" integrazione con la pagina web, che avrebbe consentito all'utente di accedere con immediatezza e facilità ai contenuti 3D, senza dover passare per complessi rompicapo quali la scelta e l'installazione di uno dei tanti player messi a disposizione. Infine, ma non per ultimo, va detto che a limitare la diffusione di X3D ha contribuito anche l'adozione di un modello di sviluppo chiuso e proprietario dei sistemi di playback, di editing e di authoring. Come spesso accade (tralasciando la quasi totale assenza di prodotti cross-platform) molti di questi software non sono sopravvissuti alla scomparsa dell'azienda produttrice, lasciando i propri utenti "con un pugno di mosche in mano".

Mentre X3D viveva la sua "crisi", un'altra tecnologia si stava facendo avanti, un nuovo standard che avrebbe potuto riportare X3D alla gloria di un tempo.

[6]ISO/IEC 19775/19776/19777

[7]Guida a VRML. In *HTML.it*. Retrieved 11:00, August 27, 2011, from `http://xml.html.it/guide/lezione/1812/il-3d-sul-web-storia-e-futuro/`

1.2 Un nuovo standard all'orizzonte: HTML5

Molto, forse troppo, ci sarebbe da dire sulla nascita e l'evoluzione di HTML. Ci si limiterà in questo contesto a ripercorrere brevemente la sua storia, soffermandosi su quei cambiamenti e punti di svolta degni di nota, per poi dedicare ampio spazio a quelle caratteristiche fondamentali per l'evoluzione del 3d nel web.

HTML nasce nel 1989 ad opera di Tim Berners-Lee, oggi ritenuto uno dei padri fondatori del World Wide Web. Berners-Lee cercava uno strumento per descrivere in maniera efficiente e semplice ipertesti, ovvero le future pagine web. Quello che fece fu scrivere la primissima specifica[8] dell'Hypertext Markup Language e il software che la implementava. Da qui, come si usa dire, il resto è storia.

Dopo oltre 20 anni, siamo giunti ad HTML5[9], quinta (ma non ultima!) revisione di quella prima embrionale specifica, ancora in corso di definizione (in gergo *draft*). Quali sono i suoi obiettivi? Come ricorda il W3C[10] stesso nella documentazione di HTML5, *"The main area that has not been adequately addressed by HTML is a vague subject referred to as Web Applications. This specification attempts to rectify this, while at the same time updating the HTML specifications to address issues raised in the past few*

[8] La prima informale specifica era un ristretto documento denominato *HTML Tags*, `http://www.w3.org/History/19921103-hypertext/hypertext/WWW/MarkUp/Tags.html`

[9] HTML5 Specification. In *HTML5 Editor's Draft*. Retrieved 12:20, August 29, 2011 from `http://dev.w3.org/html5/spec/Overview.html`

[10] Il World Wide Web Consortium (W3C) è un'associazione internazionale che si occupa di definire standard aperti per il Web e di assicurarne la crescita ed l'interoperabilità. `http://www.w3.org/Consortium/mission.html`

years". HTML5 fa da "capello" ad un insieme di tecnologie pensate per agevolare la creazione di web application.

In primo luogo il linguaggio è stato arricchito di nuovi elementi per migliorare la semantica delle pagine. I tag `<nav>`, `<section>`, `<article>`, `<header>` e `<footer>`,ad esempio, individuano parti ben precise e immediatamente riconoscibili di un documento HTML5. Il markup è stato snellito con l'eliminazione dei molti elementi deprecati in HTML 4.01 e sono stati introdotti miglioramenti per quanto riguarda le regole di parsing. Vi sono poi i tag `<audio>` e `<video>`, che consentono l'inserimento di contenuti multimediali, senza l'ausilio di plugins esterni, all'interno della pagina. Questi elementi sono parte del DOM e dunque possono essere liberamente modificati e gestiti grazie a JavaScript. Sono state poi aggiunte numerose API per semplificare la vita dello sviluppatore, come WebStorage, un'interfaccia di programmazione, evoluzione dei cookies, per la persistenza dei dati lato client, oppure WebSockets, un'API che consente di creare una connessione full-duplex tra client e server.

Per quanto riguarda questo progetto, però, le novità più importanti sono sicuramente quelle introdotte nel campo della grafica e del disegno all'interno della pagina web. Oltre ad avere integrato pienamente la specifica Scalable Vector Graphics (SVG) per la gestione di immagini ed elementi in grafica vettoriale, è stato introdotto un nuovo elemento per il disegno, il tag `canvas`. Quest'ultimo consente di definire un'area da utilizzare come una tela, un foglio sul quale disegnare. Il codice Javascript consente di accedere all'oggetto Canvas e, da questo, ci permette di recuperare un'istanza del

tipo Context. Un "contesto" è un oggetto che espone un'API per il disegno. Un Canvas può fornire molteplici Context. L'esempio seguente mostra l'utilizzo del Context "2d", l'unico per ora formalmente definito nella specifica HTML5. Il risultato è mostrato in Figura 1.1.

Listing 1.1: Draw into canvas using 2d Contex

```
<!DOCTYPE html>
<html>
<head>
<title>Canvas Examples</title>
<script type="text/javascript">
window.onload = function () {
    var draw = function (c) {
        c.fillStyle = "red";
        c.fillRect(50, 100, 100, 100);
        c.fillStyle = "green";
        c.fillRect(150, 100, 100, 100);
        c.fillStyle = "blue";
        c.fillRect(250, 100, 100, 100);
    };
    var canvas = document.getElementById("paint");

    if (canvas.getContext) {
        var context = canvas.getContext("2d");
        draw(context);
    } else {
        // fallback
    }
};
</script>
</head>
<body>
    <h1>Canvas</h1>
    <canvas id="paint" width="400px" height="300px" style="border: 1px
        solid">
    </canvas>
</body>
</html>
```

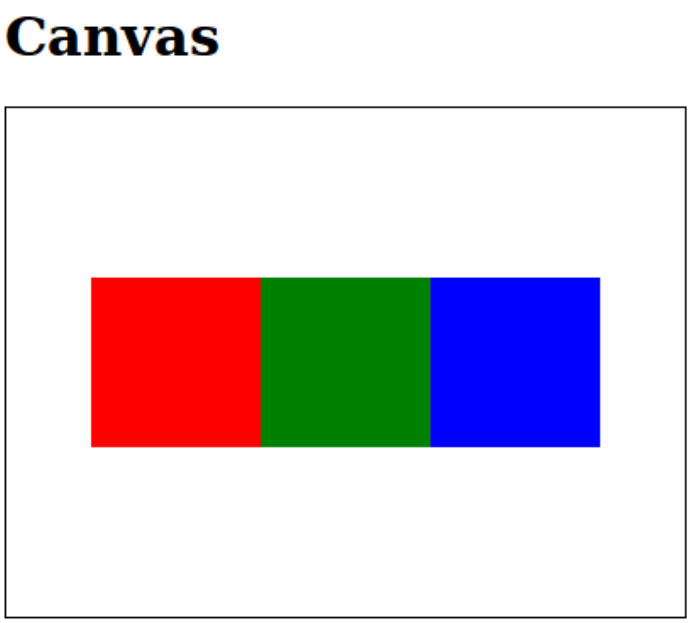

Figura 1.1: Tre rettangoli disegnati nel canvas

Il codice del listato 1.1 è decisamente semplice e autoesplicativo. Il contesto viene recuperato attraverso il metodo `getContext(2d)` dell'oggetto canvas. Qualora questo metodo non sia presente è segno che il browser in uso non supporta il tag `<canvas>` e si può scatenare un meccanismo di fallback (ad esempio sostituendone le funzionalità con Flash o simili). Il Context espone quella un'interfaccia di programmazione detta Canvas2D, dei semplici metodi per il disegno in due dimensioni, utilizzati in questo caso per riempire il canvas con tre rettangoli dei colori specificati.

Sebbene il Context "2d" sia il solo definito nella specifica, viene riportato tra i contesti supportati anche "webgl", il cui obiettivo è quello di fornire un'API per il 3D.

1.3 WebGL: il 3d (realmente) nel web

Le prime sperimentazioni di un context 3D per l'elemento canvas iniziano nel 2006 all'interno dei laboratori Mozilla ad opera di Vladimir Vukićević[11]. Qualche anno più tardi, nel 2009, visto l'interesse suscitato, fu istituito il WebGL Working Group. Ad oggi è uno standard cross-platform, cross-browser e royalty-free gestito dal consorzio Khronos Group, responsabile anche di altre importanti tecnologie in campo 3d quale, ad esempio, OpenGL. Nel Marzo 2011 è stata rilasciata la versione 1.0 della specifica[12].

WebGL, acronimo di Web-based Graphics Library, è un'interfaccia di programmazione che consente di creare e gestire contenuti 3d all'interno della pagina web. Questa libreria espone, per mezzo del tag `<canvas>`, una DOM API di basso livello, utilizzabile, dunque, attraverso qualsiasi linguaggio "DOM-compatibile" (vedi Javascript, Java). Queste caratteristiche consentono di mettere da parte i plugin di terze parti (spesso soluzioni proprietarie) e di accedere ad un 3d nativo del web. Un browser WebGL compatibile si occuperà di fornire l'accelerazione grafica sfruttando l'hardware presente sulla macchina, interfacciandosi direttamente con i driver, con risultati performanti e di alta qualità.

Non a caso infatti WebGL poggia su due tecnologie consolidate e ampiamente diffuse quali OpenGL ES2, API per la grafica 2D e 3D in sistemi embedded (console, telefono, veicoli, etc...), e l'OpenGL Shading Language

[11] Corso su WebGL. In *HTML5Today*. Retrieved 12:10, August 14, 2011 from `http://www.html5today.it/tutorial/corso-webgl--introduzione`

[12] WebGL Specification. In *Khronos Group*. Retrieved 9:20, July 29, 2011 from `https://www.khronos.org/registry/webgl/specs/1.0/`

Figura 1.2: Cubo con texture in un canvas renderizzato con WebGL

(GLSL) per la creazione di "Shader"[13]. L'utilizzo di queste tecnolgie fa di WebGL un'interfaccia di basso livello e sebbene da una parte questo porti enormi benefici sul piano dell'efficienza e della qualità, dall'altra introduce un alto livello di complessità, anche per la gestione di contenuti semplicissimi come quelli in Figura 1.2: *"WebGL is a low-level API, so it's not for the faint of heart. OpenGL's shading language, GLSL, is itself an entire programming environment. As a result, even simple things in WebGL take quite a bit of code. You have to load, compile, and link shaders, set up the variables to be passed in to the shaders, and also perform matrix math to animate shapes."*. Il bagaglio di conoscenze richieste per utilizzare questa tecnologia è enorme come lo sono, di certo, lo sforzo e il tempo richiesti per apprenderle, malgrado questo sia poi il compito di uno sviluppatore.

In conclusione WebGL non è uno strumento "ready-to-use" e questo potrebbe non giocare a suo favore. Tuttavia, stanno già nascendo librerie di più alto livello che fungono da "wrapper" e agevolano la vita al programma-

[13]Gli shader sono insiemi di istruzioni, altamente flessibili e riutilizzabili, che definiscono in ogni parte l'aspetto finale che l'oggetto avrà dopo la fase di rendering.

tore[14], mascherando la complessa natura di WebGL e consentendogli dunque di concentrarsi pienamente sulla creazione di contenuti per web. Tra queste vi è sicuramente X3DOM, un framework che tenta di unire il vecchio X3D, consolidata tecnologia di ieri, senza dubbio "user-friendly", con il giovane e complicato WebGL.

1.4 X3DOM

Quanto visto finora con WebgGL consente di introdurre e gestire elementi di grafica 3d nel contesto web utilizzado un paradigma di programmazione imperativo. Ciò che manca è un modo semplice per integrare in maniera dichiarativa contenuti 3d all'interno del DOM di una pagina HTML. Il percorso iniziato dall'X3D Working Group proponeva X3D per una piena integrazione con HTML5. La scelta di questa tecnologia come base per un 3d dichiarativo appare ovvia, in quanto è uno standard basato su XML, consolidato e assai diffuso tra gli sviluppatori. Inoltre è proprio la specifica HTML5 che nella sua promessa di un "declarative-3d" fa riferimento ad X3D, senza specificare però un modello di integrazione. L'obiettivo finale è quello di incorporare una scena X3D all'interno del DOM, come già avvenuto per altri linguaggi XML-based come SVG o MathML, consentendone la manipolazione semplicemente interagendo con i nodi stessi e grazie anche al supporto degli eventi HTML. A questo punto il codice per rappresentare un

[14]GLGE Official Website. In *GLGE*. Retrieved 7:22, August 18, 2011 from http://www.glge.org/

semplice cubo all'interno di una pagina HTML5 con X3D risulta assi snello e leggibile, come mostra il listato 1.2.

Listing 1.2: A simple cube with X3D+HTML5

```
<!DOCTYPE html>
<html>
<head>
<title>X3D example</title>
<link rel="stylesheet" type="text/css" href="http://www.x3dom.org/x3dom/
    release/x3dom.css"></link>
<script type="text/javascript" src="http://www.x3dom.org/x3dom/release/
    x3dom.js"></script>
</head>
<body>
    <h1>X3D - A cube</h1>
    <x3d id='x3d_element' width='880px' height='400px'>
        <scene>
            <shape>
                <appearance>
                    <material diffuseColor='blue'></material>
                </appearance>
                <box></box>
            </shape>
        </scene>
    </x3d>
</body>
</html>
```

Per ottenere il risultato presentato in Figura 1.3 dal listato 1.2 è necessario renderizzare in qualche modo il contenuto X3D della pagina web. Fraunhofer Society, un importante ente di ricerca internazionale, ha lavorato a lungo sia sull'integrazione X3D-HTML5 che sul motore di rendering e il risultato è la suite opensource X3DOM. Questa definisce un modello di integrazione X3D-HTML5 che si può dividere principalmente in due parti: una di interazione con il browser web e una di comunicazione con l'X3D runtime.

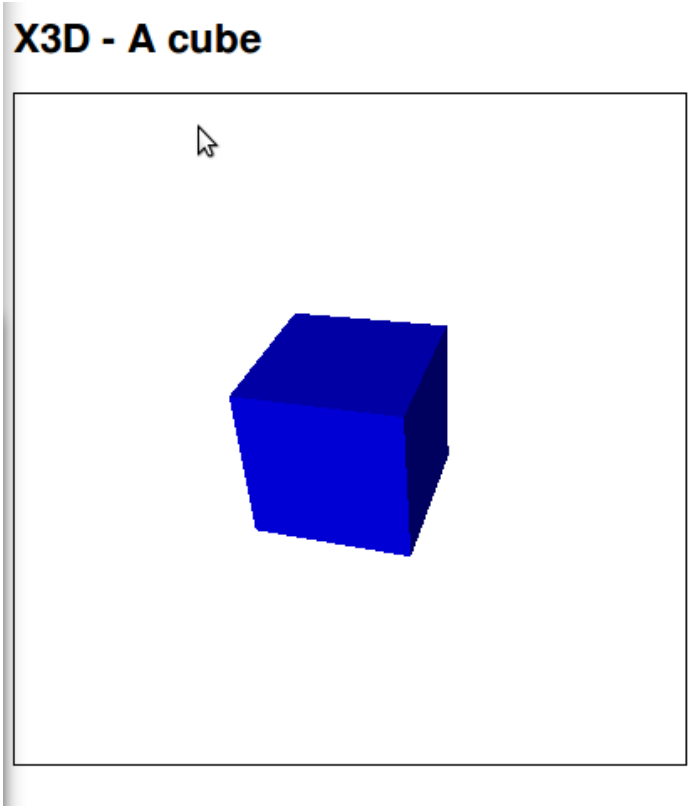

Figura 1.3: Un cubo blu realizzato con X3DOM

L'architettura di X3DOM consente di supportare diversi frontend, ovvero i browser, e backend, vale a dire gli engine con cui la scena viene renderizzata. Su questo fronte vi sono diverse possibilità, attualmente supportate dalla suite X3DOM:

- Rendering con plugin classico tramite plugin X3D-SAI esterno di terze parti.
- Rendering tramite un motore Flash 11, compreso nella suite.
- Rendering nativo grazie a WebGL

Chiaramente gli sforzi sono concentrati su quest'ultima possibilità, lasciando le prime due opzioni per garantire compatibilità con quei browser che non supportano WebGL o con sistemi obsoleti o privi di accelerazione hardware.

Per poter utilizzare X3DOM basta includere nella nostra pagina web l'engine javascript `x3dom.js` e il foglio di stile `x3dom.css`, come si può notare alle righe 5 e 6 del listato 1.2. Sarà questo motore ad occuparsi del recupero della scena X3D dal DOM della pagina, lato frontend, e di sincronizzarla con la rappresentazione della stessa scena lato backend, comunicando, qualora necessario, i cambiamenti in un verso o nell'altro. A seconda del motore di rendering utilizzato, l'engine si preoccuperà di tradurre la scena X3D utilizzando l'interfaccia corretta. Nell'esempio in Figura 1.3 viene utilizzata la Canvas3D API messa a disposizione da WebGL.

Nel capito seguente si affronterà nel dettaglio il software Medea realizzato combinando le tecnologie viste fin qui, allo scopo di metterle alla prova e valutarne la maturità e l'efficienza.

Capitolo 2

L'editor Medea

In questa seconda parte si analizzerà il software Medea realizzato per il progetto. Dapprima verranno illustrati i requisiti funzionali e poi si scenderà nel dettaglio per approfondire la caratteristiche tecniche dell'implementazione data. Alla base ci sono come colonne portanti le tecnologie già ampiamente descritte nella prima parte di questo lavoro. Il compito di questo software è mostrarne potenzialità e limiti.

2.1 Prerequisiti

Per poter utilizzare il software Medea è necessario un browser di ultima generazione che supporti le tecnologie HTML5, WebGL e il relativo Context. Attualmente i layout engines che soddisfano questi requisiti sono Gecko e WebKit vale a dire, elencando solo i browser più popolari che ne fanno uso,

Firefox, Chrome e Safari. É inoltre necessario disporre di una scheda grafica e relativi driver che supportino come minimo la specifica OpenGL 2.0 o in alternativa, per gli utenti Windows, l'ultima versione delle DirectX[1]. Va aggiunto che per usufruire agevolmente del software è necessario, soprattutto per quanto riguarda il rendering 3d, che il sistema non rasenti i requisiti minimi, pena una navigazione lenta che costringe spesso al riavvio del browser. Sono caldamente raccomandate schede grafiche ATI e NVidia con gli ultimi driver al seguito, specialmente per sistemi Linux. I driver disponibili per hardware Intel, malgrado supportino WebGL, utilizzano il rendering via software per la visualizzazione delle scene, che risulta accettabile per immagini di piccole dimensioni con pochi oggetti, ma si rivela assolutamente inadatto per visualizzare ambienti complessi con numerosi poligoni.

2.2 Strumenti di sviluppo

L'ambiente di sviluppo è costituito da una macchina su cui gira il sistema operativo GNU/Linux Ubuntu 11.04. La macchina monta una scheda grafica Intel GMA 4500MHD, un hardware che non brilla certo per le sue prestazioni. Proprio per questo è servito anche ad evidenziare quelli che sono i limiti a livello di fruibilità dell'applicazione.

[1]Lesson 0: Getting started with WebGL. In *Learning WebGL Blog*. Retrieved 21:00, August 31, 2011, from `http://learningwebgl.com/blog/?p=11`

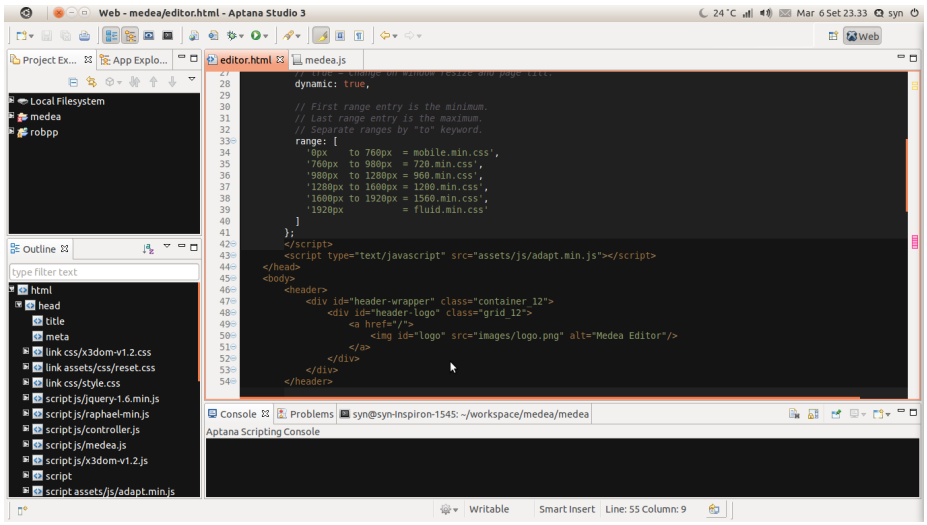

Figura 2.1: Interfaccia di Aptana Studio 3

2.2.1 Aptana Studio IDE

Aptana Studio è un Integrated Development Environment opensource improntato allo svillupo di web applications Ajax. Integra già ora il supporto a tecnologie base quali HTML5 e CSS3, DOM e Javascript. Inoltre fornisce gli strumenti per utilizzare importanti framework per applicazioni web quali Ruby on Rails, PHP, Python e Perl. Basato su un solido progetto quale Eclipse, è gratuito e multipiattaforma, ed offre funzionalita di code-completion, code-assist, debugging avanzato e molte altre.

2.2.2 Blender per il modeling 3D

Lo scoglio più grande da superare quando si parla di modellazione 3d è sicuramente quello di trovare un valido tool di supporto a questo difficile compito, in modo particolare se si ha a che fare con oggetti e scene com-

plesse. In questi casi occorre uno strumento che consenta visivamente di modellare le mesh fin nei minimi dettagli. Vi sono molti software proprietari che potrebbero risolvere il problema, ma il prezzo è spesso proibitorio e non sono multipiattaforma. Viene in nostro soccorso Blender, un software per la grafica 3D gratuito, opensource, che supporta molteplici sistemi operativi.

Nato dalla mente di Ton Roosendal nel 1995 all'interno dello studio di animazione NeoGeo, nel 1998 viene rilasciato come prodotto freeware allo scopo di stimolare interesse da parte degli utenti e quindi offrire servizi di supporto a pagamento. Malgrado l'interesse suscitato e l'enorme community che si era creata intorno, NaN (Not a Number), la società che distribuiva Blender, va in bancarotta nel 2002. Roosendaal però non voleva abbandonare Blender al proprio destino e fonda la "Blender Foundation" con lo scopo di portarne avanti lo sviluppo. Nel Luglio 2002 parte la campagna "Free Blender", il cui intento é raccogliere i fondi (100.000€) per riacquistare i diritti su Blender e rilasciarlo poi come open source. C'è un'enorme risposta da parte della community e il 13 Ottobre 2002, circa due mesi dopo, l'obiettivo é raggiunto. Blender viene rilasciato con licenza GPL.

Oggi Blender è giunto alla versione 2.59 e già si prepara il terreno per la 2.60. É un'applicazione completa per la creazione di contenuti 2D e 3D. Contiene tutti gli strumenti di base per il modeling, rendering, lighting, texturing e animazione, oltre a funzionalità avanzate quali un game engine, physic simulation engine e strumenti per la post-produzione. É estensibile e programmabile grazie ad un motore di scripting Python che ne assicura la

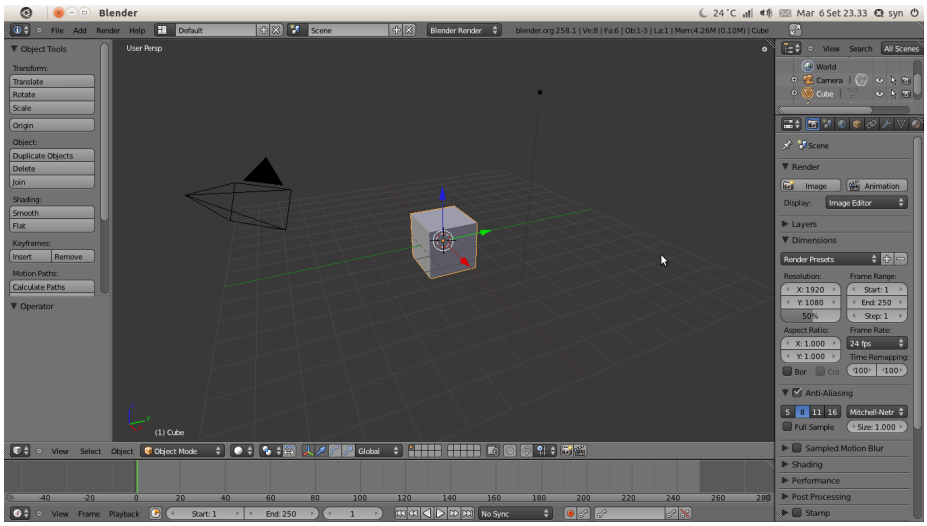

Figura 2.2: Interfaccia di Blender

massima flessibilità.

Blender integra inoltre un comodo exporter verso numerosi formati, tra cui anche X3D. Questa e le caratteristiche prima citate lo hanno reso strumento indispensabile per la realizzazione di questo progetto.

2.2.3 SVG con Raphael JavaScript Library

Merita di essere citata tra gli strumenti utilizzati anche RaphaelJS. Questa piccola libreria da la possibilità allo sviluppatore di creare attraverso codice Javascript elementi di grafica vettoriale all'interno della pagina web. Alla base c'è lo standard SVG, il markup language per la descrizione di contenuti 2D dichiarativi. Raphael espone dei metodi per il disegno che altro non fanno che creare all'interno del DOM il rispettivo scene-graph SVG (vedi il listato 2.1. Ogni oggetto creato nel DOM è connesso ad un oggetto Javascript e può

essere manipolato con facilità, sia grazie agli eventi HTML che da Javascript stesso.

L'obiettivo finale di Raphael, come dice il suo autore Dmitry Baranovskiy, è quello di fornire un libreria per il disegno vettoriale che possa garantire compatibilità cross-browser e al tempo stesso facilità d'uso.

In questo progetto Raphael è stato utilizzato per dar vita ad un controller interattivo che consentisse all'utente di spostare un oggetto all'interno della scena 3d e che fosse allo stesso tempo semplice ed intuitivo. L'intento era anche quello di "pensionare" una volta per tutte i tanto abusati slider.

Listing 2.1: Esempio di utilizzo della libreria RaphaelJS

```
// Crea una tela su cui disegnare 400x300 at 10, 10
var paper = Raphael(10, 10, 400, 300);

// Crea un rettangolo di dimensioni 50x50 alle coordinate 25,150
var rect = paper.rect(25, 150, 50,50);
rect.attr("fill", "#f00"); // Lo riempe con il colore rosso
```

2.3 Caso d'uso

L'idea alla base di questo progetto è quella di fornire all'utente un'ambiente che gli consenta di navigare in un mondo virtuale, con la possibilità di collocarvi degli oggetti a piacere. Le scene a disposizione e gli oggetti faranno parte di una libreria che verrà creata in base alle specifiche esigenze. L'obiettivo del progetto è difatti quello di sviluppare la struttura che consenta

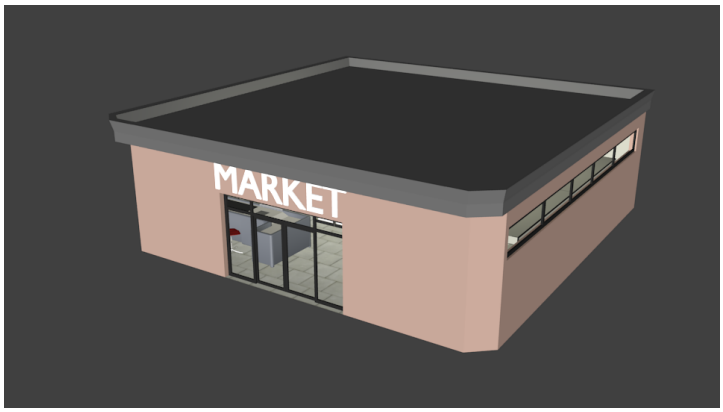

Figura 2.3: Market rendering: esterno

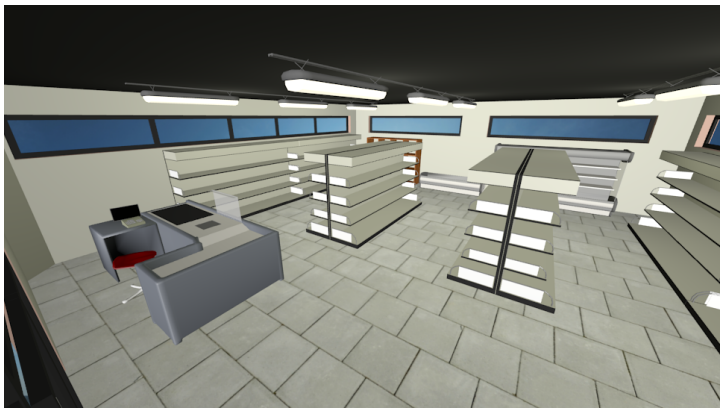

Figura 2.4: Market rendering: interno

di contenere le scene e di inserire e muovere gli oggetti, e non quello di disegnare gli oggetti stessi. Va da se che un caso concreto è stato comunque creato. Come ambiente di prova è stato modellato un minimarket e alcuni dei relativi prodotti. Nelle figure 2.3 e 2.4 si possono vedere due rendering dell'ambiente creato, effettuati con blender.

L'interfaccia, come già anticipato, sarà il browser e tutti i contenuti dovranno essere fruibili direttamente dalla pagina web, che si comporrà di tre componenti fondamentali: il canvas per la scena 3d, un controller per la

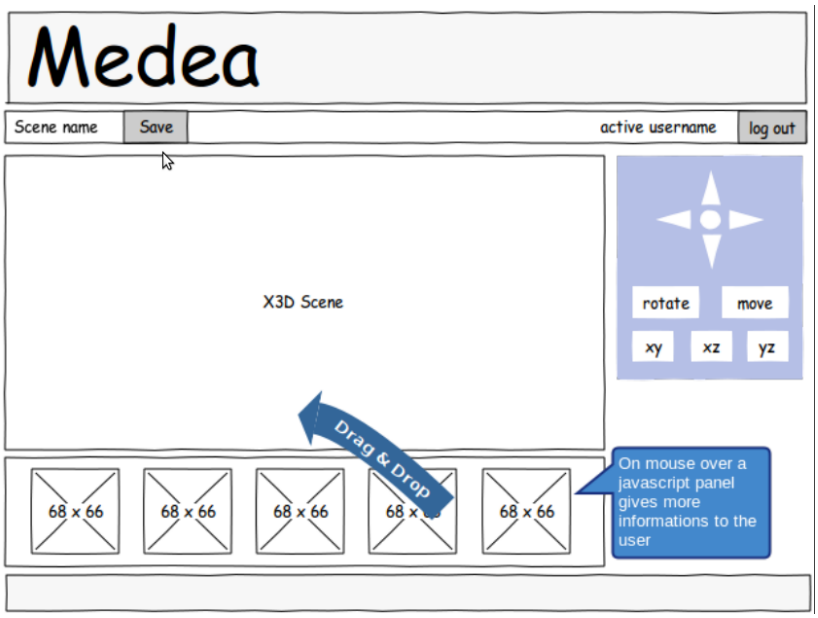

Figura 2.5: Medea User Interface

navigazione, lo spostamento, rotazione e rimozione degli oggetti, e la lista degli oggetti presenti in libreria da poter inserire nella scena. Infine serve un motore che faccia da collante e tenga insieme il tutto. In Figura 2.5 si può vedere una bozza dell'interfaccia utente finale. Non tutte le features presentate saranno implementate in questo progetto.

Quest'interfaccia mira ad essere semplice ed immediata per l'utente, operando alcune scelte fondamentali:

- Utilizzo del Drag 'n Drop HTML5 per l'inserimento degli oggetti di libreria nella scena;
- Controller SVG semplificato ed intuitivo in sostituzione dei classici slider;

2.4 Le caratteristiche tecniche

2.4.1 La struttura

Medea si divide in due parti: client e server. La parte client è tenuta insieme da un motore javascript che si occupa dell'interazione utente in tutti i suoi aspetti. Dalla creazione del controller, alla gestione del darg 'n drop e della posizione degli oggetti nella scena. Il lato server è mantenuto estremamente semplice e al momento si preoccupa solo di caricare dinamicamente la libreria degli oggetti.

Per una maggiore chiarezza della struttura del progetto, di seguito viene mostrata l'organizzazione nel filesystem con le componenti principali in evidenza:

Listing 2.2: Struttura filesystem medea

```
medea
|-- app.py # Applicazione server
|-- config.py # Modulo di configurazione server
|-- static # File statici
| |-- assets
| |-- css
| |-- images
| |-- js
| o-- x3d
| |-- library # Libreria oggetti
| |-- market # Scena minimarket
| o-- textures
|-- templates # Template html utilizzati
| o-- editor.html
o-- view.py # Modulo di rendering dei template html
```

2.4.2 Il server: Webpy framework

Alla base del software Medea c'è Webpy, un framework Python per creare web-application dinamiche. Webpy ci consente di creare un mapping tra url e oggetti python, che serviranno le richieste fatte dal client. Il codice server creato è semplice e di immediata comprensione.

Listing 2.3: Medea server

```
import web, os
import config
from view import render

urls = ('/', 'editor',)

application = web.application(urls, globals())

class editor(object):
    def GET(self):
        lib = parse_library(config.LIBRARY_PATH)
        return render.editor(lib)

def parse_library(path):
    import fnmatch
    files = map(lambda x: os.path.join(path,x), os.listdir(path))

    inlines = sorted(fnmatch.filter(files, "*.x3d"))
    thumbs = sorted(fnmatch.filter(files, "*.png"))

    return zip(inlines, thumbs)
```

Il modulo python `web` è il cuore del framework e ci consente di creare l'oggetto `application` che si occuperà di servire le richieste, secondo il mapping definito a linea 6. In questo caso la mappatura è elementare: sono consentite solo richieste indirizzate alla 'root' e queste vengono servite da

un'istanza della classe `editor`, la quale definisce un comportamento per il solo metodo HTTP GET.

Questo metodo carica tutti i file di libreria, collocata in un path specificato nel modulo di configurazione, chiamando la funzione `parse-library`. Dopodiché renderizza il template `editor.html` passandogli come parametro la lista ottenuta e restituisce il risultato al client. Il listato 2.4 mostra un estratto del template, in particolare la parte in cui viene creata la libreria. Il motore di templating utilizza il carattere '$' come escape per includere codice python all'interno della pagina html. Quando il template viene processato il codice incluso viene riconosciuto ed eseguito per generare la pagina dinamicamente. É possibile passare al template dei parametri in input. In questo caso viene fornita la lista degli oggetti in libreria. Ogni oggetto contiene l'url del file x3d e dell'immagine. Il ciclo `$for x3d, thumb in library` produce un elemento `<li>` con questi dati.

Listing 2.4: Medea: template libreria

```
<div id="library" class="grid_12">
        $if library:
            <ul>
            $for x3d, thumb in library:
                <li>
                    <div class="grid_1 draggable">
                        <img src="$thumb" width="50px"></img>
                        <inline url="$x3d"></inline>
                    </div>
                </li>
            </ul>
        $else:
            <p><big>Your library is empty!</big><p>
</div>
```

2.4.3 Il client: medea.js

Lato client il motore `medea.js` si occupa di inizializzare i componenti che compongono l'interfaccia, connettendo gli eventi ai rispettivi metodi di gestione. Nel codice si fa uso della libreria jQuery per rendere la sintassi meno prolissa e più chiara.

Listing 2.5: Medea client

```
// Loading controller
var controller = Raphael.controller(0, 0, 200, "controller");
var medea = {};
medea.range = 2; // This defines the sensibility of the controller
medea.step = 0.785; // Rotation step
medea.plane = null;
medea.selected = null;
medea.select = function () {
    var highlight = '<shape class="highlight"><sphere radius="0.2"/><
        appearance><material diffuseColor="1 0 0" transparency=".9"/></
        appearance></shape>';

    return function (ev) {
        console.log(ev.target);
        $(".highlight").remove();

        if (ev.target === medea.selected) {
            medea.selected = null;
        } else {
            medea.selected = ev.target;
            $(ev.target).append(highlight);
        }
    };
}();
medea.remove = function () {
    if (!medea.selected) return;

    $(medea.selected).remove();
    $(".highlight").remove();
}
```

```
$("#x3d_element_transform.selectable").click(medea.select);
$("#remove_bt").click(medea.remove);
```

`medea.js` è eseguito alla fine del caricamento della pagina. Il primo ad essere creato è il controller, del quale saranno dati i dettagli implementativi più avanti. Il metodo `Raphael.controller` prende in input la posizione (coordinate x, y), la dimensione e l'id dell'elemento HTML nel quale disegnare il controller. Vengono poi inizializzati alcuni parametri che specificano la sensibilità del controller durante il movimento, il piano su cui l'oggetto si muove e lo step di rotazione degli oggetti in radianti. La rotazione è stata vincolata sul solo asse y a step di 45 gradi per facilitare l'utente nel posizionamento. Segue la funzione `medea.select` che gestisce il click sull'oggetto della scena 3d marcandolo come selezionato, sia internamente, valorizzando la variabile di stato `medea.selected` con il rispettivo oggetto javascript, sia visivamente per l'utente, aggiungendo una sfera semitrasparente intorno all'oggetto. Nelle ultime due istruzioni le funzioni sopra descritte vengno connesse agli eventi degli elementi che li gestiscono. Da notare che per la selezione la funzione `medea.select` viene agganciata ai soli nodi transform che abbiano classe `selectable`, così da impedire all'utente di spostare qualsiasi mesh presente nell'ambiente.

L'inserimento viene gestito grazie al Drag'n Drop che HTML5 implementa nativamente. Nella pratica, ogni elemento che abbia l'attributo `draggable` impostato a `true` è trascinabile. Di seguito il codice che gestisce questa funzionalità.

Listing 2.6: Medea client: il Drag'n Drop

```
//Indica al browser quali elementi sono trascinabili, in questo caso attraverso una
    classe.
$(".draggable").attr("draggable", "true")
//Specifica cosa fare quanto un elemento viene trascinato
.bind('dragstart', function(ev) {
    var dt = ev.originalEvent.dataTransfer;
    var html = $('<div>').append($(this).find("inline").clone()).remove().html();
    dt.setData("node", html);
    return true;
});

//Indica al browser l'elemento in cui puoi trascinare gli oggetti.
$("#drop").bind("dragenter", function (ev) {
    return false;
}).bind("dragleave", function (ev) {
    return false;
}).bind('dragover', function(ev) {
    return false;
}).bind("drop", function (ev) {
    //con questa riga ottengo i dati salvati quando ho iniziato a trascinare l'
        oggetto'
    var tmp = ev.originalEvent.dataTransfer;
    var node = $(medea.transform).append(tmp.getData("node"));

    $(this).find("scene").append(node);

    // Binding click event to selection function
    $(this).find("scene transform.selectable:last").click(medea.select);
    return false;
});
```

Il codice può sembrare di difficile comprensione; in realtà la parte interessante è nelle due funzioni che gestiscono gli eventi "dragstart" e "drop". La prima specifica le azioni da intraprendere quando inizia un'azione di trascinamento. In questo caso viene memorizzato il contenuto html del-

l'elemento trascinato nell'oggetto `dataTransfer`[2] con la chiave node. Nel contenuto html dell'elemento "draggable" è stato preventivamente predisposto dal server il tag `<inline>` che fa riferimento all'oggetto. La funzione legata al "drop" a questo punto non fa altro che recuperare il contenuto del `dataTransfer` con la stessa chiave, lo inserisce in un un nodo `<transform>` selezionabile e lo incorpora nella scena X3D.

Ultime funzioni degne di nota sono `medea.rotate` e `medea.moving` nel listato 2.7. La prima manipola l'attributo `rotation` del nodo selezionato e viene banalmente agganciata al click dei pulsanti con id "rleft" e "rright". La seconda, più interessante, calcola la posizione dell'elemento selezionato sulla base dei due parametri in input `vu` e `vv`. La funzione si aspetta che questi valori siano compresi tra 0 e 1 per essere poi moltiplicati per la variabile `medea.range`, che ne definisce dunque la sensibilità. `medea.moving` viene passata come funzione di "callback" al controller, il quale, ad ogni variazione, la richiamerà impostando opportunamente i due valori richiesti in input.

Listing 2.7: Medea client: movimento e rotazione

```
medea.rotate = function (clockwise) {
    if (!medea.selected) return;

    var r = medea.getRotation($(medea.selected).attr('rotation'));
    r.y = 1;

    r.g += clockwise ? -medea.step : medea.step;
    $(medea.selected).attr('rotation', r.x +" "+ r.y +" "+ r.z +" "+ r.g);
}
$("#rleft").click(function (ev) {
```

[2]L'oggetto dataTransfer può essere visto come un'area di comunicazione condivisa disponibile solo se si verifica uno degli eventi legati al drag'n drop

```
    medea.rotate(true);
});
$("#rright").click(function (ev) {
    medea.rotate(false);
});

medea.moving = function (vu, vv) {
    if (!medea.selected) return;
    var pos = medea.getPos($(medea.selected).attr('translation'));

    pos[medea.plane[0]] += vu*medea.range;
    pos[medea.plane[1]] += vv*medea.range;
    $(medea.selected).attr('translation', pos.x +" "+ pos.y +" "+ pos.z);
}

controller.bind(medea.moving);
```

2.4.4 Il controller

Il componente `controller.js` ha il compito di disegnare il controller SVG per il movimento degli oggetti. Grazie alla libreria RaphaelJS, questo compito può essere svolto interamente dal codice javascript.

Quello che si è cercato di fare è creare un elemento che estende le funzionalità della libreria Raphael con un nuovo componente, istanziabile grazie al metodo `Raphael.controller`. L'oggetto Controller è stato reso indipendente dal progetto Medea ed è quindi sganciabile e riutilizzabile.

Listing 2.8: Medea client: il controller SVG

```
(function (Raphael) {
    Raphael.controller = function (x, y, size, element) {
        return new Controller(x, y, size, element);
    };
```

```
var Controller = function (x, y, size, element) {
    // *** start drawing controller
    size = size || 200;
    var size2 = size / 2,
        size5 = size / 5,
        size10 = size / 10;
        padding = size * 0.02,
        width = size * 0.02;

    var r = element ? Raphael(element, size, size) : Raphael(x, y, size, size);

    // Controller area
    // [... code omitted ...]

    // Events
    // Cursor drag
    var t = this;
    t.cursor.drag(
        function (dx, dy) {
            this.attr({
                cx: Math.min(Math.max(this.ox + dx, lbound), rbound),
                cy: Math.min(Math.max(this.oy + dy, lbound), rbound)
            })

            if (typeof t._movecback === 'function') {
                var vx = (dx / this.ox) - (this.odx / this.ox),
                    vy = (-dy / this.oy) - (this.ody / this.oy);

                t._movecback(vx, vy);
                this.odx = dx, this.ody = -dy;
            }
        },
        function () {
            this.ox = this.attr("cx");
            this.oy = this.attr("cy");

            this.odx = 0;
            this.ody = 0;

            if (typeof t._startcback === 'function') t._startcback();
        },
        function () {
```

```
                this.stop();
                this.attr({cx: this.ox, cy: this.oy, opacity: 1});

                if (typeof t._stopcback === 'function') t._stopcback();
            }
        );
    };

    // Binding callback functions
    Controller.prototype.bind = function (move, start, stop) {
        if (typeof move === 'function') this._movecback = move;

        if (typeof start === 'function') this._startcback = start;

        if (typeof stop === 'function') this._stopcback = stop;
    };
})(window.Raphael);
```

Il listato 2.8 mostra la crezione dell'area di controllo, tralasciando la parte di disegno, specifica della libreria Raphael. Vengono invece riportate i metodi che gestiscono il trascinamento del cursore per mostrare come sono state agganciate le callback. Oltre alla funzione `movecback` è stata prevista la possibilità di fornire, sempre tramite il metodo `Controller.bind`, due funzioni per gli eventi di "dragstart" e "dragstop".

2.4.5 L'interfaccia finale

Non resta altro che mettere insieme i pezzi del puzzle. Con poche righe di codice, si carica il motore x3dom.js, responsabile dell'integrazione e del rendering della scena X3D. Al caricamento della pagina, X3DOM verifica la presenza di un tag `<x3d>` nel DOM della pagina e, qualora fosse presente,

aggiunge un elemento `canvas` per la visualizzazione della scena tramite WebGL. Nel listato 2.9 la scena x3d è inclusa direttamente nella pagina. Ciò non toglie che sarebbe auspicabile una funzione lato server che la carichi dinamicamente all'interno di un `<div>` predefinito, similmente a quanto fatto per la gestione della libreria.

Listing 2.9: Medea: editor.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Medea Scene Editor</title>
        <!-- x3dom -->
        <link rel='stylesheet' type='text/css' href='http://www.x3dom.org/
            x3dom/release/x3dom.css'></link>
        <script type='text/javascript' src='http://www.x3dom.org/x3dom/
            release/x3dom.js'></script>

        <script type="text/javascript" src="/static/js/controller.js"></script>
        <script type="text/javascript" src="/static/js/medea.js"></script>
    </head>
    <body>
        <!-- [ ... code omitted ...] -->

        <div id="drop" class="grid_9">
            <x3d id='x3d_element' showStat='false' showLog='false' x='0px' y
                ='0px' width='880px' height='400px'>
                <scene id="root_scene">
                    <background skyColor='0 0 0'></background>
                    <navigationinfo headlight="true" type="EXAMINE"
                        avatarSize="0.25, 0.75, 0.75"></navigationinfo>
                    <viewpoint DEF='FrontView' description='Front View'
                        position='0.01 0.80 2.80'></viewpoint>
                    <inline url="x3d/market/minimarket_out.x3d"></inline>
                </scene>
            </x3d>
        </div>
```

La Figura 2.6 mostra l'interfaccia utente finale del software Medea, mentre in Figura 2.7 è immortalato un utente in fase di posizionamento dell'oggetto selezionato.

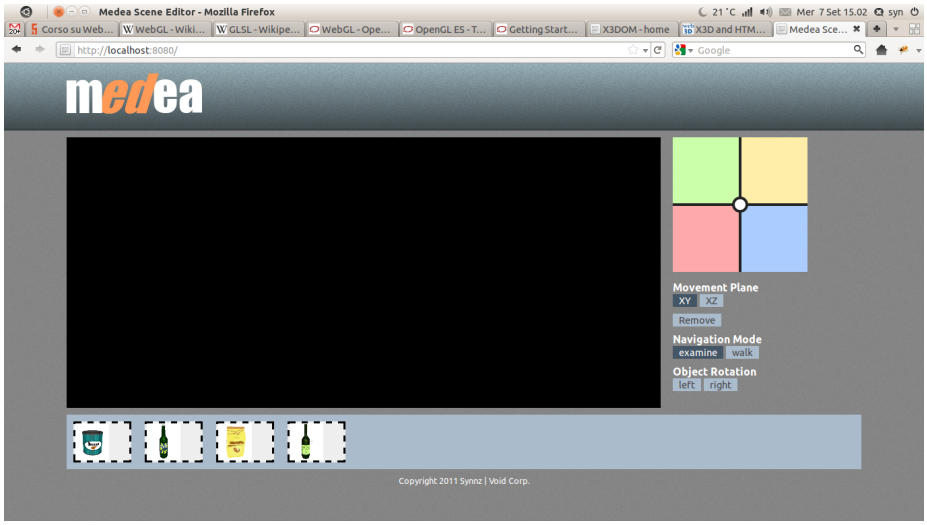

Figura 2.6: Interfaccia utente finale

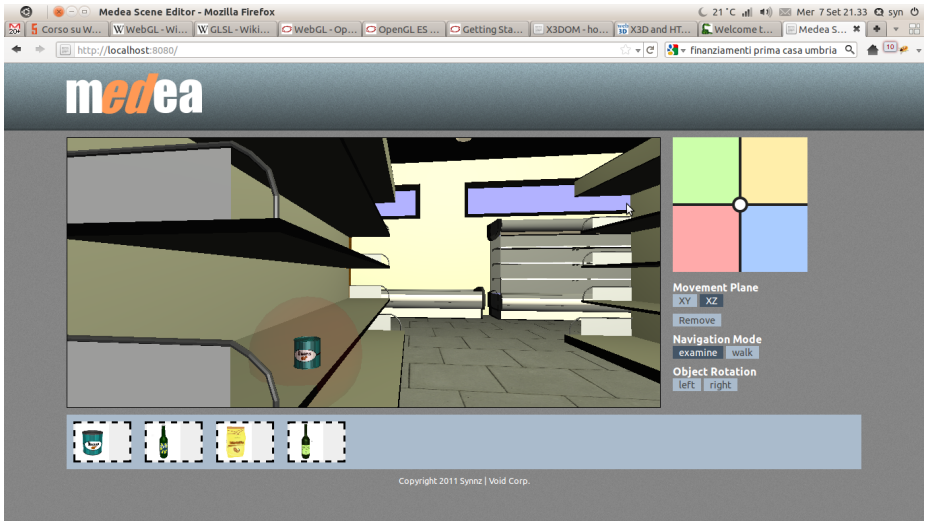

Figura 2.7: Utente durante il posizionamento di un oggetto selezionato

Conclusioni

Durante la progettazione e lo sviluppo del software Medea, ho avuto modo di approfondire quelle che sono le ultimi novità del mondo web. Tecnologie giovanissime come HTML5, WebGL e X3DOM che già riescono a dimostrare l'enorme potenziale che hanno. Nelle varie fasi di sviluppo, la tenera età a volte pesa e i limiti intrinseci vengono a galla. Inefficienza, API poco accessibili, documentazione di sviluppo carente. A compensare tutto questo c'è però una community disponibile e molto attiva. In pochi mesi una libreria come X3DOM ha implementato numerose features e risolto innumerevoli bug, senza contare i miglioramenti apportati alla documentazione. Certo, se si pensa all'entusiasmo che c'era agli albori di X3D e a come fu, lentamente, quasi dimenticato, si potrebbe pensare che la storia stia per ripetersi.

Quello che invece è emerso grazie a questo progetto è che le tecnologie di cui parliamo, questa volta ce la possono fare davvero. Gli scogli che hanno impedito l'affermazione del "vecchio" web3d sono stati superati. L'eliminazione dei plugin avvicina l'utente e i contenuti 3d alla distanza di un semplice click. Anche il contesto è cambiato. L'hardware che fornisce accelerazione grafica è sicuramente più diffuso di un tempo, si potrebbe anzi

affermare che è oramai essenziale.

C'è sicuramente ancora del lavoro da fare. La community di X3DOM sta già lavorando per rendere il motore più efficiente e l'API più completa. Il W3C ha da poco aperto un gruppo di discussione per il 3D dichiarativo, segno che il messaggio è stato recepito. In conclusione, la strada è spianata. Il web sta per acquistare definitivamente una terza dimensione.

Bibliografia

[1] WebGL Official Website http://www.khronos.org/webgl/

[2] Corso su WebGL http://www.html5today.it/tutorial/corso-webgl--introduzione

[3] X3DOM – A DOM-based HTML5/ X3D Integration Model www.web3d.org/x3d/wiki/images/3/30/X3dom-web3d2009-paper.pdf

[4] X3DOM Official Website http://www.x3dom.com

[5] X3D Specification http://www.web3d.org/x3d/specifications/

[6] D. Brutzman & L. Daly X3D: extensible 3D Graphics for Web Authors. Morgan Kaufmann publishers, (2007).

[7] Raphael Javascript Library http://raphaeljs.com/

[8] OpenGL Specification. http://www.opengl.org/documentation/

[9] OpenGL ES 2.0 Specification http://www.khronos.org/opengles/

[10] W3C Community Group on Declarative 3D for the Web Architecture http://www.w3.org/community/declarative3d/

[11] HTML 5 Specification http://dev.w3.org/html5/spec/Overview.html

[12] SVG http://www.w3.org/Graphics/SVG/

Appendice A

Codice

Listing A.1: Medea: app.py

```
import web, os
import config

from view import render

urls = (
        '/', 'editor',
)


app = web.application(urls, globals())


class editor(object):
    def GET(self):
        lib = parse_library(config.LIBRARY_PATH)
        config.LOG.debug(lib)
        return render.editor(lib)


def parse_library(lib):
    import fnmatch
    # list in lib directory and prepend the path
    files = map(lambda x: os.path.join(lib,x), os.listdir(lib))
```

```
    config.LOG.debug(files)

    inlines = sorted(fnmatch.filter(files, "*.x3d"))
    thumbs = sorted(fnmatch.filter(files, "*.png"))

    return zip(inlines, thumbs)


if __name__ == '__main__':
    app.run()
```

Listing A.2: Medea: config.py

```
import web, os
import logging

DEBUG = False

# Setup logging
def setup_logger():
    """ Configure logger """
    logger = logging.getLogger("medea")
    formatter = logging.Formatter(
                    "%(asctime)s - %(name)s - %(levelname)s - %(message)
                        s")

    shandler = logging.StreamHandler()
    shandler.setLevel(DEBUG and logging.DEBUG or logging.WARNING)
    shandler.setFormatter(formatter)

    fhandler = logging.FileHandler("medea.log")
    fhandler.setLevel(DEBUG and logging.DEBUG or logging.WARNING)
    fhandler.setFormatter(formatter)

    logger.addHandler(shandler)
    logger.addHandler(fhandler)
    logger.setLevel(logging.DEBUG)

    return logger

LOG = setup_logger()
```

```
BASE_PATH = os.path.dirname(os.path.abspath(__file__))
TEMPLATES = os.path.join(BASE_PATH, 'templates/')

STATIC_FILES = "static/"
X3D_PATH = os.path.join(STATIC_FILES, 'x3d')
LIBRARY_PATH = os.path.join(X3D_PATH, 'library')

# Web.py caching
cache = False
```

Listing A.3: Medea: view.py

```
import web
import config

t_globals = dict(
  datestr=web.datestr
)

render = web.template.render( 'templates/',
                              cache=config.cache,
                              globals=t_globals)

def editor(**kw):
    """ Renders editor """
    return render.editor(**kw)
```

Listing A.4: Medea: editor.html

```
$def with (library=None)

<!DOCTYPE html>
<html>
    <head>
        <title>Medea Scene Editor</title>
        <meta http-equiv='Content-Type' content='text/html;charset=utf
            -8'></meta>
        <link rel='stylesheet' type='text/css' href='http://www.x3dom.org/
            x3dom/release/x3dom.css'></link>
```

```
<!--<link rel='stylesheet' type='text/css' href='/static/css/x3dom-
    v1.2.css'></link>-->

<!--css-->
<link rel="stylesheet" type="text/css" href="/static/assets/css/reset.
    css" />
<link rel="stylesheet" type="text/css" href="/static/css/style.css" />

<script type="text/javascript" src="/static/js/jquery-1.6.min.js"></
    script>
<script type="text/javascript" src="/static/js/raphael-min.js"></
    script>
<script type="text/javascript" src="/static/js/controller.js"></script>
<script type="text/javascript" src="/static/js/medea.js"></script>
<script type='text/javascript' src='http://www.x3dom.org/x3dom/
    release/x3dom.js'></script>
<!--<script type='text/javascript' src='/static/js/x3dom-v1.2.js'><
    /script>-->

<script type="text/javascript">
// Edit to suit your needs.
var ADAPT_CONFIG = {
  // Where is your CSS?
  path: '/static/assets/css/',

  // false = Only run once, when page first loads.
  // true = Change on window resize and page tilt.
  dynamic: true,

  // First range entry is the minimum.
  // Last range entry is the maximum.
  // Separate ranges by "to" keyword.
  range: [
    '0px to 760px = mobile.min.css',
    '760px to 980px = 720.min.css',
    '980px to 1280px = 960.min.css',
    '1280px to 1600px = 1200.min.css',
    '1600px to 1920px = 1560.min.css',
    '1920px = fluid.min.css'
  ]
};
</script>
<script type="text/javascript" src="/static/assets/js/adapt.min.js"></
```

```
        script>
</head>
<body>
    <header>
        <div id="header-wrapper" class="container_12">
            <div id="header-logo" class="grid_12">
                <a href="/">
                    <img id="logo" src="/static/images/logo.png" alt="
                        Medea_Editor"/>
                </a>
            </div>
        </div>
    </header>

    <div id="content" class="container_12_clearfix">
        <div id="drop" class="grid_9">
            <x3d id='x3d_element' showStat='false' showLog='false' x='0px'
                 y='0px' width='880px' height='400px'>
                <scene id="root_scene">
                    <background skyColor='.7 .7 1'></background>
                    <navigationinfo headlight="false" type="EXAMINE"
                        avatarSize="0.25,_0.75,_0.20"></navigationinfo>
                    <viewpoint DEF='FrontView' description='Front View'
                        position='0 1 2'></viewpoint>

                    <inline url="/static/x3d/market/minimarket_out.x3d">
                        </inline>
                </scene>
            </x3d>
        </div>

        <!-- Scene controls -->
        <div id="side" class="grid_3">
            <div id="controller">

            </div>

            <div id="planes" class="controls">
                <h1>Movement Plane</h1>
                <button id="xy" type="button">XY</button>
                <button id="xz" type="button">XZ</button>
            </div>
            <div class="controls">
```

```
                    <button id="remove_bt" type="button">Remove</
                        button>
                </div>

                <div id="nav_mode" class="controls">
                    <h1>Navigation Mode</h1>
                    <button id="examine" type="button">examine</button
                        >
                    <button id="walk" type="button">walk</button>
                </div>
                <div id="rotation" class="controls">
                    <h1>Object Rotation</h1>
                    <button id="rleft" type="button">left</button>
                    <button id="rright" type="button">right</button>
                </div>
            </div>

            <!-- Object Library -->
            <div id="library" class="grid_12">
                    $if library:
                        <ul>
                        $for x3d, thumb in library:
                            <li>
                                <div class="grid_1_draggable">
                                    <img src="$thumb" width="50px"></
                                        img>
                                    <inline url="$x3d"></inline>
                                </div>
                            </li>
                        </ul>
                    $else:
                        <p><big>Your library is empty!</big><p>
            </div>
        </div>

        <footer>Copyright 2011 Synnz | Void Corp.</footer>
    </body>
</html>
```

Listing A.5: Medea: style.css

```
body {
```

```
    background-image: url("../images/body-bg.png");
    font-family: "Ubuntu", Arial, Helvetica, sans-serif;
    font-style: normal;
    color: white;
}

header {
    background-image: url("../images/header-bg.png");
    background-repeat: repeat-x;
    height: 100px;
}
footer {
    margin-top: 10px;
    text-align: center;
    font-size: 80%;
}

#logo {
    top: 15px;
    position: absolute;
}

button {
    background-color: #aabbcc;
    border-style: none;
}
button:active, button.checked {
    background-color: #445566;
    color: white;
}

nav {
    margin-top: 10px;
    margin-bottom: 10px;
    font-weight: bold;
}
ul li {
    margin-right: 10px;
    display: inline;
}
nav ul li:hover {
    cursor: default;
    color: #445053;
```

```
}

#library {
    background-color: #aabbcc;
    margin-top: 10px;
}

#content {
    margin-top: 10px;
}

.controls {
    padding-top: 10px;
}

.draggable {
    background-color: #eeeeee;
    /*width: 100px;
    height: 60px;*/
    border: 3px #000000 dashed;
    margin: 10px 10px;
}
```

Listing A.6: Medea: medea.js

```
$(function () {
    // Loading controller
    var controller = Raphael.controller(0, 0, 200, "controller");

    // object selection and movement
    var medea = {};
    medea.transform = '<transform class="selectable" translation="0 1 0"/>';
    //TODO: put a slide in the UI to control this parameter
    medea.range = 1; // This defines the sensibility of the controller
    medea.step = 0.785; // Rotation step
    medea.plane = null;
    medea.selected = null;
    medea.select = function () {
        var highlight = '<shape class="highlight"><sphere radius="0.2"/><
            appearance><material diffuseColor="1 0 0" transparency
            =".9"/></appearance></shape>';
```

```
    return function (ev) {
        console.log(ev.target);
        $(".highlight").remove();

        if (ev.target === medea.selected) {
            medea.selected = null;
        } else {
            medea.selected = ev.target;
            $(ev.target).append(highlight);
        }
    };
}();
medea.remove = function () {
    if (!medea.selected) return;

    $(medea.selected).remove();
    $(".highlight").remove();
}

//TODO: maybe there is a better (and reliable) way to get the object
    translation
// need to investigate x3dom
medea.getPos = function (coords) {
    var pos = {x: 0, y: 0, z: 0};
    if (coords) {
        var str = coords.split(" ");
        pos.x = parseFloat(str[0]);
        pos.y = parseFloat(str[1]);
        pos.z = parseFloat(str[2]);
    }

    return pos;
}
medea.getRotation = function (coords) {
    var rotation = {x: 0, y: 0, z: 0, g: 0};
    if (coords) {
        var str = coords.split(" ");
        rotation.x = parseFloat(str[0]);
        rotation.y = parseFloat(str[1]);
        rotation.z = parseFloat(str[2]);
        rotation.g = parseFloat(str[3]);
    }
```

```
    return rotation;
}
medea.moving = function (vu, vv) {
    if (!medea.selected) return;

    // Delta
    var pos = medea.getPos($(medea.selected).attr('translation'));

    // TODO: i'd like to implement continuos movement while the cursor is
        grabbed aroud
    // varying movement speed based on vx and vy

    pos[medea.plane[0]] += vu*medea.range;
    pos[medea.plane[1]] += vv*medea.range;
    $(medea.selected).attr('translation', pos.x +" "+ pos.y +" "+ pos.z);
}
medea.rotate = function (clockwise) {
    if (!medea.selected) return;

    var r = medea.getRotation($(medea.selected).attr('rotation'));
    r.y = 1;

    r.g += clockwise ? -medea.step : medea.step;
    $(medea.selected).attr('rotation', r.x +" "+ r.y +" "+ r.z +" "+ r.g);
}

controller.bind(medea.moving);


// On scene loading i need to bind all selectable transform to the select
    function
$("#x3d_element_transform.selectable").click(medea.select);
$("#remove_bt").click(medea.remove);


// Drag&drop handling
//Cambia il cursore sugli elementi trascinabili per renderli visibili
$(".draggable").css("cursor", "move");

$(".draggable")
//Indica al browser quali elementi sono trascinabili, in questo caso attraverso
    una classe.
.attr("draggable", "true")
```

```
//Specifica cosa fare quanto un elemento viene trascinato. In questo caso salvo
      il contenuto tramite
// innerHtml
.bind('dragstart', function(ev) {
    var dt = ev.originalEvent.dataTransfer;
    var html = $('<div>').append($(this).find("inline").clone()).remove().
        html();
    dt.setData("node", html);
    return true;
});

//Indica al browser l'elemento in cui puoi trascinare gli oggetti.
$("#drop")
//Dragenter indica l'evento "entro nel target mentre sto trascinando qualcosa"
.bind("dragenter", function (ev) {
    //$(this).css("background-color", "white");
    return false;
})
//Dragleave indica l'evento "esco dal target mentre sto trascinando qualcosa"
.bind("dragleave", function (ev) {
    //$(this).css("background-color", "#acd6ec");
    return false;
})
//Dragover indica l'evento "sono sul target mentre sto trascinando qualcosa"
.bind('dragover', function(ev) {
    //$(this).css("cursor", "copy");
    return false;
})
//Dragover indica l'evento "deposito l'elemento che ho trascinato nel target".
.bind("drop", function (ev) {
    //$(this).css("background-color", "#acd6ec");
    //con questa riga ottengo i dati salvati quando ho iniziato a trascinare l'
        oggetto'
    var tmp = ev.originalEvent.dataTransfer;
    var node = $(medea.transform).append(tmp.getData("node"));

    $(this).find("scene").append(node);

    //console.log(window.x3dom.runtime.viewpoint());
    //console.log($("#x3d_element").runtime.getActiveBindable('Viewpoint'))
        ;

    // Binding click event to selection function
```

```
    $(this).find("scene transform.selectable:last").click(medea.select);
    return false;
});

// Plane selector
var planeToggle = function (plane) {
    medea.plane = plane.toLowerCase();

    console.log("Active plane: "+ medea.plane);
}

// Handles the radio button for plane selection
$("#planes").click(function (ev){
    if ($(ev.target).hasClass("checked")) return;
    $("#planes > button").toggleClass("checked");

    planeToggle($(ev.target).html());
});

// Navigation mode
$("#nav_mode").click(function (ev) {
    if ($(ev.target).hasClass("checked")) return;
    $("#nav_mode > button").toggleClass("checked");

    window.x3dom.runtime[$(ev.target).html()]();

    console.log("Nav mode: "+ $(ev.target).html())
});

// Rotation controllers
$("#rleft").click(function (ev) {
    medea.rotate(true);
});
$("#rright").click(function (ev) {
    medea.rotate(false);
});

// Initial setup
// TODO: create a init() function

// set xy as the active plane
$("#xy").toggleClass("checked");
planeToggle($("#xy").html());
```

```
    // set navigation mode to examine
    $("#examine").toggleClass("checked");
    //window.x3dom.runtime.examine();
});
```

Listing A.7: Medea: controller.js

```
(function (Raphael) {
    Raphael.controller = function (x, y, size, element) {
        return new Controller(x, y, size, element);
    };

    var Controller = function (x, y, size, element) {
        // *** start drawing controller
        size = size || 200;
        var size2 = size / 2,
            size5 = size / 5,
            size10 = size / 10;
            padding = size * 0.02,
            width = size * 0.02;

        var r = element ? Raphael(element, size, size) : Raphael(x, y, size, size);

        // Controller area
        var rect = r.rect(-size2 + padding, -size2 + padding, size - padding,
            size - padding);
        rect.attr({stroke: "#222", "stroke-width": width, fill: "#cfa"});

        var area = r.set();
        area.push(rect,
            rect.clone().translate(size - padding, 0).attr({fill: "#fea"}),
            rect.clone().translate(0, size - padding).attr({fill: "#faa"}),
            rect.clone().translate(size - padding, size - padding).attr({fill: "#acf
                "})
        );
        //area.rotate(45, size2, size2);
        this.area = area;

        // Controller cursor
        var cursor = r.circle(size2, size2, size10/2);
        cursor.attr({stroke: "#222", "stroke-width": width, fill: "#fff"});
```

```
this.cursor = cursor;
// *** end drawing controller

// Private callback on cursor drag
this._movecback = null;
this._startcback = null;
this._stopcback = null;

// Events
// Cursor drag
var lbound = padding*2;
var rbound = size - lbound;
var t = this;
this.cursor.drag(
    function (dx, dy) {
        this.attr({
            cx: Math.min(Math.max(this.ox + dx, lbound), rbound),
            cy: Math.min(Math.max(this.oy + dy, lbound), rbound)
        })

        if (typeof t._movecback === 'function') {
            var vx = (dx / this.ox) - (this.odx / this.ox),
                vy = (-dy / this.oy) - (this.ody / this.oy);

            t._movecback(vx, vy);
            this.odx = dx, this.ody = -dy;
        }
    },
    function () {
        this.ox = this.attr("cx");
        this.oy = this.attr("cy");

        this.odx = 0;
        this.ody = 0;

        if (typeof t._startcback === 'function') t._startcback();
        /*
        // makes the cursor blinking
        var that = this;
        var glowing = function () {
            that.animate(
                {"50%": {opacity: .2},
                 "100%": {opacity: 1, callback: glowing}}, 1000);
```

```
                    };
                    glowing();
                    */
                },
                function () {
                    this.stop();
                    this.attr({cx: this.ox, cy: this.oy, opacity: 1});

                    if (typeof t._stopcback === 'function') t._stopcback();
                }
            );
    };

    // Binding callback functions
    Controller.prototype.bind = function (move, start, stop) {
        if (typeof move === 'function') this._movecback = move;

        if (typeof start === 'function') this._startcback = start;

        if (typeof stop === 'function') this._stopcback = stop;
    };
})(window.Raphael);
```