



Università
Ca'Foscari
Venezia

Corso di Laurea
in Informatica

Tesi di Laurea

PDBjs: un tool per la visualizzazione 3D di proteine

Relatrice
Prof.ssa Marta Simeoni

Laureando
Nicolas Pietro Martignon
Matricola 870034

Anno Accademico
2019 / 2020

Abstract

Argomento di questa tesi è la visualizzazione tridimensionale della struttura delle proteine. La tesi ha comportato lo sviluppo di una parte prettamente teorica seguita dallo sviluppo di un applicativo software. In particolare, nella prima parte vengono presentati e catalogati i principali tool di visualizzazione 3D di proteine. Dalla classificazione emergono delle linee guida che sono state seguite per la realizzazione di PDBjs, un applicativo multipiattaforma per la visualizzazione di proteine. PDBjs è un tool sviluppato in Javascript: nella tesi vengono presentate le sue caratteristiche e funzionalità.

Indice

1 Introduzione	4
2 Le proteine	5
2.1 Cosa sono le proteine	5
2.2 Struttura delle proteine	5
2.3 PDB	7
2.4 Visualizzazione	7
3 I tool di visualizzazione di proteine	9
3.1 Classificazione tool	10
3.2 Analisi tool	14
4 PDBjs: un tool per la visualizzazione di proteine	18
4.1 Overview progetto	18
4.2 Soluzioni software adottate	18
4.3 Descrizione funzionamento dell'applicativo	19
4.4 Ottimizzazione grafica	21
4.5 Interfaccia utente	22
4.6 Requisiti applicativo	23
4.7 Testing	23
4.8 Evoluzioni	31
5 Conclusioni	32
A Codice programma	37

Elenco delle figure

2.1 Struttura proteina	5
2.2 α elica	6
2.3 β sheet	6
2.4 CPK	8
2.5 Ball and Stick	8
2.6 Ribbon	8
3.1 Licenza applicativo	14
3.2 Appartenenza ad una suite	14
3.3 Tipologia applicativo	15
3.4 Linguaggio di programmazione	15
3.5 Librerie grafiche usate	16
3.6 Compatibilità sistemi operativi	17
4.1 Flow chart funzionamento PDBjs	19
4.2 Interfaccia grafica contenente la proteina 4HHB (deossiemoglobin)	22
4.3 Toast pop-up	23
4.4 Secondi caricamento proteina 1MH1	25
4.5 Fotogrammi per secondo medi proteina 1MH1	25
4.6 Le tre visualizzazioni della proteina 1MH1 elaborate da PDBjs	25
4.7 Le tre visualizzazioni della proteina 1MH1 elaborate da Jmol 	26
4.8 Secondi caricamento proteina 3MDD	26
4.9 Fotogrammi per secondo medi proteina 3MDD	26
4.10 Le tre visualizzazioni della proteina 3MDD elaborate da PDBjs	27
4.11 Le tre visualizzazioni della proteina 3MDD elaborate da Jmol 	27
4.12 Secondi caricamento proteina 4FP9	28
4.13 Fotogrammi per secondo medi proteina 4FP9	28
4.14 Le tre visualizzazioni della proteina 4FP9 elaborate da PDBjs	29
4.15 Le tre visualizzazioni della proteina 4FP9 elaborate da Jmol 	30

Capitolo 1

Introduzione

Le proteine possono essere rappresentate in formato digitale attraverso file con specifiche formattazioni. Diversi tool permettono la visualizzazione di proteine nella computer grafica 3D, questo è utile, perchè rende comprensibile la loro struttura molecolare, le loro proprietà e le loro interazioni. Inoltre, dalla struttura molecolare di una proteina può essere ricavata la sua specifica funzione biochimica.

Scopo del presente elaborato è catalogare i principali tool di visualizzazione 3D reperibili. Di ogni programma in elenco vengono indicate diverse caratteristiche e informazioni. Lo scopo di questo lavoro di catalogazione è avere una visione chiara di cosa è stato fatto in letteratura e con quali modalità e strumenti. Inoltre, la catalogazione è utile per capire quali sono le funzionalità ricorrenti dei tool attualmente in uso. Questa catalogazione darà origine ad un'analisi sulle tecnologie maggiormente utilizzate dagli applicativi, alla quale seguiranno delle linee guida utili per la realizzazione di un analogo tool per la visualizzazione 3D di proteine. L'applicativo creato, PDBjs, è una web app sviluppata in JavaScript, che utilizza la nota libreria grafica three.js [2], che si basa a sua volta su WebGL [3].

La tesi è organizzata come segue: innanzi tutto vengono introdotte le nozioni di base sulle proteine, la loro struttura e la loro rappresentazione digitale.

Successivamente, viene presentato lo studio degli applicativi presenti in letteratura e la loro catalogazione.

L'ultimo capitolo è dedicato a PDBjs, in cui vengono discusse le soluzioni software adottate, il funzionamento dell'applicativo, l'interfaccia ideata e le sue dipendenze tecnologiche. Viene inoltre svolto un test di correttezza dell'applicativo e un confronto tra il funzionamento di PDBjs e di Jmol [1], un noto tool di visualizzazione di proteine.

Capitolo 2

Le proteine

In questo capitolo viene presentata una breve introduzione sulle proteine. In seguito vengono presentate le principali modalità per visualizzare la struttura tridimensionale di una proteina. Il materiale descritto in questo capitolo è tratto da: [4, 5, 6, 7, 8].

2.1 Cosa sono le proteine

In chimica, le proteine (o protidi) sono macromolecole biologiche costituite da catene di amminoacidi legati l'uno all'altro da un legame peptidico. Esse differiscono tra loro per numero e sequenza di aminoacidi, inoltre, a causa delle proprietà chimiche e fisiche delle molecole, ogni sequenza è organizzata in modo differente nella sua struttura tridimensionale, la quale determina la specifica funzione della proteina stessa. Le proteine svolgono numerose funzioni all'interno degli organismi viventi, ad esempio: la sintesi come replicazione del DNA, la catalisi delle reazioni metaboliche, il trasporto di molecole da un luogo ad un altro e la risposta agli stimoli.

2.2 Struttura delle proteine

Una proteina è nel suo complesso una molecola in cui vengono convenzionalmente distinti 4 livelli di organizzazione.

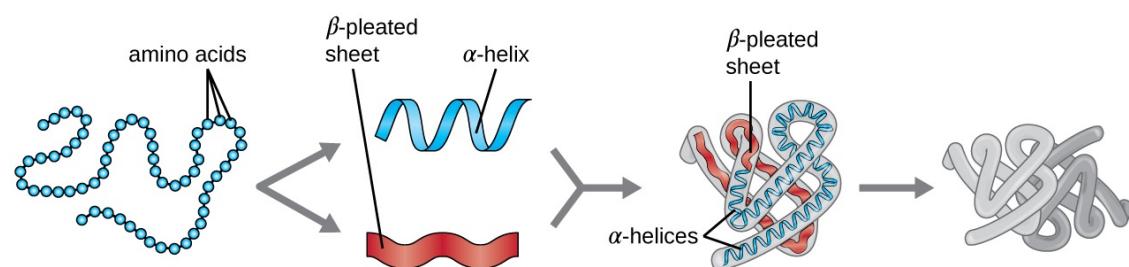


Figura 2.1: Struttura proteina

2.2.1 Struttura Primaria

Essa è formata da una sequenza specifica di amminoacidi che la compongono e data sequenza determina da sola il ripiegamento della proteina.

2.2.2 Struttura Secondaria

Consiste nella conformazione spaziale delle catene aminoacidiche, ad esempio: la conformazione a spirale (α elica), quella planare (β sheet) e quelle irregolari (Loop). All'interno di una singola proteina possono convergere combinazioni di sequenze di α eliche, β sheet e Loop.

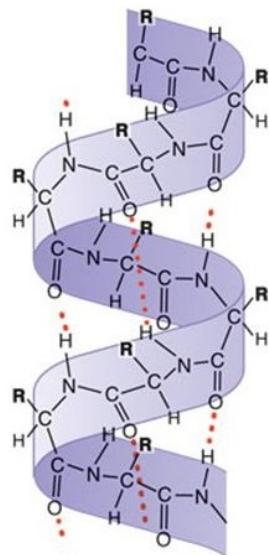


Figura 2.2: α elica

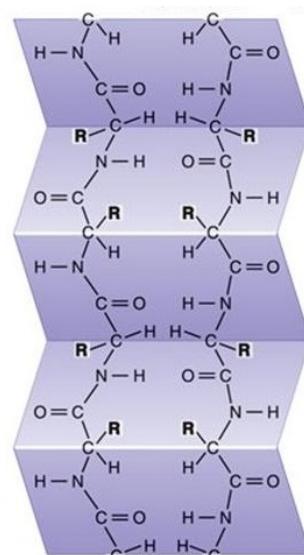


Figura 2.3: β sheet

2.2.3 Struttura Terziaria

Dal punto di vista della termodinamica è la forma con la più bassa energia libera, è rappresentata dalla configurazione tridimensionale completa che la catena polipeptidica assume nell'ambiente in cui si trova. Questa struttura viene consentita e mantenuta da diversi fattori, come i ponti disolfuro e le forze di Van der Waals. La struttura terziaria è l'unione delle strutture secondarie che compongono una catena polipeptidica.

2.2.4 Struttura Quaternaria

È la struttura che deriva dall'associazione di due o più unità polipeptidiche, unite tra loro da legami deboli e, a volte, ponti disolfuro.

2.3 PDB

Il Protein Data Bank (PDB) [9] è una banca dati di proteine e acidi nucleici. Le macromolecole presenti nella banca dati sono ottenute soprattutto grazie alla cristallografia ai raggi X e sono depositati da biologi e biochimici di tutto il mondo. I dati sono di pubblico dominio e sono accessibili gratuitamente. Ogni molecola è salvata nel PDB mediante un file con formato .PDB, il quale ne descrive la struttura tridimensionale con la relativa disposizione degli atomi e delle strutture secondarie. Ognuno di questi file è composto da diversi record, di cui i principali sono mostrati sotto.

ATOM 22 HB VAL A 2 10.230 34.930 14.018 1.00 19.80 H

Record che descrive un atomo.

HETATM 95 C3A HEM A 12 10.636 10.499 -16.796 1.00 18.36 C

Record che descrive un eteroatomo.

HELIX 7 AG ASP A 94 HIS A 112 1 19

Record che descrive un' α elica.

SHEET 1 A 3 ALA A 1 VAL A 5 0

Record che descrive un β sheet.

Nelle pagine precedenti abbiamo già spiegato cosa sono le α eliche e β sheet. Gli eteroatomi invece sono degli atomi non uniti tramite legami covalenti alle catene polipeptidiche formanti la proteina in esame. In altre parole, sono atomi che non fanno parte della proteina, ma sono stati cristallizzati con essa.

Molti altri record che descrivono le caratteristiche della proteina possono essere trovati nel manuale del PDB [10].

2.4 Visualizzazione

Esistono molti tool di visualizzazione delle proteine. Essi, di base, offrono tre tipi di rappresentazione, che possono essere chiamati in modo differente. Nel capitolo successivo verranno esaminati nel dettaglio molti tool di visualizzazione.

- Rappresentazione CPK: In questo tipo di rappresentazione sono mostrati soltanto gli atomi, essi sono disegnati come sfere i cui diametri sono proporzionali ai loro raggi di Van Der Waals [11, 12];
- Rappresentazione Ball and Stick: In questo tipo di rappresentazione sono mostrati sia gli atomi, che i relativi legami tra loro. Gli atomi sono disegnati

allo stesso modo della Rappresentazione CPK, solamente appaiono più piccoli, mentre i legami tra atomi sono rappresentati da cilindri;

- Rappresentazione Ribbon: Attraverso questo tipo di rappresentazione è raffigurata la struttura terziaria/quaternaria di una proteina. Questa rappresentazione viene generata creando una linea curva che attraversa la backbone della proteina. Le α eliche sono mostrate come nastri a spirale, i β sheet, invece, come nastri con inizio o fine a forma di freccia, infine, i loop sono rappresentati da un filo curvo.

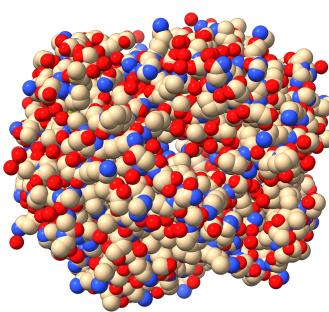


Figura 2.4: CPK

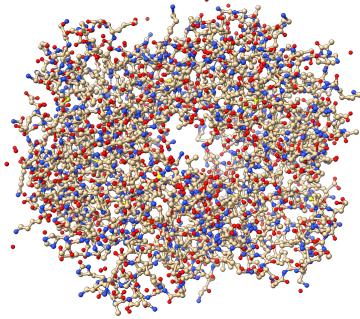


Figura 2.5: Ball and Stick

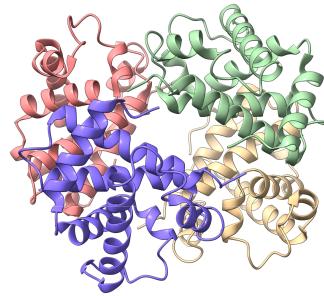


Figura 2.6: Ribbon

I tre tipi di rappresentazione sopra descritti della proteina 4HBB (deossiemoglobina). Render creato con il programma UCSF ChimeraX [13].

Capitolo 3

I tool di visualizzazione di proteine

In questo capitolo vengono presentati i principali programmi di visualizzazione 3D di proteine, reperibili dal sito RCSB PDB . L'obiettivo di presentarli in modo omogeneo ci ha portato a classificarli rispetto alle seguenti caratteristiche:

- Licenza: tipologia di licenza, che può essere:
 - Open source: software il cui codice sorgente è liberamente utilizzabile e a disposizione del pubblico;
 - Freeware: il programma è distribuito gratuitamente. È perfettamente funzionante e non richiede alcun pagamento;
 - Shareware: programmi che, pur essendo liberamente distribuibili, richiedono un pagamento, per poter essere utilizzati in tutte le loro funzioni e senza limitazioni di tempo.
- Tipologia: la tipologia di applicativo, che può essere:
 - Standalone application (S.A.): software tradizionale che deve essere installato nella macchina client;
 - Web app object (W.A.O.): oggetto manipolabile attraverso API che può essere importato e utilizzato nella propria pagina web;
 - Web applications (W.A.): vera e propria applicazione web che può essere raggiunta mediante URL;
 - Web app helper (W.A.H.): applicazione che viene avviata per visualizzare il contenuto recuperato utilizzando un browser Web. Un'applicazione helper non fa parte del browser, ma è esterna;
 - Framework (F.W.): frammento di codice che può essere utilizzato da programmatori attraverso API.
- Linguaggio: il linguaggio di programmazione utilizzato per la creazione.
- Sistema operativo: sistemi operativi compatibili in tutte le versioni dell'applicativo.
- Librerie: librerie note usate dall'applicativo.

- Suite: se il programma fa parte o meno di una suite.
- Referenze: eventuale articolo di riferimento e sito web.

Terminata la classificazione degli applicativi, vengono formulate delle considerazioni riguardanti le tipologie e i linguaggi dei tool catalogati. Date considerazioni saranno utilizzate ai fini della creazione di un applicativo analogo che verrà presentato nel capitolo successivo.

3.1 Classificazione tool

In questa sezione vengono presentati i vari tool di visualizzazione delle proteine presi in considerazione per l'analisi. Si procede elencando prima i programmi open source, poi freeware ed infine shareware.

Programmi open source

Lista programmi						
Nome	Tipologia	Linguaggio	Sistema operativo	Librerie	Suite	Referenze
BioBlender	Modulo per Blender	Python	S.O. compatibile con Blender	Blender	No	[14]
BioJava	F.W.	Java	S.O. con J.V.M	-	No	[15, 16]
CCP4MG	S.A.	C++, Python	(Windows, Linux, MacOS X) 64bit	OpenGL	No	[17, 18]
Cm2 MembraneEditor	S.A.	Java	S.O con J.V.M.	Jmol	Si	[19]
Cn3D	W.A.H.	C++	Windows, Linux, MacOS X	OpenGL, NCBI C++ Toolkit	Si	[20]
iCn3D	W.A.O.	JavaScript	S.O con browser compatibile	3Dmol.js, jQuery UI, three.js, WebGL	Si	[21]
Jmol	S.A.	Java	S.O con J.V.M.	-	No	[1]
JSmol (Jmol porting)	W.A.O.	JavaScript	S.O con browser compatibile	WebGL	No	[1]
KiNG Kinemages	S.A.	Java	S.O con J.V.M.	OpenGL	Si	[22]

Lista programmi						
Nome	Tipologia	Linguaggio	Sistema operativo	Librerie	Suite	Referenze
MolScript	S.A.	C	SGI IRIX 6.3, Unix	OpenGL, GLUT, Mesa, VRML	No	[23, 24]
PMV	S.A.	Python	Windows, Unix, MacOS X	OpenGL	Si	[25]
RasMol	S.A.	C	Unix, Windows, MacOS	OpenGL, NearTree, CQRlib, CVector	No	[26]
Raster3D	S.A.	Fortran, C	Windows, Unix, MacOS	-	No	[27, 28]
RasTop	S.A.	C	Windows	RasMol	No	[29]
RCSB-Viewers	F.W.	Java	S.O. con J.V.M	BioJava, JOGL	Si*	[30]
PyMOL	S.A.	C, C++, Python	Windows, MacOS X, Unix	OpenGL, OpenVr, Libxml2	Si	[31]
QuteMol	S.A.	C++	Linux, MacOS X, Windows	OpenGL	No	[32, 33]
SPADE	S.A.	Python	Linux, Windows	-	No	[34, 35]
Unipro UGENE	S.A.	C++	MacOS X, Linux, Windows	OpenGL, OpenCL, Cuda, Qt	No	[36, 37]
VMD	S.A./ W.A.H.	C/C++, Python, Tcl	MacOS X, Unix, Windows	OpenGL, Cuda	No	[38, 39]
VRmol	W.A.	JavaScript	S.O con browser compatibile	WebVR, three.js, WebGL	No	[40]

* è una suite di programmi

Programmi freeware

Lista programmi						
Nome	Tipologia	Linguaggio	Sistema operativo	Librerie	Suite	Referenze
BRAGI	S.A.	C/C++	Windows, SGI IRIX 6.5, Linux	OpenGL, Qt (toolkit)	No	[41]
EzMol	W.A.	JavaScript	S.O con browser compatibile	3Dmol.js, jQuery UI, Spectrum, WebGL	Si	[42, 43]
ICM-Browser	S.A.	C/C++	Windows (64bit), Linux, MacOS X (64bit)	OpenGL	Si	[44]
icmJS (ICM- Browser porting)	W.A.O.	ASM.JS	S.O con browser compatibile	WebGL	Si	[45]
iMol	S.A.	Objective- C, C	Mac OS X v.10.2 o superiori	OpenGL	No	[46]
MarvinView	S.A.	Java	S.O con J.V.M.	-	Si	[47]
POLYVIEW	W.A.	Perl	S.O con browser compatibile	JSmol	Si*	[48]
STRAP	S.A./ W.A(Java Web Start)	Java	S.O con J.V.M.	Java Web Start	No	[49]
Swiss PDB viewer	S.A.	C	[4.1] MacOS X, Windows [3.7 ≥] SGI IRIX 6.5, Linux, Windows, MacOS	OpenGL	No	[50]
UCSF Chimera	S.A.	C++	Windows, Linux, MacOS X	OpenGL	Si	[51]
UCSF ChimeraX	S.A.	C++	Windows, Linux, MacOS X	OpenGL	Si	[13]

* è una suite di programmi

Lista programmi						
Nome	Tipologia	Linguaggio	Sistema operativo	Librerie	Suite	Referenze
YASARA View	S.A.	C/C++	Windows, Linux, MacOS, Android(cpu intel)	OpenGL	Si	[52]
Zeus	S.A.	[Win] Delphi, Pascal [Linux] Java	Windows, Linux	[Win] OpenGL, DirectX [Linux] Java3D, JavaFX	No	[53]
ZMM	S.A.	-	Windows, MacOS, Unix	-	No	[54]

Programmi shareware

Lista programmi						
Nome	Tipologia	Linguaggio	Sistema operativo	Librerie	Suite	Referenze
CrystalMaker	S.A.	C++	Windows (64bit), MacOS X (64 bit)	OpenGL	Si	[55]
Molecular Operating Environment	S.A.	-	Windows, Linux, MacOS X	-	Si	[56]
Nanome	S.A.	-	Windows, Android (Oculus quest)	-	Si	[57]
ChemDoodle 3D	S.A.	-	(Windows, Linux, MacOS X) 64bit	OpenGL v2+	Si	[58]

3.2 Analisi tool

Nella stesura di una tesi scientifica in bionfirmatica, è necessario l'uso di un programma di visualizzazione 3D di proteine Open Source, questo perchè il risultato mostrato da esso deve poter essere confermato ed esaminato da chiunque osservi la tesi. Molti dei tool catalogati sono stati utilizzati nella stesura di tesi scientifiche e quindi risultano essere Open Source, vedi figura 3.1. Molti di questi tool essendo utilizzati in ambito scientifico non sono tool commerciali e quindi non appartengono a nessuna suite, vedi figura 3.2.

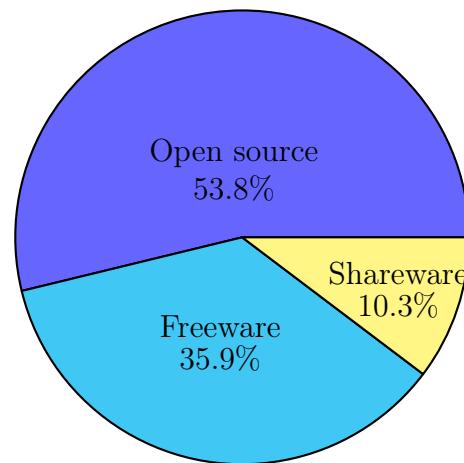


Figura 3.1: Licenza applicativo

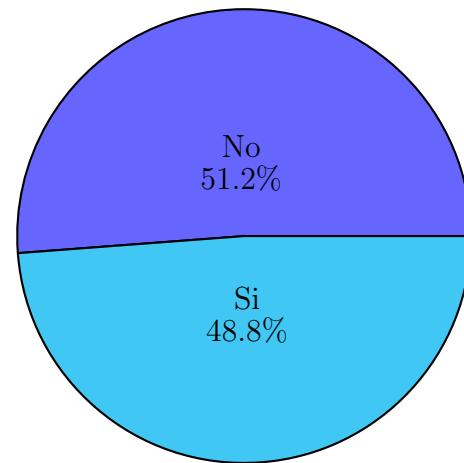


Figura 3.2: Appartenenza ad una suite

Altri tool invece essendo creati a scopo commerciale, offrono licenze differenti, gratuite ma senza la possibilità di avere il codice sorgente oppure licenze acquistabili. Molto spesso l'azienda che propone tool con licenza acquistabile (Shareware), offre

altri tool simili, facenti parte di una stessa suite.

Dei tool analizzati, possiamo notare dal grafico in figura 3.3, che la maggior parte è di tipo S.A., necessitano quindi di essere installati/scaricati per essere eseguiti. La loro portabilità tra sistemi operativi e architetture differenti dipende dal linguaggio con cui sono stati scritti.

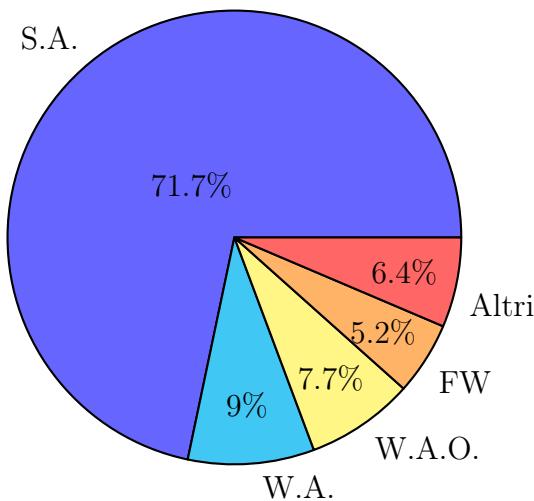


Figura 3.3: Tipologia applicativo

Linguaggi compilati come C, C++ che rappresentano la percentuale maggiore espresa nel grafico in figura 3.4, sono dipendenti dalla piattaforma e dal S.O., per questo comportano un'impegnativa operazione di portabilità. Essendo però linguaggi compilati, essi permettono un uso oculato delle risorse del sistema, che conferisce loro una elevata velocità di esecuzione.

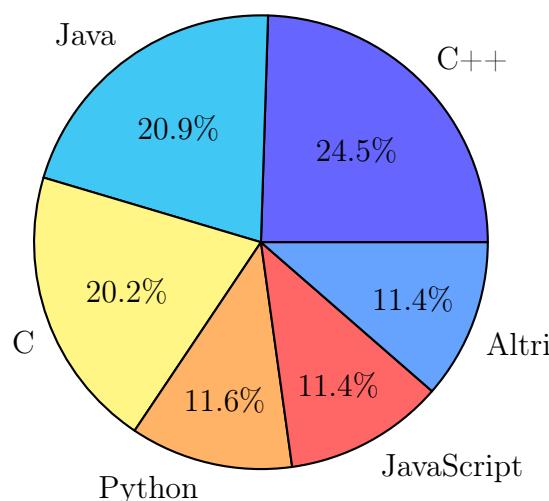


Figura 3.4: Linguaggio di programmazione

Nel caso in cui la S.A. sia stata scritta in Java, vedi figura 3.4, la portabilità è ad esso intrinseca, in quanto Java è un linguaggio indipendente dalla piattaforma e dal S.O.. Una volta scritta, l'applicazione può funzionare in tutti i sistemi operativi dotati di una Java Virtual Machine (JVM), la quale non è presente però nei dispositivi mobili (Android e iOS).

Altri tool S.A. sono stati invece scritti in Python, vedi figura 3.4. Python permette l'indipendenza dalla piattaforma e dal S.O. se vengono utilizzano determinati moduli e funzioni progettati per essere indipendenti.

Per quanto riguarda le librerie grafiche utilizzate, è possibile notare dal grafico in figura 3.5, che la maggior parte dei tool utilizza OpenGL [59], una libreria open source, e multipiattaforma.

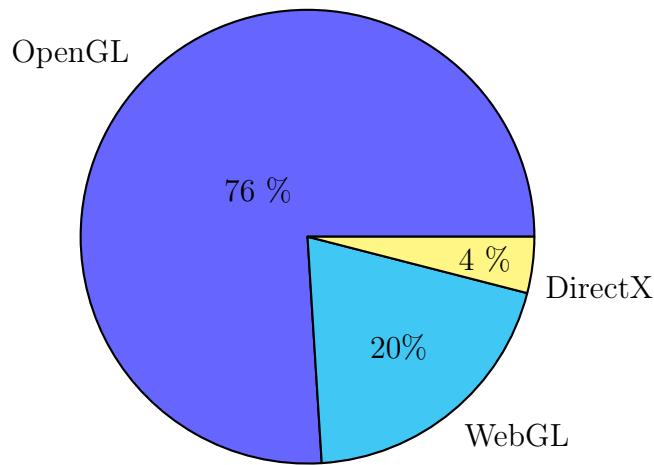


Figura 3.5: Librerie grafiche usate

DirectX invece risulta essere meno utilizzata in quanto si tratta di una libreria per lo sviluppo di videogiochi.

Le W.A che possono essere considerate, W.A + W.A.O., coprono invece una percentuale minore del grafico in figura 3.3, in quanto diffuse solo verso la fine degli anni Novanta. Le W.A., a differenza delle S.A., non necessitano di alcuna installazione e inoltre esse non sono eseguite direttamente dal S.O., ma dal browser web che interpreta il loro codice. Questo conferisce loro una velocità di esecuzione inferiore rispetto ad applicazioni sviluppate con linguaggi compilati o semi-interpretati.

Proprio per questo le Web Application presentate offrono meno funzionalità complesse dal punto di vista computazionale. Il punto di forza delle Web Application è che esse possono raggiungere tutti i dispositivi, sia desktop, sia mobile, purchè dispongano di un browser web compatibile. Quanto appena descritto rende le W.A. indipendenti dalla piattaforma e dal S.O., permettendo quindi di funzionare potenzialmente in tutti i sistemi operativi presenti in figura 3.6.

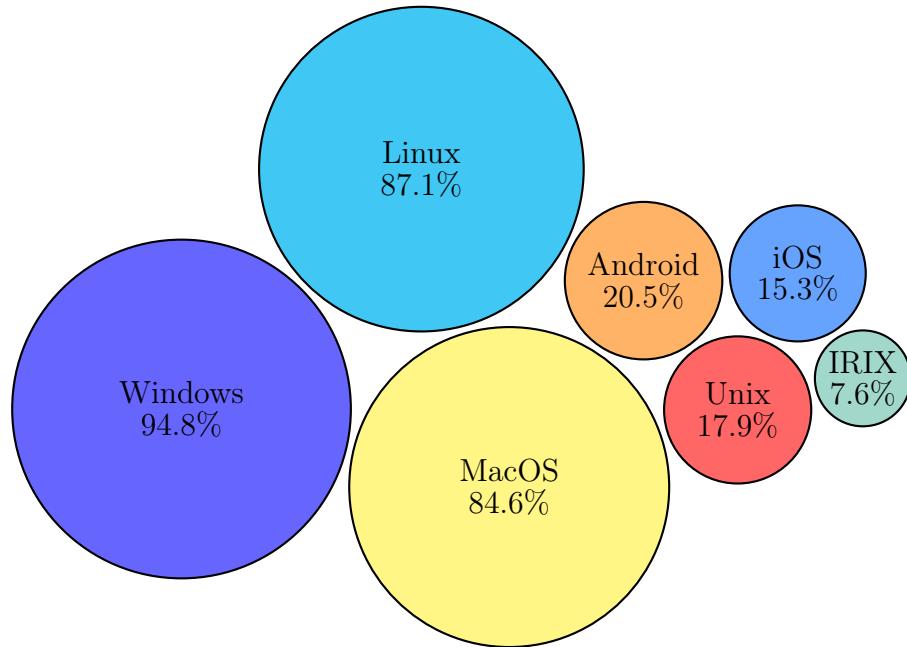


Figura 3.6: Compatibilità sistemi operativi

Ai fini di avere una maggiore compatibilità verso qualsiasi sistema operativo e device è stato scelto di sviluppare PDBjs attraverso JavaScript e quindi di creare una Web App, questa scelta inoltre ha permesso di interfacciarsi con la libreria grafica WebGL [3], che rappresenta la controparte web di OpenGL [59].

Capitolo 4

PDBjs: un tool per la visualizzazione di proteine

In questo capitolo viene presentato PDBjs, il tool di visualizzazione sviluppato per la visualizzazione tridimensionale della struttura delle proteine.

4.1 Overview progetto

Per la realizzazione della seguente tesi, si è voluto realizzare un applicativo che, prendendo in input un file PDB mostri a video la struttura tridimensionale della proteina descritta nel file utilizzando i tre principali tipi di visualizzazione descritti nella sezione 2.4, ovvero: rappresentazione CPK, rappresentazione Ball and Stick e rappresentazione Ribbon. Inoltre, nell'applicativo, dovrà essere possibile poter ruotare la proteina, ingrandirla, renderla più piccola e, mediante click su un atomo ottenere le sue caratteristiche.

4.2 Soluzioni software adottate

Sulla base delle considerazioni effettuate riguardo i tool di visualizzazione descritti nel capitolo 3, è stata scelta come opzione un'applicazione web, in quanto le caratteristiche richieste non sono troppo complesse rispetto alla capacità computazionale dei device disponibili al giorno d'oggi. Inoltre, tale soluzione risulta essere la più adatta, poichè, come spiegato nella capitolo precedente, consente la massima compatibilità verso qualsiasi sistema operativo e device.

- Linguaggio di programmazione: È stato scelto di utilizzare il linguaggio di programmazione JavaScript nativo, a discapito di altri linguaggi basati su JavaScript, al fine di potersi interfacciare con le librerie di seguito presentate.
- Librerie: Sono state utilizzate le seguenti librerie per lo sviluppo di PDBjs in tutte le sue funzionalità:
 - **three.js** [2]: libreria JavaScript open source, usata per creare applicazioni tridimensionali. three.js [2] usa WebGL [3], è facile da usare ed

è leggera, ma, al tempo stesso, offre potenti strumenti di rendering con molte ottimizzazioni.

- **OrbitControls.js** [60]: libreria in simbiosi con three.js [2], essa permette di avere un punto di focus attorno al quale ruotare, avvicinarsi e allontanarsi, all'interno di una scena tridimensionale.
- **BufferGeometryUtils.js** [61]: contiene funzioni utili per le istanze BufferGeometry di three.js [2].
- **THREE.MeshLine.js** [62]: libreria per creare linee con più personalizzazioni rispetto alla soluzione standard offerta da three.js [2] (THREE.Line).
- **jQuery** [63]: popolare framework JavaScript, permette la gestione degli eventi e l'animazione di elementi DOM in pagine HTML.

4.3 Descrizione funzionamento dell'applicativo

In questa sezione viene descritto il flusso dell'applicazione e in particolare il funzionamento di ogni componente.

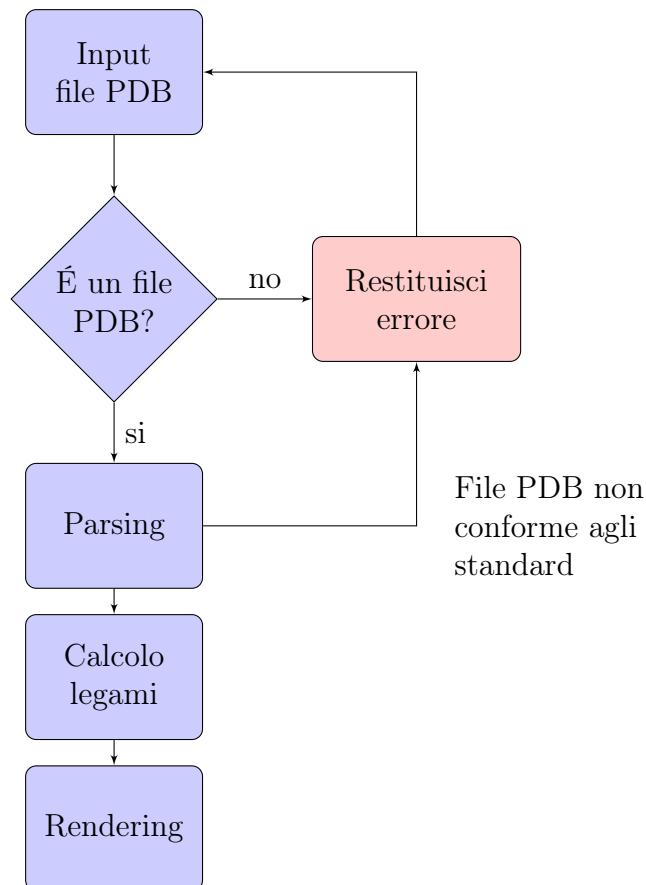


Figura 4.1: Flow chart funzionamento PDBjs

4.3.1 Parsing

Dopo il caricamento di un file PDB da parte dell'utente, il programma procede effettuando il parsing delle α eliche, dei β sheet e degli atomi (esclusi gli Hetatm), memorizzandole in strutture dati che contengono le informazioni utili per creare successivamente la visualizzazione 3D.

Se durante questa fase sorgono degli errori di parsing, il programma restituisce un errore e non procede alle fasi successive.

4.3.2 Calcolo legami

Se il parsing non ha generato errori, il programma procede calcolando tutti i legami tra gli atomi per realizzare la visualizzazione di tipo ball and stick. Questo è necessario perché nel file PDB non sono presenti i legami: essi devono essere calcolati dai tool di visualizzazione.

Come accade in Rasmol [20], il legame tra due atomi viene creato se la loro distanza rispetta il seguente vincolo:

$$\text{Raggio 1} + \text{Raggio 2} + \text{TOLLERANZA} > \text{Distanza}$$

Dove per Raggio 1 e Raggio 2 si intende il raggio covalente rispettivamente dell'atomo di partenza e di destinazione, per TOLLERANZA si intende, invece, un valore arbitrario costante, posto a 0,45 Å in analogia a Jmol [1].

Se tale vincolo è rispettato, il collegamento viene salvato nella struttura dati adibita. Il test per verificare l'esistenza di un legame non viene effettuato per ogni coppia di atomi, altrimenti vi sarebbe un'eccessivo spreco di risorse. L'algoritmo avrebbe dunque una complessità di $O(n^2)$, che non permetterebbe all'applicativo di terminare il calcolo dei collegamenti in tempi ragionevoli, nel caso in cui l'input sia una proteina complessa. È stato adoperato quindi un algoritmo simile al counting sort, dove gli atomi vengono inseriti in un array tridimensionale, dove ogni cella rappresenta un determinato punto nello spazio tridimensionale.

Effettuato l'inserimento di tutti gli atomi, occorre effettuare il test dell'effettiva esistenza del legame solo sugli atomi posizionalmente vicini nell'array.

L'algoritmo adoperato ha quindi una complessità $\Theta(n)$.

4.3.3 Rendering

Dopo il calcolo dei legami, l'applicativo inizia a comporre le varie scene tridimensionali: CPK, Ball and Stick e Ribbon, che verranno mostrate a video solo se la relativa checkbox è selezionata.

CPK

Per ogni atomo presente nella struttura dati degli atomi viene creata dunque una sfera con un determinato raggio e colore, a seconda del nome dell'elemento dell'atomo e viene posizionata nelle coordinate rispettive.

Come raggio degli atomi è stato usato il Raggio di Van Der Waals [11, 12].

Per il colore degli atomi, invece, è stata usata la colorazione CPK.

Ball and Stick

I legami calcolati sono disegnati con un cilindro, che va dall'atomo iniziale a quello finale del legame.

È stato scelto un raggio del cilindro di 0,150 Å, poichè adatto alla grandezza dell'atomo.

È stato scelto di utilizzare il colore viola per tutti i cilindri, che identificano i legami. Come per la scena CPK, gli atomi sono disegnati allo stesso modo, avendo come unica differenza il raggio: per ogni atomo è stato scelto di utilizzare il proprio raggio di Van Der Waals moltiplicato per 0,22, questo per permettere anche ai legami di essere visibili, altrimenti sarebbero nascosti dagli atomi.

Ribbon

La rappresentazione Ribbon viene creata partendo dalle relative strutture dati contenenti le α eliche e β sheet. Ogni record di queste strutture dati memorizza l'aminoacido di partenza e quello di arrivo della struttura secondaria. Il programma, quindi, scorre tutti gli atomi: dall'aminoacido di partenza a quello di arrivo e salva in un vettore tutti gli atomi con il campo "name" a "C" (colonna 13-16): questo perchè la backbone di una proteina tocca appunto tutti gli atomi C.

Viene quindi creata una **Catmull rom curve** [64], ovvero una linea curva passante per questi punti, che forma appunto la struttura secondaria (α elica, β sheet). I loop, non venendo indicati nel file PDB, vengono calcolati per esclusione, ovvero: considerando gli aminoacidi che non fanno parte di nessuna α elica o β sheet, viene tracciato il loop allo stesso modo di quanto precedentemente spiegato. La suddetta linea che forma la struttura secondaria verrà estesa per formare il ribbon.

4.4 Ottimizzazione grafica

È stata effettuata un'operazione di ottimizzazione grafica nella fase di rendering, per permettere all'applicativo di funzionare anche su dispositivi modesti, e per permettere la visualizzazione di molecole con numerosi atomi ad esempio più di 20.000. three.js [2] offre due tipologie di oggetti, **BufferGeometry** e **Geometry**. La prima memorizza in modo efficiente mesh, linee e geometrie, salvando la posizioni dei vertici, indici delle facce, normals, colori, UV e attributi personalizzati, all'interno di buffer (Array tipati), riducendo così il costo del passaggio di tutti questi dati alla GPU.

La seconda invece è un'alternativa più user-friendly rispetto a BufferGeometry. Gli oggetti Geometry memorizzano gli attributi (posizioni dei vertici, facce, colori, ecc.), usando oggetti come Vector3D o Color che sono più facili da leggere e modificare, ma meno efficienti di Array tipati.

Per lo sviluppo dell'applicativo è stato scelto quindi di utilizzare istanze BufferGeometry. Successivamente, gli oggetti BufferGeometry compatibili sono stati uniti in una singola istanza, attraverso l'uso della libreria **BufferGeometryUtils.js** illustrata nella sezione [4.2].

Così facendo, sono state ridotte significativamente le chiamate al renderer WebGL [3], questo rende l'applicativo molto più fluido. Ciò nonostante, questa operazione di

merging è molto costosa in termini di tempo, risultando però accettabile, in quanto il tempo di caricamento di una proteina di medie dimensioni su pc avviene in pochi secondi.

4.5 Interfaccia utente

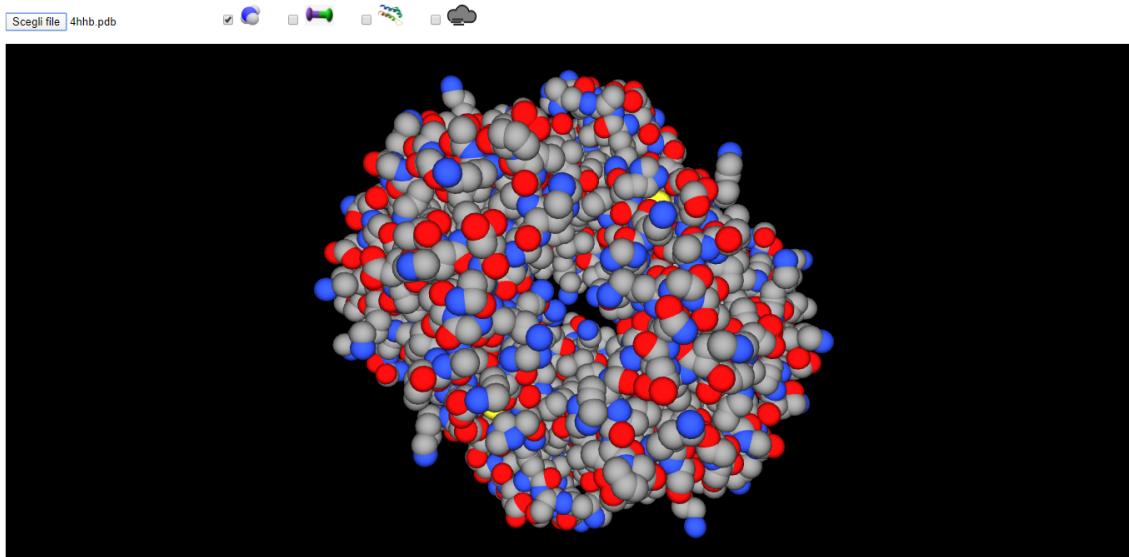


Figura 4.2: Interfaccia grafica contenente la proteina 4Hhb (deossiemoglobina)

L’interfaccia utente nel suo insieme è composta da una form di controllo e da un riquadro nero (canvas).

La form di controllo è costituita da un pulsante e quattro checkbox, che sono poste nella parte superiore della schermata. Partendo da sinistra, il primo pulsante permette di scegliere il file .PDB da elaborare e mostrare a video. Le successive tre checkbox servono per nascondere oppure visualizzare le tre diverse modalità di visualizzazione. L’ultima checkbox, invece, serve per aggiungere all’intera scena 3D un effetto nebbia, utile per comprendere meglio la prospettiva della macromolecola. Sottostante alla form di controllo, è presente il canvas che viene popolato dal renderer WebGL [3]: esso contiene le visualizzazioni desiderate della proteina. All’interno del canvas la proteina può essere ruotata, ingrandita e resa più piccola.

Altro elemento dell’interfaccia utente è il toast popup con le informazioni dell’atomo premuto con il mouse. Tale funzione è disponibile solo da pc dato che cliccare sopra un atomo con un dispositivo mobile risulta scomodo data la ridotta dimensione dello schermo.

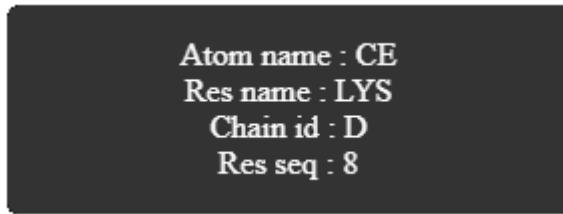


Figura 4.3: Toast pop-up

4.6 Requisiti applicativo

Sono riassunti i seguenti punti per il corretto funzionamento dell'applicativo:

- È consigliata la seguente versione minima dei suddetti browser:

Edge	Firefox	Chrome	Safari	iOS Safari	Chrome Android	UC Android	Samsung Internet
12	44	49	11	11	49	12.12	5

- Il dispositivo deve inoltre essere compatibile con la tecnologia WebGL [3] e di conseguenza OpenGL [59];
- L'algoritmo di parsing è stato scritto per funzionare con file PDB formattati secondo lo standard PDB versione 3.30 [10];
- I record degli atomi devono essere ordinati in modo lessicografico in base al campo "chainID" (colonna 22) e in modo numerico nella stessa catena polipeptidica in base al campo "resSeq" (colonna 23 - 26), come richiesto dallo standard PDB 3.30.

4.7 Testing

È stato condotto un test dell'applicativo su due dispositivi diversi, per verificare la corretta esecuzione del visualizzatore di proteine. Si è deciso di utilizzare le seguenti proteine: (1MH1, 3MDD, 4FP9), perché in esse sono presenti tutte e tre le strutture secondarie (α elica, β sheet e loop), inoltre rappresentano rispettivamente una proteina di piccole, medie e grandi dimensioni.

Nome	1mh1.PDB	3mdd.PDB	4fp9.PDB
Numero Atomi (esclusi Hetatm)	1395	5964	18569
α eliche	8	22	148
β sheet	6	14	48
Catene polipeptidiche	1	2	8

Nella tabelle che seguono vengono descritte le caratteristiche del computer e dello smartphone utilizzati per i test:

Computer	
S.O	Windows 10 Education 64-bit
Browser	Google Chrome 80.0.3987.132
Cpu	Intel Core i7-4720HQ CPU (Quad core @ 2.60GHz)
Gpu	NVIDIA GeForce GTX 960M
Memoria Ram	16 GB

Smartphone	
Nome dispositivo	LG Nexus 5X
Anno produzione	2015
S.O	Android 8.1 (Oreo)
Browser	Google Chrome 80.0.3987.132
Cpu	Qualcomm Snapdragon 808 (Quad core @ 1.44 GHz + Dual core @ 1.8)
Gpu	Adreno 418
Memoria Ram	2 GB

Per ogni test viene indicato il tempo di caricamento della proteina e la fluidità nella rotazione ed esecuzione dello zoom sulla macromolecola, espressa in FPS (fotogrammi per secondo).

Alla fine di ogni test vengono confrontate le tre visualizzazioni create da PDBjs con le analoghe visualizzazioni create da Jmol. Le rappresentazioni potrebbero risultare leggermente dissimili se poste in confronto nei due programmi, a causa di diverse implementazioni e impostazioni grafiche, ad esempio: motore grafico, FOV, ecc.

4.7.1 Primo test 1MH1

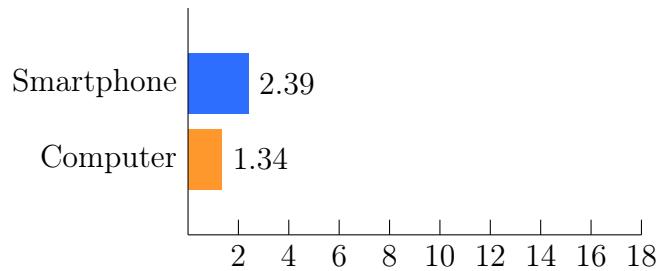


Figura 4.4: Secondi caricamento proteina 1MH1

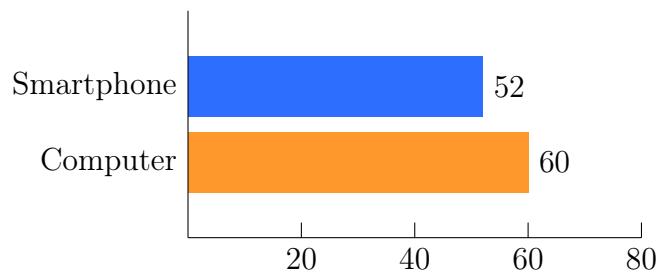


Figura 4.5: Fotogrammi per secondo medi proteina 1MH1

Come si può notare dal grafico in figura 4.4, i due dispositivi impiegano breve tempo per caricare la proteina e la visualizzazione, ruotando, rendendo più piccola e ingrandendo la proteina, risulta molto fluida come indicato nella figura 4.5.

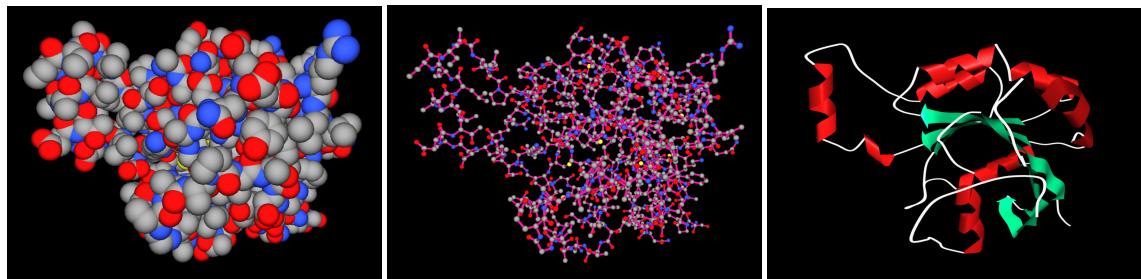


Figura 4.6: Le tre visualizzazioni della proteina 1MH1 elaborate da PDBjs

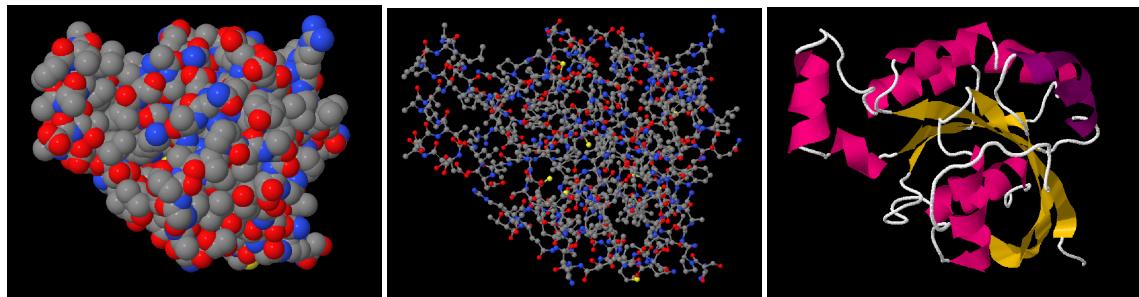


Figura 4.7: Le tre visualizzazioni della proteina 1MH1 elaborate da Jmol

4.7.2 Secondo test 3MDD

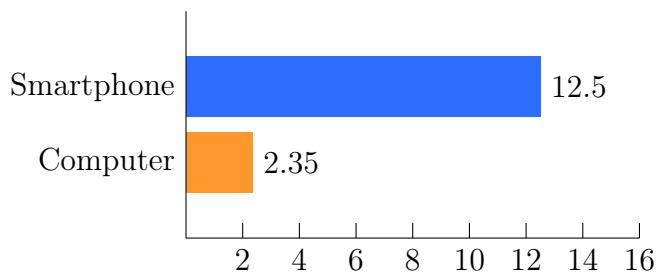


Figura 4.8: Secondi caricamento proteina 3MDD

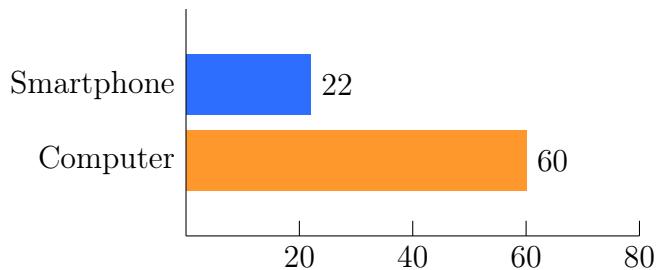


Figura 4.9: Fotogrammi per secondo medi proteina 3MDD

Come è possibile notare dal grafico in figura 4.8, il computer impiega breve tempo per il caricamento della proteina, al contrario, lo smartphone impiega un tempo più ampio, ma esso risulta essere comunque accettabile, considerando che lo smartphone utilizzato sia un dispositivo datato.

Come espresso dal grafico in figura 4.9, la visualizzazione, ruotando, rendendo più piccola e ingrandendo la proteina, risulta molto fluida su pc e sufficientemente fluida su smartphone.

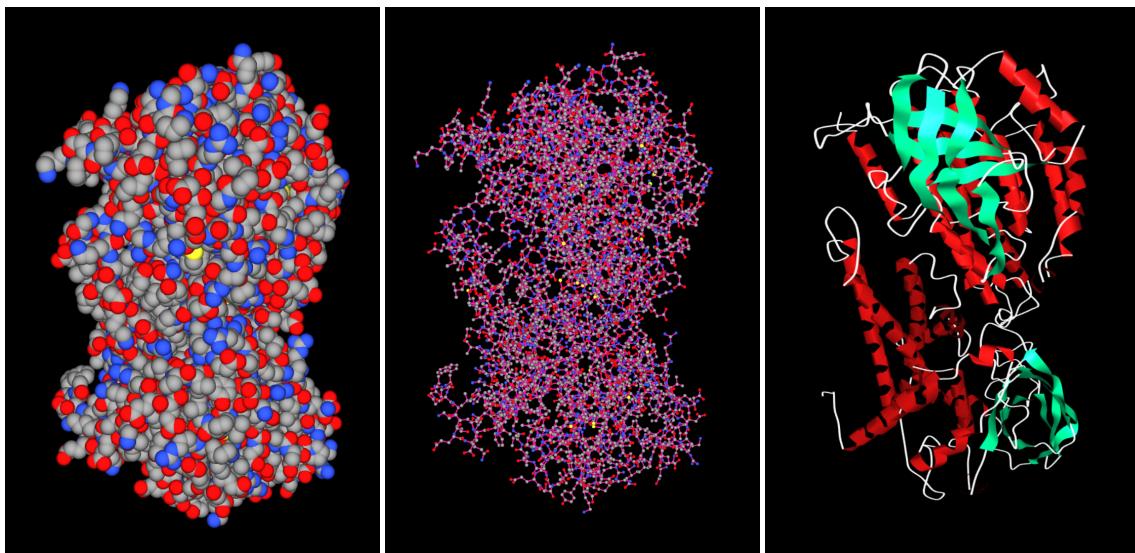


Figura 4.10: Le tre visualizzazioni della proteina 3MDD elaborate da PDBjs

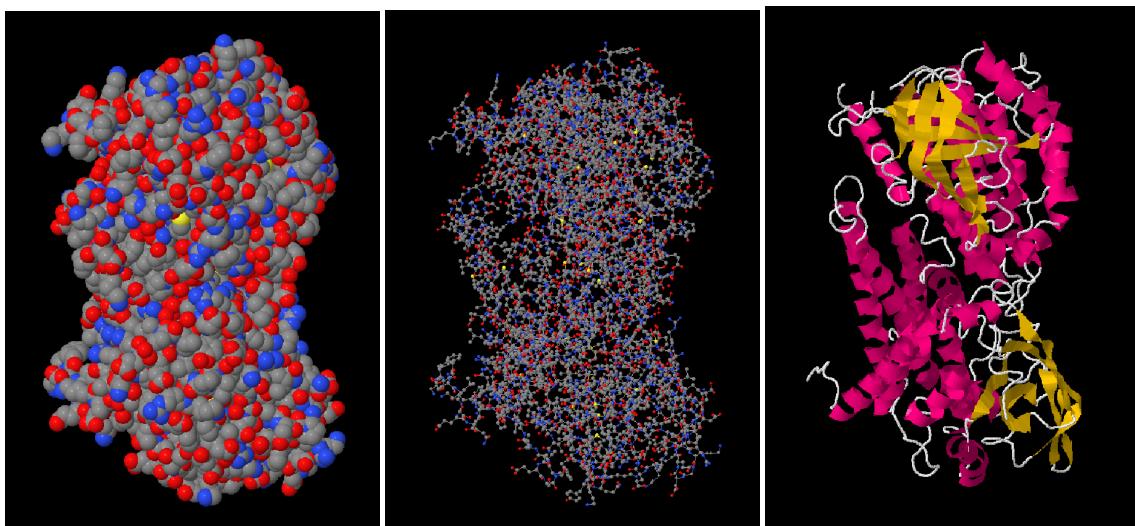


Figura 4.11: Le tre visualizzazioni della proteina 3MDD elaborate da Jmol 

4.7.3 Terzo test 4FP9

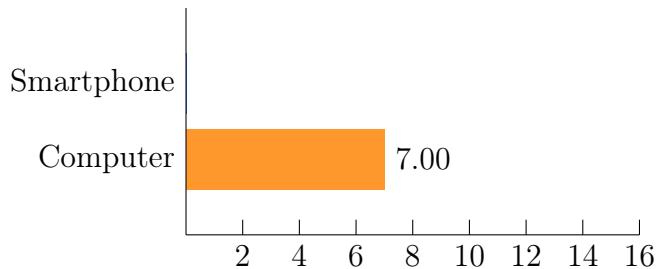


Figura 4.12: Secondi caricamento proteina 4FP9

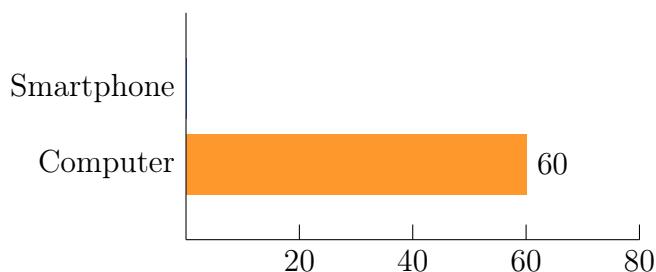


Figura 4.13: Fotogrammi per secondo medi proteina 4FP9

È possibile notare che nel grafico in figura 4.12, non è stato possibile effettuare il terzo test sullo smartphone, a causa della sua insufficiente memoria RAM, la quale causa la chiusura del tab di Google Chrome. La stessa proteina è stata correttamente visualizzata su un altro smarphone dotato di 4GB di memoria RAM.

Su computer la proteina viene caricata in sette secondi, un'attesa ragionevole, in quanto essa risulta essere eccezionale.

Come espresso dal grafico in figura 4.13, la visualizzazione, ruotando, rendendo più piccola e ingrandendo la proteina, risulta comunque molto fluida su pc.

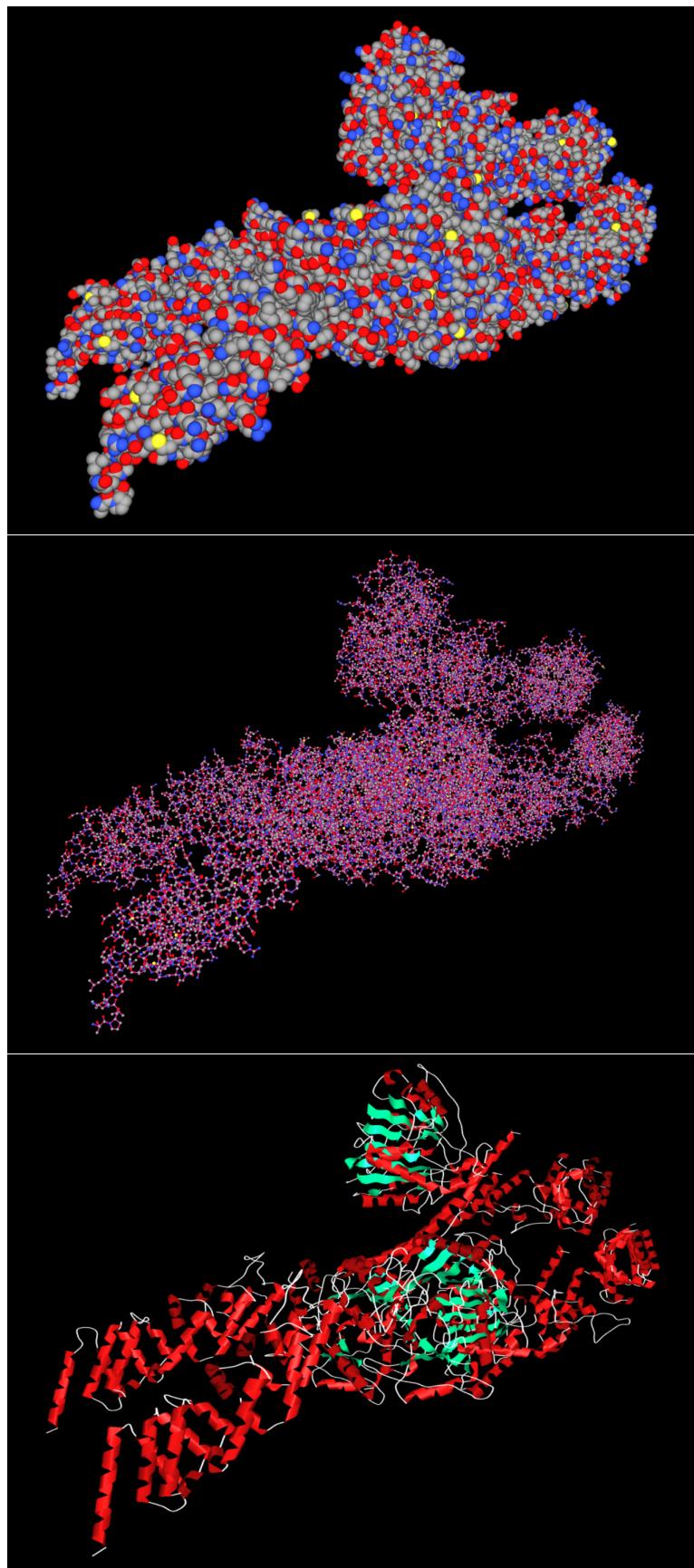


Figura 4.14: Le tre visualizzazioni della proteina 4FP9 elaborate da PDBjs

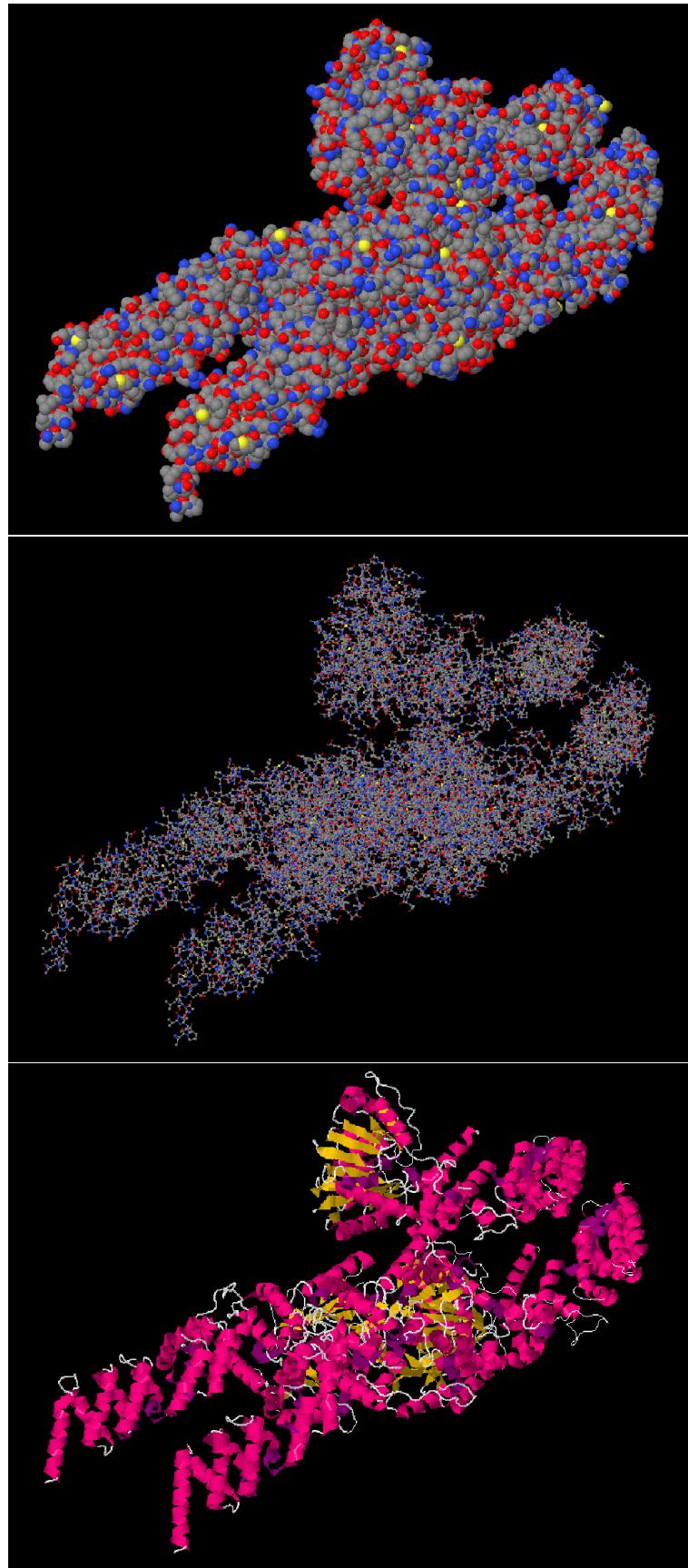


Figura 4.15: Le tre visualizzazioni della proteina 4FP9 elaborate da Jmol [\[1\]](#)

4.7.4 Conclusioni test

Alla luce di quanto emerso dagli esiti dei test effettuati, è possibile concludere che l'applicativo realizzato soddisfa le caratteristiche richieste. È possibile confermare la correttezza dell'output del tool sviluppato, dato che, durante la fase di parsing, non è stato restituito alcun errore, pertanto l'applicativo è riuscito a ricavare e convertire le informazioni necessarie correttamente. Conseguentemente, la posizione nello spazio 3D di atomi e delle strutture secondarie può essere considerata corretta, tale affermazione trova riscontro mediante il confronto effettuato con Jmol, dove emerge la fedeltà e l'attendibilità delle viste create.

Inoltre, è possibile mostrare soddisfazione per quanto concerne il tempo di caricamento e la fluidità del programma tramite PC: le tre proteine sono state caricate in tempi ottimi. In aggiunta, la visualizzazione è rimasta fluida in tutte le tre viste di ciascuna proteina ed è stato possibile ottenere l'informazione dell'atomo, premendo tramite il mouse su ogni macromolecola.

Analizzando quanto accaduto utilizzando l'applicativo sullo smartphone, risulta evidente che non sia stato raggiunto lo stesso successo che con il pc: lo smartphone non è riuscito a caricare tutte e tre le proteine. Quanto accaduto è dovuto al fatto che il device utilizzato era obsoleto e con RAM limitata. Ciononostante il dispositivo è riuscito a caricare le prime due proteine in tempi ragionevoli, con una fluidità sufficiente nella visualizzazione delle viste.

In conclusione, è possibile affermare che la fase di test è stata superata dall'applicativo.

4.8 Evoluzioni

Eventuali evoluzioni dell'applicativo sviluppato potrebbero essere:

- Aggiungere possibilità di colorazione degli atomi, dei legami e della struttura secondaria in base alla catena polipeptidica;
- Aggiungere possibilità di colorazione dei legami, in base al tipo di atomi che essi collegano. Il collegamento sarebbe dunque metà di un colore e metà di un altro, se gli atomi collegati sono di diverso tipo;
- Possibilità di effettuare calcoli tra vari atomi come: distanza tra di essi e angolo planare formato;
- Possibilità di visualizzare una proteina, immettendo il codice che la identifica;
- Aggiungere eventuale virtual reality.

Capitolo 5

Conclusioni

In questo capitolo si illustrano le conclusioni riguardanti i temi affrontati nei vari capitoli di questa tesi.

Nella prima parte della tesi viene condotta una classificazione dei principali tool di visualizzazione 3D reperibili nella letteratura scientifica. Dall’analisi dei tool effettuata sono emerse le caratteristiche tecnologiche ideali per lo sviluppo di PDBjs. Si è dunque optato per un’applicazione web utilizzando il linguaggio di programmazione JavaScript nativo, questo ha permesso di creare un tool che possa operare su moltissimi device differenti e aventi un sistema operativo diverso.

Il linguaggio JavaScript è meno veloce nell’esecuzione di altri linguaggi compilati. Nonostante questo le funzionalità implementate in PDBjs non sono troppo complesse e pesanti dal punto di vista computazionale e questo permette al tool di visualizzare anche proteine di grandi dimensioni.

Nel capitolo dedicato allo sviluppo di PDBjs sono state discusse le soluzioni tecnologiche adottate come ad esempio la tipologia di applicativo, il linguaggio di programmazione e le librerie adoperate. È stato anche discusso il funzionamento dell’applicativo nelle sue subroutine di parsing, calcolo collegamenti e rendering, dove vengono spiegate nel dettaglio le geometrie create nelle varie rappresentazioni della proteina. Viene inoltre condotto un test di correttezza dell’applicativo e un confronto tra il funzionamento di PDBjs e di Jmol [1], dal quale è possibile confermare la correttezza del tool. Infine, vi è una sezione dedicata alla discussione delle possibili evoluzioni dell’applicativo.

Bibliografia

- [1] “Jmol: an open-source java viewer for chemical structures in 3d..” <http://jmol.sourceforge.net/>.
- [2] Three.js. <https://threejs.org/>.
- [3] WebGL. https://www.khronos.org/webgl/wiki/Main_Page.
- [4] C. Mathews and K. Van Holde, *Biochemistry*. Benjamin/Cummings series in the life sciences and chemistry, Benjamin/Cummings Publishing Company, Incorporated, 1996.
- [5] R. Pain, *Mechanisms of Protein Folding*. Frontiers in molecular biology, Oxford University Press, 2000.
- [6] P. Wilson, K. Wilson, and J. Walker, *Principles and Techniques of Practical Biochemistry*. Principles and Techniques of Practical Biochemistry, Cambridge University Press, 2000.
- [7] D. Voet, J. Voet, and C. Pratt, *Principles of Biochemistry*. Wiley, 2008.
- [8] C. Tanford and J. Reynolds, *Nature’s Robots: A History of Proteins*. Oxford Paperbacks, OUP Oxford, 2003.
- [9] RCSB-PDB. <https://www.rcsb.org/>.
- [10] Pdb File Format. <http://www.wwpdb.org/documentation/file-format>.
- [11] R. S. Rowland and R. Taylor, “Intermolecular nonbonded contact distances in organic crystal structures: Comparison with distances expected from van der waals radii,” *The Journal of Physical Chemistry*, vol. 100, no. 18, pp. 7384–7391, 1996.
- [12] A. Bondi, “van der waals volumes and radii,” *The Journal of Physical Chemistry*, vol. 68, no. 3, pp. 441–451, 1964.
- [13] UCSF ChimeraX. <http://www.cgl.ucsf.edu/chimerax/>.
- [14] BioBlender. <http://www.bioblender.org/>.
- [15] A. Lafita, S. Bliven, A. Prlić, D. Guzenko, P. W. Rose, A. Bradley, P. Pavan, D. Myers-Turnbull, Y. Valasatava, M. Heuer, M. Larson, S. K. Burley, and J. M. Duarte, “Biojava 5: A community driven open-source bioinformatics library,” *PLOS Computational Biology*, vol. 15, pp. 1–8, 02 2019.

- [16] BioJava. <https://biojava.org/>.
- [17] S. McNicholas, E. Potterton, K. S. Wilson, and M. E. M. Noble, “Presenting your structures: the *CCP4mg* molecular-graphics software,” *Acta Crystallographica Section D*, vol. 67, pp. 386–394, Apr 2011.
- [18] CCP4MG. <http://www ccp4.ac.uk/MG/>.
- [19] Cm2 MembraneEditor. <https://www.cellmicrocosmos.org/index.php/cm2-project>.
- [20] Cn3D. <https://www.ncbi.nlm.nih.gov/Structure/CN3D/cn3d.shtml>.
- [21] iCn3D. https://www.ncbi.nlm.nih.gov/Structure/icn3d/docs/icn3d_about.html.
- [22] KiNG Kinemages. <http://kinemage.biochem.duke.edu/software/king.php>.
- [23] P. J. Kraulis, “*MOLSCRIPT*: a program to produce both detailed and schematic plots of protein structures,” *Journal of Applied Crystallography*, vol. 24, pp. 946–950, Oct 1991.
- [24] MolScript. <https://kraulis.se/MolScript/>.
- [25] PMV. <http://mgltools.scripps.edu/>.
- [26] RasMol. <http://www.bernstein-plus-sons.com/software/rasmol/>.
- [27] E. A. Merritt and D. J. Bacon, “Raster3d: Photorealistic molecular graphics,” in *Macromolecular Crystallography Part B*, vol. 277 of *Methods in Enzymology*, pp. 505 – 524, Academic Press, 1997.
- [28] Raster3D. <http://skuld.bmsc.washington.edu/raster3d/>.
- [29] RasTop. <http://www.geneinfinity.org/rastop/>.
- [30] RCSBViewers. https://biojava.org/wiki/RCSB_Viewers:About.
- [31] PyMOL. <https://pymol.org/2/>.
- [32] M. Tarini, P. Cignoni, and C. Montani, “Ambient occlusion and edge cueing for enhancing real time molecular visualization,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1237–1244, 2006.
- [33] QuteMol. <http://qutemol.sourceforge.net>.
- [34] Deacon J. Sweeney, Gerald M. Alter, and Michael L. Raymer, “Introducing SPADE, the Structural Proteomics Application Development Environment. Ohio Collaborative Conference on Bioinformatics (OCCBIO),” 2008. University of Toledo.
- [35] SPADE. <https://sites.google.com/view/spade>.

- [36] K. Okonechnikov, O. Golosova, M. Fursov, and the UGENE team, “Unipro UGENE: a unified bioinformatics toolkit,” *Bioinformatics*, vol. 28, pp. 1166–1167, 02 2012.
- [37] Unipro UGENE. <http://ugene.net/>.
- [38] W. Humphrey, A. Dalke, and K. Schulten, “VMD – Visual Molecular Dynamics,” *Journal of Molecular Graphics*, vol. 14, pp. 33–38, 1996.
- [39] VMD. <http://www.ks.uiuc.edu/Research/vmd/>.
- [40] VRmol. <https://vrmol.net/>.
- [41] BRAGI. <https://bragi.helmholtz-hzi.de/>.
- [42] C. R. Reynolds, S. A. Islam, and M. J. Sternberg, “Ezmol: A web server wizard for the rapid visualization and image production of protein and nucleic acid structures,” *Journal of Molecular Biology*, vol. 430, no. 15, pp. 2244 – 2248, 2018. Computation Resources for Molecular Biology.
- [43] EzMol. <http://www.sbg.bio.ic.ac.uk/~ezmol/>.
- [44] ICM-Browser. http://www.molsoft.com/icm_browser.html.
- [45] IcmJS. <http://www.molsoft.com/activeicmjs.html>.
- [46] iMol. <http://www.pirx.com/iMol/index.shtml>.
- [47] MarvinView. <https://chemaxon.com/>.
- [48] POLYVIEW. <http://polyview.cchmc.org/>.
- [49] “Alignment-annotator web server: rendering and annotating sequence alignments.” <http://www.bioinformatics.org/strap/>, 2014. Nucleic Acids Res.
- [50] Swiss PDB viewer. <https://spdbv.vital-it.ch/>.
- [51] UCSF Chimera. <http://www.cgl.ucsf.edu/chimera/>.
- [52] YASARA View. <http://www.yasara.org/>.
- [53] Zeus. <http://www.al-nasir.com/portfolio/zeus/>.
- [54] ZMM. <https://pymol.org/2/>.
- [55] CrystalMaker. <http://www.crystalmaker.com/crystalmaker/>.
- [56] Molecular Operating Environment. <https://www.chemcomp.com/>.
- [57] Nanome. <https://nanome.ai/>.
- [58] ChemDoodle 3D. <https://www.chemdoodle.com/3d>.
- [59] OpenGL. <https://www.opengl.org/>.

- [60] OrbitControls.js. <https://threejs.org/docs/#examples/en/controls/OrbitControls>.
- [61] BufferGeometryUtils.js. <https://threejs.org/docs/#examples/en/utils/BufferGeometryUtils>.
- [62] THREE.MeshLine.js. <https://github.com/spite/THREE.MeshLine>.
- [63] jQuery. <https://jquery.com/>.
- [64] E. E. Catmull and R. Rom, “A class of local interpolating splines,” 1974.
- [65] DirectX. <https://www.microsoft.com/it-it/download/details.aspx?id=35>.

Appendice A

Codice programma

Qui di seguito il codice sorgente di PDBjs, non è incluso il codice delle librerie esterne discusse nella sezione [4.2](#)

Index.html

```
1 <html>
2
3 <head>
4 <title>Visualizzatore Proteine</title>
5 <link rel="stylesheet" href="toast.css">
6 </head>
7 <body>
8
9 <div id="snackbar"></div>
10
11 <form id="myform" enctype="multipart/form-data" method="post">
12   <input id="file" type="file" />
13   <span class="checkbox-wrapper">
14     <input id="Check_atoms" type="checkbox" name="value" checked/>
15     </img>
17   </span>
18   <span class="checkbox-wrapper" style="margin-left: 25px;">
19     <input id="Check_sticks" type="checkbox" name="value" />
20     
21     </img>
22   </span>
23   <span class="checkbox-wrapper" style="margin-left: 25px;">
24     <input id="Check_Secondary" type="checkbox" name="value" />
25      </img>
27   </span>
28   <span class="checkbox-wrapper" style="margin-left: 25px;">
29     <input id="Check_fog" type="checkbox" name="value"/>
30     
31     </img>
32   </span>
33 </form>
34 </body>
35 <script
```

```

36  src="https://code.jquery.com/jquery-3.4.1.min.js"
37  integrity="sha256-CSXorXvZcTkaix6Yvo6HppcZGetbYMGWSflBw8HfCJo="
38  crossorigin="anonymous">></script>
39  <script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/109/three.min.js"
40  crossorigin="anonymous" ></script>
41 <script src="src/Color_Radius.js"></script>
42 <script src="src/OrbitControl.js"></script>
43 <script src="src/Rendering.js"></script>
44 <script src="src/Auxiliary_functions.js"></script>
45 <script src="src/BufferGeometryUtils.js"></script>
46 <script src="src/JavascriptCod.js"></script>
47 <script src="src/THREE.meshLine.js"></script>

```

PDBjs/Index.html

Toast.css

```

1 #snackbar {
2     visibility: hidden; /* Hidden by default. Visible on click */
3     min-width: 250px; /* Set a default minimum width */
4     margin-left: -125px; /* Divide value of min-width by 2 */
5     background-color: #333; /* Black background color */
6     color: #fff; /* White text color */
7     text-align: center; /* Centered text */
8     border-radius: 6px; /* Rounded borders */
9     padding: 16px; /* Padding */
10    position: fixed; /* Sit on top of the screen */
11    z-index: 1; /* Add a z-index if needed */
12    left: 50%; /* Center the snackbar */
13    bottom: 30px; /* 30px from the bottom */
14 }
15
16 /* Show the snackbar when clicking on a button (class added with JavaScript) */
17 #snackbar.show {
18     visibility: visible; /* Show the snackbar */
19     /* Add animation: Take 0.5 seconds to fade in and out the snackbar.
20        However, delay the fade out process for 2.5 seconds */
21     -webkit-animation: fadein 0.5s;
22     animation: fadein 0.5s;
23 }
24
25 #snackbar.hide {
26     visibility: hidden; /* Show the snackbar */
27     /* Add animation: Take 0.5 seconds to fade in and out the snackbar.
28        However, delay the fade out process for 2.5 seconds */
29     -webkit-animation: fadeout 0.5s;
30     animation: fadeout 0.5s;
31 }
32
33 /* Animations to fade the snackbar in and out */
34 @-webkit-keyframes fadein {
35     from {bottom: 0; opacity: 0;}
36     to {bottom: 30px; opacity: 1;}
37 }
38
39 @keyframes fadein {
40     from {bottom: 0; opacity: 0;}

```

```

41     to {bottom: 30px; opacity: 1;}
42 }
43
44 @-webkit-keyframes fadeout {
45     from {bottom: 30px; opacity: 1;}
46     to {bottom: 0; opacity: 0;}
47 }
48
49 @keyframes fadeout {
50     from {bottom: 30px; opacity: 1;}
51     to {bottom: 0; opacity: 0;}
52 }
```

PDBjs/Toast.css

JavascriptCod.js

```

1 //////////////////////////////////////////////////////////////////
2 //VARIABILI GLOBALE
3 //////////////////////////////////////////////////////////////////
4
5 var Atomi = [], AtomBonds = [], Oxigen= [],
6     Helix = [], Sheet = [], Backbone=[];           //strutture dati
7 var AlsoConnect = [];                            //connect
8 var AtomPool1 = [];                            //pool
9 var AtomPool2 = [];
10
11 var media;
12 var mouse = new THREE.Vector2();
13 var mouse2 = new THREE.Vector2();
14 var raycaster = new THREE.Raycaster();
15
16 const GRAFICA = 16;                           //costanti
17 const TOLLERANZA = 0.45;
18 const MOLTATOMI = 0.22, MOLTCPK = 1.0 ,MOLTCOLLEGAMENTI = 0.150;
19 const FOV = 45;
20
21 var ScenaAtomi1 = new THREE.Scene();           //Scene
22 var ScenaAtomi2 = new THREE.Scene();
23 var ScenaAtomBonds = new THREE.Scene();
24 var ScenaSecondary = new THREE.Scene();
25 var Scene = new THREE.Scene();
26
27
28 var renderer = new THREE.WebGLRenderer({antialias : true});
29 var camera = new THREE.PerspectiveCamera(FOV,
30     window.innerWidth/window.innerHeight, 0.1, 500);
31 var controls, ter, disengaRibbon , minY , maxY , minX , maxX, minZ , maxZ;
32
33 var CheckboxAtomi, CheckboxCollegamenti, CheckboxSecondary,
34     CheckboxFog, toast;                         //checkbox and button
35
36
37 //////////////////////////////////////////////////////////////////
38 //Document Ready
39 //////////////////////////////////////////////////////////////////
40 $(document).ready(function() {
```

```
41 //SETTING VARIABILI A DOCUMENTO PRONTO
42 mouse2.x=9999;
43 ScenaSecondary.name = "scenaSecondary";
44 ScenaAtomBonds.name = "collegamenti";
45 ScenaAtomi2.name = "atomi2";
46 ScenaAtomi1.name = "atomi1";
47
48
49 //TOAST POPUP
50 toast = document.getElementById("snackbar");
51 //CHECKBOX
52 CheckboxAtomi = $("#Check_atoms");
53 CheckboxCollegamenti = $("#Check_sticks");
54 CheckboxSecondary = $("#Check_Secondary");
55 CheckboxFog = $("#Check_fog");
56
57
58 /////////////////
59 //CALLBACK CHECKBOX
60 /////////////////
61
62 CheckboxAtomi.on('click', () =>{
63     if(CheckboxAtomi[0].checked==true)
64         Scene.add(ScenaAtomi1);
65     else
66         Scene.remove(Scene.getObjectByName("atomi1"));
67 });
68
69 CheckboxCollegamenti.on('click', () =>{
70     if(CheckboxCollegamenti[0].checked==true){
71         Scene.add(ScenaAtomBonds);
72         Scene.add(ScenaAtomi2);
73     }
74     else{
75         Scene.remove(Scene.getObjectByName("collegamenti"));
76         Scene.remove(Scene.getObjectByName("atomi2"));
77     }
78 });
79
80 CheckboxSecondary.on('click', () =>{
81     if(CheckboxSecondary[0].checked==true)
82         Scene.add(ScenaSecondary);
83     else
84         Scene.remove(Scene.getObjectByName("scenaSecondary"));
85 });
86
87 CheckboxFog.on('click', () =>{
88     if(CheckboxFog[0].checked==true)
89         Scene.fog = new THREE.FogExp2("#262626", 0.02);
90     else
91         Scene.fog = undefined;
92 });
93
94
95 /////////////////
96 //CALLBACK MOUSE, servono per toast popup
97 /////////////////
```

```
98
99 renderer.domElement.addEventListener("mousemove", () =>
100   toast.className = "hide" );
101
102 renderer.domElement.addEventListener('mousedown', (event) => {
103   mouse.x = (event.offsetX / renderer.domElement.width) * 2 - 1;
104   mouse.y = -(event.offsetY / renderer.domElement.height) * 2 + 1;
105 });
106
107 renderer.domElement.addEventListener('mouseup', (event) => {
108   if (mouse.x == ((event.offsetX / renderer.domElement.width) * 2 - 1) &&
109       mouse.y == (-(event.offsetY / renderer.domElement.height) * 2 + 1)) {
110     mouse2.x = mouse.x;
111     mouse2.y = mouse.y;
112   }
113   else
114     mouse2.x = 9999;
115 });
116
117
118 /////////////////
119 //LETTURA FILE PDB
120 /////////////////
121 document.forms['myform'].elements['file'].onchange = function(evt) {
122   if(!window.FileReader) return; // Browser is not compatible
123
124   const name = evt.target.files[0].name;
125   const lastDot = name.lastIndexOf('.');
126   const ext = name.substring(lastDot + 1);
127
128   if(ext != "pdb"){ //file non ha estensione pdb
129     alert("Il file inserito non ha estensione .pdb ! \n"
130       +"Inserire un file con estensione .pdb");
131     return;
132   }
133
134   var reader = new FileReader();
135
136   reader.onload = function(evt) {
137     if(evt.target.readyState != 2) return;
138     if(evt.target.error) {
139       alert('Errore nella lettura del file');
140       return;
141     }
142
143     //inseriamo contenuto file pdb dentro stringa
144     var pdb = evt.target.result;
145
146
147     //reset strutture dati atomi e collegamenti
148     Atomi = [] ; Backbone = [];
149     AtomBonds = [] ; AlsoConnect = [];
150     Helix = [] ; Sheet = [];
151     media = new THREE.Vector3(); ter=0, disengaRibbon=true;
152
153
154     /////////////////
155     //RAGGIUNGIMENTO HELIX SHEET
```

```
155 //////////////////////////////////////////////////////////////////
156 var appo = pdb.indexOf("REMARK 900");
157 if(appo != -1) pdb = pdb.slice(appo, pdb.length);
158 appo = pdb.indexOf("SEQRES");
159 if(appo != -1) pdb = pdb.slice(appo, pdb.length);
160 appo = pdb.indexOf("HETNAM");
161 if(appo != -1) pdb = pdb.slice(appo, pdb.length);
162 appo = pdb.indexOf("FORMUL");
163 if(appo != -1) pdb = pdb.slice(appo, pdb.length);
164
165 //////////////////////////////////////////////////////////////////
166 //PARSING HELIX, SHEET
167 //////////////////////////////////////////////////////////////////
168 //la fase di parsing [U+FFFFD] interamente copera da un blocco try catch
169 //in caso sorgano errori
170
171 try{
172
173     var occur = pdb.indexOf("HELIX");
174     while (occur!=-1){
175         pdb = pdb.slice(occur+4, pdb.length);
176         let startChainId = pdb.slice(15, 16);
177         let start = parseInt(pdb.slice(17, 21));
178         let endChainId = pdb.slice(27, 28);
179         let end = parseInt(pdb.slice(29, 33));
180
181         Helix.push({ startChainId : startChainId , start : start
182                     , endChainId : endChainId , end : end});
183
184         occur = pdb.indexOf("HELIX");
185     }
186     Helix.sort(compareSheetHelix);
187
188     occur = pdb.indexOf("SHEET");
189     while (occur!=-1){
190         pdb = pdb.slice(occur+4, pdb.length);
191         let startChainId = pdb.slice(17, 18);
192         let start = parseInt(pdb.slice(18, 22));
193         let endChainId = pdb.slice(28, 29);
194         let end = parseInt(pdb.slice(29, 33));
195
196         Sheet.push({ startChainId : startChainId , start : start
197                     , endChainId : endChainId , end : end});
198
199         occur = pdb.indexOf("SHEET");
200     }
201
202     Sheet.sort(compareSheetHelix);           //riordino
203     Sheet = Sheet.filter(checkDuplicates);   //rimuovo duplicati
204
205
206 //////////////////////////////////////////////////////////////////
207 //RAGGIUNGIMENTO ATOMI
208 //////////////////////////////////////////////////////////////////
209 appo = pdb.indexOf("SITE");
210 if(appo != -1) pdb = pdb.slice(appo, pdb.length);
211 appo = pdb.indexOf("CRYST1");
```

```
212     if(appo != -1) pdb = pdb.slice(appo, pdb.length);
213     appo = pdb.indexOf("ORIGX1");
214     if(appo != -1) pdb = pdb.slice(appo, pdb.length);
215     appo = pdb.indexOf("SCALE1");
216     if(appo != -1) pdb = pdb.slice(appo, pdb.length);
217     appo = pdb.indexOf("MTRIX1");
218     if(appo != -1) pdb = pdb.slice(appo, pdb.length);
219
220
221 ///////////////////////////////////////////////////////////////////
222 //PARSING ATOMI
223 ///////////////////////////////////////////////////////////////////
224 //prendo in considerazione il primo modello
225 if(pdb.indexOf("MODEL") != -1){
226     pdb = pdb.slice(pdb.indexOf("MODEL"), pdb.indexOf("ENDMDL"));
227     console.log("Proteina multi modello");
228 }
229
230 minY = 1000; maxY = -1000;
231 minX = 1000; maxX = -1000;
232 minZ = 1000; maxZ = -1000;
233
234 occur = pdb.indexOf("ATOM");
235 var occur2 = Number.MAX_SAFE_INTEGER, chainIdPrec;
236
237 while (occur!=-1){
238     pdb = pdb.slice(occur+4, pdb.length);
239
240     //parsing caratteristiche atomo
241     let name = pdb.slice(8, 12).trim();
242     let resName = pdb.slice(13, 16);
243     let chainId = pdb.slice(17, 18);
244     let resSeq = parseInt(pdb.slice(18,22));
245
246     //coordinate
247     let pos = new THREE.Vector3(parseFloat(pdb.slice(26, 34)),
248                               parseFloat(pdb.slice(34, 42)),
249                               parseFloat(pdb.slice(42, 50)));
250     let elem = pdb.slice(72, 74).trim();
251
252     //controllo che gli atomi sia ordinati in modo lessicografico
253     //in base al campo chainID
254     if(chainIdPrec>chainId && disengaRibbon){
255         let string = 'Atomi non ordinati in modo lessicografico' +
256             ' in base al campo \'chainID\'! \n La struttura ' +
257             'secondaria della proteina non verrà disegnata';
258
259         alert(string);
260         console.log(string);
261         disengaRibbon=false;
262     }
263     chainIdPrec=chainId;
264
265
266     //variabili per settare zoom e per calcolo collegamenti
267     minY = (minY > pos.y)? pos.y : minY;
268     maxY = (maxY < pos.y)? pos.y : maxY;
```

```
269     minX = (minX > pos.x)? pos.x : minX;
270     maxX = (maxX < pos.x)? pos.x : maxX;
271     minZ = (minZ > pos.z)? pos.z : minZ;
272     maxZ = (maxZ < pos.z)? pos.z : maxZ;
273
274     media.add(pos);    //media
275
276     let atomo = {pos : pos, elem : elem, name : name,    //atomo
277                  resName : resName, chainId : chainId, resSeq : resSeq};
278
279     Atomi.push(atomo);
280
281     //vengono memorizzati tutti gli atomi con nome 'C'
282     //che compongono la backbone della proteina
283     if(name == "C") Backbone.push(atomo);
284     //vengono memorizzati tutti gli atomi con nome 'O'
285     // questo [U+FFF] necessario per disegnare le beta sheet.
286     if(name == "O")
287     { Oxigen[chainId.toString() + resSeq.toString()] = pos; }
288
289     //TER
290     {
291     occur = pdb.indexOf("ATOM");
292
293     //ottimizzazione TER
294     if(occur > 100 || occur == -1)
295         occur2 = pdb.indexOf('TER');
296
297     //caso atomi finiti e TER finale
298     if(occur===-1 && occur2!=-1){
299         Atomi.push("TER");    ter++;    Backbone.push("TER");
300     }
301
302     //caso fine catena
303     if(occur2 < occur && occur2!=-1 && occur!=-1){
304         Atomi.push("TER");    ter++;    Backbone.push("TER");
305     }
306     }
307 }
308
309 console.log("Catene polipeptidiche trovate : " + ter);
310 media.divideScalar(Atomi.length);
311
312 }catch(all){
313     console.log("Errore fase parsing: " + all.message );
314     alert("Errore fase parsing: " + all.message );
315     return;
316 }
317
318 ////////////////////////////////CALCOLO COLLEGAMENTI -> (raggio1 + raggio2 + TOLLERANZA) T(n)
319 //////////////////////////////
320
321 let nVectX = (maxX-minX)/2;      let nVectY = (maxY-minY)/2;
322 let nVectZ = (maxZ-minZ)/2;      let cont = new Array;
323
324
```

```
326     for(i=0; i<nVectX; i++){
327         cont[i] = new Array();
328         for(j=0; j<nVectY; j++)
329             cont[i][j] = new Array();
330     }
331
332     //INSERIMENTO ATOMI NELLA MATRICE POSIZIONALE
333     Atomi.forEach( (x) => {
334         if(x == "TER") return;
335
336         let indeX = Math.floor(((x.pos.x) - minX)/2);
337         let indeY = Math.floor(((x.pos.y) - minY)/2);
338         let indeZ = Math.floor(((x.pos.z) - minZ)/2);
339         if( cont[indeX][indeY][indeZ] == undefined)
340             cont[indeX][indeY][indeZ] = new Array();
341
342         cont[indeX][indeY][indeZ].push(x);
343     });
344
345     //CALCOLO COLLEGAMENTI
346     let cache = CovalentRadius;
347     for(i=0; i<nVectX; i++)
348         for(j=0; j<nVectY; j++)
349             for(k=0; k<nVectZ; k++){
350                 if(!cont[i][j][k]) continue;
351
352                 let original_length = cont[i][j][k].length;
353                 for(l=0; l<original_length; l++){
354
355                     let atom = cont[i][j][k].pop();
356
357                     let iXs = (i -1 < 0)? 0 : i -1;
358                     let iYs = (j -1 < 0)? 0 : j -1;
359                     let iZs = (k -1 < 0)? 0 : k -1;
360
361                     for(iXs2 = iXs; (iXs2 < i+2) && (iXs2 < nVectX); iXs2++)
362                         for(iYs2 = iYs ; (iYs2 < j+2) && (iYs2 < nVectY); iYs2++)
363                             for(iZs2 = iZs ; (iZs2 < k+2) && (iZs2 < nVectZ); iZs2++){
364
365                             if(!cont[iXs2][iYs2][iZs2]) continue;
366
367                             cont[iXs2][iYs2][iZs2].forEach((other) =>{
368                                 let dist = calcola_Distanza(atom, other);
369                                 let raggio1 = (cache[atom.elem])? cache[atom.elem] : 1.04;
370                                 let raggio2 = (cache[other.elem])? cache[atom.elem] : 1.04;
371
372                                 if(raggio1+raggio2 + TOLLERANZA > dist)
373                                     AtomBonds.push({start : atom.pos, end : other.pos });
374                             });
375                         }
376                     }
377                 }
378
379                 cont.splice(0,cont.length); //viene svuotato il vettore posizionale
380
381                 ////////////////////////////////PARSING COLLEGAMENTI FORZATI
382             
```

```
383 //////////////////////////////////////////////////////////////////
384 occur = pdb.indexOf("CONNECT");
385
386 while (occur!=-1){
387     pdb = pdb.slice(occur+6, pdb.length);
388
389     let atom1 = pdb.slice( 0, 5).trim();           //coordinate
390     let atom2 = pdb.slice( 5, 10).trim();
391     let atom3 = pdb.slice(10, 15).trim();
392     let atom4 = pdb.slice(15, 20).trim();
393     let atom5 = pdb.slice(20, 25).trim();
394     AlsoConnect.push(atom1);
395
396     try{
397         if(collegamentoBuono(atom2)
398             AtomBonds.push({start : Atomi[atom1 -1].pos,
399                             end : Atomi[atom2 -1].pos});
400         if(collegamentoBuono(atom3))
401             AtomBonds.push({start : Atomi[atom1 -1].pos,
402                             end : Atomi[atom3 -1].pos});
403         if(collegamentoBuono(atom4))
404             AtomBonds.push({start : Atomi[atom1 -1].pos,
405                             end : Atomi[atom4 -1].pos});
406         if(collegamentoBuono(atom5))
407             AtomBonds.push({start : Atomi[atom1 -1].pos,
408                             end : Atomi[atom5 -1].pos});
409     }catch(all){
410         console.log("Record CONNECT non formattati correttamente.")
411     }
412
413     occur = pdb.indexOf("CONNECT");
414 }
415     renderizza();
416 };
417     reader.readAsText(evt.target.files[0]);
418 };
419
420
421 //renderer
422 renderer.setSize(window.innerWidth, window.innerHeight);
423 document.body.appendChild(renderer.domElement);
424
425 //control
426 controls = new THREE.OrbitControls( camera, renderer.domElement );
427
428 //illuminazione
429 var light = new THREE.PointLight( 0xffffffff, 1.2, 1000,0 );
430 light.position.set( 0,5,0 );
431 camera.add( light );
432 light = new THREE.AmbientLight( 0x404040, 0.5 ); // soft white light
433 Scene.add( light );
434 Scene.add( camera );
435
436 requestAnimationFrame( animate );
437
438 function animate() {
439     requestAnimationFrame( animate );
```

```

440
441 // algoritmo raycaster serve per conoscere atomo cliccato con mouse
442 if(mouse2.x!=9999 && mouse2.x!=99999 ){
443     raycaster.setFromCamera( mouse, camera );
444     var atom, atom1, atom2;
445
446     Scene.children.forEach((elem) =>{
447         if      (elem.name == "atomi1") atom1 = elem;
448         else if  (elem.name == "atomi2") atom2 = elem;
449     });
450
451     var sceneObj = (atom1) ? atom1 : atom2;
452
453     if(sceneObj){
454         //atomi ball & stick
455         var intersects = raycaster.intersectObject(sceneObj , true);
456
457         if(intersects[0]){
458             var hexColor = intersects[0].object.material.color.getHexString();
459             var atomName = getNameByColor(color, "#" + hexColor.toUpperCase());
460             atom = findClosest(atomName, intersects[0].point);
461         }
462     }
463
464     showToast(atom);
465     mouse2.x=9999;
466 }
467
468 controls.update();
469 renderer.render( Scene, camera );
470 }
471 });

```

PDBjs/JavascriptCod.js

Rendering.js

```

1 function renderizza() {
2
3     //vengono rimosse le scene della proteina precedente
4     Scene.remove(Scene.getObjectByName("atomi1"));
5     Scene.remove(Scene.getObjectByName("atomi2"));
6     Scene.remove(Scene.getObjectByName("collegamenti"));
7     Scene.remove(Scene.getObjectByName("scenaSecondary"));
8
9     renderAllAtoms();
10    renderizzaCollegamenti();
11    if(disengaRibbon) renderizzaSecondary();
12    else{
13        ScenaSecondary = new THREE.Scene();
14        ScenaSecondary.name = "scenaSecondary";
15    }
16
17    if (CheckboxAtomi[0].checked == true) Scene.add(ScenaAtomi1);
18    if (CheckboxCollegamenti[0].checked == true){
19        Scene.add(ScenaAtomBonds); Scene.add(ScenaAtomi2);
20    }

```

```
21 if (CheckboxSecondary[0].checked == true) Scene.add(ScenaSecondary);
22
23 //calcolo fog
24 var c = maxY - minY;
25 var cos = 2-((Math.cos(FOV * Math.PI/180))*2);
26 var distance = Math.sqrt(((c)/cos)*((c)/cos))- (c*c)) * 1.05;
27
28 if (CheckboxFog.checked == true) Scene.fog = new THREE.FogExp2("#262626", 0.02);
29 else Scene.fog = undefined;
30
31 //orbit control e posizione camera
32 var appo = media.clone();
33 controls.target.set(media.x, media.y, media.z);
34 var dist = distance / appo.length();
35 var appo2 = appo.clone().multiplyScalar((dist*1.80));
36 camera.lookAt(appo);
37 camera.position.set(appo2.x, appo2.y, appo2.z);
38 controls.update();
39 renderer.render(Scene, camera);
40}
41
42function renderAllAtoms(){
43
44 //scena CPK
45 ScenaAtomi1 = new THREE.Scene();    ScenaAtomi1.name = "atomi1";
46 //scena ball and stick
47 ScenaAtomi2 = new THREE.Scene();    ScenaAtomi2.name = "atomi2";
48 //variabili contenenti l'unione di tutte le buffergeometry
49 var SingleGeometry = [], SingleGeometry2 = [],
50 valScale = MOLTATOMI/ MOLTCPK;
51
52 Atomi.forEach( (Atom) => {
53
54     if(Atom == "TER") return;
55
56     /////////////////////////////////
57     //CPK ATOMS
58     /////////////////////////////////
59     if(AtomPool1[Atom.elem] == undefined)
60         AtomPool1[Atom.elem] =
61             new THREE.SphereBufferGeometry((radius[Atom.elem] || 1.50) * MOLTCPK,
62                                         GRAFICA, GRAFICA );
63
64     let positionHelper = new THREE.Object3D();
65     positionHelper.position.set(Atom.pos.x, Atom.pos.y, Atom.pos.z);
66     positionHelper.updateWorldMatrix(true, false);
67
68     let geometry = AtomPool1[Atom.elem].clone();
69     geometry.applyMatrix(positionHelper.matrixWorld);
70
71     if(SingleGeometry[Atom.elem] == undefined)
72         SingleGeometry[Atom.elem] = [];
73
74     SingleGeometry[Atom.elem].push(geometry);
75
76     /////////////////////////////////
77 }
```

```
78 //ATOMI BALL AND STICKS
79 /////////////////////////////////
80 if(AtomPool2[Atom.elem] == undefined)
81   AtomPool2[Atom.elem] =
82     AtomPool1[Atom.elem].clone().scale(valScale,valScale,valScale);
83
84 geometry = AtomPool2[Atom.elem].clone();
85 geometry.applyMatrix(positionHelper.matrixWorld);
86
87 if(SingleGeometry2[Atom.elem] == undefined)
88   SingleGeometry2[Atom.elem] = [];
89
90 SingleGeometry2[Atom.elem].push(geometry);
91 });
92
93 for(par in SingleGeometry){
94
95   let material = new THREE.MeshLambertMaterial(
96     { color: (color[par] || "#FF1493" ), fog : true});
97
98   let MergedGeometry = THREE.BufferGeometryUtils.mergeBufferGeometries(
99     SingleGeometry[par], false);
100  ScenaAtomi1.add(new THREE.Mesh(MergedGeometry, material));
101 }
102
103 for(par in SingleGeometry2){
104
105   let material = new THREE.MeshLambertMaterial(
106     { color: (color[par] || "#FF1493" ), fog : true});
107
108   let MergedGeometry = THREE.BufferGeometryUtils.mergeBufferGeometries(
109     SingleGeometry2[par], false);
110   ScenaAtomi2.add(new THREE.Mesh(MergedGeometry, material));
111 }
112
113 console.log("Atomi disegnati: " + (Atomi.length - ter) );
114 }
115
116
117 function renderizzaCollegamenti(){
118
119   ScenaAtomBonds = new THREE.Scene();  ScenaAtomBonds.name = "collegamenti";
120
121   var materiale_cilindro = new THREE.MeshLambertMaterial(
122     { color: "#FF1493" , fog : true});
123   var HALF_PI = Math.PI * .5;
124
125   var SingleGeometry = [];
126
127   AtomBonds.forEach( (Coll) => {
128
129     if(Coll.start == undefined || Coll.end == undefined){
130       console.log("Presenza di un collegamento errato.");
131       return ;
132     } //caso in cui il collegamento sia errato
133
134     let distanza = Coll.start.distanceTo(Coll.end);
```

```
135 let divideScalar2 = Coll.end.clone().add(Coll.start).divideScalar(2);
136
137 let cylinder = new THREE.CylinderBufferGeometry( MOLTCOLLEGAMENTI,
138     MOLTCOLLEGAMENTI, distanza, GRAFICA );
139
140 let orientation = new THREE.Matrix4(); //matrice di orientamento
141 let offsetRotation = new THREE.Matrix4(); //matrice di rotazione
142
143 //guarda a destinazione
144 orientation.lookAt(Coll.start, Coll.end, new THREE.Vector3(0,1,0));
145 offsetRotation.makeRotationX(HALF_PI); //ruotiamo di 90 su asse x
146 //combiniamo l'orientamento con la matrice di rotazione
147 orientation.multiply(offsetRotation);
148
149 orientation.setPosition(divideScalar2.x, divideScalar2.y, divideScalar2.z);
150
151 cylinder.applyMatrix(orientation);
152 SingleGeometry.push(cylinder);
153 );
154
155 if(SingleGeometry.length){
156     var MergedGeometry = THREE.BufferGeometryUtils.mergeBufferGeometries(
157         SingleGeometry, false);
158     ScenaAtomBonds.add(new THREE.Mesh(MergedGeometry, materiale_cilindro));
159 }
160
161 console.log("Legami disegnati: " + AtomBonds.length );
162 }
163
164
165 function renderizzaSecondary(){
166     ScenaSecondary = new THREE.Scene();    ScenaSecondary.name = "scenaSecondary";
167
168     //materiali
169     var materiale_loop = new MeshLineMaterial(
170     { color : "#f8f8f8", lineWidth : 0.3});
171     var materiale_helix = new THREE.MeshPhongMaterial( { color : "#CDOFOF",
172         fog : true, side: THREE.DoubleSide});
173     var materiale_sheet = new THREE.MeshPhongMaterial( { color : "#00ff99",
174         fog : true, side: THREE.DoubleSide});
175
176
177     // rispettivamente, indice for, indice helix, indice sheet
178     var i=0, j=0, k=0,
179     GroupVector = new THREE.Group();    // contenitore THREE.meshline -> loop
180     var SingleGeometry = [] ;          // contenitore bufferGeometry -> sheet, helix
181     var max=0, prec=-5;
182
183
184     ////////////////////////////////CALCOLO CORDINATE HELIX, SHEET, E LOOP////////////////
185     ////////////////////////////////CALCOLO CORDINATE HELIX, SHEET, E LOOP////////////////
186
187
188     while(i< Backbone.length){
189
190         if(Backbone[i]==="TER") {i++; continue;}
191 }
```

```
192 let first = i;
193 let seq = Backbone[i].resSeq;
194 let chainId = Backbone[i].chainId
195 let Vector = [], where;
196 max = (max>i)? max : i;
197
198
199 if(Helix[j] && seq == Helix[j].start && chainId == Helix[j].startChainId){
200     for(; Backbone[i].resSeq <= Helix[j].end; i++)
201         Vector.push(Backbone[i].pos.clone());
202
203     j++; i--; where="helix";
204 }
205 else if(Sheet[k] && seq == Sheet[k].start &&
206         chainId == Sheet[k].startChainId){
207     for(; Backbone[i].resSeq <= Sheet[k].end; i++)
208         Vector.push(Backbone[i].pos.clone());
209
210     k++; i--; where="sheet";
211 }
212 else{
213     for(;(Sheet[k]==undefined || !(Backbone[i].resSeq == (Sheet[k].start+1)
214 && Backbone[i].chainId == Sheet[k].startChainId))
215 && (Helix[j]==undefined || !(Backbone[i].resSeq == (Helix[j].start+1)
216 && Backbone[i].chainId == Helix[j].startChainId)); i++){
217
218         if(Backbone[i] != "TER")
219             Vector.push(Backbone[i].pos.clone());
220
221         else { i=i+2; break; }
222     }
223
224     i--; where="loop";
225 }
226
227
228 if(Vector.length<2){
229     //if per meccanismo anti loop, questo potrebbe succedere quando la
230     //struttura secondaria ha inizio o fine in un etereoatomo che in
231     //questo programma non sono presi in considerazione
232     if(prec == i) i=max+1;
233     else prec=i;
234     continue;
235 }
236
237
238 /////////////////////////////////
239 //DISEGNO HELIX, SHEET AND LOOP
240 /////////////////////////////////
241 let curve = new THREE.CatmullRomCurve3(Vector, false, "chordal", 0.1);
242 var pointsCount = Vector.length*7;
243 var pointsCount1 = pointsCount+1;
244 let startHead = Math.floor(pointsCount/Vector.length);
245 let endHead = pointsCount-startHead;
246
247 switch(where) {
248     case "loop":
```

```
249         disegnaLoop();
250         break;
251     case "sheet":
252         disegnaSheet();
253         break;
254     case "helix":
255         disegnaHelix();
256         break;
257     }
258
259     function disegnaLoop(){
260
261         let loopGeom = new THREE.Geometry().setFromPoints(
262             curve.getPoints(pointsCount));
263         var line = new MeshLine();
264         line.setGeometry( loopGeom );
265         var mesh = new THREE.Mesh( line.geometry, materiale_loop );
266
267         GroupVector.add(mesh);
268     }
269
270     function disegnaSheet(){
271         pointsCount = Vector.length*17;
272         pointsCount1 = pointsCount+1;
273
274         //startHead /= 1.0;
275         endHead = pointsCount-startHead;
276         let endHeadPlus = endHead + startHead/2;
277
278         let SeconVector=[];
279         Vector.forEach( (p) => SeconVector.push(p.clone()) );
280
281         let firstRight = first;
282
283         //PARAMETRO LARGHEZZA SHEET
284         let reverse = 1.1;
285         Vector.forEach(planeSheet);
286
287         firstRight = first, reverse = -reverse;
288         SeconVector.forEach(planeSheet);
289
290         function planeSheet(p, index) {
291             let oxyRight = Backbone[firstRight].chainId.toString() +
292                 Backbone[firstRight++].resSeq.toString() ;
293
294             let appo = Oxigen[oxyRight].clone().add(
295                 p.clone().multiplyScalar(-1));
296
297             if(index%2==0)
298                 p.add(appo.multiplyScalar(reverse));
299             else
300                 p.add(appo.multiplyScalar(-reverse));
301         };
302
303         curve = new THREE.CatmullRomCurve3(Vector, false, "chordal");
304         let curve2 = new THREE.CatmullRomCurve3(SeconVector, false, "chordal");
305
```

```
306 let pts = curve.getPoints(pointsCount);
307 let pts2 = curve2.getPoints(pointsCount);
308
309
310 //////////////////////////////////////////////////////////////////
311 //DISEGNO FRECCIA SHEET
312 //////////////////////////////////////////////////////////////////
313 for(l = Math.floor(endHead)+1 ; l < pts.length; l++){
314     if(l > endHead && l < endHeadPlus){
315         let vect = pts2[l].clone().add(
316             pts[l].clone().multiplyScalar(-1));
317         vect.multiplyScalar(-0.45);
318         pts[l].add(vect.clone().multiplyScalar(
319             (endHeadPlus-1)/(startHead/2)));
320         vect.multiplyScalar(-1);
321         pts2[l].add(vect.clone().multiplyScalar(
322             (endHeadPlus-1)/(startHead/2)));
323     }
324     else if(l > endHeadPlus){
325         let vect = pts2[l].clone().add(
326             pts[l].clone().multiplyScalar(-1));
327         vect.multiplyScalar(0.5);
328         pts[l].add(vect.clone().multiplyScalar(
329             (l-endHeadPlus)/(startHead/2)));
330         vect.multiplyScalar(-1);
331         pts2[l].add(vect.clone().multiplyScalar(
332             (l-endHeadPlus)/(startHead/2)));
333     }
334
335     pts = pts.concat(pts2);
336
337     let sheetGeom = new THREE.BufferGeometry().setFromPoints(pts);
338
339     let indices = [];
340
341     for (ix = 0; ix < pointsCount; ix++) {
342         let a = ix;
343         let b = ix + pointsCount1;
344         let c = (ix + 1) + pointsCount1;
345         let d = (ix + 1);
346         // faces
347         indices.push(a, b, d);
348         indices.push(b, c, d);
349     }
350
351     sheetGeom.setIndex(indices);
352     sheetGeom.computeVertexNormals();
353
354     if(SingleGeometry[where] == undefined)
355         SingleGeometry[where] = [];
356
357     SingleGeometry[where].push(sheetGeom);
358 }
359
360 function disegnaHelix(){
361     var direction = Vector[Vector.length-1].clone().add(
362
```

```
363     Vector[0].clone().multiplyScalar(-1));
364     direction.divideScalar(Vector.length*0.95);
365
366     let pts = curve.getPoints(pointsCount);
367     let pts2 = curve.getPoints(pointsCount);
368
369     pts2.forEach(planeHelix);
370     direction.multiplyScalar(-1);
371     pts.forEach(planeHelix);
372
373     function planeHelix(p, index, arr) {
374         if(index<startHead-1)
375             p.add(direction.clone().multiplyScalar((index+1)/startHead));
376         else if (index>endHead)
377             p.add(direction.clone().multiplyScalar(
378                 (arr.length-index)/ startHead));
379         else
380             p.add(direction)
381     };
382
383     pts = pts.concat(pts2);
384
385     let ribbonGeom = new THREE.BufferGeometry().setFromPoints(pts);
386
387     let indices = [];
388
389     //PlaneBufferGeometry style
390     for (ix = 0; ix < pointsCount; ix++) {
391         let a = ix;
392         let b = ix + pointsCount1;
393         let c = (ix + 1) + pointsCount1;
394         let d = (ix + 1);
395         // faces
396         indices.push(a, b, d);
397         indices.push(b, c, d);
398     }
399
400     ribbonGeom.setIndex(indices);
401     ribbonGeom.computeVertexNormals();
402
403     if(SingleGeometry[where] == undefined)
404         SingleGeometry[where] = [];
405
406     SingleGeometry[where].push(ribbonGeom);
407 }
408 }
409
410 for(par in SingleGeometry){
411
412     let material = (par == 'helix') ? materiale_helix :  materiale_sheet;
413
414     let MergedGeometry = THREE.BufferGeometryUtils.mergeBufferGeometries(
415         SingleGeometry[par], false);
416     ScenaSecondary.add(new THREE.Mesh(MergedGeometry, material));
417 }
418
419
```

```
420 ScenaSecondary.add(GroupVector);
421
422 console.log("Alfa eliche disegnate: " + j + "/" + Helix.length);
423 console.log("Beta sheet disegnate: " + k + "/" + Sheet.length);
424 }
```

PDBjs/Rendering.js

AuxiliaryFunctions.js

```
1
2 function calcola_Distanza(atom1, atom2) {
3     var CordX = Math.pow((atom2.pos.x - atom1.pos.x), 2);
4     var CordY = Math.pow((atom2.pos.y - atom1.pos.y), 2);
5     var CordZ = Math.pow((atom2.pos.z - atom1.pos.z), 2);
6     var dist = Math.pow((CordX + CordY + CordZ), 1 / 2);
7     return dist;
8 }
9
10 function collegamentoBuono(atom) {
11     if (atom == "" || isNaN(atom))
12         return false;
13     else if (AlsoConnect.includes(atom) || atom > Atomi.length)
14         return false;
15     else
16         return true;
17 }
18
19 function findClosest(atomName, point) {
20     var distance = Number.MAX_SAFE_INTEGER, min;
21     Atomi.forEach((Atom) => {
22         if (Atom.elem != atomName)
23             return;
24         var atomDistance = Atom.pos.distanceTo(point);
25         if (distance > atomDistance) {
26             min = Atom;
27             distance = atomDistance;
28         }
29     });
30     return min;
31 }
32
33 function showToast(x) {
34     if (x){
35         toast.className = "show";
36         toast.innerHTML = "Atom name : " + x.name + "<br>Res name : " + x.resName +
37                         "<br>Chain id : " + x.chainId + "<br>Res seq : " + x.resSeq;
38     }
39 }
40
41 function compareSheetHelix(a, b) {
42     if (a.startChainId < b.startChainId)
43         return -1;
44     else if (a.startChainId > b.startChainId)
45         return 1;
46     else if (a.start < b.start)
47         return -1;
```

```
48 else if (a.start > b.start)
49     return 1;
50 else
51     return 0;
52 }
53
54 function checkDuplicates(a, b, array) {
55     var duplicate = true;
56     for (var i = 0; i < b; i++) {
57         if (array[i].startChainId == a.startChainId &&
58             array[i].endChainId == a.endChainId &&
59             array[i].start == a.start) {
60             duplicate = false;
61             break;
62         }
63     }
64     return duplicate;
65 }
```

PDBjs/AuxiliaryFunctions.js

ColorRadius.js

```
1 //CPK color
2 var color = {
3     H : '#FFFFFF',
4     C : '#909090',
5     N : '#3050F8',
6     O : '#FF0D0D',
7     F : '#90E050',
8     Cl : '#1FF01F',
9     Br : '#A62929',
10    I : '#940094',
11    He : '#D9FFFF',
12    Ne : '#B3E3F5',
13    Ar : '#80D1E3',
14    Xe : '#429EB0',
15    Kr : '#5CB8D1',
16    P : '#FF8000',
17    S : '#FFF30',
18    B : '#FFB5B5',
19    Li : '#CC80FF',
20    Na : '#AB5CF2',
21    K : '#8F40D4',
22    Rb : '#702EB0',
23    Cs : '#57178F',
24    Fr : '#420066',
25    Be : '#C2FF00',
26    Mg : '#8AFF00',
27    Ca : '#3DFF00',
28    Sr : '#00FF00',
29    Ba : '#00C900',
30    Ra : '#007D00',
31    Ti : '#BFC2C7',
32    Fe : '#E06633',
33 };
34 }
```

```
35 //funzione inversa: per ricevere nome elemento sapendo colore
36 function getNameByColor(object, value) {
37   return Object.keys(object).find(key => object[key] === value);
38 }
39
40 //raggi di Van Der Waals
41 var radius = {
42   H : 1.1,
43   C : 1.7,
44   N : 1.6,
45   O : 1.55,
46   F : 1.5,
47   P : 1.95,
48   S : 1.8,
49   Cl : 1.8,
50   Cu : 2.0,
51   Zn : 2.1,
52   He : 1.40,
53   Be : 1.9,
54   Ne : 1.54,
55   Hg : 2.05,
56   Cd : 2.2,
57   Ni : 2.0,
58   Pd : 2.05,
59   Au : 2.1,
60   Ag : 2.1,
61   Mg : 2.2,
62   Pt : 2.05,
63   Li : 2.2,
64   Al : 2.1,
65   As : 2.05,
66 };
67
68
69 //raggi Covalenti
70 var CovalentRadius = {
71   H : 0.31,
72   C : 0.73,
73   N : 0.71,
74   O : 0.66,
75   F : 0.57,
76   P : 1.07,
77   S : 1.05,
78   Cl : 1.02,
79   Cu : 1.32,
80   Zn : 1.22,
81   He : 0.28,
82   Be : 0.96,
83   Ne : 0.58,
84   Hg : 1.32,
85   Cd : 1.44,
86   Ni : 1.24,
87   Pd : 1.39,
88   Au : 1.36,
89   Ag : 1.45,
90   Mg : 1.41,
91   Pt : 1.36,
```

```
92     Li : 1.28,  
93     Al : 1.21,  
94     As : 1.19,  
95 };
```

PDBjs/ColorRadius.js