UNIVERSITÀ DEGLI STUDI ROMA TRE

Dipartimento di Ingegneria
Corso di Laurea in Ingegneria Informatica

Tesi Di Laurea

Strumenti per la navigazione di mappe geografiche nella vista prospettica

Laureando

Cristiano Tofani

Matricola 445202

Relatore

Prof. Maurizio Patrignani

Anno Accademico 2014/2015

A mia zia e ai miei nonni,
perchè anche se da lontano
mi siete e sarete sempre accanto

Ringraziamenti

Grazie a tutti

Introduzione

Molto spesso utilizziamo le mappe o ci imbattiamo nell'annoso problema di voler visualizzare un'area ben definita di una mappa su un browser web, il problema che ci troviamo davanti è che l'area visualizzata è ridotta o non corretta.

Sono diversi i servizi di visualizzazione di mappe presenti in rete, ma in nessuno di essi è possibile visualizzare la mappa con una vista prospettica. Più in particolare in nessuno è possibile visualizzare la mappa *stradale* con una prospettiva di 45 gradi; Google Maps permette di utilizzare le sue mappe *satellitari* con un'inclinazione variabile tra i 15 e i 45 gradi, andando oltre la semplice visualizzazione della mappa, mostrando la struttura tridimensionale della terra e degli edifici, questo, però, risulta spesso fuorviante o, in alcuni casi, anche troppo pesante per il proprio computer.

Indice

Elenco delle figure

Capitolo 1

Servizi di visualizzazione di carte geografiche

Per *servizio di visualizzazione di carte geografiche* si intende un sistema, disponibile in rete, che permette l'utilizzo e la consultazione di mappe geografiche. Nel corso di questo capitolo verrà illustrato il funzionamento e le caratteristiche principali dei maggiori competitor del settore.

1.1 OpenStreetMap

OpenStreetMap (OSM) è molto più di un semplice servizio di visualizzazione di mappe, volendo citare proprio la comunità che ha creato e gestisce il progetto: "OpenStreetMap is a map of the world, created by people like you and free to use under an open license" (*OpenStreetMap è una mappa del mondo, creata da persone come te e utilizzata sotto licenza libera*).[Fou15]

OSM, come si può capire dalla citazione, è un progetto sviluppato e realizzato da una comunità di persone. Fondato nel 2004 da Steve Coast con lo scopo di creare una mappa libera del mondo, nel 2006 ebbe inizio il processo per la trasformazione della società in fondazione non a scopo di lucro, da cui anche la rinominazione in **OpenStreetMaps Foundation**.

La *"mappa"* negli anni ha subito diverse modifiche e soprattutto è cresciuta grazie

alle donazioni non solo di utenti privati, ma sopratutto di enti e aziende quali ad esempio *Automotive Navigation Data*, che nel 2007 donò il database stradale completo dei Paesi Bassi e quello delle principali strade di India e Cina.

La particolarità e l'unicità di questo progetto è sicuramente la sua licenza. Il database è pubblicato secondo la *Open Database License*, la licenza che permette di condividere e adattare il database e creare opere a partire dallo stesso. Le uniche condizioni da rispettare secondo la *ODbL* sono:

- attribuire l'appartenenza del database ad ogni suo utilizzo e ad ogni utilizzo di una banca dati derivata;
- se viene pubblicato il database con una modifica rispetto all'originale o si creano nuovi database a partire da esso è obbligatorio distribuire il nuovo database secondo la licenza *OBdL*;
- anche se il database venisse redistribuito in una nuova versione che ne restringa la libertà d'utilizzo, deve rimanere disponibile una versione aperta del database priva delle restrizioni.

Allo stesso modo del database, anche i dati inseriti e caricati dagli utenti devono rispettare una licenza compatibile con la Creative Commons e gli stessi contributori devono essere registrati al progetto.

Le modifiche operate sulla mappa vengono registrate in un elenco visibile sul sito web della fondazione, come visibile nella figura 1.1.

I dati di *OpenStreetMap* sono utilizzati inoltre da uno dei principali provider di questo settore, ovvero **Mapbox**. L'utilizzo di quest'ultimo lo abbiamo inconsciamente visto diverse volte navigando su internet, su siti web come *Foursquare*, *Pinterest*, *Evernote*, *Financial Times* e *National Geographic.*[Map15]

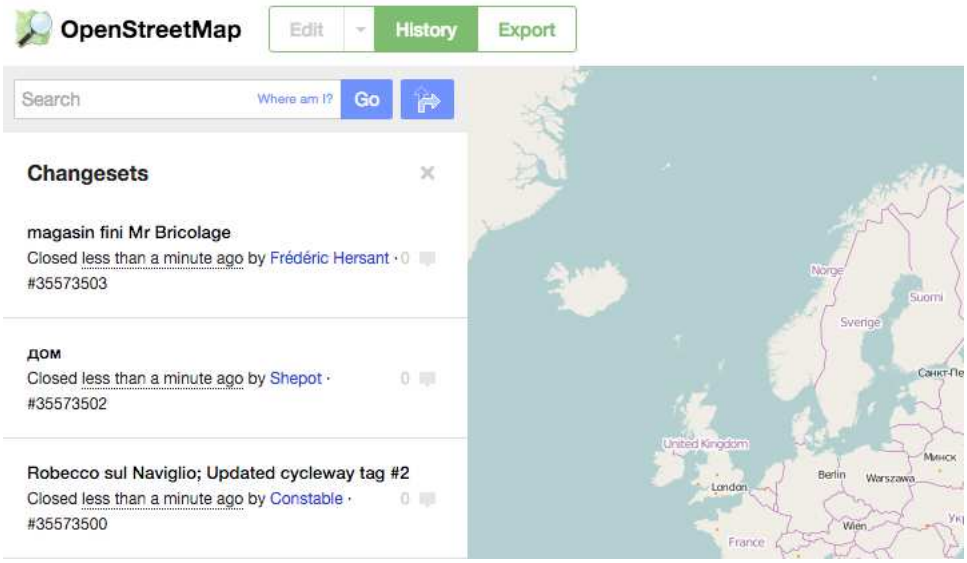

Figura 1.1: Records di modifica della mappa ad opera di utenti

1.2 Bing Maps

Dopo aver visto un servizio *open source*[1] come *OSM*, è importante vedere anche il competitor forse più *"demonizzato"* dal mondo delle licenze libere. Infatti in questa sezione, descriveremo il funzionamento e le caratteristiche di **Bing Maps**, il sistema di *web mapping*[2] creato e sviluppato dalla Microsoft.

In questo servizio a differenza di *OpenStreetMap* le mappe non sono curate da una comunità di developer ma raccolte ed elaborate in diversi modi. Anche Bing Maps come *Mapbox* attinge ai database di OSM per le mappe stradali, che rappresentano la visualizzazione di default, ovvero la mappa che possiamo vedere appena accediamo alla pagina web del portale di Microsoft. È necessario però sottolineare, che a differenza dei due servizi sopra citati - OSM e Mapbox -, quello di Microsoft offre altre possibilità di visualizzazione delle cartine, consentendone anche una vista aerea, una cosiddetta "*bird's-eye view*" (vista a volo d'uccello, che ne permette una visualizzazione prospettica di terreni e edifici), una veduta stradale che consente di vedere la strada a 360 gradi e infine troviamo la possibilità di accedere a mappe 3D in cui a prendere forma, oltre

[1] Open Source[Wik15d]
[2] Con "Web Mapping" si intende il processo di utilizzo o offerta di mappe sul World Wide Web

ai terreni e alle montagne, ci sono anche palazzi e monumenti. Di seguito vengono illustrati alcuni esempi di visualizzazione delle mappe di Bing:

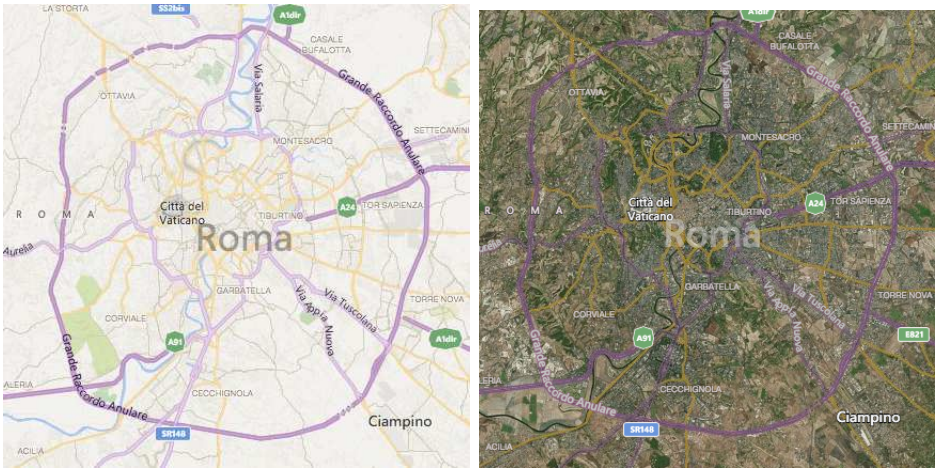

Figura 1.2: Esempi di mappa stradale (a sinistra) e mappa aerea (a destra)

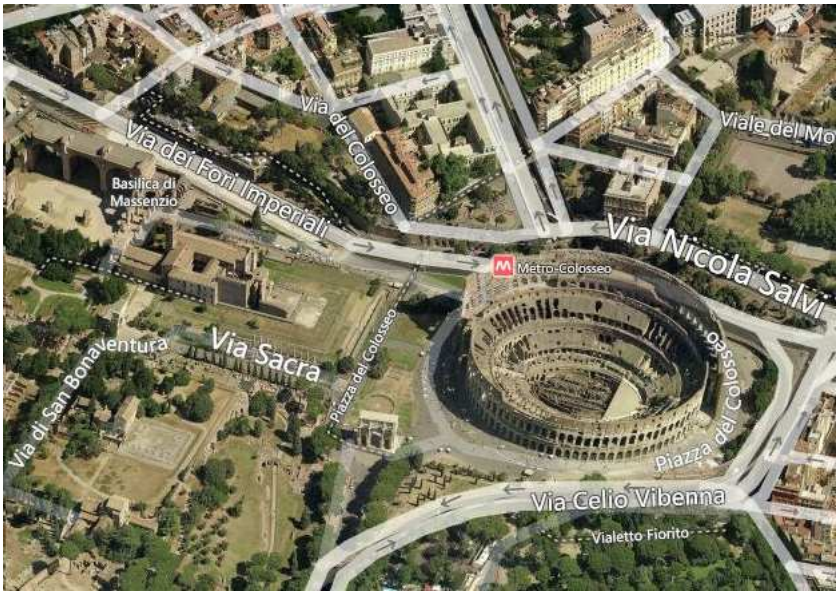

Figura 1.3: Esempio di visualizzazione Bird's-eye

Una delle peculiarità probabilmente più interessanti di Bing Maps, comunque, è la presenza al suo interno di differenti "Map Apps" delle vere e proprie applicazioni interne al servizio che ne sfruttano funzionalità e informazioni e ne arricchiscono così anche

l'esperienza d'uso da parte dell'utente finale. Per fornire un esempio di questo utilizzo, si può far riferimento all'edizione del 2010 del *Tour de France* che utilizzava la mappa Bing per mostrare risultati e tracciati delle differenti tappe della nota competizione ciclistica.[Bin10]

1.3 Google Maps

Dopo aver trattato di *OpenStreetMap* e *Bing Maps*, era inevitabile parlare di quello che è, molto probabilmente, il sistema di visualizzazione cartografico più utilizzato nel mondo. Infatti in questo paragrafo, verrà descritto e analizzato *Google Maps*, che è il servizio di mappe offerto e creato da *Google Inc.*. Nato e reso disponibile nel 2005, a differenza dei due precedenti servizi non era stato ideato e pensato come servizio online, ma doveva essere un programma scritto in *C++*[3] e ideato dal cofondatore di *Google* Lars Rasmussen e suo fratello Jens Eilstrup Rasmussen, successivamente il progetto venne convertito e rilasciato online come tutt'oggi lo conosciamo.

Le funzionalità di visualizzazione sono molto simili ai due servizi prima descritti. Infatti anche *Google Maps* come *Bing Maps*, offre agli utenti la possibilità di utilizzare e consultare mappe con vista satellitare, stradale o ibrida[4]. Negli anni *Google Maps* ha esteso molto le sue funzionalità e le sue caratteristiche. Nel tempo, infatti la società di Mountain View[5] ha unito molti dei suoi progetti o prodotto *ad hoc* servizi aggiuntivi, per aggiungerli al proprio servizio di mappe. Di seguito verranno elencati i principali servizi aggiuntivi offerti da *Google Maps*:

- **Google Traffic**: è un servizio disponibile in oltre di 50 paesi, che permette all'utente di conoscere la situazione del traffico in tempo reale;
- **Google Transit**: è il sistema che permette all'utente la pianificazione di un itinerario utilizzando esclusivamente i mezzi pubblici partito nel 2007, ad oggi serve centinaia di città in tutto il mondo;

[3] Il C++ è un linguaggio di programmazione orientato agli oggetti[Wik15b]

[4] La visualizzazioen ibrida permette all'utente di vedere il terreno con vista satellitare avendo anche il dettaglio delle strade

[5] La sede principale di *Google Inc.* si trova a Mountain View in California

- **Google Places**: attraverso questo servizio si può segnare sulla mappa una qualsiasi attività presente sul territorio (negozi, ristoranti, studi, ambulatori). I luoghi segnalati, verranno memorizzati e sarà disponibile una scheda dell'attività sotto la barra di ricerca ogin volta che selezioneremo il luogo sulla mappa;

- **Google Street View**: è probabilmente la caratteristica più conosciuta del sistema Maps. Il suo funzionamento permette di poter percorrere le strade di numerose città in tutto il mondo, immedesimandosi nella macchina di google che periodicamente percorre le strade per aggiornare le immagini.

Ho introdotto questo sistema di visualizzazione descrivendo come fosse, probabilmente, il più utilizzato in tutto il mondo. È importante, infatti, citare alcune delle applicazioni o società che utilizzano le API[6] di *Google Maps*. Infatti tra di esse spiccano i nomi di *Whatsapp*, *Airbnb*, *Runtastic* o il *New York Times*[Inc15a]

Entriamo, dunque, nel mezzo del mio progetto, perchè per realizzare il mio sistema di visualizzazione di mappe ho utilizzato proprio le API fornite da *Google*, che avrò modo di illustrare maggiormente nel capitolo successivo.

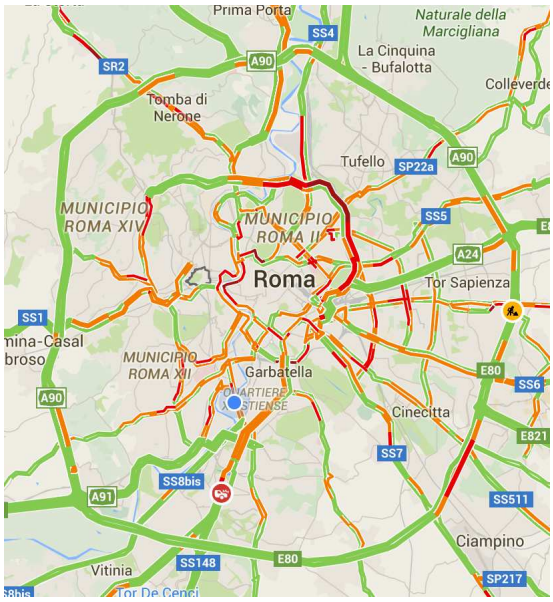

Figura 1.4: Applicazione di Google Traffic sulla mappa di Roma

[6] Application Programming Interface[Wik15a]

Capitolo 2

Tecnologie utilizzate

2.1 Node.js, un framework javascript server-side

Per creare e strutturare una web app è necessario, molto spesso, seguire paradigmi e framework differenti tra di loro. Non è, in alcuni casi, sufficiente conoscere un linguaggio unico per creare la nostra applicazione. Se si pensa alla struttura di un tradizionale sito web, occorre dunque conoscere svariati linguaggi o metalinguaggi di programmazione come Html, Javascript e CSS per il solo sviluppo *front end*[1] e di uno tra quelli più tradizionali come PHP, Pearl, Java e molti altri, rivolti invece allo sviluppo *back end*[2]. Molto spesso è, inoltre, necessario conoscere anche linguaggi come PL/SQL che consentono la gestione e gli accessi del database.

Node.js ha la sua praticità proprio in questo, ovvero la possibilità di unire in una sola figura i due profili prima descritti di sviluppatore *back end* e *front end*, perchè le conoscenze di un web designer esperto di *Javascript* possono essere applicate senza problemi al framework. Il vero e proprio punto di forza di *Node.js* rimane la sua scalabilità, ovvero la possibilità di poterlo applicare a qualunque tipo di progetto, da uno più grande e strutturato fino ad uno più piccolo e semplice.

Analizzando la struttura di questo framework, possiamo vedere come alla sua base si trova il motore **Javascript V8** di **Google**. V8 è un motore di interpretazione,

[1] Si intende con sviluppo front end la parte di programmazione rivolta al client, ovvero la componente grafica del sito web

[2] Si intende con sviluppo back end la programmazione della componente logica del sito web o della applicazione web, ovvero lo sviluppo rivolto al lato server

sviluppato da *Google*, originariamente per poter leggere e compilare codici Javascript all'interno del proprio browser proprietario[3], però grazie alle sue altissime performance si è permesso anche di eseguirlo stand alone sui server. Il *Javascript V8 engine* ha nella diretta compilazione in linguaggio macchina di Javascript la sua maggiore caratteristica, ed è proprio questa possibilità che lo rende altamente performante. Un'altra importante proprietà è la portabilità di V8, che unito al suo stato *open source*, rende V8, ed indirettamente *Node.js*, stabile, sicuro ed eseguibile su qualsiasi sistema operativo.

Un'ulteriore importante caratteristica di questo framework, è la sua semplicità. Con poche righe è possibile creare la struttura base del server. Questo è un piccolo esempio di creazione del server in Node.js:

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

Un'altra caratteristica altrettanto importante è il suo carattere asincrono. La programmazione in *Javascript* permette, rispetto ad altri linguaggi tradizionali, la possibilità di essere affrontata in maniera asincrona.

Prima di vedere nel dettaglio cosa sia una programmazione asincrona, è bene analizzare come un tradizionale server **PHP** affronta e gestisce le richieste. In genere, in una web app, ci sono uno o più server in ascolto su una porta specifica in attesa di una richiesta. Nel momento in cui questa arriva viene attivato un nuovo processo sul server o un thread che fa partire l'elaborazione dell'interrogazione. Questo processo molto spesso deve contattare un altro server di elaborazione esterna, un database o magari il file system stesso. Solamente a elaborazione conclusa, il thread verrà segnalato come disponibile e allora sarà possibile gestire nuove richieste. Se pensiamo alle milioni di richieste che provengono dai siti web e dalle applicazioni di tutto il mondo, è facile capire come non sia in tutti i casi raccomandabile e utile utilizzare un processo come questo, nonostante sia il più lineare e semplice. Proprio per questo motivo, un approccio

[3]Google Chrome

asincrono aiuta a gestire molto più efficientemente, le richieste che sopraggiungono al server. In questo contesto si colloca anche *Node.js*. Il server *Node.js*, grazie alla sua struttura *event-driven*, supera la struttura *multi thread* messa in evidenza prima con il riferimento al server PHP, portando l'esecuzione di più azioni su un singolo thread. Si ha dunque lo svolgimento contemporaneo di più azioni, a differenza dei sistemi sincroni dove un'operazione viene eseguita solamente alla conclusione della precedente. Di seguito vengono mostrate i due tipi di chiamata, nel primo caso sincrona, nel secondo caso asincrona.

```
\\chiamata sincrona
function getData(url)
var a = getData("www.ricevidato.it");
alert(a);

\\chiamata asincrona
function getData(url, callback){...}
getData("www.ricevidato.it", function(result){
        alert(result);
});
```

2.2 WebGL: Web Graphics Library

Per il client del mio progetto ho utilizzato una libreria Javascript dedicata alla grafica tridimensionale e bidimensionale, o meglio una API da essa derivata di cui, però, parlerò in seguito nel corso di questo paragrafo. Analizziamo ora *WebGL* che come si può evincere dal nome, **Web Graphics Library**, è una libreria grafica studiata proprio per il web. Uno dei punti di forza indiscutibili di questa *interface* è sicuramente la possibilità di essere utilizzato in differenti contesti. *WebGL* è una libreria basata su OpenGL ES 2.0[Gro15], di cui ne utilizza il linguaggio di programmazione per la gestione dello *shader*[4], il GLSL.

[4]Il termine shader indica uno strumento della computer grafica 3D che generalmente è utilizzato per determinare l'aspetto finale della superficie di un oggetto

Essendo una API studiata per il web, garantisce una completa interazione con gli elementi della *canvas* di HTML5 e questo la rende anche completamente compatibile con tutte le interfacce del DOM[5]. È proprio per questo motivo che può essere utilizzata con tutti i linguaggi di programmazione compatibili con il DOM stesso: Javascript, Java, Objective C. *WebGL* è inoltre una libreria gratuita e *open source*. Infatti è gestita e curata dal Khronos Group, un consorzio di sviluppatori di grafica 3D di cui fanno parte anche i provider dei maggiori browser web quali Mozilla (Firefox), Google (Chrome), Apple (Safari), Opera (Opera) e le principali aziende produttrici di hardware video dedicato, ovvero AMD e NVIDIA. Questo fatto la rende sicura e molto efficace.

La programmazione *WebGL*, non segue lo schema tradizionale di programmazione web. Ogni progetto ha bisogno di una parte scritta in Javascript e una in GLSL. In pratica, con il codice JS, gli sviluppatori definiscono oggetti, immagini e colori presenti sulla scena, mentre attraverso il GLSL ci si occupa dei programmi *shader* che rendono possibile l'esecuzione dei colori, immagini e vettori sulla GPU. Questo è possibile perchè il GLSL è un linguaggio basato sul linguaggio C e proprio per questo è un linguaggio di basso livello in grado di accedere direttamente alle informazioni della GPU, in modo tale da attivarne all'occorrenza anche l'accelerazione grafica.

Una delle caratteristiche però negative di WebGL è l'assenza di funzioni proprie per azioni abbastanza comuni nella computer grafica, ad esempio la riduzione in scala o la rotazione di oggetti. È necessario, inoltre, aggiungere un altro importante dettaglio, quello della definizione e costruzione di oggetti come cubi, sfere ed altre forme geometriche. Nella programmazione grafica base, quindi senza far riferimento ad altri tipi di librerie, per definire ad esempio una sfera occorre far riferimento ad una matrice di coordinate, ognuna delle quali definisce un vertice di un triangolo, utilizzando poi la trigonometria per calcolare i singoli punti.

Proprio per superare questi e altri problemi nello sviluppo di un progetto gi grafica 3D, gli sviluppatori fanno uso di librerie appositamente semplificate per l'esecuzione di calcoli più complessi o per operazioni che potrebbero risultare ripetitive. È il caso, per esempio di **Three.js**.

[5]Document Object Model[Con15]

2.2.1 Three.js, la libreria javascript semplificata di grafica 3D

Abbiamo analizzato prima il funzionamento di *WebGL*, vedendo come fosse molto complicato, o meglio articolato, affrontare la programmazione grafica 3D senza l'utilizzo di alcuna libreria di supporto. Proprio in questo contesto si colloca *Three.js*.

Questa è una libreria basata su *WebGl*, utilizzata in almeno la metà dei progetti web di grafica 3D. Come la stessa API di riferimento, anche *Three.js* è una libreria gratuita ed open source. Originariamente venne messa in rete dal suo creatore, Ricardo Cabello che nell'aprile del 2010 pubblicò il progetto su GitHub[6]. Cabello, conosciuto in rete come Mr.doob (il suo username su GitHub), scrisse inizialmente il progetto in ActionScript[Wik14] ma alla fine convertì il tutto in Javascript, questo perchè Javascript permette di non compilare il codice prima di ogni sua esecuzione e soprattutto la sua completa indipendenza dalla piattaforma da cui lo si sta utilizzando. Negli anni sono aumentati sempre di più i contributori al progetto iniziale al punto da essere ormai più di 390 contributori alla repository online. Il compito e merito di Cabello è stato, oltre allo sviluppo iniziale dei renderer principali, anche quello di unire tra di loro le modifiche fatte al progetto dai vari contributori.

Dopo aver descritto brevemente la storia e l'importanza di questa libreria, è opportuno vederne la sua struttura e funzionamento.

La caratteristica, probabilmente, che ne segna l'assoluta praticità è sicuramente il fatto che non richiede alcun tipo di installazione, infatti una volta scaricato il progetto da GitHub sarà sufficiente includere il file *three.min.js* all'interno della nostra pagina html come da esempio:

```
<script scr="directory/three.min.js"></script>
```

La struttura di uno *script* che fa uso della libreria è abbastanza standard, sarà infatti necessario dichiarare le variabili, che rappresenteranno gli oggetti presenti sulla nostra canvas[7]. In questa parte andremo, quindi, ad analizzare le componenti fondamentali di una scena 3D. Le prime tre variabili da dichiarare sono senza dubbio **camera**, **scene** e

[6] GitHub è un servizio web che utilizza il sistema di controllo di versione Git per lo sviluppo di progetti software

[7] Canvas è una estensione dell'HTML standard che permette il rendering dinamico di immagini bitmap gestibili attraverso un linguaggio di scripting[Wik15c]

renderer, questi tre elementi rappresentano l'anatomia base si un progetto di grafica 3D e definiscono rispettivamente:

- la camera, ovvero il punto di vista da cui l'utente sta guardando il nostro disegno e da cui controllerà se richiesto i movimenti e le azioni da eseguire;
- la scena sulla quale andreamo a "disegnare" e aggiungere forme geometriche o gli elementi di cui abbiamo bisogno;
- il renderer rappresenta invece la componente che effettivamente "*disegna*" la nostra scena, il suo compito è quello di far materializzarele componenti sulla canvas html;

Ovviamente, oltre queste tre variabili sarà necessario all'occorrenza dichiarare la geometria dell'oggetto o degli oggetti da utilizzare, il/i materiale/i per gli oggetti, e l'oggetto in se.

Una volta dichiarate tutte le variabili di cui abbiamo bisogno il nostro script dovrebbe avere una struttura all'incirca simile alla seguente:

```
var camera, scene, renderer;
\\altre variabili necessarie
init();
animate();

function init() {
        scene = new THREE.Scene();
        camera = ...
        \\resto del codice
}

function animate(){
        requestAnimationFrame( animate );
        render();
}
```

```
function render(){
        \\azioni da eseguire
        renderer.render(scene, camera);
}
```

Le tre funzioni, sono necessarie rispettivamente per inizializzare il nostro ambiente grafico, aggiornamento e animazione degli oggetti sulla scena e disegno vero e proprio sulla scena. Questa è, ovviamente, una struttura base del nostro progetto, ogni progetto è differente, potrebbe quindi aver bisogno di altre funzioni.

Abbiamo dunque visto la definizione di una scena con il semplice comando **new THREE.Scene()**, la cosa non è altrettanto semplice per la camera poichè sarà necessario definire il tipo di camera da utilizzare e le sue specifiche. All'interno della libreria sono comunque disponibili due tipi di camere: OrtographicCamera, una camera che genera una visione assonometrica, e PerspectiveCamera, una camera che utilizza una proiezione prospettica, la più tradizionale ed usata anche nel mio progetto.

Abbiamo parlato prima della presenza anche di oggetti, geometrie e materiali. La presenza dei tre all'interno di un progetto è abbastanza correlata. Questo perchè ogni oggetto è definito a partire da una geometria e un materiale che lo plasmeranno. Vediamo dunque l'esempio di costruzione di un semplice cubo:

```
geometry = new THREE.BoxGeometry(200,200,200);
material = new THREE.MeshBasicMaterial({color: 0xHEXCOLOR, ...});
mesh = new THREE.Mesh(geometry, material);
scene.add(mesh);
```

In questo esempio *geometry* rappresenta la struttura 3D del nostro cubo, di dimensioni 200 pixels per lato, *material* è, appunto, il nostro materiale che ha come attributi il colore espresso in codice esadecimale, che sarà il colore di base del nostro cubo, *mesh*, infine, è il cubo vero e proprio, costruito a partire da *geometry* e *material*. Infine ho aggiunto anche l'oggetto alla nostra scena utilizzando il metodo add definito in *THREE.Scene*. Alla fine il nostro script sul file Html, una volta definiti forma e materiali utilizzati, dovrebbe essere più o meno questo:

```
<script>
        var camera, scene, renderer,
    geometry, material, mesh;

    init();
    animate();

    function init() {
        scene = new THREE.Scene();
                camera = new THREE.PerspectiveCamera(75,
                        window.innerWidth/window.innerHeight,
                                1, 10000);
        camera.position.z = 1000;
        geometry = new THREE.BoxGeometry(200, 200, 200);
        material = new THREE.MeshBasicMaterial({color:0xff0000,
                wireframe: true});
        mesh = new THREE.Mesh(geometry, material);
        scene.add(mesh);
        renderer = new THREE.WebGLRenderer();
        renderer.setSize(window.innerWidth,window.innerHeight);
        document.body.appendChild(renderer.domElement);
    }
        function animate() {
        requestAnimationFrame(animate);
        render();
    }
    function render() {
        renderer.render(scene, camera);
    }
</script>
```

Il risultato di questo codice può essere visto nella figura 2.1

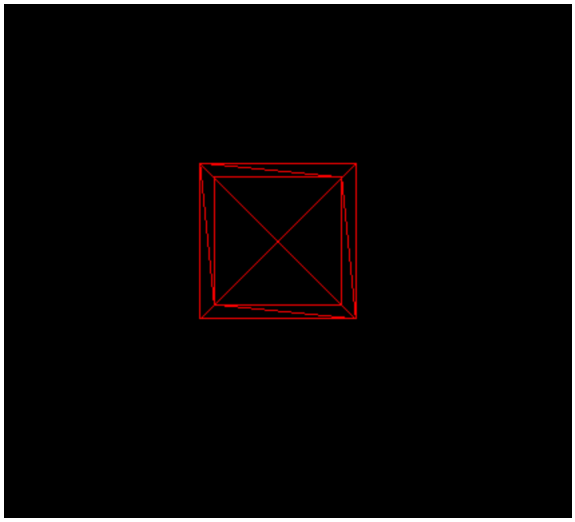

Figura 2.1: Risultato della scena precedentemente creata

2.3 Google Maps API

Nel primo capitolo di questa tesi ho descritto i più utilizzati sistemi di visualizzazione e navigazione di mappe geografiche disponibili sul web. In particolare, avevo evidenziato come *Google Maps* fosse, probabilmente, il sistema più conosciuto ed utilizzato nel mondo. Sicuramente ciò è dovuto anche alla larga scala di diffusione di smartphones, siano essi con sistema operativo Android di Google o iOS di Apple. All'inizio dello studio del contesto intorno al quale ruotava il mio progetto di tirocinio, una delle prime fasi è stata comprendere quale libreria o API sarebbe stata la prescelta per l'acquisizione delle mappe. Le alternative proposte erano principalmente due: **Leaflet.js** e **Google Maps API**. Per descriverle entrambe è necessario sottolineare che la prima è una libreria javascript basata sulle mappe di *OpenStreetMap*, mentre la seconda come si può evincere dal nome è la libreria di riferimento per gli sviluppatori che intendano utilizzare le mappe di Google. La decisione finale, ha portato all'utilizzo delle primitive offerte da Google.

Le API di Google Maps, sono delle librerie scritte in javascript arrivate alla terza versione, che permettono di interagire, adattare o aggiungere informazioni alla mappa stessa da visualizzare su di un qualsiasi sito web. Per utilizzare le API all'interno di

un progetto, è necessario iscriversi gratuitamente alla portale che Google ha destinato agli sviluppatori[8]. Una volta registratosi, lo sviluppatore che intende utilizzare un determinato servizio deve far richiesta di una *API Key*, che permetterà a Google di tracciare l'utilizzo delle mappe all'interno di un sito web. Questo è legato al fatto che l'utilizzo è sì gratuito, ma fino ad un certo limite di richieste giornaliere, il piano standard consente la richiesta di 25000 mappe giornaliere per un massimo di 90 giorni consecutivi, in eccesso Google chiede il pagamento di 0,50 dollari ogni altre 1000 richieste. Queste librerie sono disponibili per differenti target. Accedendo alla documentazione[Inc15b], è possibile vedere che la stessa è divisa in quattro principali sezioni: **Android**, **iOS**, **Web** e **Web Services**. Le prime due sezioni riguardano lo sviluppo di applicazioni per i dispositivi mobili con rispettivamente il sistema della stessa casa di Mountain View e quello di Apple. Le altre due sezioni invece riguardano le altrettante componenti di un sito web, perchè con *Web* si intendono tutte quelle operazioni atte a mostrare la mappa sul client, ovvero il nostro browser web. Con *Web Services* si vuole intendere tutte le restanti operazioni di elaborazione che ci servono nel nostro backend, come riportato sulla documentazione ufficiale potremo dunque: accedere al *Geocodice*, ovvero tutte quelle operazioni di conversione tra coordinate geografiche e indirizzi veri e propri; utilizzare servizi di *Geolocalizzazione*, i servizi di posizionamento fruibili da un cellulare con sensore GPS o tramite i nodi WiFi; l'altitudine di ogni posizione della terra, Google ha infatti all'interno dei propri database le informazioni di altitudine di ogni zona della terra (informazioni visibili anche sul portale maps nella vista satellitare prospettica). Quelle elencate sono solo alcune delle principali funzioni disponibili per i Web Services.

Per importare le API all'interno del nostro progetto sarà sufficiente aggiungere un tag "script" all'interno del file HTML, come nell'esempio che segue.

```
<script src="https://maps.googleapis.com/maps/api/js?key=API_KEY
        &callback=initMap"async defer></script>
```

Nell'esempio richiamiamo una funzione, initMap, che serve ad inizializzare la nostra mappa. La funzione, prima citata può essere scritta nel modo seguente:

```
var map;
function initMap() {
```

[8] developers.google.com

```
  map = new google.maps.Map(document.getElementById('map'), {
    center: {lat: -34.397, lng: 150.644},
    zoom: 8
  });
}
```

Nella funzione viene creata una mappa con centro nel punto di latitudine -34.397 e longitudine 150.644 e zoom pari a 8. La definizione dello zoom la vedremo più avanti, in quanto è stato uno degli argomenti di studio iniziali del mio progetto.

La natura del mio progetto non era però probabilmente adatta per l'impiego di una mappa già interattiva come quella offerta da Google all'interno del suo servizio. Dovendo lavorare su di un piano tridimensionale, disegnato con Three.js, non era completamente funzionale far interagire tra loro due sistemi già interattivi. Per questo motivo ho deciso di utilizzare un altro servizio più semplificato messo a disposizione da Google. Sto parlando delle Google Static Maps. Le mappe statiche offerte da google sono delle immagini definite in base alla richiesta da noi avanzata. Infatti non c'è bisogno di alcun codice definito o particolare per richiedere la mappa. Questa è disponibile grazie ad un url di richiesta, una vera e propria query, a cui vengono concatenati i parametri necessari: centro, che può essere definito con coordinate polari o con un indirizzo testuale; la grandezza dell'immagine, definita come *LARGHEZZA*x*ALTEZZA* e inifine il livello di zoom desiderato. All'url è possibile attaccare anche altri parametri nel caso volessimo modificare lo stile della mappa, i colori, se volessimo aggiungere dei fermaposto sulla nostra tile o se volessimo disegnare un tracciato su di essa. Un esempi di url di richiesta può essere questo:

```
https://maps.googleapis.com/maps/api/staticmap?center=Roma,IT
        &zoom=11&size=640x640
```

In questo url troveremo una mappa centrata in Roma di zoom 11 e grandezza 640x640 pixels, che è la grandezza massima che Google consente di utilizzare nelle API gratuite. Il risultato dell'url può essere visto nella figura 2.2

Figura 2.2: Esempio di mappa statica di Roma

Capitolo 3

Requisiti per un navigatore geografico in 3D

Capitolo 4

Progetto del navigatore

Capitolo 5

Verifiche funzionali

Conclusioni e sviluppi futuri

La tesi è finita

Bibliografia

[Bin10] Bing. New bing map app: 2010 tour de france, 2010.

[Con15] World Wide Web Consortium. Document object model, 2015.

[Fou15] OpenStreetMap Foundation. Openstreetmap, 2015.

[Gro15] Khronos Group. Opengl es - the standard for embedded accelerated 3d graphics, 2015.

[Inc15a] Google Inc. Google maps apis, 2015.

[Inc15b] Google Inc. Google maps apis | google developers, 2015.

[Map15] Mapbox. Showcase | mapbox, 2015.

[Wik14] Wikipedia. Actionscript — wikipedia, l'enciclopedia libera, 2014. [Online; in data 27-novembre-2015].

[Wik15a] Wikipedia. Application programming interface — wikipedia, l'enciclopedia libera, 2015. [Online; in data 26-novembre-2015].

[Wik15b] Wikipedia. C++ — wikipedia, l'enciclopedia libera, 2015. [Online; in data 26-novembre-2015].

[Wik15c] Wikipedia. Canvas (elemento html) — wikipedia, l'enciclopedia libera, 2015. [Online; in data 28-novembre-2015].

[Wik15d] Wikipedia. Open source — wikipedia, l'enciclopedia libera, 2015. [Online; in data 25-novembre-2015].