Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

UNIDAD ZACATENCO

Departamento de Computación

Optimización del rendimiento y extensión de la funcionalidad de cuantificación de la aplicación web de visualización molecular HTMoL a través de la implementación de funciones WebGL y un lenguaje de comandos sintácticos

Tesis que presenta:

Omar Israel Lara Ramírez

Para obtener el grado de:

Maestro en Ciencias

En:

Computación

Directores de la tesis:

Dr. Mauricio Carrillo Tripp

Dr. Sergio Víctor Chapa Vergara

Ciudad de México **Diciembre, 2016**

Resumen

El estudio de la estructura y dinámica molecular es fundamental dentro del campo de las ciencias químico-biológicas. En este sentido, visualizar y analizar información a nivel atómico de sistemas biológicos es de esencial importancia. Actualmente, una de las formas más empleadas para el análisis de la estructura y dinámica de las moléculas biológicas involucra la creación de modelos visuales representativos.

La representación de dichos modelos se realiza por medio de diversos algoritmos y metodologías enmarcadas dentro del campo de la Biología Computacional, los cuales permiten realizar estudios a profundidad de las principales características estructurales de interés. Dentro del área de la Biología Molecular, los sistemas de visualización molecular constituyen una poderosa herramienta de análisis, sin embargo, la implementación de dichas metodologías requiere un amplio conocimiento computacional.

El presente proyecto consistió en desarrollar una herramienta que permite simplificar y potenciar el proceso de visualización molecular tridimensional. Se extendieron las capacidades del programa web de visualización molecular HTMoL, generando un sistema interno de análisis y visualización ad hoc basado en un lenguaje de comandos y en un esquema de selección para su uso en el estudio de complejos biomoleculares. Además se optimizó el proceso de renderizado realizando todas las funciones necesarias directamente con WebGL; Se diseñó un API para su fácil integración en páginas web y se trabajó en paralelo con tres tesistas de licenciatura aportando la lectura de archivos estándar con información de dinámica molecular, la integración de una interfaz gráfica eficiente, amigable e intuitiva al usuario, y la implementación de representaciones moleculares avanzadas, en particular, Trace, Ribbon y Spline.

HTMoL v3 es la primer aplicación web para la visualización remota de estructuras y dinámicas moleculares de sistemas biológicos. La herramienta es de código abierto y está disponible de forma gratuita en http://htmol.tripplab.com.

Abstract

The study of the structure and molecular dynamics is essential in the field of chemical and biological sciences. Here, displaying and analyzing information at the atomic level of biological systems is of essential importance. Currently, one of the most used forms for analyzing the structure and dynamics of biological molecules involves the creation of representative visual models.

The representation of these models is performed by various altorithms and methodologies framed within the field of Computational Biology, with allow in depth studies fo the major structural features of interest. Within the area of Molecular Biology, molecular visualization systems are a powerful tool of analysis, however, the implementation of these methods requires extensive computer knowledge.

The present project consisted of developing a tool that simplifies and enhances the process of three-dimensional molecular visualization. The capabilities of the HTMoL molecular visualization web program were extended, generating an internal ad hoc analysis and visualization system based on a command language and a selection scheme for use in the study of biomolecular complexes. In addition, the rendering process was optimized by performing all necessary functions directly with WebGL; An API was designed for easy integration into web pages and worked in parallel with three undergraduate theses providing standard file reading with molecular dynamics information, integration of an efficient, user-friendly and intuitive graphical user interface, and implementation Of advanced molecular representations, in particular, Trace, Ribbon and Spline.

HTMoL v3 is the first web application for the remote visualization of structures and molecular dynamics of biological systems. The tool is open source and available for free at http://htmol.tripplab.com.

Agradecimientos

A Dios

El ser supremo de todo el universo por bendecirme con una hermosa familia y por la oportunidad de llegar hasta este momento, porque sin ti sería imposible realizar este sueño anhelado.

Al CONACyT

Por el gran apoyo que se nos brinda con el cual podemos seguir adelante y darnos cuenta que no hay límites ni pretextos para seguir estudiando, sin su apoyo económico muchos nos quedaríamos a la mitad del camino. Gracias por su confianza puesta en mí.

Al CINVESTAV

Por abrime sus puertas y poder ser parte de esta familia de grandes investigadores que buscan siempre mostrarnos el camino hacia un estilo de vida de excelencia en el estudio. Por bridarme todos los recursos necesarios en sus instalaciones para mi formación, y contar las 24 horas con sus instalaciones.

Al Dr. Mauricio

Por su generosidad al brindarme la oportunidad de recurrir a su capacidad y experiencia en un marco de confiaza, afecto y amistad, fundametales para la finalización de este trabajo de tesis.

Al Dr. Chapa

Una gran persona que puso su confianza en mí al emprender este proyecto. Su orientación y paciencia fueron la clave fundamental para mi correcto aprendizaje.

A mi amada esposa

Eres la ayuda idónea que Dios puso en mi camino, gracias por tu apoyo incondicional en todo momento, por tu comprension en mis noches de desvelo, por tu motivación y regaños a concentrarme en realizar un buen proyecto. Te amo chiquita.

Al personal del Departamento de Computación y compañeros

A Sofi, gracias por brindarnos todo tu apoyo y amistad y ser como una madre que siempre está protegiendo a sus hijos. A los auxiliares de investigación del departamento, ustedes son personas dedicadas completamente al servicio de los demás, hicieron sentir mi estancia en el Cinvestav como si fuese mi hogar. A mis compaẽros, siempre me exhortaron a seguir adelante, juntos pasamos momentos difíciles y de alegría en este proceso de formación.

A mis hijos

Jordy, Joselín y una bebé que viene en camino, son mi orgullo y más grande

motivación, me impulsan a superarme día con día y ofrecer siempre lo mejor de mí. Su presencia en mi vida hace que todo mi esfuerzo valga la pena al final del día.

A mi papá y a mi mamá

Mamá eres un ángel que Dios puso en mi camino, todo lo que tengo lo he logrado gracias a los consejos que me dabas de niño, tu amor para mí es invaluable. Tu junto con mi papá me educaron para ser una persona con buenos principios, y al servicio de los demás. Son el ejemplo a seguir en mi vida. Papá admiro mucho tu integridad ante cualquier circunstancia, aunque haya una tormenta tú siempre estás de pie. Gracias los amo.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La Biología Computacional es la interrelación entre el área de la Biología y la Computación. En esta disciplina la Biología se apoya grandemente en la Ciencia de la Computación, los métodos matemáticos-computacionales y la tecnología. La Bioinformática es la aplicación de tecnologías computacionales a la gestión y análisis de datos biológicos. Esta busca aportar soluciones capaces de manejar dicha información. En la actualidad se ha tenido un avance impresionante en las técnicas experimentales para la determinación de estructuras moleculares. Entre los principales métodos resaltan la resonancia magnética nuclear y la cristalografía de rayos X. Gracias a este avance, se ha incrementado en un número significativo los portales públicos de bases de datos conteniendo información tridimensional de estructuras moleculares [1][4][9].

Debido al incremento exponencial de este tipo de información la Biología Computacional constantemente está buscando el proveer aplicaciones capaces de realizar un análisis eficiente de las estructuras moleculares almacenadas en las distintas bases de datos. Una de las aportaciones más importantes de las últimas décadas son los programas de *visualización molecular*, los cuales permiten explorar y visualizar en distintas formas diferentes tipos de biomoléculas, proporcionadas generalmente en un formato estándar llamado Protein Data Bank (PDB).

Por lo general, dichas aplicaciones son de escritorio, y permiten ejecutar comandos interactivos tecleados por el usuario con la finalidad de realizar un análisis estructural preciso y detallado. No obstante una de las desventejas que se han presentado es que requieren del seguimiento de un proceso en muchas ocasiones complicado para ser instalado en alguna de sus versiones existentes para los principales sistemas operativos. En la actualidad, existen visualizadores moleculares de escritorio (altamente funcionales) y web (altamente distribuibles).

En particular, HTMoL es una aplicación web ligera de fácil instalación que permite al navegador desplegar la estructura tridimensional de moléculas en una manera dinámica e interactiva la cual no requiere de la instalación de ningún otro plugin para su funcionamiento. HTMoL es la primer apllicación de su tipo que permite visualizar datos de dinámica molecular de forma remota. Sin embargo, su usabilidad se ve muy limitada en su versión 2 dado que no posee la capacidad de interacción por medio de comandos, si no que sólo permite cambios generales de una representación molecular

a otra a través de una GUI. Además la eficiencia de renderizado se ve muy afectado debido a que utiliza la ayuda de un framework desarrollado por terceros para la programación de la API de WebGL.

Con el presente proyecto se pretende extender la funcionalidad de HTMoL al proveer una herramienta ad hoc, la cual permita la interacción directa con la información estructural de interés. Esto permitirá al usuario obtener un conocimiento claro y preciso de las estructuras macromoleculares y su dinámica molecular como es el caso de las cápsides virales y otras proteínas, con un renderizado óptimo agilizando así el conocimiento de frontera en el campo de la biología estructural.

1.1. Antecedentes y motivación

La visualización tridimensional de estructuras moleculares en general es un problema computacional que ha llamado mucho la atención. Desde sus inicios, uno de los mayores enigmas de la biocomputación fue la determinación de estructuras tridimensionales para diferentes moléculas. Este campo fue abordado por cristalógrafos, matemáticos y físicos que construyendo modelos teóricos llegaron a ciertos grados interesantes de predicción.

Los logros obtenidos en la visualización estructural, llevarón, junto con otras investigaciones, a la creación de bancos de datos en los que se almacena la información de las pocisiones espaciales de cada uno de los átomos que componen a una molécula [2]. Enseguida, los datos fueron puestos a disposición del público en plataformas accesadas a través de internet. Finalmente, su administración se volvió entonces una tarea de especialistas, llamados curadores. Actualmente, existen diferentes clases de bancos de datos con información estructural desde proteínas hasta ácidos nucleicos [3].

Con el avance de las tecnologías y el incremento de la información se hizo evidente la necesiad de permitir, a quienes no contaban con una estación de trabajo, la visualización y el análisis de estructuras moleculares representadas en archivos de coordenadas moleculares a través de la web. Surgen entonces, programas de gráficos moleculares, capaces de interpretar y presentar al usuario la estructura molecular tridimensional, con el objetivo de proveer un análisis completo. HTMoL es uno de estos programas, el cual permite de manera inicial visualizar moléculas, pero además brinda la posibilidad de manipularlas inclusive a nivel atómico.

HTMoL analiza un archivo con información estructural de la molécula y despliega en pantalla el modelo gráfico, el cual es resultado de la localización de cada uno de los átomos de la molécula sobre un espacio de coordenadas cartesianas x, y, z y de la construcción de un grafo que relaciona las entidades que componen el sistema.

La representación de una molécula se entiende como una sucesión de puntos localizados en el espacio tridimensional x, y, z interconectados mediante ciertas reglas predeterminadas, por ejemplo, aquellas coordenadas relativas a los ángulos y distancias de enlace [7]. Al leer el archivo que contiene esta información HTMoL procesa los datos y muestra en pantalla un sistema coherente.

Es deseable que una vez que HTMoL despliega la estructura, permita analizarla mediante un conjunto de sencillos comandos que, por ejemplo permitan seleccionar elementos de interés, además de la creación de scripts o animaciones, los cuales permiten mostrar y analizar regiones importantes de una molécula.

1.2. Planteamiento del problema

Las moléculas biológicas, como las proteínas y los ácidos nucleicos, son los motores principales de la célula viva. Conocer su estructura tridimensional y la manera en que interactúan entre ellas contribuye a entender el funcionamiento de la maquina celular. Además, la gran cantidad de información genómica y estructural disponible requiere del uso de herramientas computacionales para conseguir una perspectiva sintética de los mecanismos, de la interacción y de las redes subyacentes en la acción de la célula.

Los programas de visualización molecular son una herramienta sumamente útil y algunos de ellos se encuentran disponibles de forma gratuita, siendo además utilizados en diversos trabajos científicos para el análisis de moléculas. Además, su uso se relaciona con el manejo de información científica publicada en las diferentes bases de datos disponibles [1].

El análisis de macroestructuras moleculares complejas se ve afectado por la poca interacción que maneja la mayoría de estos programas, los cuales no permiten una selección específica de la estructura para resaltarla y realizar cuantificaciones sobre ella, y además por el rendimiento poco óptimo de gráficos que presentan al tener que renderizar una gran cantidad de estructuras geométricas.

En el caso específico de HTMoL en su versión 2.0 también se ve afectado el renderizado de proteínas y macromoléculas, debido a que este ocupa un framework llamado three.js el cual se emplea para la programación de gráficos 3D dejando a un lado la optimización de recursos de la tarjeta gráfica.

En este sentido, los problemas que se plantean abordar en este trabajo son, por un lado, implementar un lenguaje que permita la interacción directa con la información estructural que se esté analizando a través de comandos, y por otro, evitar el uso de frameworks de forma que se optimize el despliegue de tales estructuras en HTMoL.

1.3. Justificación

Para el estudio de estructuras moleculares que contienen una gran cantidad de datos atómicos espaciales es necesario contar con una herramienta eficiente y amigable que permita un análisis adecuado y eficaz hacia el usuario. Por tal motivo surge la siguiente pregunta de investigación:

¿Es posible optimizar el análisis de macroestructuras biomoleculares en HTMoL a través de la implementación de un lenguaje de comandos y la eliminación de trhee.js?

Como se puede observar a través de la pregunta de investigación la motivación de este trabajo tiene sus origenes en la necesidad de una herramienta capáz de analizar de manera eficiente las estructuras macromoleculares.

En este trabajo se aborda la integración de un lenguaje de comandos amigable al usuario para que se pueda tener una mejor interacción con las selecciones moleculares, y a su vez se estudia la optimización del renderizado de gráficos 3D acelerados por la tarjeta gráfica (GPU) con la ayuda de WebGL. Adicionalmente, se coordinó un equipo de desarrollo para implementar y optimizar ciertas funciones adicionales, como la lectura de todos los formatos de información estructural y dinámica molecular, inclusión de representaciones moleculares avanzadas, y un menú intuitivo y poco intrusivo.

1.4. Objetivos del proyecto

1.4.1. Objetivo general

Incorporar el soporte de un lenguaje de scripts y funciones de renderización basada en WebGL en la aplicación web de visualización 3D remota HTMoL, de forma constitutiva para así proveer funcionalidades de caracterización y análisis estructural a nivel atómico para agilizar el estudio de macroestructuras biomoleculares. Esto elevará el versionamiento de HTMoL de su versión actual, 2.0, a 3.0.

1.4.2. Objetivos particulares

Para alcanzar el objetivo general, se plantean los siguientes objetivos particulares:

- Analizar la gramática de RasMol e implementar las funciones requeridas en HTMoL para la correcta selección y representación visual de biomoléculas en el contexto de la macroestructura biológica.
- Diseñar un analizador sintáctico basado en el empleado por RasMol e implementarlo en HTMoL para el manejo de comandos, integrándolo a la funcionalidad del despliegue de dinámica molecular.
- Diseñar la arquitectura adicional necesaria en HTMoL para la implementación de funcionalidades de análisis estructural cuantitativo.
- Incorporar el soporte de Pipeline que ofrece WebGL para la optimización del renderizado.
- Diseñar un API en HTMoL para que este pueda ser integrado fácilmente a cualquier sitio web de análisis estructural molecular que soporte el estándar de WebGL.

Capítulo 2

Marco teórico

En el presente capítulo se describen los principales conceptos de los diferentes temas relacionados con el proyecto con la finalidad de tener una mejor comprensión de su desarrollo.

2.1. Conceptos Moleculares

2.1.1. Átomo

Desde un punto de vista clásico, los átomos son la unida básica de toda la materia con propiedades químicas bien definidas [10].

2.1.2. Molécula

La molécula se encuentra formada por dos o más átomos. Los átomos que forman una molécula pueden ser iguales (como es el caso de la molécula del oxígeno, la cual cuenta con dos átomos de oxígeno) o distintos. Las moléculas se encuentran en constante movimiento, y esto se conoce como vibraciones moleculares. Los átomos se mantienen unidos por enlaces covalentes en los que los átomos comparten electrones. En la figura 2.1 se muestra un ejemplo de la molécula del metano, la cual contiene cuatro hidrógenos enlazados a un carbono.

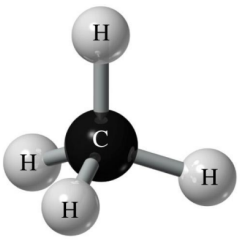

Figura 2.1: Representación de la molécula del metano.

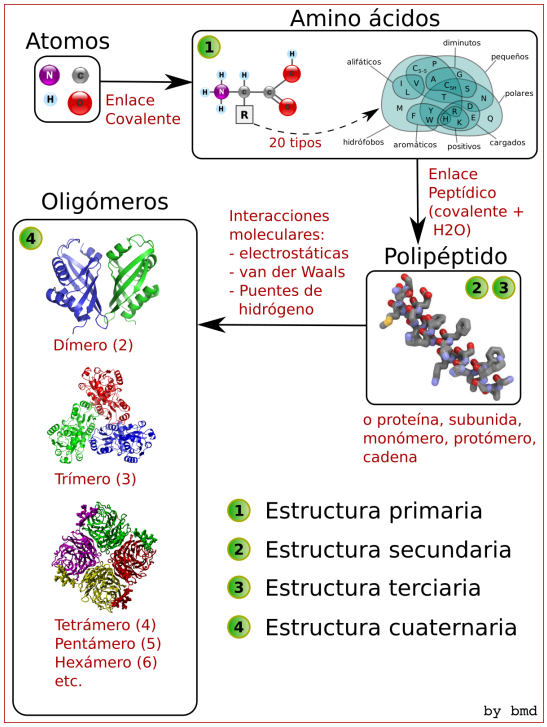

Figura 2.2: Jerarquía de los niveles de organización de la proteína

2.1.3. Aminoácido

Es un ácido orgánico que contiene un grupo amino (NH2) y un grupo carboxilo (COOH) unidos a un mismo carbono central denominado carbono alfa (Ca). A este carbono también se encuentra unido un hidrógeno y un grupo R. Los aminoácidos son moléculas orgánicas que se combinan para formar proteínas, formando así los pilares fundamentales de la vida [15].

2.1.4. Proteína

Las proteínas son biomoléculas formadas por cadenas lineales de aminoácidos. Estas desempeñan un papel fundamental para la vida y son las biomoléculas más versátiles y diversas. La organización de una proteína está definida por cuatro niveles estructurales denominados: estructura primaria, secundaria, terciaria y cuaternaria. Cada una de estas informa la disposición de la anterior en el espacio [15]. En la figura 2.2 se muestra el proceso que contempla los cuatro niveles de organización de una proteína.

2.2. Visualización molecular

La visualización de biomoléculas de manera tridimensional es una de las áreas que ha tenido mayor difusión y en general que ha acaparado mayor atención en el ámbito de

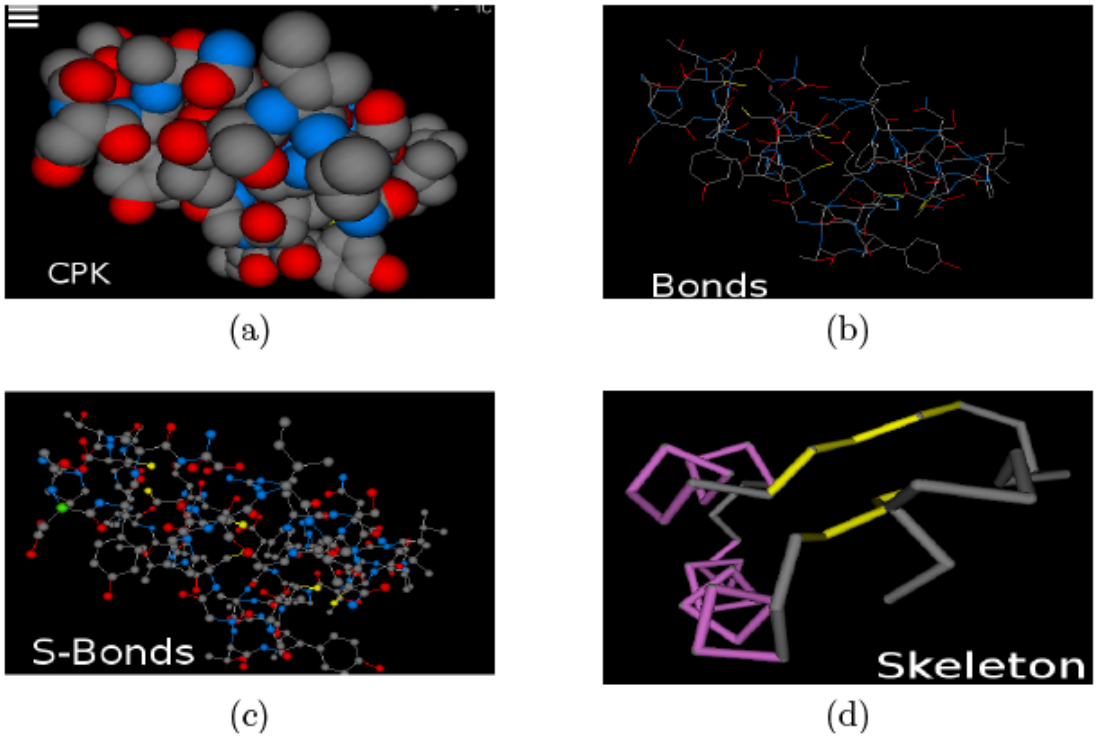

Figura 2.3: Representaciones moleculares en el visualizador HTMoL

la biología molecular moderna. El uso de un software para examinar en forma interactiva modelos moleculares digitales 3D representa un gran avance en la biología y otras ciencias afines. La tecnología de visualización molecular 3D ofrece posibilidades muy amplias para apreciar mejor el tamaño, el volumen y la disposición relativa de los átomos en el espacio. Así mismo la visualización molecular en 3D ha introducido profundos cambios en la manera de concebir la enseñanza, por lo que se ha ido incorporando como una poderosa herramienta en el proceso de enseñanza-aprendizaje de las áreas biológicas [34].

2.2.1. Representaciones moleculares

Existe un amplio rango de representaciones moleculares básicas, estilos de coloración, transparencia, y propiedades de los materiales que ayudan a analizar de mejor manera la estructura de las moléculas; Algunos ejemplos de estas representaciones son:

- CPK: Corey-Pauling-Koltun, también llamado esfera de radio de Van der Waals. En la figura 2.3 inciso *a* se muestra la representación CPK de la de la proteína con PDBID 1CRN.

- Bonds: Dibuja lineas simples representando los enlaces entre los átomos. En la figura 2.3 inciso *b* se muestra la representación Bonds de la proteína con PDBID 1CRN.

- S-Bonds: Escala el radio de Van der Waals de los átomos y dibuja líneas en los enlaces. En la figura 2.3 inciso *c* se puede observar la representación S-Bonds de la proteína con PDBID 1CRN.

- Backbone: Dibuja un tubo por cada enlace de los carbono alfa de cada aminoácido. En la figura 2.3 inciso *d* se muestra la representación Skeleton de la proteína con PDBID 1CRN

2.2.2. Bibliotecas de desarrollo 3D

Existen distintas tecnologías para desplegar gráficos 3D en computadoras personales; Las que se utilizan más ampliamente son *OpenGL*, *DirectX* y *WebGL*. En esta sección se describen las principales características de cada una de las tecnologías mencionadas.

Open GL

OpenGL fue originalente desarrollado por Sillicon Graphics Inc. y publidado como un estándar abierto en 1992. OpenGL ha evolucionado a través de varias versiones desde 1992 y ha tenido un efecto profundo sobre el desarrollo de gráficos en 3D, el desarrollo de productos de software e incluso la producción de películas. OpenGL fue diseñado para la infografía en animación o efectos especiales principalmente en películas o comerciales [22]. OpenGL facilita la interacción con el hardware gráfico de nuestra máquina. Consta de distintas funciones, que permiten producir aplicaciones interactivas en la que intervengan gráficos en tres dimensiones. Está diseñado de forma completamente independiente del hardware, por lo que puede implementarse en plataformas muy diversas (PC, SGI, Digital, SUN, etc.) y con diversos sistemas operativos (Linux, Windows, UNIX, etc.). El precio que hay que pagar en aras de esta portabilidad es que OpenGL no incluye comandos para gestionar el sistema de ventanas, ni para capturar órdenes de los usuarios, ya sea por ratón o por teclado. En lugar de esto, debemos trabajar con la ayuda de cualquier sistema de ventanas que se utilice en la máquina sobre la que estemos desarrollando nuestra aplicación. Igualmente OpenGL tampoco incluye comandos de alto nivel para modelar objetos complejos. Para este trabajo, se incluye una serie de primitivas geométricas muy básicas, tales como la definición de puntos, líneas y polígonos. Existen herramientas que internamente utilizan las bibliotecas de OpenGL y que ofrecen un mayor nivel de abstracción a la hora de desarrollar aplicaciones gráficas. Un ejemplo de ello lo constituyen las bibliotecas de OpenInventor, con las que se puede implementar fácilmente aplicaciones interactivas sin necesidad de conocer en profundidad los comandos de bajo nivel. Las operaciones que se pueden realizar con OpenGL son las siguientes:

- Modelar figuras a partir de las primitivas básicas, creando descripciones geométricas de los objetos (puntos, líneas, polígonos, fotografías, mapas de bits, etc.).

- Situar los objetos en el espacio tridimensional de la escena y seleccionar el punto de vista desde el cual queremos observarla.

- Calcular el color de todos los objetos. El color puede asignarse explícitamente a cada pixel, o bien puede calcularse a partir de las condiciones de iluminación, o también puede asignarse por medio de una textura colocada sobre los objetos.

- Convertir la descripción matemática de los objetos y la información de color asociada, en pixeles de la pantalla, en forma de imagen virtual 3D.

A la vez que se realiza este proceso, OpenGL desarrolla otras operaciones complejas como, por ejemplo, la eliminación de partes de objetos que quedan ocultas para el usuario por estar tapadas por otros objetos de la escena.

DirectX

DirectX es un conjunto de librerías para el desarrollo de juegos o aplicaciones relacionadas con gráficos, sobre plataformas de Microsoft, incluyendo Windows, Windows Phone, Xbox 360 y Xbox One. Ha evolucionado desde mediados de los 90 y ha dado lugar al desarrollo de gráficos modernos. DirectX está estrechamente integrado a la plataforma Windows, por lo que no trabaja con ningún otro sistema operativo [20].

Direct 3D es la biblioteca de renderizado que proporciona acceso a la tarjeta gráfica para renderizar gráficos 2D y 3D. Es el componente más importante de DirectX. Direct3D es el sistema que se utiliza para dibujar los gráficos 3D, y define una secuencia de pasos que se utilizará para presentar los gráficos en la pantalla.

Web-based Graphics Languaje (WebGL)

Es una nueva tecnología que permite crear potentes gráficos 3D dentro de un navegador Web. La forma en que esto se logra es mediante el uso de una API de JavaScript que interactúa con la unidad de procesamiento gráfico (GPU). Algunos críticos anteriores de Direct3D reconocen que ahora es tan bueno o mejor que OpenGL en términos de capacidades y facilidad de uso [16]. Aunque WebGL tiene sus origenes en OpenGL, este es actualmente un derivado de una versión de OpenGL específicamente diseñada para computadoras empotradas, tales como teléfonos inteligentes y consolas de videojuegos. Esta versión conocida como OpenGL ES (para Sistemas Empotrados), fue originalmente desarrollada en 2003-2004 y actualizada en 2007 (ES 2.0) y posteriormente en 2012 (ES 3.0). WebGL esta basado en la versión ES 2.0. En los últimos años, el número de dispositivos y procesadores que soportan WebGL se ha incrementado rápidamente incluyendo teléfonos inteligentes, tablets y consolas de videojuego [17]. WebGL es una tecnología que permite dibujar, desplegar e interactuar con gráficos de computadora tridimensionales interactivos mediante navegadores Web. Tradicionalmente los gráficos 3D han sido restringidos a computadoras de alta gama o a consolas de videojuegos dedicadas, requiriendo programación compleja. Sin embargo, tanto las computadoras personales, así como los navegadores Web más importantes, se han sofisticado para hacer posible la creación y el despliegue de gráficos 3D, utilizando tecnologías Web accesibles y bien conocidas.

2.2.3. Lectura del archivo Protein Data Bank

El formato de archivos Protein Data Bank (PDB) es una forma estándar para representar la estructura tridimensional de las proteínas y ácidos nucléicos. Estos datos, generalmente obtenidos mediante cristalografía, rayos X y resonancia magnética nuclear, son generados por biólogos y bioquímicos de todo el mundo. Estos archivos están bajo el dominio público y puden ser usados libremente. El almacenamiento de los datos se realiza a través de un formato uniforme que se denomina *PDB*, el cual contiene las coordenadas y enlaces parciales atómicos, como las que se derivan de los estudios de la cristalografía. La obtención de información de un archivo en este formato nos brinda una descripción que contiene las coordenadas atómicas de una molécula en cuestión. El estándar en que está organizada la información de los átomos se divide en columnas [38]. Un complejo molecular proteínico (oligómero) puede contener una o más cadenas, y a su vez cada una de estas cadenas contienen un conjunto de aminoácidos los cuales están formados por un grupo de átomos. En la figura 2.4 la primer columna del archivo pdb corresponde al nombre del registro, entre estos los casos que nos interesa es cuando se encuentra la palabra *Header* lo cual nos indicaría que la información de las siguientes columnas es el nombre de la molécula. En otro caso que nos interesa es cuando el registro es *ATOM*, de esta manera nos indicaría que la información es relativa a un átomo del cuál aparecerán sus atributos en las siguientes columnas.

Se pueden observar las columnas que contienen los atributos de cáda átomo leído en cada renglón. De la columna seis a la columna diez nos indica el índice del átomo respecto a su aparición en el pdb. De la columna once a la columna 16 nos indica el nombre del elemento del átomo. De la columna 20 a la 21 nos indica la cadena a la que pertece el átomo. De la columna 22 a la 25 nos indica el Aminoácido al que pertenece el átomo. De la columna 30 a la 52 contine las posiciones espaciales x, y y z en números flotantes del átomo.

2.2.4. Dinámica molecular

La Dinámica Molecular (DM) es una técnica de simulación numérica por computadora en la que se describe la interacción de átomos y moléculas a lo largo del tiempo, generando una trayectoria del movimiento de las partículas. En la DM se resuelven las ecuaciones de movimiento de la física clásica planteadas por Newton de forma interactiva dando pasos en el tiempo. Estos cálculos son computacionalmente costosos, por lo que generalmente se realizan en equipos de alto rendimiento. Originalmente fue concebida dentro de la física teórica, aunque hoy en día se utiliza sobre todo en biofísica y ciencia de materiales. Su campo de aplicación va desde superficies catalíticas hasta sistemas biológicos como las proteínas. Si bien los experimentos de cristalografía de rayos X permiten tomar fotografías estáticas y la técnica de resonancia magnética nuclear (NMR, por su siglas en inglés) nos da indicios del movimiento molecular, ninguna técnica experimental actual es capaz de acceder a todas las escalas de tiempo y espacio involucradas. Hay quienes describen a la DM como un *microscopio virtual*

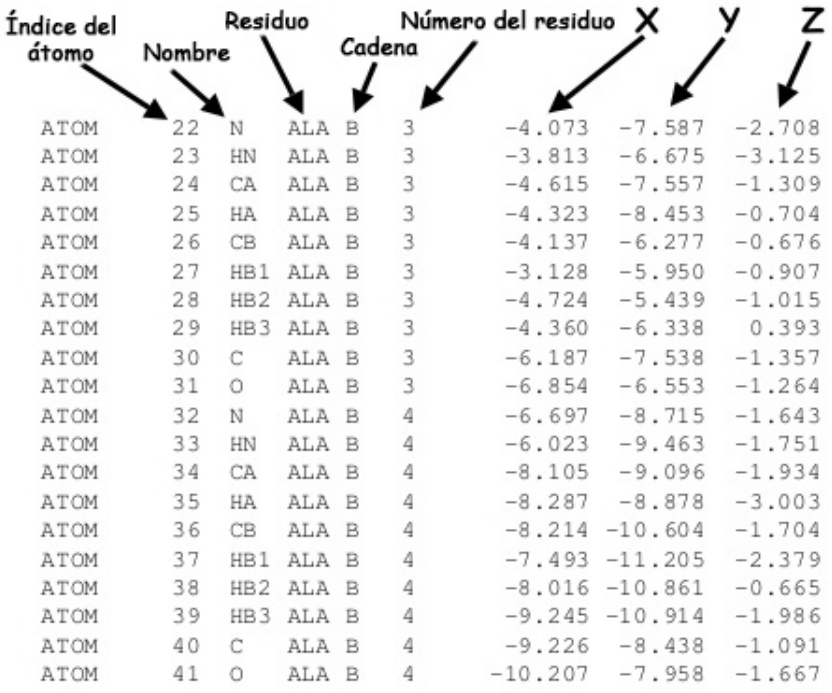

Figura 2.4: Pedazo de un archivo con formato pdb para una estructura molecular donde se muestran algunos registros de átomos

con alta resolución espacial y temporal [39].

2.2.5. Lectura del archivo de trayectoria de DM con formato XTC

Los archivos xtc, generados por el paquete de simulación numérica *Gromacs* estan en un formato portable para trayectorias de Dinámica Molecular. Estos utilizan rutinas *xdr* para leer y escribir datos que fueron creados por el sistema NFS Unix.

Las trayectorias son escritas utilizando un algoritmo de reducción de precisión que funciona de la siguiente manera: las coordenadas (en nm) se multiplican por un factor de escala, por lo general 1000, así entonces tenemos las coordenadas en pm. Estos se redondean a valores enteros. Después se realizan otros trucos, por ejemplo, se hace uso del hecho de que los átomos cercanos en secuencias son usualmente cercanos en el espacio, por ejemplo una molécula de agua. Para este fin la biblioteca xdr se extiende con una rutina especial para escribir coordenadas 3D de tipo flotante [40].

```
HDR     NSET    ISTRT   NSAVC   5-ZEROS NATOM-NFREAT    DELTA   9-ZEROS
`CORD'  #files  step 1  step    zeroes  (zero)          timestep  (zeroes)
                        interval
C*4     INT     INT     INT     5INT    INT             DOUBLE  9INT
================================================================================
NTITLE          TITLE
INT (=2)        C*MAXTITL
                (=32)
================================================================================
NATOM
#atoms
INT
================================================================================
X(I), I=1,NATOM         (DOUBLE)
Y(I), I=1,NATOM
Z(I), I=1,NATOM
================================================================================
```

Figura 2.5: Archivo para dinámica molecular con formato DCD Xplor

2.2.6. Lectura del archivo de trayectoria de DM con formato DCD

El formato de un archivo DCD puede estar dato en dos distintos formatos; El formato Xplor, el cuál usa datos de tipo double (8 Bytes/64 bits), y el formato CHARM, el cuál usa datos de tipo Flotante (4 Bytes/32 bits) y representa al formato estándar, debido a que es más eficiente utilizar datos en 32 bits que en 64. La Figura 2.5 nos muestra el formato Xplor. En este se tienen varias secciones dividas por bloques, cada uno conteniendo información diferente de la trayectoria y el tipo de dato en el que está almacenada dicha información. Entre los datos más significativos de este formato están la palabra *CORD*. Todos los archivos deben tener esta palabra al inicio del primer bloque, si no la tienen entonces no es un archivo con formato DCD. La palabra *NSET*, la cual indica la cantidad de cuadros que hay en la trayectoria, en base a este se establecen el número de coordenadas que se leerán en el archivo. La palabra *NATOM*, la cual indica la cantidad de átomos que hay en la trayectoria. Con este dato se verifica que el números de átomos de la trayectoria coincidan con el de la molécula caargada. Y las coordenadas X, Y y Z, ya que en base a ellas se genera la animación (http://www.ks.uiuc.edu/Research/vmd/plugins/molfile/dcdplugin.html).

2.3. Navegadores web

Un navegador Web es un programa que permite visualizar páginas web en la red además de acceder a otros recursos, documentos almacenados y guardar información. Este dispone del software necesario para interpretar el contenido de una página, que será normalmente escrita en lenguaje HTML. El presente proyecto se basa en la librería ofrecida para la implementación de gráficos llamada WebGL. A continuación se describen algunas particularidades de estos navegadores Web.

2.3.1. Google Chrome

Google Chrome es un navegador web desarrollado por Google y compilado con base en varios componentes e infraestructuras de desarrollo de aplicaciones (frameworks) de código abierto, como el motor de renderizado Blink (bifurcación o fork de WebKit). Está disponible gratuitamente bajo condiciones específicas del software privativo o cerrado. El nombre del navegador deriva del término en inglés usado para el marco de la interfaz gráfica de usuario («chrome»).

2.3.2. Mozilla Firefox

Mozilla Firefox (llamado simplemente Firefox) es un navegador web libre y de código abierto desarrollado para Microsoft Windows, Android, OS X y GNU/Linux coordinado por la Corporación Mozilla y la Fundación Mozilla. Usa el motor Gecko para renderizar páginas web, el cual implementa actuales y futuros estándares web.

2.3.3. Internet Explorer

Internet Explorer (usualmente abreviado a IE), es un navegador web desarrollado por Microsoft para el sistema operativo Microsoft Windows desde 1995. En el año 2015 se anunció que a partir de Windows 10 se sustituye por Microsoft Edge.

2.3.4. Safari

Safari es un navegador web de código cerrado desarrollado por Apple Inc. Está disponible para OS X, iOS (el sistema usado por el iPhone, el iPod touch y iPad) y Windows (sin soporte desde el 2012).

2.4. Conceptos de Graficación Web

2.4.1. Modelo de Objetos del Documento

El Modelo de Objetos del Documento (DOM por sus siglas en inglés) es una interfaz de plataforma que proporciona un conjunto estándar de objetos para representar documento HTML, XHTML y XML. A través del DOM, los programas pueden acceder y modificar el contenido, estructura y estilo de los documentos HTML y XML [14].

2.4.2. Renderizado

Renderizar es un término usado para referirse al proceso de generar una imagen desde un modelo o escenario en 3D. Este modelo se somete a diversos procesos, que con el uso de técnicas de texturizado de materiales e iluminación crean una serie de efectos ópticos que se asemejan a una situación específica en el mundo real.

2.4.3. Canvas

Canvas es un elemento HTML que puede ser usado para dibujar gráficos usando secuencias de comandos (normalmente JavaScript). Esto puede, por ejemplo, ser utilizado para dibujar gráficos, hacer la composición de foto o animaciones [12].

2.4.4. Frame o cuadro

Se denomina frame en inglés, a un fotograma o cuadro, una imagen particular dentro de una sucesión de imágenes que componen una animación. En el contexto de la DM, un frame representa un instante en el tiempo capturado dentro de la trayectoria.

2.4.5. Three.js

Three.js es una biblioteca de código abierto que permite crear y producir escenas 3D directamente en el navegador Web. Three.js proporciona una amplia API para su gran conjunto de funciones Three.js es probablemente la biblioteca disponible más popular para la generación de gráficos en 3D usando WebGL. Esta biblioteca trabaja sobre WebGL, simplificando la mayoría de las tareas proporcionando todas las herramientas que necesitamos para controlar las cámaras, luces, objetos, texturas y más. La bilioteca incluye varios constructores para generar los objetos básicos requeridos para trabajar con gráficos 3D. Estos objetos incluyen los métodos y propiedades necesarias para crear la escena, la cámara y mallas que representan los objetos físicos en la pantalla, y también todas la herramientas para trabajar con estos elementos y construir y animar el mundo tridimensional en la Web [13]. A pesar de su flexibilidad, esta biblioteca no está optimizada del todo, y su eficiencia es baja cuando se aplica a problemas complejos, como la visualización molecular.

2.4.6. GPU

La unidad de procesamiento gráfico (GPU por sus siglas en inglés), es un procesador dedicado al procesamiento de gráficos provenientes de la unidad central de procesamiento (CPU por sus siglas en inglés) y transformarlos en infromación comprensible y representable en el dispositivo de salida, por ejemplo un monitor. La GPU está optimizada para el cálculo de punto flotante, el cuál es muy predominante en las funciones 3D.

2.5. Tecnologías utilizadas en HTMoLv3

2.5.1. HTML5

HTML (Hypertext Markup Language) se desarrolló hace más de 20 años. En 1997 apareció en el mercado HTML4 y XHTML re-surgió, pero hubo muy pocos avances en todo este tiempo a la estructura general de HTML. HTML5 empezó a desarrollarse

en el 2007, con el objetivo de desarrollar un estándar de HTML capaz de ejecutar aplicaciones completas en un navegador Web. Las nuevas tecnologías incluidas en HTML5 incluyen características como: geolocalización, soporte a audio y video, canvas gráficos de SVG, CSS3, animaciones en dos y tres dimensiones y JavaScript 2.0. Gracias a estas características, actualmente los desarrolladores Web pueden construir sitios aun mejores que aplicaciones de escritorio [11].

2.5.2. AJAX

AJAX, acrónimo de Asynchronous JavaScript And XML (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas. Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas por completo, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.

2.5.3. jQuery

jQuery es un framework JavaScript libre y Open Source, del lado cliente, que se centra en la interacción entre el Modelo de Objetos del Documento (DOM por sus siglas en inglés), JavaScript, AJAX y HTML. El objetivo de esta librería JavaScript es simplificar los comandos comunes de JavaScript. De hecho, el lema de jQuery es "Escribir menos para hacer más" ("Write less, do more") [59].

Las especificaciones de jQuery son numerosas, pero la principal es asegurar la flexibilidad que aporta para acceder a todos los elementos del documento HTML a través de la multitud de selectores que existen.

2.6. Estado del arte

Actualmente es posible acceder a un gran número de programas de visualización molecular, algunos de ellos comerciales, otros distribuidos bajo licencias de software abierto o con licencias para su uso gratuito con fines académicos. De manera general, todos ellos llevan a cabo el mismo propósito: la visualización de estructuras moleculares y la representación de diversas propiedades asociadas a estas. Por este motivo, dada la amplia oferta de software disponible, la elección del software más adecuado dependerá de la experiencia previa del usuario con este tipo de programas y de las necesidades de análisis y de visualizacación que se tengan [21].

Existen varias herramientas de visualización molecular que pueden interpretar múltiples formatos de archivos y generar múltiples representaciones, a continuación se describen las más relevantes.

2.6.1. Antecedentes

HTMoL

Aplicación web ligera de nueva generación para visualización molecular, desarrollada en el Laboratorio de la Diversidad Molecular de la Unidad de Genómica Avanzada, Cinvestav Sede Irapuato Guanajuato. Permite al navegador web desplegar la estructura tridimensional de moléculas en una manera dinámica e interactiva. La aplicación es fácil de instalar y configurar. Puesto que se ha desarrollado utilizando tecnologías web de última generación (HTML5 + WebGL) HTMoL no requiere ningún otro plugin para ser utilizado en el lado del cliente. Funciona en todos los navegadores web principales: Chrome, Mozilla Firefox, Internet Explorer, Opera y Safari (http://htmol.tripplab.com).

A continuación se mencionan las particularidades de HTMoLv1 y HTMoLv2:

- HTMoLv1 se desarrolló en la tesis de licenciatura de Álvarez Rivera Leonardo. Desarrollo de un sistema de visualización molecular utilizando tecnología web/móvil llamado HTMoL, el cual muestra información estructural de biomoléculas de una forma visual a los usuarios, permitiendo contar con un sistema que permite estudiar información detallada y extensa de forma visual. Haciendo uso de la especificación estándar llamada WebGL, la cual permite mostrar gráficos 3D acelerados por hardware en páginas web, sin la necesidad de plug-ins, que en conjunto con el motor 3D three.js puede ser utilizado en los principales navegadores miembros del WebGL Working Group.
  Entre sus desventajas está que no posee una interacción de lenguaje de comandos con la molécula, sino que sólo presenta algunas opciones muy generales en un menú gráfico, lo que nos deja muy limitados en el análisis de la estructura molecular. Así mismo como ya se mencionó anteriormente, utiliza el framework three.js, lo que provoca que no se pueda optimizar el renderizado mediante algunas técnicas de WebGL como el Pipeline.

- HTMoLv2 se desarrolló en la tesis de maestría de Javier García Vieyra. Desarrollo de una plataforma Web para la visualización remota de dinámica molecular a través de la implementación de estrategias de streaming apoyado en la tecnológia de javascript Node.js.
  Sus limitantes es que al igual que HTMoLv1 depende de three.js, por lo que el renderizado es muy deficiente, además de que Node.js tiene que estar ejecutado en un puerto diferente al del servicio de publicación de la página que lo contenga. En las funciones de visualización de la trayectoria de dinámica molecular no posee ningúna función de regresar o pausar la visualización, sino sólo la de reproducir la trayectoria, y al momento de ejecutarla siempre se vuelve a cargar toda la información descartando toda la que ya se había descargado volviendo a reproducirse desde el inicio. El Anexo I contiene la información de cómo funciona el *Streamming* en HTMoLv2.

Su gran impacto en el campo de la Biología Estructural moderna radica en que

HTMoL posee la peculiaridad única de desplegar trayectorias derivadas de una DM de forma remota lo cual no puede ser realizado por ninguna de las demás aplicaciones de visualización molecular web hasta el momento.

2.6.2. Trabajos relacionados

En el siguiente apartado se describen los principales visualizadores moleculares existentes hasta el momento. Es importante mencionar que la peculiaridad de HTMoL es la de ser el único navegador web que permite mostrar la trayectoria de una simulación de DM sin la necesidad de tener que instalarlo como una aplicación de escritorio en la computadora.

RasMol

Programa de escritorio desarrollado por Roger Sayle, el cual permite visualizar la estructura tridimensional de una molécula. Es un programa de libre distribución y además dispone de código abierto, de modo que cualquiera que tenga los conocimientos adecuados puede introducir mejoras o adaptar el programa a su uso particular [36].

PyMol

Es un visualizador molecular de escritorio de código abierto y auspiciado por usuarios, creado por Warren Lydord Delano y comercializado por Delano Scientific LLC. Pymol proporciona la mayoría de las capacidades y el desempleño de visualizadores gráficos escritos en C y FORTRAN. La parte Py de su nombre alude al hecho de que extiende a, y es extendible mediante el lenguaje de programación Python, debido a lo cual puede ser extendido para realizar análisis complejos de estructuras moleculares utilizando bibliotecas disponibles para Python como NumPy o pylab (http://www.pymol.org/)[29].

Jmol

Jmol es un software gratuito de código abierto desarrollado inicialmente por Dan Gezelter, para visualización molecular web interactiva. Está escrito en Java, es compatible con la mayoría de los sistemas operativos y con la la mayoría de los navegadores Web modernos en forma de applet. Requiere la instalación de plug-ins para su ejecución, teniendo problemas de interoperabilidad y seguridad (http://jmol.sourceforge.net/).

JSmol

Versión de Jmol escrita en JavaScript, la cual incluye una implementación completa de las funcionalidades de Jmol y proporciona representaciones en HTML5, garantizando el acceso a dispositivos móviles modernos y ayuda a eludir las cuestiones de seguridad que han sido un problema con Java (http://sourcegorge.net/projects/jsmol/) [27]. La

desventaja de este visualizador radica en que sólo se pueden visualizar estructuras sin dinámica molecular.

GLmol

Es un visualizador molecular 3D basado en WebGL y Javascript creado por biochemfan. Con GLmol se pueden incluir modelos moleculares en páginas web sin necesidad de utilizar plugins. GLmol es de código abierto bajo la licencia dual GPL3 o licencia MIT. Acepta archivos PDB, SDF/MOL y XYZ cargándolos ya sea de manera local, o directamente desde el servidor RCSB PDB o desde el servidor NCBI PubChem. Tiene un conjunto de distintas representaciones y códigos de color para las estructuras moleculares y se ejecuta en la mayoría de los navegadores web modernos (http://webglmol.osdn.jp/index-en.html)[30]. Las principales desventajas que presenta es que tiene funcionalidades limitadas, no cuenta con una interacción por medio de lenguaje de comandos y no posee visualización de dinámicas moleculares.

Iview

Es un visualizador WebGL de fácil uso, del complejo proteína ligando desarrollado en la Universidad China de Hong Kong. Aprovecha la aceleración del hardware en lugar del software de renderización. Iview es de uso gratuito y de código abierto y puede ser integrado fácilmente en aplicaciones bioinformáticas que requieren visualización interactiva proteína ligando (http://webglmol.sourceforge.jp/indexen.hmtl)[31]. sus principales desventajas es que depende del framework three.js, teniendo un renderizado limitado complejos proteínicos con una cantidad no muy grande de átomos. Además no posee una interacción por comandos, ni es posible visualizar dinámicas moleculares.

NGL Viewer

NGL Viewer es una aplicación web para la visualización de estructuras macromoleculares. Al adoptar capacidades de los modernos navegadores web, como WebGL, para gráficos moleculares, el visualizador puede interactivamente mostrar moléculas grandes y complejas, así mismo no se ve afectado por la necesidad de instalación de plugins adicionales [35]. La desventaja de NGL Viewer radica en que se tiene que realizar un proceso de instalación complicado para poder ser usado por fuera de su servidor.

Capítulo 3

Codificación con WebGL

Como ya se mencionó anteriormente, HTMoL en sus versiones uno y dos utiliza la ayuda de *three.js* para realizar la programación con la API de WebGL. *Three.js* es un framework que tiene integrados una amplia gama de funciones que facilitan en gran manera la implementación de los gráficos con WebGL. Entre estas funciones se encuentran por ejemplo, el dibujar varios tipos de figuras, iluminar con distintos tipos de luces la escena esto con la finalidad de darle un efecto más realista a los gráficos, ingresar texto dentro de la escena, mostrar el modelado en distintas posiciones por medio de un objeto llamado cámara y además contiene los métodos necesarios para realizar selecciones de objetos dentro del escenario.

No obstante el uso de este framework deteriora en gran manera la eficiencia del renderizado cuando se están procesando miles de objetos, esto debido a que en *three.js* cada objeto dibujado es procesado individualmente por la tarjeta gráfica, generando así más veces la necesidad de sincronización del CPU con esta por cada vez que se pinta la escena, lo cual genera un gran costo con este procesamiento tomándole mayor tiempo realizar las operaciones necesarias. Esto se ve reflejado al querer interactuar con macromoleculas que contienen miles de átomos, ya que el simple hecho de mover con el mouse esta estructura produce que la imagen se mantenga pausada un lapso de tiempo en lo que se realiza el cálculo requerido, dejando así un sabor amargo en su uso. Por esta razón se decidió eliminarlo y escribir las funciones correspondientes a la parte de la graficación sobre WebGL necesarias para la implementación de HTMoL, de tal manera que se pudieran procesar varias decenas de objetos por cada vez que se pintara la escena.

3.1. Funcionalidad de WebGL

Para poder trabajar con la API de WebGL a través del navegador se utiliza un *contexto*, este cambia en los diferentes navegadores que actualmente lo soportan. La lista actual de nombres de este contexto en los navegadores que lo soportan es: *webgl*, *experimental-webgl*, *webkit-3d*, *moz-webgl*. Este *contexto* nos ayuda a acceder a todas las funciones y atributos para dibujar sobre el elemento incorporado en *HTML5*

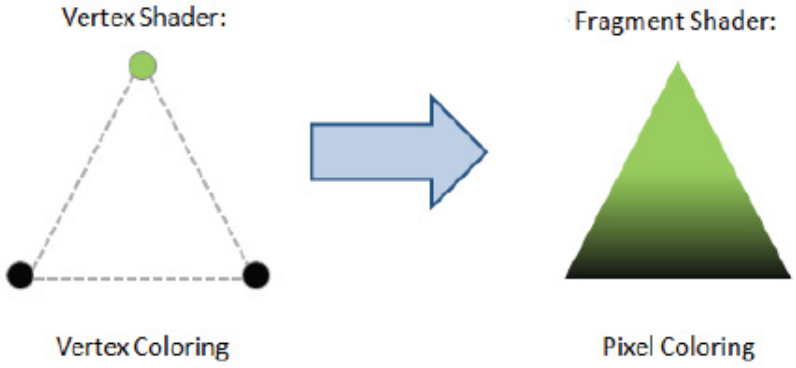

Figura 3.1: Coloreado de vértices y pixeles en los shaders

llamado *canvas*. El canvas es el lugar donde la escena será renderizada y puede ser accesada usando el *Document Object Model* (DOM) a través de JavaScript. En el presente proyecto se creó la variable llamada *gl* esto por convención de OpenGL a la que se le asignó el contenido del contexto anteriormente descrito. De esta forma se accedieron a todas las propiedades y funciones que nos brinda la API de WebGL con el formato *gl.function()*.

3.1.1. Shaders

WebGL se ejecuta en la parte de la GPU de la computadora, por este motivo se tiene que proporcionar a la GPU el código a procesar. Este código se escribe en el lenguaje conocido como GL Shader Language (GLSL) y es similar a C/C++. Es escrito en funciones pares llamadas *vertex shader* y *fragment shader*, y juntas en WebGL son conocidas como un programa. El *vertex shader* es el que se encarga de procesar las posiciones de los vértices, basado en ellas se pueden rasterizar varios tipos de *primitivas*, como son los puntos, las líneas o los triángulos. Cuando se rasterizan estas *primitivas* se llama a la función secundaria *fragment shader*. Esta función se encarga de colorear por medio de *interpolación lineal* cada pixel de la primitiva que está siendo dibujada. Cada primitiva es procesada en paralelo por la GPU a través de una serie de pasos conocidos como el *renderizado pipeline* [43]. En la Figura 3.1 Se muestra el coloreado de los vértices y pixeles en el vertex shader y fragment shader respectivamente.

Los shaders permiten tres diferentes tipos de variables, estas son los *atributos*, *varyings* y *uniformes*. Los *Atributos* son usados para ingresar por medio de los buffers los arreglos que contienen la información respecto a los vértices. Los *varyings* sirven para pasar la información del *Vertex Shader* al *Fragment Shader*. Las posiciones de los vértices obtenidas en el *Vertex Shader* son pasadas al *Fragment Shader* por medio de un *varying* llamado *gl_position*. Y las *uniformes* que son variables que mantienen la misma información para todos los elementos procesados tanto en el *Vertex Shader* como en el *Fragment Shader*.

Implementación del programa que contiene a los shaders

Para crear un programa con el cuál podamos utilizar los shaders, primeramente se utiliza la función *gl.createProgram()*, guardando la referencia a este programa en alguna variable de JavaScript. Después se tiene que hacer un enlace de cada uno de nuestros shaders en el programa que creamos, esto es con la función *gl.attachShader(nombre, shader)*, la cuál recibe como atributos el nombre de la variable donde se almacenó el programa y el nombre del shader que se va a enlazar.

Con la función *gl.linkProgram(nombre)* nuestro programa creado es enlazado a los shader. Y se finaliza con la función *gl.useProgram(nombre)* con la cuál podemos usar nuestro programa creado previamente.

3.1.2. Buffers

Un buffer es un bloque de memoria sobre el cual se puede leer, escribir y almacenar datos. Las posiciones, colores y vectores de los vértices son ingresados por medio de arreglos de JavaScript a los Buffers y estos a su vez llevan el contenido a la GPU como arreglos binarios. Para crear un buffer se utiliza la función *gl.createBuffer(tipo de buffer, nombre)*. Para decir con cuál buffer trabajaremos se utiliza la función *gl.bindBuffer(tipo de buffer, nombre)*. Y para almacenar información en el buffer con el cuál estamos trabajando se utiliza la función *gl.bufferData(tipo de buffer, arreglo de JavaScript)*. Es una buena práctica decirle a WebGL cuando ya no trabajaremos con algún buffer, por esta razón se utiliza la función *gl.bindBuffer(tipo de buffer, null)* una vez que terminamos de ingresar los datos en ese arreglo.

Estos buffers son procesados en los shaders como atributos de cada primitiva, los vértices son tomados en paquetes de tres, esto es para la posición x, y y z de un punto, y los colores en paquetes de cuatro, obteniendo así un conjunto de cuatro elementos en un formato conocido como *RGBA* el cual expresa el color en la configuración *rojo*, *verde*, *azul*, mas un elemento llamado *alpha*, el cual corresponde a la transparencia del color. Por esta razón por cada paquete de tres vértices, serán procesados paquetes de cuatro elementos que contienen la representación de *RGBA* [43].

3.2. Dibujar una esfera en WebGL

WebGL no tiene ningún método para dibujar una esfera; Como ya se mencionó anteriormente, cualquier figura por más compleja que sea es dibujada por medio de triángulos, representados en paquetes de tres vértices. La esfera es necesaria ya que es la figura base que utilizan los visualizadores moleculares para representar a los átomos.

3.2.1. Cálculo de los vértices de la esfera

Una esfera es dibuja por medio de triángulos dividiendola en latitudes y longitudes. Cada par de latitudes y longitudes forman un cuadrado, y si dividimos estos cuadrados

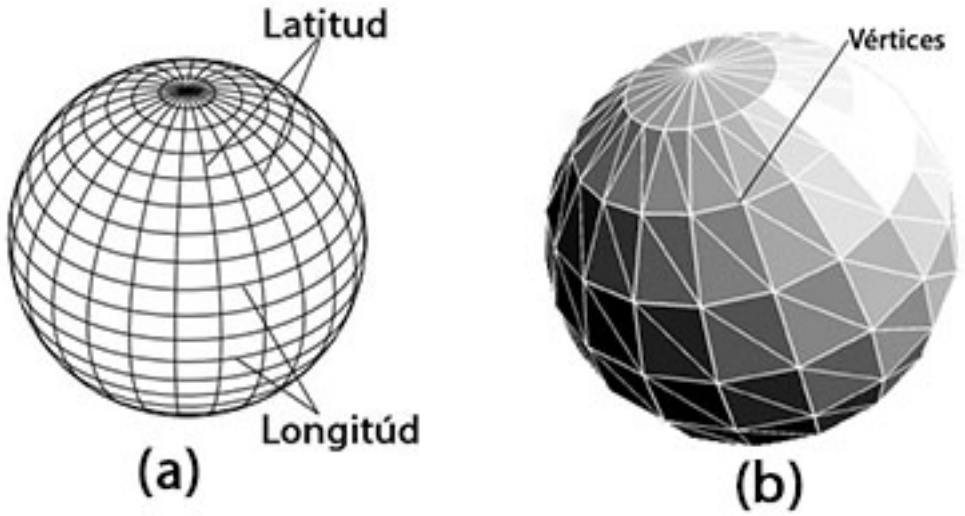

Figura 3.2: a) Esfera dividida por latitudes y longitudes, b) Cuadrados formados por intersecciones de latitudes y longitudes

por una diagonal obtendremos dos triángulos. A su vez cada triángulo contendrá tres vértices dibujados por las intersecciones de su par de latitudes y longitudes, los cuales serán las *primitivas* que usaremos. Mientras más latitudes y longitudes tengamos, mejor será la definición del contorno de la esfera.

En la figura 3.2 en el inciso *a* se muestra la esfera dividida por latitudes y longitudes, en el inciso *b* se muestran los cuadrados formados por las intersecciones de las latitudes y las longitudes.

En este proyecto se dibujó cada esfera con un total de 16 latitudes y 16 longitudes, debido a que con este número se obtiene un contorno visual realista del átomo. En general, una esfera con radio r con latitudes *lat* y con longitudes *lon* puede generar las posiciones x, y y z tomando un rango de valores para θ dividiendo π en *lat* partes, y teniendo un rango de valores para ϕ dividiendo 2π en *lon* partes. Finalmente, calculamos:

$x = r * \text{sen}(\theta) * \cos(\phi)$
$y = r * \cos(\theta)$
$z = r * \text{sen}(\theta) * \text{sen}(\phi)$

Con esto obtenemos las posiciones espaciales de los vértices para la plantilla de la esfera que se utilizará. Esta esfera está ubicada en el origen de nuestro sistema coordenado, esto es la posición *0,0,0* corresponde al centro de la esfera. Para evitar realizar miles de multiplicaciones, a cada átomo se le realizó un *offset* sumando a cada posición x, y y z contenida en la plantilla el valor x, y y z del átomo respectivamente. Es importante mencionar que se tiene que ingresar un buffer que contenga un arreglo con todos los índices para decirle a WebGL cómo se formarán los triángulos con los vértices proporcionados. En la figura 3.3 Se muestra el manejo de los índices para representar cada triángulo a dibujar. Dentro de esta imágen se puede observar una

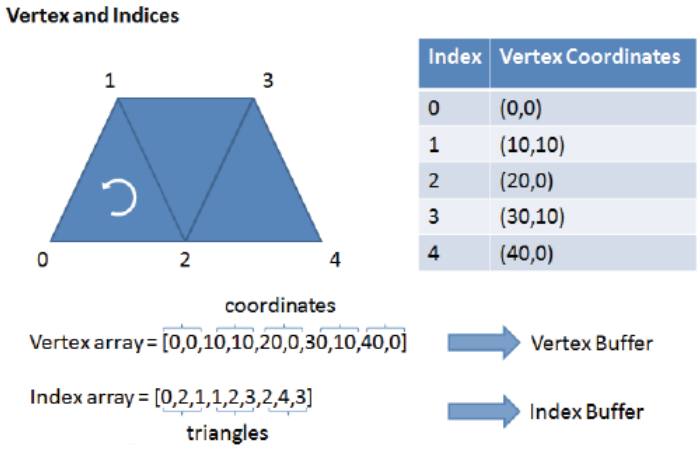

Figura 3.3: Manejo de índices para formar triángulos

tabla la cual contiene los índices utilizados y las posiciones x y y de dicho índice. Es común ingresar los vértices en sentido de las manecillas del reloj, pero también se pueden ingresar en sentido contrario sin afectar la visualización de los gráficos.

3.2.2. Cálculo de las normales de la esfera

Una *normal* es un vector con una magnitud de valor igual a uno, la cual indica la dirección a la que apunta una línea perpendicular saliendo de una cara. En este caso para una esfera con radio de una unidad, es un vector que va desde su centro hasta la superficie para cada cara, es decir, los senos y cosenos del ángulo que queremos hallar para un punto en cuestión.

En este proyecto las *normales* se calcularon dentro de la plantilla de la esfera, y después simplemente se le sumó el valor x, y y z de la posición del átomo a los elementos ingresados en el arreglo correspondientes a estos valores.

3.3. Proceso del renderizado

Los elementos que se utilizan para la implementación de una escena en WebGL son:

- Arreglos de JavaScript conteniendo la información relevante a los vértices de los objetos (color, posición, normales, etc.).
- Los buffers que son los que ingresan los datos de los arreglos de JavaScript a los atributos declarados en el *Vertex Shader*.
- Los shaders formando un pequeño programa que contiene al *Fragment Shader* y al *Vertex Shader* el cuál se ejecuta en la parte de la GPU.
- Atributos. Por medio de estos se puede ingresar toda la información contenida en los buffers al *Vertex Shader*.

- La matriz de tranformación la cual mantiene las transformaciones que sufren los vértices al ser procesados por alguna rotación o traslación.
- *Frame Buffer* Es el buffer que contiene la imágen final que va a ser mostrada en la pantalla.

La Figura 3.4 nos muestra el proceso del pipeline del renderizado en WebGL. Este se inicializa cuando se llama a la función *drawElements* o *drawArrays*. Los arreglos de JavaScript que contienen toda la información de los vértices son ingresados al *Vertex Shader* por medio de los atributos. Estos tienen que ser habilitados en JavaScript por medio de la función *gl.enableVertexAttribArray(positionAttibute)* la cuál toma como parámetro la ubicación del atributo en la tarjeta gráfica. El *Vertex Shader* es ejecutado por cada vértice ingresado. Se calcula la posición de cada uno de los vértices que forman a la primitiva. En esta etapa también se calculan otros atributos del vértice como su color. Después se procede al proceso del ensamblaje de la primitiva. Aquí la primitiva es ensamblada y pasada a la fase de rasterización.

En la fase de rasterización se calculan los pixeles que formarán la imagen de la primitiva en dos etapas:

- *Culling* En esta etapa se determina la orientación del polígono. En el caso de las esferas se determina la orientación de cada uno de los triángulos, aquellos que tengan una orientación en la cuál no sean visibles se descartan.
- *Clipping* En esta etapa se determinan los triángulos que están fuera del área de visibilidad. Al igual que el paso anterior, estos triángulos son descartados.

Finalmente el *Fragment Shader* obtiene los datos del *Vertex Shader* en variables de tipo *varyings* y las primitivas del proceso de rasterización para calcular el valor de cada uno de los pixeles que se encuentran dentro de cada una de estas primitivas.

3.4. Implementación de las representaciones

En el presente proyecto se puede visualizar una molécula en representación *Spheres Bonds*, *Bonds*, *CPK*, *Skeleton* además de las representaciones *Trace*, *Spline* y *Ribbon* agregadas con la colaboración de otro tesista. La configuración por defecto de HTMoLv3.0 es mostrar la molécula en representación *Spheres Bonds*. No obstante, esta representación puede ser modificada por el usuario cambiando la variable que la contiene en la configuración inicial. Esta se enecuentra en el archivo con el nombre *config.js* ubicado en la carpeta *js*. Además en todo momento se tienen todas las representaciones inicializadas en la escena, pero las representaciones que no se requieren no son visualizadas. Esto es gracias a que se ingresa el valor 0 al elemento alpha del color de cada vértice al inicializar las representaciones que no van a ser visibles.

A cada vértice que es procesado en el *Vertex Shader* se le pregunta el valor de su elemento *alpha*, si este tiene un valor igual a 1 se calcula su primitiva y pasa a la parte del *Fragment Shader* donde es calculado el color de cada uno de sus pixeles. Si

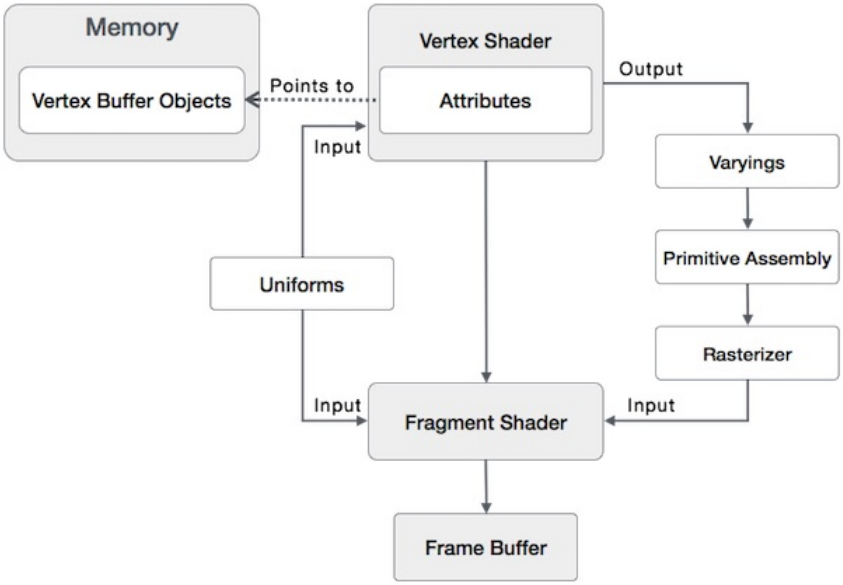

Figura 3.4: Pipeline del renderizado de WebGL

el valor del *alpha* es igual a 0, entonces es descartado del *Vertex Shader*, ya no pasa al *Fragment Shader* y no es visible en la escena.

Los atributos que se ingresan por cada vértice en el *Vertex Shader* en el presente proyecto son:

- Los atributos de posiciones. Este atributo es declarado como un vector de tres componentes de tipo flotante. Para las componentes x, y y z.

- Atributo de color. Este atributo es declarado como un vector de cuatro componentes de tipo flotante. En este se ingresa el color en formato RGBA con cada componente con valor entre 0 y 1.

- Atributo de las normales. Este atributo es declarado como un vector de tres componentes de tipo flotante y es usado para el procesamiento de la luz.

- Atributo de los colores difusos. Al igual que el color es declarado como un vector de cuatro componentes de tipo flotante. En este atributo se ingresa el color único para indentificar a cada átomo.

- Atributo llamado *chain* el cuál nos indica a qué cadena pertenece cada vértice. Este atributo es ingresado en un vector de dos componentes de tipo flotante. Esto debido a que los atributos sólo reciben números de tipo flotante, y el vector mínimo aceptado es de dos componentes. Por lo que este atributo provoca un desperdicio de un número flotante por cada vértice en memoria. Pero tiene la ventaja de que el proceso de prender y apagar una cadena es muy rápido.

3.4.1. Representación CPK

La representación CPK muestra a los átomos dibujados por esferas con radios de Vander Walls, esto es, cada átomo tiene un radio definido dependiendo del elemento al que representa. Debido a esto se definieron dentro de la plantilla de esferas los arreglos que contienen los vértices que forman a cada esfera con los diferentes radios posicionada en el origen. Para esto se ingresaron los vértices para el átomo del Hidrógeno, Carbono, Nitrógeno, Sulfuro, Fósforo, Oxígeno y para los demás elementos se utilizó un radio general. De esta forma una vez que se utiliza representar una átomo en CPK simplemente se modifica el arreglo de JavaScript que tiene contenidos estos vértices, colocando los del arreglo de la plantilla de esfera en CPK del átomo correspondiente agregándole la posición x, y y z del átomo leído del pdb.

3.4.2. Representación Spheres Bonds

Esta representación consiste en mostrar todos los átomos conectados por sus enlaces covalentes por medio de una esfera representando al átomo y una línea representando al enlace. Esta esfera tiene un radio de 0.2, pero este dato puede ser configurado por el usuario en la configuración inicial. Si actualmente tengo un átomo en representación CPK y quiero cambiarlo a Spheres Bonds, simplemente se modifica el arreglo de JavaScript que tiene contenidos estos vértices, colocando los del arreglo de la plantilla de la esfera y sumándole la posición x, y y z del átomo leído del pdb.

3.4.3. Representación Bonds

Esta representación consiste en mostrar todos los enlaces covalentes existentes en la molécula. Cada enlace es representado por una línea ingresando las posiciones de los vértices de los átomos que se encuentran en cada extremo del enlace. Los arreglos de esta representación son los mismos usados para la representación de Spheres Bonds.

3.4.4. Representación Skeleton

Como ya se mencionó en el capítulo dos, cada aminoácido está formado por un sólo carbono alfa (CA). Esta representación consiste en conectar todos los CA que contienen los aminoácidos de cada cadena. De esta forma se inicializó un arreglo conteniendo las líneas con las posiciones x, y y z de los vértices de los CA que se encuentran en los extremos.

3.5. Selección de un objeto

Una aplicación de gráficos 3D necesita proveer de algún mecanismo para la interacción entre el usuario y la escena mostrada en pantalla. Una de las interacciones más común es la de selección de algún elemento mostrado en la escena por medio del mouse.

En WebGL los objetos pueden ser seleccionados mediante dos principales técnicas, una es la de *Ray Casting* y otra es mediante *Colores difusos únicos*. En HTMoLv3 implementamos la técnica de *Ray Casting* debido a que utiliza un proceso óptimo y fácil de implementar. De esta forma, ahora el usuario puede seleccionar un átomo representado por una esfera con el mouse o varios al mantener presionada la tecla *shift*. A continuación se describen las técnicas del *Ray Casting* y *Colores difusos únicos* debido a que en este trabajo probamos ambas para determinar cuál de ellas es más eficiente.

3.5.1. Técnica del Ray Casting

En la técnica del *Ray Casting* se crea un rayo con origen en el espectador (osea desde el punto de localización de la cámara), y con dirección hacia el punto de selección proyectado en 3D, con una magnitud suficientemente capáz de intersectar el objeto que se encuentre en la posición z más retirada. Después se calculan los objetos intersectados por el rayo, esto se puede realizar por medio de ecuaciones matemáticas complejas [42]. Estas ecuaciones matemáticas pueden disminuir la rapidéz de aplicaciones que contienen miles de objetos, esto debido a que por cada objeto se tiene que aplicar la multiplicación e inversión de matrices. Esto es, por ejemplo en el caso de los átomos representados por eferas, se tiene que procesar cada uno de los centros de estas aplicándole la multiplicación de las matrices que usamos en el vertex shader, debido a que no se sabe el valor actual de nuestro centro de la esfera el cual pudo haber cambiado con la interacción del usuario, ya que esto se procesa en la parte de la GPU y no podemos obtener valor alguno en estos procesos. Una vez obtenida la posición actual se procede a calcular la distancia más corta formada por la recta que sale del origen de la cámara al punto de selección del mouse. Si esta distancia es menor al radio de la esfera, etonces idicaría que el objeto será agregado a una lista de selección. Una vez realizado este proceso con todos los elementos se optiene el objeto seleccionado correspondiente al que tenga la menor distancia a la cámara en la posición z. Descartamos esta técnica debido a que este procesamiento disminuyó la velocidad al procesar miles de objetos. En la figura 3.5 se muestra la representación de el rayo que sale del origen de la cámara comúnmente llamado *Eye* hacia la posición del mouse en el canvas para optener el objeto seleccionado.

3.5.2. Técnica de Colores difusos únicos

Cada átomo tiene un color correspondiente al elemento que pertenece y además un único color difuso, estos dos colores son asignados conforme se van leyendo del archivo PDB. Esto es cada que se lee un átomo el asignador de colores únicos va aumentando en uno el valor de la columna que le corresponda de las tres que contienen al color rojo, verde y azul. Dicho valor se encuentra entre un intervalo de 0 a 255 siendo así que si alguna columna llega a su valor máximo esto es 255, vuelve a iniciar de cero incrementando en uno la columna siguiente ubicada a la derecha y así sucesivamente.

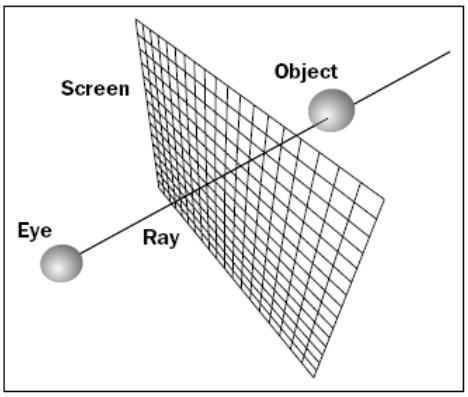

Figura 3.5: Rayo para realizar un casting en la escena

Por lo cuál se podrían asignar un máximo de una combinación de $255 * 255 * 255 - 1$ colores sin tomar en cuenta el color correspondiente al elemento canvas.

En esta técnica se utilizan dos escenas, la escena On-Screen y la Off-Screen. Cada vez que se realiza un click en el canvas se manda una variable uniforme al vertex y fragment shader para decirle que a cada vértice se le coloree por su color difuso sin tomar en cuenta las luces en la interpolación lineal en la parte del *Fragment Shader*, de esta manera se obtiene el valor del pixel leído por la selección en la pantalla el cuál es procesado y convertido en el átomo que le corresponde a ese pixel. Una vez que se obtiene el átomo leído se le manda la variable al vertex y fragment shader para decirle que ahora coloree todos los vértices por medio de los colores reales. Todo esto ocurre en el mismo método, por lo que el usuario no verá la representación en colores difusos [30]. En la Figura 3.6 se muestra el proceso de cambio entre una escena con colores difusos sin luces a una con colores de cada elemento.

Una vez que se conoce qué átomo fué seleccionado es coloreado de color verde para diferenciarlo de los demás. Cabe mencionar que inicialmente se tenían arreglos de JavaScript conteniendo 100 átomos cada uno y cuando se seleccionaba un átomo este pasaba a formar parte de otro arreglo para mostrarlo en visualización *wireframe*, por lo tanto se tenían que generar otras llamadas a la tarjeta gráfica, esto debido a que para mostrar un objeto en forma sólida se utiliza la función *drawElements*, y para mostrarlo en forma *wireframe* la función *drawArrays*, por lo que se generaban más llamadas a la tarjeta gráfica y se retardaba el procesamiento. Además se escribió un algoritmo para generar el menor número de arreglos ya fuese de los que contenían a los objetos sólidos o a los objetos en representación *wireframe*, lo cual incrementó aún más el tiempo de procesamiento. Debido a esto se optó por simplemente cambiar el color del átomo seleccionado, cambiando los valores en el arreglo correspondiente al átomo sin tener que integrarse a otro arreglo. Esto optimizó en gran manera el procesamiento de los datos y el algoritmo de selección pasó de ser de más de 600 líneas de código a 200 líneas. Esta reducción de un factor de tres disminuyó considerablemente el tiempo de descarga de HTMoL. En la Figura 3.7 se muestra en el inciso *a* la representación sólida de un átomo de carbono enlazado a un átomo de nitrógeno. En el inciso *b* se

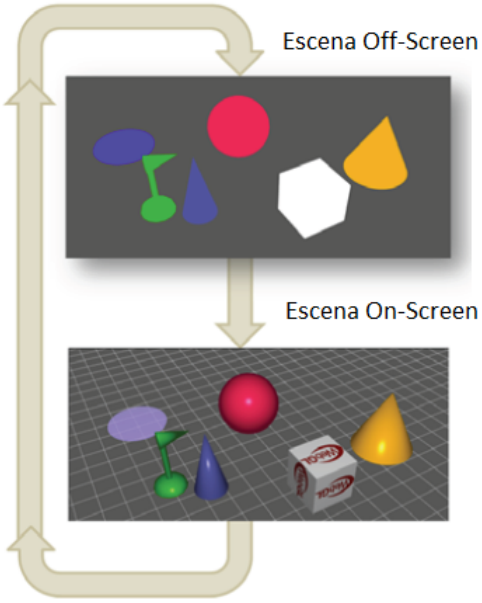

Figura 3.6: Proceso Off-Screen y On-Screen

muestran estos mismos átomos en representación *wireframe*.

3.6. Luces

En WebGL se hace uso de los vertex y fragment shaders para crear un modelo matemático con la ayuda de las normales para la iluminación de la escena. La luz es escencial ya que como en el mundo real en WebGL un objeto es visible cuando este refleja cierta porción de luz que le llega, y es entonces cuando se pueden distinguir los colores del objeto.

3.7. Cámara

WebGL no tiene un objeto que realize las funciones de la cámara, pero se puede asumir que lo que se renderiza en el cavans es lo que la cámara captura. Para esto se utilizan matrices de 4x4, que contienen las componentes x, y y z que forman a nuestro espacio euclidaneo, mas un cuarto componente llamado la *coordenada homogénea*. Este componente nos deja en un nuevo espacio llamado el espacio proyectivo y nos dá la posibilidad de realizar transformaciones de rotación, traslación y proyección representadas en una matriz de 4x4. Con la ayuda de operaciones de matrices se puede obtener el efecto de mover la escena con el mouse. Para esto se utiliza una matriz llamada *mvMatrix* la cual guarda la información del cambio de posiciones y rotaciones generado por el mouse. Esta matriz se multiplica inicialmente por otra que contiene la perspectiva deseada de la escena y después por cada vértice que está siendo procesado

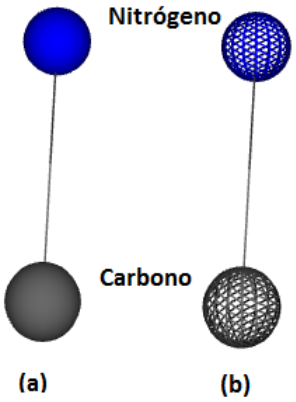

Figura 3.7: a) Representación sólida del átomo de carbono y nitrógeno. b) Representación wireframe del átomo de carbono y nitrógeno

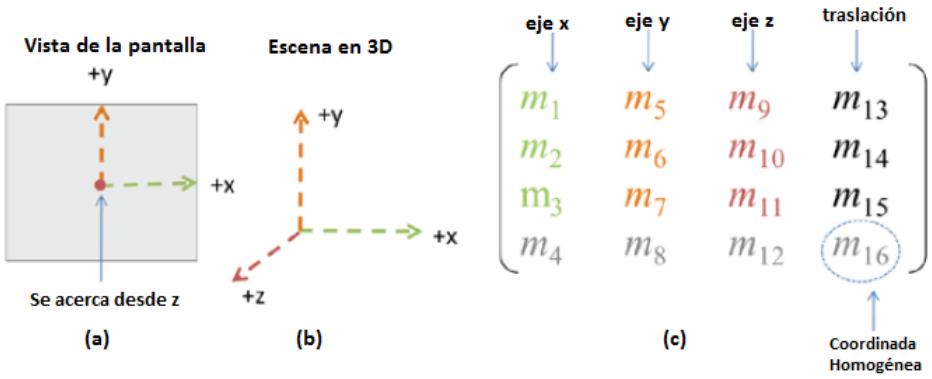

Figura 3.8: Matriz modelo vista

dentro del vertex shader obteniendo así la posición final después del movimiento de la escena generado por el mouse [43]. La figura 3.8 en el inciso *a* nos muestra la visualización de los ejes desde la pantalla. Nos hacercamos a esta desde el eje positivo z, a la derecha está ubicado nuestro eje x positivo, y la parte vertical corresponde al eje y. El inciso *b* nos muestra la distribución de los ejes dentro de nuestra escena en WebGL. El inciso *c* nos muestra cómo está integrada la matriz modelo vista. La primer columna corresponde al eje x, la segunda al eje y y la tercera al eje z. La última columna corresponde a la traslación que efectúa nuestro sistema. Cabe mencionar que la última fila de nuestras columnas corresponde a la *coordenada homogénea*.

WebGL no contiene una biblioteca para ejecutar operaciones de matrices y vectores, no obstante existen varias desarrolladas por terceros que nos brindan todo el soporte que se requiere. En este proyecto se utiliza la biblioteca *glMatrix*, creada por *Brandon Jones* especialmente para ejecutar operaciones de matrices y vectores (http://glmatrix.net/).

3.8. Cambiar los vértices contenidos en un buffer

Para poder cambiar cierta selección de una representación a otra, es necesario el cambio de los vértices correspondientes a esa selección por la nueva representación. Esto se puede hacer de dos distintas formas, una es usando el método *bufferSubData*, el cual utiliza como parámetro un offset para indicar la parte del buffer que se cambiará. Esta opción presentó problemas al no cambiar los elementos restantes del arreglo. La segunda forma es cambiando el último parámetro de la llamada a los buffers, esto es de Static_draw a Dinamic_draw, seguido de cambiar todo el arreglo de JavaScript, y volver a llamar el buffer a la tarjeta gráfica con el método *bufferData*.

3.9. Manejo del streamming en WebGL

Las trayectorias de la dinámica molecular que son obtenidas por medio del *streamming* son almacenadas en seis arreglos en JavaScript. Los tres primeros contienen las posiciones x, y y z nombrados como: *coordsX*, *coordsY* y *coordsZ* repectivamente. Cada uno de estos arreglos contiene en bloques cada uno de los frames con las posiciones de la componente ya sea x, y o z de todos los átomos. Si este arreglo llega a un tope, entonces se comienzan a llenar los arreglos nombrados: *coordsX1*, *coordsY1* y *coordsZ1*.

La forma en que se implementó esta funcionalidad, fué con la ayuda de dos variables dentro de nuestro método *drawScene* mencionado anteriormente, las cuales nos indican cuando se encuentran listos nuestros arreglos de JavaScript para comenzar a mostrar los frames que llevan ingresados, y otra para indicar que el botón *play* ha sido ejecutado. De esta forma si estas dos variables son verdaderas, entonces con la ayuda de la función de JavaScript *RequestAnimFrame* se procesarán todas las posiciones de los átomos en el frame que se encuentra actualmente, dando así la sensación del movimiento de la molécula.

3.10. Optimización del renderizado

Como ya se mencionó anteriormente, mientras menos llamadas se realizen a la tarjeta gráfica, más optimizado estará el renderizado. Por lo cual en HTMoLv3 se dibujan los átomos en paquetes de 100 esferas en una sola llamada a la tarjeta gráfica, de esta manera son menos las veces que se requiere sincronizar la GPU con el CPU obteniendo un óptimo renderizado para macroestructuras moleculares. Este número puede ser configurado por el usuario para aumentar la cantidad de esferas a procesar en una llamada, pero cabe mencionar que por ser arreglo de javascript de tipo *float32* el que se ingresa a los buffers, este contiene un límite de aproximadamente 200 esferas, esto debido a la cantidad de números flotantes requeridos en cada arreglo para formar cada una de las esferas de 16 latitudes y 16 longitudes. Se toma como base a las esferas, ya que en cuestión de tamaño del número de vértices que se generan,

son las que tienen más peso que los demás objetos. En particular, para una esfera formada por 16 latitudes y 16 longitudes se generan 289 vértices, requiriendo tres números flotantes indicando las componentes x, y y z de cada vértice en el buffer de posiciones, generando así un total de 867 números flotantes dentro de este arreglo para una sóla esfera. Para el buffer que contiene la información del color en formato RGBA, se requirieron cuatro números flotantes para cada uno de los vértices, esto es un total de 1,156 números flotantes para cada esféra de 289 vértices. Para el buffer de índices que nos indica la forma de conectar todos los triángulos, se requirieron 1,536 números enteros por cada esféra. Es preciso mencionar que todos arreglos de javascript a excepción de los de color y de las posiciones son borrados después de ser ingresados a los buffers de la GPU. Esto por cuestión de que nuestra aplicación no mantenga en uso una gran cantidad de memoria innecesaria provocando efectos negativos en la página web que la contenga. Los buffers de color se tienen que mantener, debido a que en HTMoLv3 es posible cambiar el color de cualquier átomo. De esta forma cada que un átomo o átomos son cambiados de color, sólamente se cambian los elementos correspondiente dentro del arreglo. Y después se procede a cambiar estos valores en los buffers de la GPU. Asimismo el arreglo de posiciones es requerido en escenarios como cuando se realiza la funcionalidad de centrar por átomo, en este caso todas las posiciones de los átomos cambiarán centrando al átomo seleccionado en la coordenada *0,0,0*; O dentro de la dinámica molecular cambiando en cada frame todas las posiciones de los átomos.

En general cualquier llamada a sincronizar el CPU con la GPU ocasiona un retraso potencial por lo que se debe evitar en gran manera realizar este proceso. Con este fin desarrollamos una función la cuál se encarga de procesar todos los átomos que estén seleccionados. Esta función utiliza un algoritmo con el cuál se van agregando a un arreglo todos los bloques de átomos que son afectados y que requieren que sean actualizados los buffers correspondientes a estos bloques. De esta manera cuando termina de hacer el proceso en cada bloque, se realiza una sincronización con la tarjeta gráfica pasándole los únicos bloques que requieren actualización. Al principio cada que se modificaba un bloque de JavaScript se realizaba una sincronización con la tarjeta gráfica, pero había bloques de átomos que se modificaban varias veces, debido a que los átomos modificados estaban contenidos en el mismo bloque. Esto probocaba un retraso enorme al sincronizar más veces el CPU con la tarjeta gráfica. En el apéndice B se describen a más detalle los algoritmos utilizados para mantener un número óptimo de sincronización de la CPU con la GPU.

Capítulo 4

Implementación del lenguaje de comandos en HTMoL

Como se ha mencionado, una molécula se puede visualizar en distintas representaciones. Gracias al lenguaje de comandos que integramos en HTMoLv3.0 se pueden hacer selecciones específicas para realizar un análisis independiente de toda la estructura sobre esta selección. Por ejemplo, se pueden hacer cambios de visualización o inclusive visualizar solo la parte seleccionada y así llevar a cabo un estudio minucioso y cuantitativo sobre esta.

4.1. Funciones requeridas por HTMoL

Después de eliminar el framework de *three.js* se tuvieron que programar todas las funciones requeridas para el lenguaje de comandos sobre WebGL directamente. A continuación se describen algunas de las principales funciones necesarias para integrar el lenguaje de comandos.

4.1.1. Dibujar mediciones de ángulos y distancias

El contexto 3D de WebGL no contiene métodos para poder mostrar texto en el objeto de HTML5 llamado *canvas*. Esto se puede hacer principalmente de dos formas. La primera es ingresar el texto con la ayuda de texturas. La segunda es por medio de triángulos formando el caracter requerido.

Primeramente se representaron los números con una textura que los contenía todos, de tal manera que cada número estaba en alguna posición de la textura, pero al presentarce en la escena el número se veía deformado, con un mala presentación. Entonces se optó por representar los caractéres necesarios formados por triángulos. Para esto se obtuvo con la ayuda de Blender 2.71 los dígitos, el punto, el signo de grado para el ángulo, y la terminación *nm* para indicar nanometros en distancias. La forma de obtener los triángulos que forman estos dígitos fue dibujando cada uno en Blender con el tipo de letra por default y el tamaño reducido a la mitad. Posteriomente se

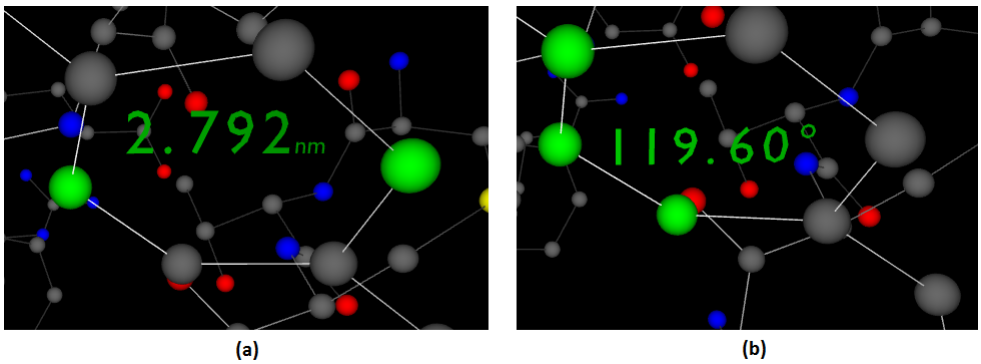

Figura 4.1: Inciso *a* medición de la distancia entre dos átomos, *b* medición del ángulo formado entre tres átomos.

exportó en formato *.x3d* seleccionando entre las propiedades de exportación la casilla *Triangulate*. Con la ayuda de esta propiedad se generan los triángulos e índices que los conectan para cada caracter.

Para calcular la distancia entre dos átomos se usó la ecuación que nos proporciona la distancia entre dos puntos y se representa de la siguiente forma:

$$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2}$$

La ecuación que se utilizó para el cálculo de los ángulos fué la siguiente:

$$angle = \arccos \frac{(x2 - x1) * (x2 - x3) + (y2 - y1) * (y2 - y3) + (z2 - z1) * (z2 - xz3)}{\sqrt{(x2 - x1)^2} + \sqrt{(y2 - y1)^2} + \sqrt{(z2 - z1)^2}}$$

En la Figura 4.1 se puede observar en el inciso *a* la medición de distancia entre dos átomos. Los dos átomos a medir están resaltados de color verde, así como la magnitud medida. En el inciso *b* se muestra la medición de un ángulo dados tres átomos, al igual que la distancia, los átomos tres átomos de esta medición están resaltados en verde al igual que la magnitud.

Estas mediciones se integraron en la visualización de la dinámica molecular que ofrece HTMoL de tal manera que por cada cambio de *frame* se calculan de nuevo todas las mediciones, esto debido a que cada átomo puede tener una posición diferente en cada *frame* de la trayectoria.

4.1.2. Vistas de la molécula

Para mostrar la molécula en cada una de las seis vistas posibles, primeramente se limpia la matriz de rotación, esta matriz es la que al multiplicarse por la matriz de proyección y por la matriz modelo vista nos dá el efecto de girar la molécula, una vez limpiada esta matriz se convierte a una matriz identidad. Después es rotada sobre el eje requerido con la ayuda de la biblioteca *glMatrix*. La aplicación de rotación con la ayuda de esta biblioteca se muestra a continuación:

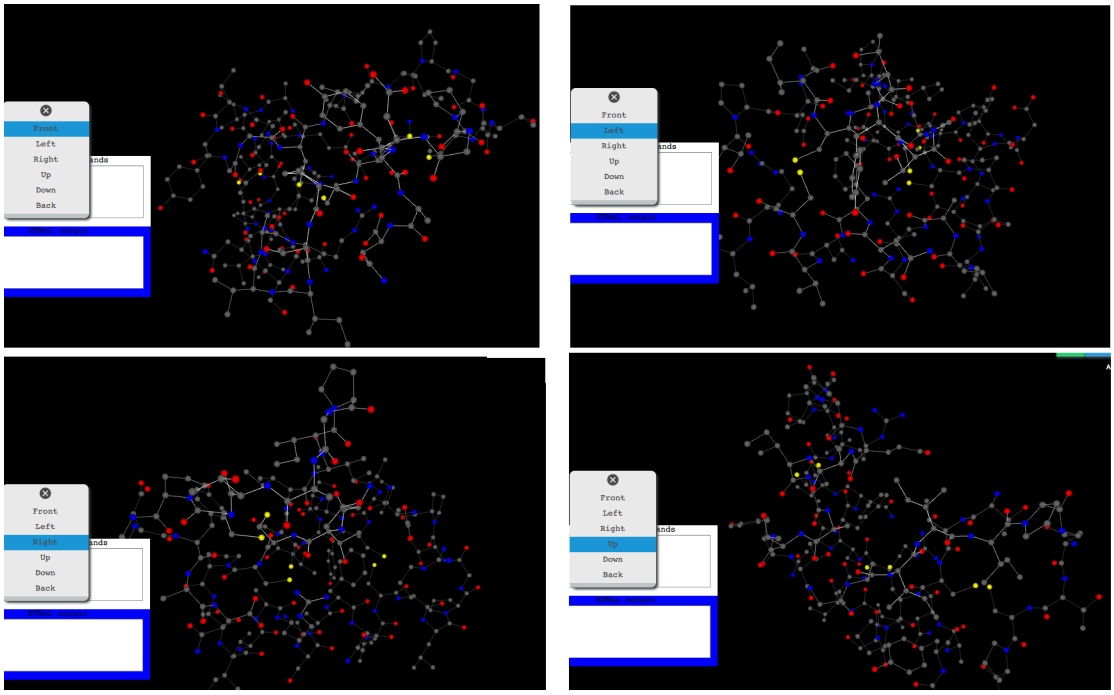

Figura 4.2: Vista frontal, izquierda, derecha y arriba de la proteína con PDBID 1CRN

- *mat4.rotate(newRotationMatrix, degToRad(0), [0, 1, 0]);* //vista frontal
- *mat4.rotate(newRotationMatrix, degToRad(90), [0, 1, 0]);* //vista izquierda
- *mat4.rotate(newRotationMatrix, degToRad(270), [0, 1, 0]);* //vista derecha
- *mat4.rotate(newRotationMatrix, degToRad(90), [1, 0, 0]);* //vista de arriba
- *mat4.rotate(newRotationMatrix, degToRad(270), [1, 0, 0]);* //vista de abajo
- *mat4.rotate(newRotationMatrix, degToRad(180), [0, 1, 0]);* //vista de atrás

Para la vista frontal se hace una multiplicación por 0 grados con un vector con valor uno en el eje y, para la vista lateral izquierda, por 90 grados con el mismo vector, para la vista de arriba se multiplica por 90 grados con un vector con un valor uno en el eje x etc. En la figura 4.2 se observa la vista frontal, izquierda, derecha y arriba de la proteína con PDBID 1CRN.

4.1.3. Centrado de la molécula en el sistema cartesiano

Para centrar la molécula primero se calcula el centro geométrico de ésta y después se traslada este al origen. Esto es, se realizó la suma de las posiciones x, y y z de todos los átomos, obteniendo un offset en los ejes x, y y z respectivamente, así mismo se dividió cada uno de estos valores entre el número de átomos de la molécula, de esta

manera a los vértices de cada una de las componentes se le restó el valor obtenido en el offset de tal manera de situar el centro de la molécula en la posición *0,0,0* de nuestros ejes.

4.1.4. Mostrar u ocultar las cadenas de un complejo proteínico

Un complejo proteínico puede estar formado por más de una cadena, las cuales a su vez contienen un grupo de aminoácidos comúnmente llamados residuos. Se implementó la funcionalidad de mostrar u ocultar estas cadenas de forma independiente cada átomo se relacionó a su cadena correspondiente por medio de un identificador que se ingresó como un atributo en los shaders; De esta manera cada que se apaga o prende una cadena, por medio de un arreglo al cual se agrega o elimina el indicador de la cadena deseada, de tal manera que en el vertex shader se checa si el vértice es de un elemento perteneciente a la cadena apagada, si es así todo vértice que muestre este indicador no será procesado, pasándose así al vértice siguiente hasta encontrar los vértices que deben de estar prendidos.

Cabe mencionar que en el lenguaje de los shaders (GLSL) no se pueden realizar loops con variables, si no que tienen que ser constantes, de esta forma para decirle al shader con cuántas cadenas se va a realizar búsqueda en el loop fué por medio de mandar la función de shaders en javascript como un string de manera que se pudiera concatenar el valor de la constante del número de las cadenas. En la Figura 4.3 se muestra la secuencia de ocultar cadenas del complejo proteínico 1al0. En la parte superior derecha se pueden observar los nombres de las cadenas que contiene dicho complejo. Estas cadenas se pueden ocultar seleccionando cada uno de estos nombres, cambiando su color de blanco a rojo para indicar que actualmente está oculta y ocultando todos los átomos correspondientes a esta cadena.

4.2. Diseño del analizador sintáctico

Un token o también llamado componente léxico es una cadena de caracteres que tiene un significado coherente en cierto lenguaje de programación. Algunos ejemplos de tokens podrían ser las palabras clave *if*, *else*, *while*, *int*, etc.; Dentro de algunos lenguajes de programación como es el caso de JavaScript y C++, identificadores, números, signos, o un operador formado de varios caracteres.

El analizador sintáctico convierte un texto dado en otras estructuras (comúnmente árboles), que son más útiles para el posterior análisis y capturan la jerarquía implícita de la entrada. Un analizador léxico crea tokens de una secuencia de caracteres de entrada y son estos tokens los que son procesados por el analizador sintáctico para construir la estructura de datos, por ejemplo un árbol de análisis o árboles de sintaxis abstracta [18].

La escritura de cada instrucción en HTMoLv3 está inicializada por un comando seguida de los atributos de ese comando a excepción del comando *Select*, el cual pro-

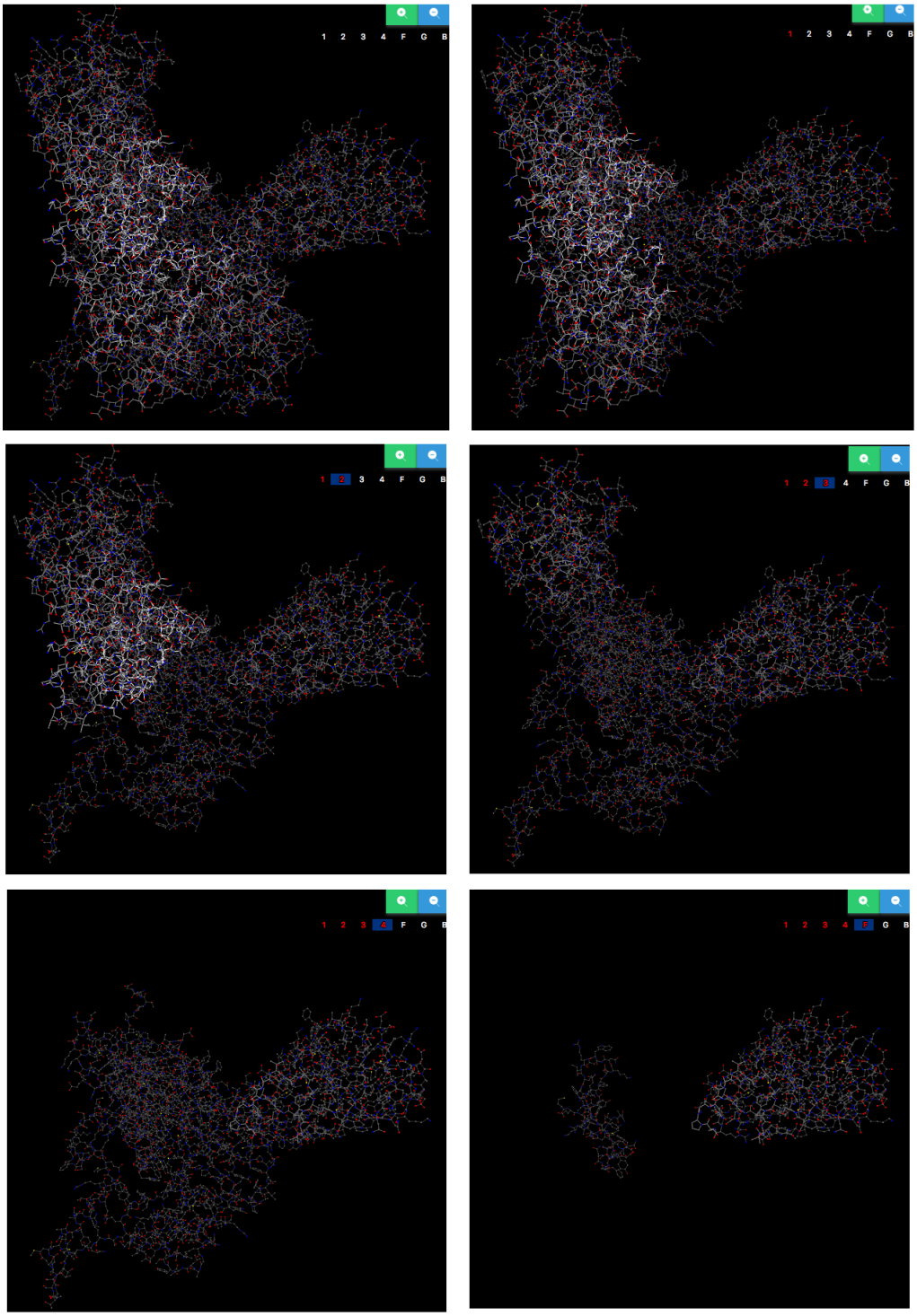

Figura 4.3: Secuencia de ocultamiento de cadenas del complejo proteínico 1al0

cesa la cadena de entrada con un analizador sintáctico para identificar el o los átomos a seleccionar. Los comandos implementados en HTMoL son algunos de los principales empleados por RasMol: *select*, *show*, *color*, *center*, *Distance*, *Angle* y *Rotate*.

El comando *show* puede contener los atributos: *sequence* para mostrar la estructura de cadenas aminoácidos contenidos en el pdb, y los atributos: *Bonds*, *SBonds*, *CPK*, *BackBone* los cuales cambian el conjunto de átomos seleccionados a la representación indicada.

En la parte de la selección se puede procesar a los átomos de dos diferentes maneras, la primera es por índice, la cual es un número que nos especifica su posición en el pdb, y la segunda es por grupo, la cuál es una combinación de símbolos que representan un conjunto de átomos. El grupo está formado por tres apartados divididos cada uno por dos puntos. El primer apartado contiene la parte correspondiente al átomo, el segundo al aminoácido y el tercero a la cadena. La parte del átomo puede contener el número 0 cuando se quiere seleccionar todos los átomos correspondientes a los siguientes apartados, o el nombre del átomo a seleccionar en conjunto con los siguientes apartados. La parte correspondiente al aminoácido puede contener el nombre del aminoácido deseado, o el número que tiene asignado dentro del pdb; si el número ingresado es un 0 quiere decir que buscará en todos los aminoácidos correspondientes a la(s) cadena(s) del siguiente apartado. La parte de la cadena puede contener un índice para indicar la posición de esta cadena en el pdb o un símbolo, en este caso se buscaría la cadena por este símbolo, si el número ingresado es un 0 al igual que en los apartados, se buscaría en todas las cadenas existentes.

La instrucción de selección se genera de diferentes formas. Puede contener un índice, un rango, un grupo o a varias conjuciones de estas separadas por una coma. Un rango se define por el símbolo -, y en sus extremos sólo puede contener índices, por lo que si se ingresa un grupo en cualquiera de sus dos extremos la instrucción no se llevaría a cabo. Las siguientes son instrucciones válidas dentro de la selección:

- *Select 0:0:0* Selecciona todos los átomos existentes
- *Select 0:0:1* Selecciona los átomos de la cadena 1
- *Select 0:0:A* Selecciona los átomos de la cadena A
- *Select 0:3:2* Selecciona los átomos del aminoácido con número 3 que se encuentren en la cadena 2
- *Select 0:THR:0* Selecciona los átomos de los aminácios de tipo THR en todas las cadenas
- *Select C:ALA:A* Selecciona todos los carbonos del aminoácido tipo ALA en la cadena A
- *Select 3-6* Selecciona los átomos del 3 al 6 que contiene el pdb

- *Select 4-23,N:THR:3* Selecciona los átomos del 4 al 23 que contiene el pdb, además selecciona los nitrógenos que contiene el aminoácido tipo THR en la cadena 3.

4.2.1. Diseño del Autómata

Los autómatas son máquinas conceptuales para el reconocimiento de patrones, una representación matemática de un sistema que recibe una cadena de símbolos pertenecientes a un alfabeto determinado y determnina si esa cadena es aceptada o no, esto es si pertenece al lenguaje que reconoce el autómata.

Existen autómatas finitos que además son un sistema determinista (AFD); Es decir, para cada estado en que se encuentre el autómata, y con cualquier símbolo del alfabeto leído, existe siempre no más de una transición posible desde ese estado y con ese símbolo. Este se define como una quíntupla de la siguiente manera:

$$A = (Q, T, \lambda, q0, F)$$

donde:

Q = Conjunto finito y no vacío de estados,

T = Alfabeto de entrada,

λ = Función de transición que toma como argumentos un estado de Q y un símbolo de T, y devuelve un subconjunto de estados de Q.

q0 = Subconjunto de Q que representa al conjunto de estados iniciales (a diferencia de los AFD, los AFND pueden admitir varios estados iniciales).

F = Subconjunto de Q que representa al conjunto de estados de aceptación (estados finales).

Los Autómatas Finitos No Deterministas abreviados comúnmente AFND, se caracterizan porque, a diferencia de los AFD, en un estado puede haber más de una transición posible para un mismo símbolo de entrada (alfabeto). Es decir/$\lambda(q, a)$/ >= 1 para algún q perteneciente a Q y para algún símbolo a perteneciente a T [18].

La Figura 4.4 nos muestra el AFND que se utilizó para integrar el lenguaje de comandos en este proyecto. Los elementos utilizados para el alfabeto son:

- *Amino.* Representa un aminoácido existente. Puede ser GLY, ALA, VAL, LEU, ILE, SER, THR, ASP, GLU, ASN, GLN, LYS, HIS, ARG, PHE, TYR, TRP, CYS, MET y PRO.

- *Element.* Representa un elemento de la tabla periódica. Puede ser H, He, Li, Be, B, C, etc.

- *N.* Representa los números naturales sin tomar en cuenta el 0. Estos serían 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, etc.

- *NameChain* Representa el nombre de una cadena como aparece en el pdb.

Estado	Entrada							
	Amino	Element	N	NameChain	0	:	,	-
q0	0	q1	q6	0	q1	0	0	0
q1	0	0	0	0	0	q2	0	0
q2	q3	0	q3	0	q3	0	0	0
q3	0	0	0	0	0	q4	0	0
q4	0	0	q5	q5	q5	0	0	0
q5	0	0	0	0	0	0	q0	0
q6	0	0	0	0	0	0	0	q7
q7	0	0	q6	0	0	0	0	0

Tabla 4.1: Tabla de transiciones del AFND empleado en HTMoLv3.0

Dejando así al alfabeto como se muestra a continuación:

T: {`Amino`, `Element`, `N`, `NameChain`, `0`, `:`, `,`, `-`}

Este AFND posee los estados: q0, q1, q2, q3, q4, q5, q6, q7. Siendo el estado inicial: q0. Y los estados de aceptación: q5 y q6.

La tabla 4.1 nos muestra las transiciones posibles entre los estados dadas diferentes entradas. Como ya se mencionó anteriormente en el diseño del analizador sintáctico, este autómata puede admitir a un átomo denotado por un índice N, este corresponde al orden en el que aparece en el pdb. Un rango denotado por N1-N2, en donde N1 y N2 se encuentran dentro del rango de índices del pdb. O por una instrucción conformada por tres apartados, definido inicialmente por el nombre del átomo, después correspondiendo a los aminoácidos y por útimo a las cadenas.

4.3. Cambio entre representaciones

Para realizar el cambio entre representaciones se diseñaron varios algoritmos, dependiendo de la situación en la que se encuentren los átomos seleccionados se procederá a las acciones requeridas para realizar dicho cambio de representación. Para esto se creó la función llamada *Cambiar representación*. Esta función recibe como entrada el nombre de la representación a la que se va a cambiar. Su forma de trabajar se describe a continuación. Primeramente checa si todos los átomos de la molécula están seleccionados, si es así, se limpian todas las representaciones actuales en la escena, además de todas las variables y arreglos de JavaScript utilizados. Después continúa con inicializar todas las representaciones mostrando sólo la requerida y ocultando las que no se visualizarán agregando 0 el valor alpha de todos los vértices de las representaciones que no se desean visualizar.

Si el caso es que no todos los átomos de la molécula están seleccionados, entonces se utiliza un algoritmo que va a procesar a cada uno de los átomos dependiendo de la representación actual en la que se encuentra y a la representación a la que se va a cambiar.

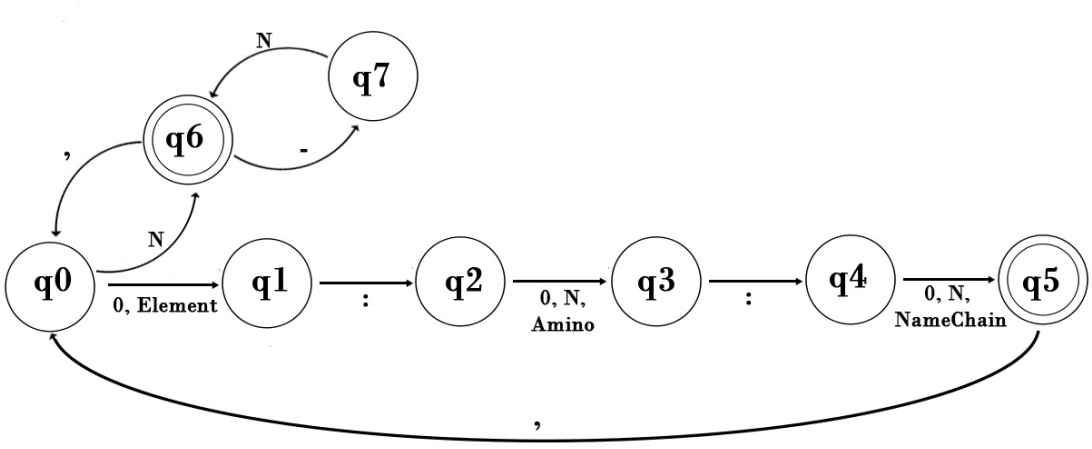

Figura 4.4: Autómata finito no determinista usado en HTMoLv3 para integrar el lenguaje de comandos

En la Figura 4.5 se puede observar la molécula con PDBID 1CRN en representación Spheres Bonds y los aminoácidos de tipo PRO mostrados en representación CPK.

4.4. Integración de la consola de comandos

Para poder utilizar el lenguaje de comandos en HTMoLv3 diseñamos una consola con la ayuda del elemento *textarea* de HTML; Dentro de este se ingresan instrucciones que son ejecutadas al teclear *enter* con la aplicación activa. Estas instrucciones son procesadas en JavaScript para realizar las funciónes requeridas.

Con la ayuda de la biblioteca de JavaScript *jQuery* se implementó la función de *dragg* de la consola, esto para poder moverla a cualquier parte de la aplicación y que no nos dificulte la visibilidad de la estructura molecular.

Esta consola está integrada por dos apartados, uno para la captura del lenguaje de comandos, y otro para mostrarnos las acciones realizadas, por ejemplo si aplicamos la instrucción *select* con los átomos que queremos seleccionar, en la parte de salida nos dirá la cantidad de átomos seleccionados, o si es el caso de alguna instrucción que esté mal escrita, nos indicará que no conoce el comando. La Figura 4.5 nos muestra la consola con la instrucción *select* arrojándonos el número de átomos seleccionados en la parte de la salida. También se puede observar la selección de estos átomos con color verde en la molécula.

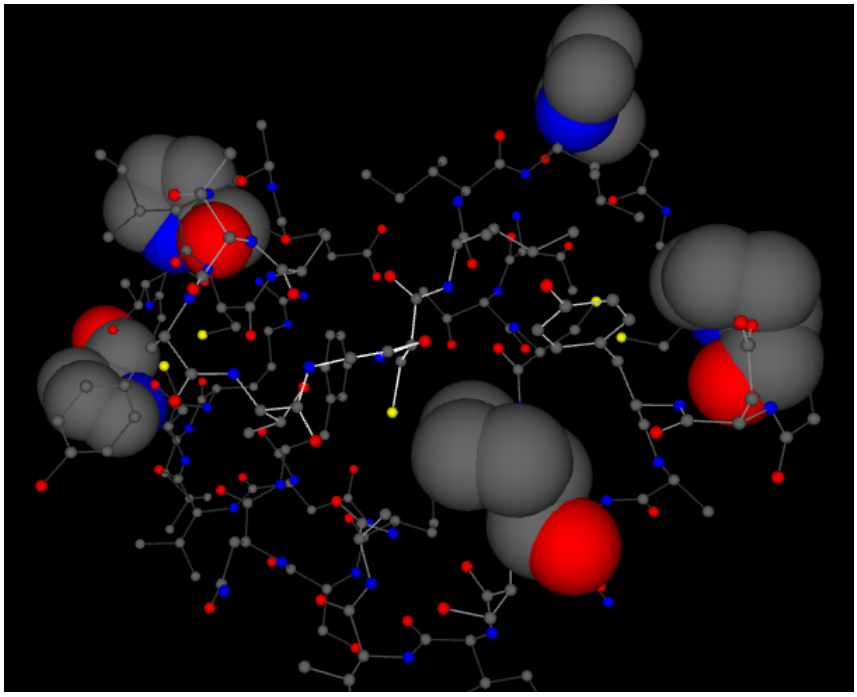

Figura 4.5: Visualización en Spheres Bonds junto con CPK de la molécula con PDBID 1CRN

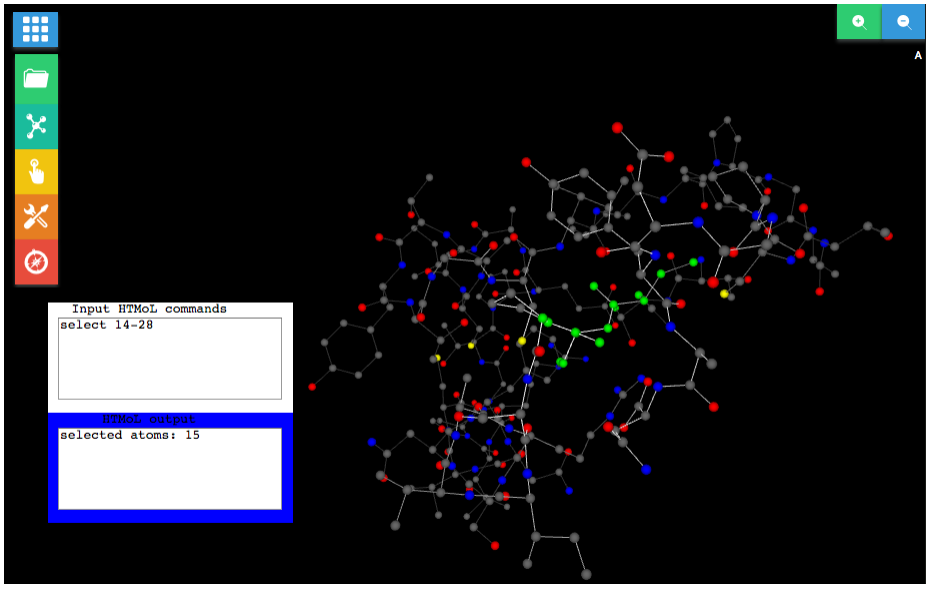

Figura 4.6: Consola de comandos en HTMoLv3

4.5. Diseño de la API

La Interfaz de Programación de Aplicaciones (API por sus siglas en inglés) es el conjunto de subrutinas, funciones y procedimientos que nos permite usar ciertas funciones ya programadas sin la necesidad de entender la complejidad de su diseño. Con esta se pueden realizar programas muy sofisticados relativamente con muy poco trabajo. Esto es en parte debido a que una API permite el acceso a grandes funciones y bibliotecas que nos ofrecen ciertos servicios desde los procesos y representa un método para conseguir abstracción en la programación entre los niveles o capas inferiores y los superiores del software.

Uno de los principales propósitos de una API consiste en proporcionar un conjunto de funciones de uso general. De esta forma, los programadores se benefician de las ventajas del API haciendo uso de su funcionalidad, evitándose el trabajo de programar todo desde el principio. Una API en sí misma es abstracta y el software que proporciona generalmente es llamado la implementación de esa API [52].

En nuestro caso la interacción con la molécula puede ser de tres maneras distintas. 1) Por medio de la Interfáz Gráfica de Usuario (GUI), la cuál contiene funciones particulares, como cargar una molécula, cambiar de representación, cambiar de vista, entre otras. 2) Por medio de la consola de comandos, en donde el usuario puede ingresar los distintos comandos como seleccionar ciertos átomos, cambiar de representación, etc. Y 3) por medio de un Script inicial, en este se pueden ejecutar varias sentencias cargando la molécula de una manera predeterminada.

La presente API se diseñó para poder hacer de HTMoLv3 una herramienta fácil de usar y poderlo incrustar en cualquier página web de una manera eficiente, pudiendo generar una interfáz gráfica personalizada sin perder el control de sus funciones. Los comandos generados por la API se hicieron de tal forma que el usuario pueda acceder a todas las capacidades principales de HTMoLv3 sin la necesidad de saber cómo está estructurada la codificación. El acceso a la API es por medio del objeto *htmol*. Este objeto se define en la variable llamada *htmol* la cuál es inicializada después de que se carga toda la molécula. El objeto *htmol* posee los métodos principales definidos a continuación:

- *htmol.chain(index, state)* Sirve para prender o apagar una cadena. Recibe dos atributos, el primero es el índice de la cadena a procesar y el segundo es el estado, este puede ser *on* para prenderla u *off* para apagarla

- *htmol.spin(axis, grados)* Esta instrucción sirve para mantener una rotación a la molécula sobre un eje. El primer parámetro es uno de los dos ejes x o y. El segundo parámetro nos indíca cuantos grados va a rotar, esto se realizará por frame, dándonos una velocidad controlable.

- *htmol.axis(state)* Esta función sirve para prender cuando se ingresa el estado *on* o apagar con el estado *off* los ejes de la molécula.

- *htmol.show(representation)* Con esta instrucción podemos cambiar la representación de la molécula, ya sea a Spheres bonds, Bonds, CPK, Back Bone o Spline.
- *htmol.console(state)* Esta instrucción muestra u oculta la consola, recibiendo el estado de *on* para mostrarla u *off* para ocultarla.
- *htmol.reset()* Vuelve la molécula a su estado inicial, siendo este el estado cuando se carga la molécula en la escena.
- *htmol.color()* Cambia de color toda la molécula, pudiendo ser uno de los predeterminados (white, red, blue, yellow) o *byChain* para mostrar el color por cadena o *bySecond* para mostrar el color por estructuras secundarias.
- *htmol.select(rango o selección)* Esta instrucción selecciona uno o varios átomos mediante una instrucción estructurada como se indica en el capítulo cuatro.

Las funciones requeridas para esta API pueden ser ingresadas en el elemento HTMoL.html después de que es declarado el objeto *htmol*.

Capítulo 5

Resultados

En el presente capítulo se describen los resultados obtenidos al realizar una comparación entre HTMoLv3 y HTMoLv2 en los navegadores Firefox, Google Chrome y Safari en diferentes entornos. Todas las siguientes pruebas se realizaron con la representación de la molécula en Spheres Bonds, esto debido a que es la representación que contiene el mayor número de elementos dentro de la escena. Para el valor obtenido de Frames por segundo (FPS) en cada navegador se tomaron diez muestras con las cuales se calculó el promedio y desviación estándar para dar un valor más preciso en cada cuantificación comparativa. A continuación se muestran las tres moléculas utilizadas para estas pruebas. En la Figura 7.1 se muestra la proteína con PDBID 1CRN. La cual contiene un total de 327 átomos. En la Figura 7.2 podemos observar el complejo proteínico 1AL0 contiendo 9,527 átomos. En la Figura 7.3 se muestra una membrana conteniendo un total de 22,335 átomos.

5.1. Pruebas realizadas en el entorno de Ubuntu

Las siguientes pruebas se realizaron en los navegadores Firefox y Google Chrome en una Computadora Dell Core i5 con 8 Gigabytes en ram. Las siguientes tablas muestran los resultados obtenidos en el sistema operativo ubuntu 14.04.

En la tabla 6.1 se muestran los resultados obtenidos de la lectura de tres moléculas con diferente número de átomos. En esta se muestra el tiempo que tomó visualizarlas así como el número de frames que se alcanzan a leer por segundo en la versión dos y tres de HTMoL en el navegador Firefox.

La tabla 6.2 nos muestra las mediciones anteriormente descritas pero ahora en el navegador Google Chrome en su versión 54.

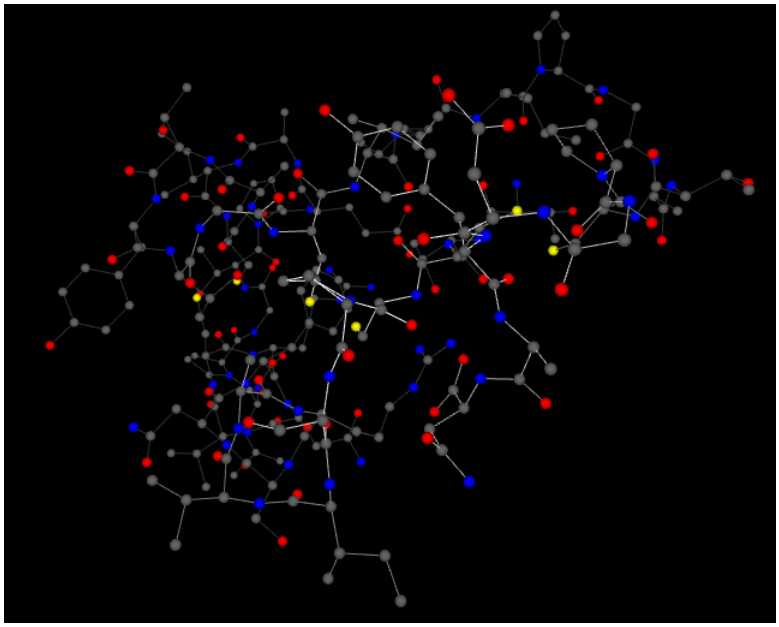

Figura 5.1: Proteína con PDBID 1CRN para pruebas conteniendo 327 átomos

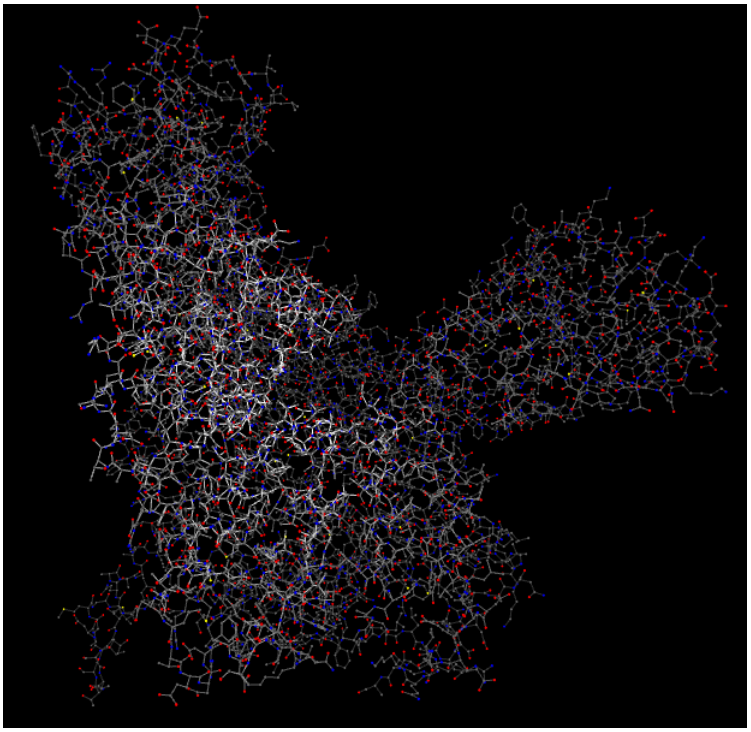

Figura 5.2: Complejo proteínico para pruebas 1AL0 con 9,527 átomos

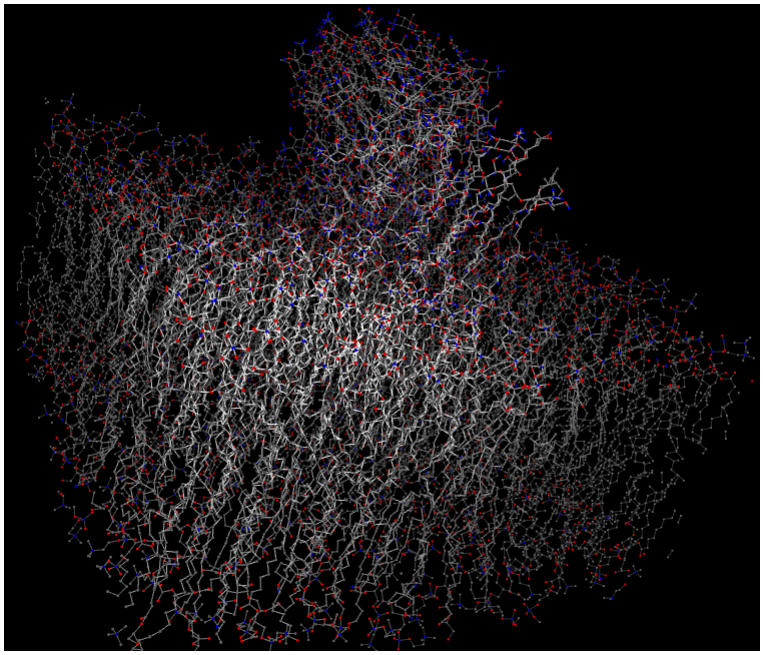

Figura 5.3: Membrana para pruebas con 22,335 átomos

Pruebas realizadas en el navegador Firefox v41.0.2				
Molécula	**Núm. de Átomos**	**HTMoL versión**	**Tiempo de carga**	**Lectura de frames por segundo**
1crn	327	versión 2	139.57 ms	12.411 ± 3.05 fps
		versión 3	86.57 ms	34.798 ± 9.68 fps
1al0	9,527	versión 2	1,549.04 ms	0.981 ± 0.18 fps
		versión 3	633.74 ms	10.544 ± 1.34 fps
Membrana	22,335	versión 2	5,622.51 ms	0.453 ± 0.05 fps
		versión 3	2,962 ms	5.196 ± 0.46 fps

Tabla 5.1: Comparación de desempeño entre HTMoLv2 y HTMoLv3 en el navegador de Firefox en el entorno de Ubuntu

Pruebas realizadas en el navegador Google Chrome v54.0.2840.71				
Molécula	**Núm. de Átomos**	**HTMoL versión**	**Tiempo de carga**	**Lectura de frames por segundo**
1crn	327	versión 2	105 ms	37.114 ± 6.54 fps
		versión 3	68.2 ms	59.164 ± 2.23 fps
1al0	9,527	versión 2	1,680 ms	1.219 ± 0.572 fps
		versión 3	387 ms	14.62 ± 1 fps
Membrana	22,335	versión 2	3,650 ms	0.578 ± 0.351 fps
		versión 3	1,250 ms	6.151 ± 1.84 fps

Tabla 5.2: Comparación de desempeño entre HTMoLv2 y HTMoLv3 en el navegador de Google Chrome en el entorno de Ubuntu

Pruebas realizadas en el navegador Safari 9.1.2				
Molécula	**Núm. de Átomos**	**HTMoL versión**	**Tiempo de carga**	**Lectura de frames por segundo**
1crn	327	versión 2	32.674 ms	50.837 ± 9.73 fps
		versión 3	40.675 ms	59.454 ± 1.94 fps
1al0	9,527	versión 2	458.594 ms	2.037 ± 0.37 fps
		versión 3	156.165 ms	20.275 ± 7.39 fps
Membrana	22,335	versión 2	1,068.17 ms	0.895 ± 0.35 fps
		versión 3	609.801 ms	8.267 ± 7.6 fps

Tabla 5.3: Comparación de desempeño entre HTMoLv2 y HTMoLv3 en el navegador de Safari en el entorno de Mac

5.2. Pruebas realizadas en el entorno de Mac OS El Capitan

Las siguientes pruebas se realizaron en los navegadores Firefox, Google Chrome y Safari en una Computadora Macbook pro core i5 con 4 Gigabytes en ram. Las siguientes tablas muestran los resultados obtenidos en el sistema operativo El Capitan.

En la tabla 6.3 se muestran los resultados obtenidos de la lectura de tres moléculas con diferente número de átomos. En esta se muestra el tiempo que tomó visualizarlas así como el número de frames que se alcanzan a leer por segundo en la versión dos y tres de HTMoL en el navegador de Safari.

La tabla 6.4 nos mestra los resultados obtenidos de la lectura de tres moléculas con diferente número de átomos. En esta se muestra el tiempo que tomó visualizarlas así como el número de frames que se alcanzan.

La tabla 6.5 se observan los resultados obtenidos de la lectura de tres moléculas con diferente número de átomos. Se muestra el tiempo de carga y el número de frames que se leen por segundo.

Pruebas realizadas en el navegador Firefox v49.02				
Molécula	**Núm. de Átomos**	**HTMoL versión**	**Tiempo de carga**	**Lectura de frames por segundo**
1crn	327	versión 2	84.19 ms	35.536 ± 17.45 fps
		versión 3	49.24 ms	52.234 ± 7.76 fps
1al0	9,527	versión 2	898.4 ms	2.394 ± 0.2 fps
		versión 3	285.15 ms	28.218 ± 2.94 pfs
Membrana	22,335	versión 2	2,453.75 ms	1.075 ± 0.04 fps
		versión 3	1,742.48 ms	14.451 ± 0.67 fps

Tabla 5.4: Comparación de desempeño entre HTMoLv2 y HTMoLv3 en el navegador de Firefox en el entorno de Mac

Pruebas realizadas en el navegador Google Chrome v54.0.2840.98				
Molécula	**Núm. de Átomos**	**HTMoL versión**	**Tiempo de carga**	**Lectura de frames por segundo**
1crn	327	versión 2	72.5 ms	52.701 ± 13.11 fps
		versión 3	49.7 ms	59.31 ± 1.61 fps
1al0	9,527	versión 2	909 ms	2.852 ± 1.09 fps
		versión 3	196 ms	31.344 ± 1.34 pfs
Membrana	22,335	versión 2	1,630 ms	1.221 ± 0.51 fps
		versión 3	599 ms	13.97 ± 0.8 fps

Tabla 5.5: Comparación de desempeño entre HTMoLv2 y HTMoLv3 en el navegador de Chrome en el entorno de Mac

Pruebas realizadas en el navegador Firefox v49.0.2				
Molécula	**Núm. de Átomos**	**HTMoL versión**	**Tiempo de carga**	**Lectura de frames por segundo**
1crn	327	versión 2	281.33 ms	4.509 ± 3 fps
		versión 3	106.01 ms	38.06 ± 13.61 fps
1al0	9,527	versión 2	2,583.5 ms	0.481 ± 0.223 fps
		versión 3	665.75 ms	9.797 ± 2.71 fps
Membrana	22,335	versión 2	6,193.27 ms	0.264 ± 0.07 fps
		versión 3	3,955.87 ms	3.965 ± 1.31 fps

Tabla 5.6: Comparación de desempeño entre HTMoLv2 y HTMoLv3 en el navegador de Firefox en el entorno de Windows 7

Pruebas realizadas en el navegador Google Chromev 54.0.2840.99 m				
Molécula	**Núm. de Átomos**	**HTMoL versión**	**Tiempo de carga**	**Lectura de frames por segundo**
1crn	327	versión 2	289.94 ms	16.024 ± 4.28 fps
		versión 3	76.3 ms	58.348 ± 2.84 fps
1al0	9,527	versión 2	2,004 ms	1.505 ± 0.105 fps
		versión 3	321 ms	4.517 ± 2.49 fps
Membrana	22,335	versión 2	6,759 ms	0.661 ± 0.1339 fps
		versión 3	1,270 ms	2.469 ± 2.47 fps

Tabla 5.7: Comparación de desempeño entre HTMoLv2 y HTMoLv3 en el navegador de Chrome en el entorno de Windows 7

5.3. Pruebas realizadas en el entorno de Windows 7

Por último se muestran las pruebas que se desarrollaron en el entorno de Windows. La tabla 6.6 nos muestra las mediciones anteriormente descritas en el navegador Firefox en su versión 54.

La tabla 6.7 nos muestra las mediciones anteriormente descritas en el navegador Google Chrome en su versión 54.

La versión 3.0 de HTMoL puede ser distribuida y accesible desde su portal de internet http://htmol.tripplab.com, conteniendo una gran variedad de ejemplos y un demo mostrando la gran peculiaridad que hace de HTMoLv3 una herramienta clave en la Biología Molecular, esto es, de visualizar la dinámica molecular.

Capítulo 6

Discusión y conclusiones

En este capítulo se discuten los resultados y se presentan las conclusiones que se tuvieron en la elaboración del presente proyecto de tesis.

Uno de los grandes aportes que se obtuvieron fué quitar la dependencia del framework three.js. Esto permite ahora interactuar directamente con WebGL, con la posibilidad de modificar cualquier funcionalidad de una forma eficiente, además de poder integrar alguna futura técnica que se desarrolle en WebGL que pueda optimizar los tiempos de renderizado de nuestra aplicación. Así mismo, podemos cambiar cualquier peculiaridad directamente con los shaders de la tarjeta gráfica, esto debido a que ya no hay ningún intermediario entre la API de WebGL con nuestra aplicación.

La GPU está diseñada especialmente para realizar operaciones de punto flotantes. Debido a que la mayoría de operaciones requeridas para realizar gráficos utilizan operaciones con números de punto flotante, por ejemplo el posicionamiento de los atributos de los vértices, se puede obtener una gran ventaja en velocidades de procesamiento con esta arquitectura, dejando al CPU para otras funciones más indicadas para él. Así mismo, mientras menos veces se tenga que sincronizar la CPU con la GPU durante la ejecuión de la aplicación, más optimizado estará el renderizado de nuestra escena.

Por el lado de la programación con JavaScript, para la integración de los arreglos que contienen la información en punto flotante, inicialmente se usaba la funcionalidad llamada *splice*, esta puede incrustar elementos en un arreglo, pero se tenía un retraso muy grande debido a que el algoritmo que usa es muy costoso en tiempo de ejecución. Al detectar este comportamiento se remplazó por la funcionalidad de igualar la posición del elemento a cambiar en el arreglo por el nuevo elemento a ingresar. Lo anterior logró optimizar en una manera significativa los procesos deseados. También se observó un retraso significativo al concatenar una cantidad grande de pequeños arreglos con el *concat*, por lo que se cambió completamente por un loop for para ingresar todos los elementos por medio de un push. Esto tuvo un impacto grande al generar los bloques de átomos. Al iniciar este trabajo no se contaba con un documento que explicara la integración que se había realizado por parte del streamming, además de que se tuvieron que resolver los problemas pendientes con esta versión, así mismo de que la codificación no presentaba coherencia ni notas que especificaran las funciones

principales, lo que se vió reflejado en un retraso significativo. Por tal motivo, toda la codificación realizada se estuvo realizando de una forma coherente y con notas para futuras modificaciones que puedan ayudar a los interesados. Este proyecto puede ser usado como base para futuros desarrollos que requieran usar la tecnología WebGL, ya que se expresean las experiencias y algunos consejos al usarla para tener un mejor rendimiento en nuestra aplicación.

Al observar detalladamente los datos que se obtuvieron en las diferentes pruebas de velocidad y carga, se puede concluir que los navegadores más recomendables para correr la aplicación HTMoLv3 es primeramente Google Chrome en un entorno de Mac OS, seguido por el entorno de Linux en Ubuntu. El segundo navegador más recomendable es Firefox en un entorno de Linux o Mac; Se observaron reindimientos muy bajos en un entorno de Windows, esto puede ser debido a que el sistema operativo no no utiliza de una forma optimizada la tarjeta gráfica.

Durante la elaboración de este proyecto se trabajó a la par con tres tesistas de licenciatura del Instituto Tecnológico Superior de Irapuato (ITESI). Una de ellas se enfocó en implementar la lectura de archivos DCD para la dinámica molecular; La segunda fué la integración de una interfaz gráfica eficiente, amigable e intuitiva al usuario. La tercera se encargó de implementar más representaciones moleculares, en particular, Trace, Ribbon y Spline.

La forma en que se trabajó fué teniendo reuniones de una o dos veces por semana con todos los integrantes del proyecto. Las primeras reuniones sirvieron para proporcionarle a los tres tesistas un conocimiento de la forma en que HTMoL en sus versiones uno y dos fueron diseñados. Así como la importancia y el impacto del trabajo a realizar. Poco a poco se les porpocionaron los fundamentos y conceptos necesarios en JavaScript y Biología para comenzar la integración de sus proyectos. Se trabajó con el apoyo de la plataforma GitHub (https://github.com/) donde primeramente se tenía a HTMoL en su versión 2.0 escrito con la ayuda del framework three.js.

Al empezar con la integración y pruebas de cada uno de los proyectos nos dimos cuenta de que era imposible trabajar con complejos proteínicos que contenían más de 20,000 átomos, esto debido a la forma de renderizar en three.js. Por esta razón se buscaron y dieron varias opciones para optimizar el renderizado con dicho framework. No obteniendo ningún resultado satisfactorio además de que cada cierto tiempo se cambia la versión de este framework junto con algunas de sus funciones, se optó definitivamente por realizar todo el trabajo directamente con WebGL. La eliminación de este framework no tuvo gran impacto en los proyectos de visualización de trajectorias con DCD y la integración de una interfáz gráfica amigable. No obstante para integrar las representaciones moleculares de Trace, Ribbon y Spline se tuvo una complejidad más grande, esto debido a la realización de las funciones requeridas directamente en WebGL.

El presente proyecto dió como resultado una tesis para obtener un grado de maestría, además de tres tesis de licenciatura en el área de las Ciencias Computacionales.

6.1. Perspectivas

En el presente capítulo se habla hacerca del trabajo futuro y las perspectivas que se tienen para la aplicación HTMoL.

La importancia e impacto en el campo de la Biología Estructural moderna de HTMoLv3 radica en que además de mostrar estructuras moleculares en 3D en cualquier navegador web que soporte el estándar WebGL, puede ser mostrada la trayectoria resultado de simulaciones de dinámica molecular contenida en los principales formatos existentes (XTC y DCT) con la ayuda de técnicas de streamming, permitiendo al usuario una visualización de los frames conforme se vayan descargando. Además de esto se tiene integrado un lenguaje de comandos de sencilla utilización para una selección especial de la molécula dejando así un potencial grande en su uso.

No obstante durante la realización de este proyecto se observaron posibles modificaciones que podrían tener un gran efecto en la optimización del renderizado en WebGL. Los cambios que se proponen son los siguientes:

- Implementación de un lenguaje de caracteres por medio de texturas y comparar el tiempo de renderizado con el actual sistema de caracteres implementado por triángulos para ver si se logra tener un mejor desempeño en los tiempos de renderizado.
- Cambiar la herramienta de Node.js en el sistema de streamming para que se pueda compartir el puerto con el servidor de publicación de la página.
- Implementar las representaciones faltantes dentro de HTMoL, esto es la representación *cartoon*, *ribbon* y *strands* para la visualización de la molécula.
- Usar la técnica de *esferas impostoras* con ray casting para dibujar los átomos.

En el presente proyecto primeramente se integraron los caractéres a dibujar en la escena por medio de texturas, no obstante al visualizar los números se veían algo deformados y se optó por la implementación de caractéres dibujados por medio de triángulos. Se propone el buscar una manera de mostrar las texturas con alguna técnica de tal forma que se tenga una buena visualización de los números. Se espera que con esto se pueda tener una optimización en el renderizado, esto debido a que por cada número se utilizaría una textura integrada en un cuadrado transparente. De esta forma se estarían usando sólo dos triángulos por cada número que se requiera en la escena.

HTMoLv3 utiliza la *primitiva* llamada triángulo en la parte del *fragment shader* para dibujar la esfera. Como ya se mencionó en el capítulo tres esta técnica utiliza tres vértices para dibujar un triángulo y realizar el pintado de los pixeles correspondientes a este, y a su vez una esfera está integrada por cientos de triángulos. Existe una técnica llamada *esferas impostoras* la cuál utiliza esta misma *primitiva*. La diferencia radica en que por cada esfera que es pintada sólo se necesitan 12 triángulos formando un cubo de longitud del diámetro de la esfera. Esto se hace con la ayuda de funciones matemáticas

usadas en el *fragment shader*, de tal forma de que cada vez que son procesados los vértices correspondientes a este cubo varios rayos con origen en la cámara son enviados realizando un casting para definir qué pixeles serán visibles con el objetivo de darle la forma de esfera, y los que no, serán descartados. Asimismo se realiza un cálculo de los pixeles que serán afectados por los rayos de la iluminación utilizada en la escena. De esta forma se reduce en gran manera la información requerida por JavaScript para dibujar las esferas. Dejando la mayor parte del procesamiento en el *fragment shader*. La mayoría de las computadoras actuales soportan esta técnica por medio de sus tarjetas gráficas. No obstante, hay algunas tecnologías que no lo hacen [51].

Bibliografía

1. Goddard, T. D., & Ferrin, T. E. (2007). Visualization software for molecular assemblies. *Current opinion in structural biology, 17*(5), 587-595.

2. O'Donoghue, S. I., Goodsell, D. S., Frangakis, A. S., Jossinet, F., Laskowski, R. A., Nilges, M., ... & Olson, A. J. (2010). Visualization of macromolecular structures. *Nature methods, 7*, S42-S55.

3. Bustos Jaimes, I., Castañeda Patlán, C., Oria Hernández, J., Rendón Huerta, E., Reyes Vivas, H., & Romero Álvarez, I. (2008). HERRAMIENTAS DE VISUALIZACIÓN MOLECULAR PARA LA ENSEÑANZA. *MENSAJE BIOQUÍMICO*, 32.

4. Flores Herrera, O., Rendón Huerta, E., Riveros Rosas, H., Sosa Peinado, A., Vázquez Contreras, E., & Velázquez López, I. (2005). LA ESTRUCTURA Y LA VISUALIZACIÓN MOLECULAR DE PROTEÍNAS. *MENSAJE BIOQUÍMICO*, 29.

5. Giordan, M., & Gois, J. (2009). Entornos virtuales de aprendizaje en química: una revisión de la literatura. educación química, 20(3), 301-303.

6. García-Ruiz, M. A., Valdez-Velazquez, L. L., & Gómez-Sandoval, Z. (2012). Estudio de usabilidad de visualización molecular educativa en un teléfono inteligente. Química Nova, 35(3), 648-653.

7. BEDOYA, A. M., & SÁNCHEZ, G. I. ESTRUCTURA MOLECULAR Y ANTIGÉNICA DE LA VACUNA CONTRA EL VIRUS DEL PAPILOMA HUMANO 16 (VPH 16).

8. Richardson, D. C., & Richardson, J. S. (2002). Teaching molecular 3D literacy. Biochemistry and Molecular Biology Education, 30(1), 21-26.

9. Chapa, S., Martínez, J., Sierra, N. & Estrada, J. *Memoria de la Base de Datos de la Colección Nacional Microbiana CDBB-500*. PorrúaPrint. p56.

10. Wilkinson, A., & McNaught, A. (1997). IUPAC Compendium of Chemical Terminology, (the "Gold Book"). p121

11. Eric Freeman and Elisabeth Robson. Head first HTML5 programming: building web apps with JavaScript. OReilly Media, Inc., USA, 1st edition, 2011.

12. Matsuda, K., & Lea, R. (2013). *WebGL programming guide: interactive 3D graphics programming with WebGL*. Addison-Wesley. p36

13. Jos Dirksen. Three.js Essentials. Packt Publishing, 1st edition, 2014.

14. Keith, J., & Scripting, D. O. M. (2005). Web Design with JavaScript and the Document Object Model. Friends of ED Publishing. p5.

15. Montenegro, R. (Ed.). (2001). Biología evolutiva. Editorial Brujas. pp55-56

16. Brian Danchilla. Beginning WebGL for HTML5. Apress, Berkeley, CA, 1st edition, 2012.

17. K. Matsuda and R. Lea. WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL. Pearson Education, USA, 1st edition, 2013.

18. Sethi, R., & Ullman, J. D. (1998). *Compiladores: principios, técnicas y herramientas. Pearson Educación* pp118-120.

19. Brooks, B. R., Brooks, C. L., MacKerell, A. D., Nilsson, L., Petrella, R. J., Roux, B., ... & Caflisch, A. (2009). CHARMM: the biomolecular simulation program. *Journal of computational chemistry, 30*(10), 1545-1614.

20. Wolfgang F. Engel. Beginning direct3D game programming. Premier Press, USA, 2nd edition, 2003.

21. Case, D. A., Cheatham, T. E., Darden, T., Gohlke, H., Luo, R., Merz, K. M., ... & Woods, R. J. (2005). The Amber biomolecular simulation programs. *Journal of computational chemistry, 26*(16), 1668-1688.

22. J.L.B. Orero and S.M. Fernández. Introducción a OpenGL. Editorial Dykinson, S.L., Madrid, España, 1st edition, 2007.

23. Wu, Y., Lee, J., & Wang, Y. (2007). A Comparative Study of GROMACS and NAMD..

24. Pronk, S., Páll, S., Schulz, R., Larsson, P., Bjelkmar, P., Apostolov, R., ... & Hess, B. (2013). GROMACS 4.5: a high-throughput and highly parallel open source molecular simulation toolkit. *Bioinformatics*, btt055.

25. Andrio, P., Fenollosa, C., Cicin-Sain, D., Orozco, M., & Gelpí, J. L. (2012). MDWeb and MDMoby: an integrated web-based platform for molecular dynamics simulations. *Bioinformatics*, 28(9), 1278-1279.

26. Makarewicz, T., & Kazmierkiewicz, R. (2013). Molecular dynamics simulation by GROMACS using GUI plugin for PyMOL. *Journal of chemical information and modeling, 53*(5), 1229-1234.

27. Hanson, R. M., Prilusky, J., Renjian, Z., Nakane, T., & Sussman, J. L. (2013). JSmol and the Next-Generation Web-Based Representation of 3D Molecular Structure as Applied to Proteopedia. *Israel Journal of Chemistry, 53*(3-4), 207-216.

28. Liu, Y., Stone, J. E., Cai, E., Fei, J., Lee, S. H., Park, S., ... & Schulten, K. (2014). VMD as a Software for Visualization and Quantitative Analysis of Super Resolution Imaging and Single Particle Tracking. *Biophysical Journal, 106*(2), 202a.

29. DeLano, W. Pymol: An open-source molecular graphics tool. CCP4 Newsletter On Protein Crystallography, 40, 2002.

30. Nakane, T. (2014). GLmol-Molecular Viewer on WebGL/Javascript, Version 0.47.

31. Li, H., Leung, K. S., Nakane, T., & Wong, M. H. (2014). iview: an interactive WebGL visualizer for protein-ligand complex. *BMC bioinformatics, 15*(1), 1.

32. Claros Díaz, M. G., & Díaz, M. G. C. (2001). *Bioquímica aplicada: Manual para el diseño experimental y el análisis de datos en bioquímica y biología molecular* (No. Sirsi) i9788495687012).

33. Riera, M. A., Caldez, M., Giorgio, E. M., Milde, L. B., & Zapata, P. D. (2010). Utilización del programa de visualización molecular RasMol como estrategia didáctica para la integración del contenido curricular ṗroteínas: *Educación médica, 13*(3), 157-162.

34. Cariaga Martínez, A. E., & Darío Zapata, P. (2006). Aplicación de un trabajo práctico autoguiado para la formación en el uso de herramientas bioinformáticas de alumnos de pregrado en Bioquímica Clínica. *Educación médica, 9*(4B), 207-211.

35. Rose, A. S., & Hildebrand, P. W. (2015). NGL Viewer: a web application for molecular visualization. *Nucleic acids research*, gkv402.

36. Sayle, R.A. & Milner-White, E.J. *RASMOL: biomolecular graphics for all. Trends Biochem.* Sci. 20, 374 (1995).

37. Protein Data Bank. Protein data bank. Nature New Biol, 223:233, 1971

38. Alberts, B., & Bray, D. (2006). *Introducción a la biología celular*. Ed. Médica Panamericana. p.45.

39. Vazquez, N., & Ines, M. (2006). *Algunos aspectos básicos de la química computacional*. UNAM. pp.25-26.

40. Hess, B., van Der Spoel, D., & Lindahl, E. (2010). Gromacs user manual version 4.5. 4. *University of Groningen, Netherland*.

41. Matsuda, K., & Lea, R. (2013). *WebGL programming guide: interactive 3D graphics programming with WebGL*. Addison-Wesley.

42. Arora, S. (2014). *WebGL Game Development*. Packt Publishing Ltd. p325-332.

43. Cantor, D., & Jones, B. (2012). *WebGL beginner's guide*. Packt Publishing Ltd.

44. Leonardo Alvarez R. Desarrollo de un sistema de visualización molecular utilizando tecnología web para su aplicación en el campo de la virología estructural. Bachelor's thesis, ITESI, Guanajuato, México, 2015.

45. M. Griss and G. Yang. *Mobile Computing, Applications, and Services: Second International ICST Conference, MOBICASE 2010, Santa Clara, CA, USA, October 25-28, 2010, Revised Selected Papers*. Springer Berlin Heidelberg, USA, 1st edition, 2012.

46. P. Lubbers, B. Albers, and F. Salim. *Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development*. Apress, USA, 1st edition, 2010.

47. Brad Dayley. *Node.js, MongoDB, and AngularJS Web Development*. Pearson Education, USA, 1st edition, 2014.

48. Alan Burns and Andrew J Wellings. *Real-time systems and programming languages: Ada 95, real-time Java, and real-time POSIX*. Pearson Education, 2001.

49. L. Van Lancker. *jQuery: el framework JavaScript de la Web 2.0*. Ediciones ENI, Barcelona, España, 2nd edition, 2014.

50. E. H. Pérez. *Tecnologías y redes de transmisión de datos*. Limusa, México, D.F., 1st edition, 2003.

51. Chavent, M., Vanel, A., Tek, A., Levy, B., Robert, S., Raffin, B., & Baaden, M. (2011). GPU-accelerated atom and dynamic bond visualization using hyperballs: A unified algorithm for balls, sticks, and hyperboloids. *Journal of computational chemistry, 32*(13), 2924-2935.

52. Stylos, J. (2009). *Making APIs more usable with improved API designs, documentation and tools*. ProQuest.

Apéndices

Apéndice A

Streamming

En el presente anexo se describe la forma en que se implementó la visualización de una trayectoria de dinámica molecular en HTMoL por medio de técnicas de Streamming, esto es, sin la necesidad de descargar el archivo completo para poder comenzar con la visualización de la dinámica molecular. Gran parte de dicha implementación fué realizada previamente por García Vieyra Javier, sin embargo, se incluye aquí para que exista la documentación correspondiente dentro del contexto del desarrollo que se presenta en esta Tesis.

A.1. Arquitectura empleada

Inicialmente se diseñó una arquitectura para poder abarcar los procedimientos necesarios para el análisis de la información binaria, la obtención de la información tanto para estructuras estáticas como para trayectorias de dinámica molecular y la comunicación entre los componentes.

A.1.1. Cliente

Como se muestra en la figura A.1, el cliente es el encargado de desplegar la información al usuario mediante una interfaz Web. La interfaz del visualizador contiene un menú despleglable que agrupa las funcionalidades relacionadas con:

- Lectura y despliegue de estructuras estáticas desde algún servidor Web.
- Lectura y reproducción de trayectorias de dinámica molecular desde el servidor de la aplicación.
- Selección de distintos elementos de las estructuras como son átomos, aminoacidos, entre otros.
- Acciones para visualizar la estructura desde distintas perspectivas, medir distancias entre átomos, mostrar los ejes, etc.

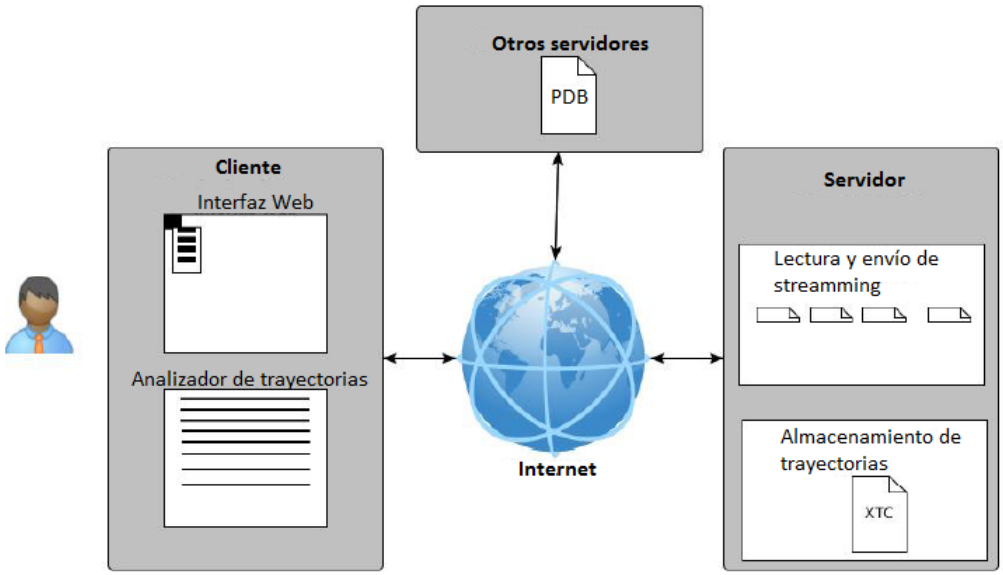

Figura A.1: Arquitectura para el proceso de Streamming

- Opciones para cambiar entre distintas representaciones de las estructuras.

A.1.2. Servidor

El servidor es la parte en donde se alojan los archivos de trayectorias de dinámica molecular. Las funciones que realiza el servidor son:

- Recepción de solicitudes de archivo.
- Búsqueda de archivos.
- Envío de archivos en forma de streaming.

A.1.3. Comunicación con servidores externos

El cliente es el que se encarga de solicitar y recibir la información necesaria para el despliegue de las estructuras estáticas o dinámicas. El cliente debe comunicarse con algún servidor externo para obtener los archivos necesarios. Como se muestra en la figura A.2, el cliente envía la URL de la ubicación de un archivo PDB, el servidor responde enviando el archivo solicitado, finalmente el cliente analiza el archivo y despliega la estructura según los datos recibidos [35].

A.1.4. Conumicación con el servidor

Por otro lado, el cliente también es el encargado de solicitar, recibir e interpretar la información binaria de los archivos que contienen las trayectorias de dinámica molecular.

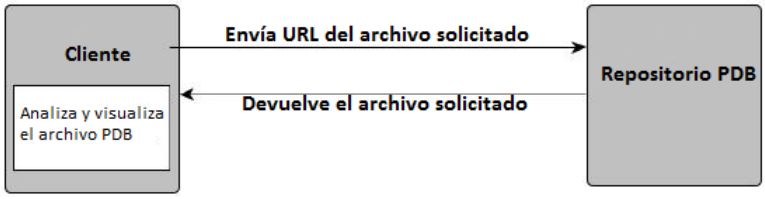

Figura A.2: Solicitud de archivo PDB

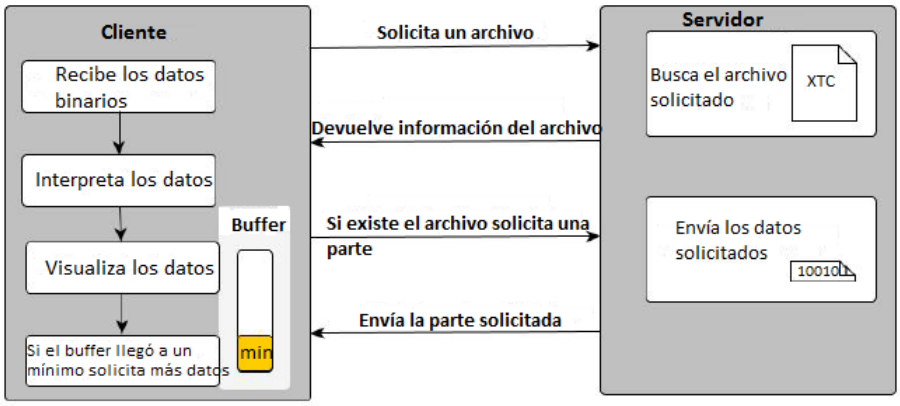

Figura A.3: Solicitud de archivo XTC

Como se muestra en la figura A.3, el cliente debe enviar la ruta de donde se encuentra el archivo alojado en el servidor, el servidor busca el archivo, y si lo encuentra el servidor devuelve una cadena JSON con información sobre el archivo como el tamaño del archivo en bytes y el número de atomos de la molécula. Cuando el cliente ya cuenta con la información necesaria, si el número de átomos de la trayectoria coincide con el número de átomos de la estructura estática, el cliente solicita la primera parte de la trayectoria. El servidor comienza a enviar los streams de la parte solicitada y el cliente los recibe y almacena en un buffer, al mismo tiempo que realiza una serie de cálculos necesarios para obtener las coordenadas de los átomos en cada uno de los frames. Cuando ya se tienen las coordenadas de cierta cantidad de frames, el visualizador comienza a reproducir la trayectoria. Si el buffer revasa un umbral mínimo de datos, el cliente solicita más datos al servidor y repite las operaciones descritas anteriormente hasta llegar al final del archivo.

A.1.5. Web Workers

Existen algunas tareas que necesitan ser llevadas a cabo en paralelo, como es el cálculo de las coordenadas de los frames de las trayectorias, que se tiene que realizar al mismo tiempo que la renderización. Además, mientras se visualiza la trayectoria es deseable que la estrucuctura se pueda manipular ya sea girándola, trasladandola o seleccionando alguno de sus elementos.

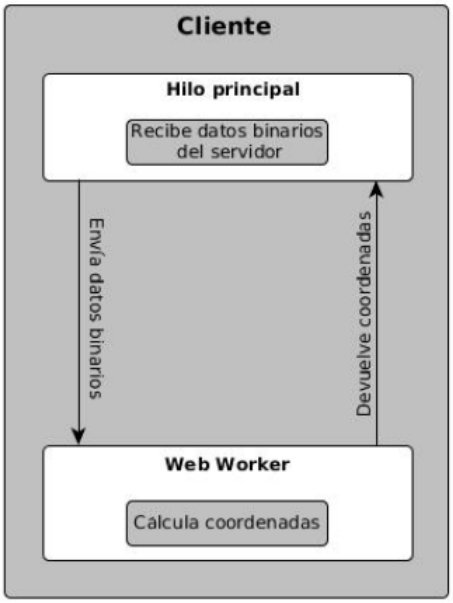

Figura A.4: Funcionamiento del webworker

Para esto agregamos a la aplicación un Web Worker. Los Web Workers le permiten al lenguaje JavaScript ejecutar tareas en paralelo sin bloquear la interfaz de usuario [36].

Los Web Workers generalmente son independientes del contexto de ejecución del navegador. Los Web Workers se ejecutan en un subproceso aislado. Como resultado, es necesario que el código que ejecutan se encuentre en un archivo independiente [37].

Como se muestra en la figura A.4, el hilo principal se comunica con el Web Worker mediante el paso de mensajes, enviando los datos binarios que recibe desde el servidor. El Web Worker recibe los datos y realiza los cálculos necesarios para obtener las coordenadas y enviarlas de regreso hacia el proceso principal. Mientras en hilo principal comienza a renderizar las coordenadas ya calculadas, el Web Worker continúa con el cálculo de las coordenadas para mantener el buffer con suficientes datos.

A.2. Tecnologías utilizadas

Para el desarrollo de la aplicación HTMoLv3 se utilizó HTML5 y el lenguaje de programación JavaScript, junto con la biblioteca WebGL para el despliegue de los gráficos. También se utilizaron las tecnologías de javascript como Node.js y JQuery. Tanto el cliente como el servidor estan implementados en Javascript. El servidor utiliza las funciones de la plataforma Node.js únicamente para crear las conexiones y transmitir los datos solicitados por el cliente. El cliente utiliza tanto las funciones de Node.js como de WebGL para realizar las solicitudes al servidor, el dibujado de los gráficos en el canvas y la transmisión de streams. En la figura A.5 podemos observar la relación entre las tecnologías utilizadas y los componentes de la aplicación. En esta

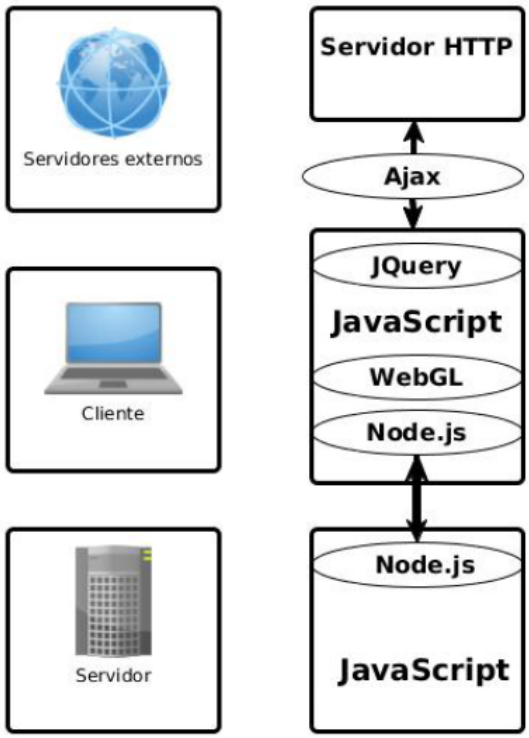

Figura A.5: Tecnologías utilizadas en HTMoLv3

sección se hace una breve descripción de las plataformas utilizadas.

A.2.1. Node.js

Node.js, de ahora en adelante Node, es un proyecto creado por Ryan Dahl a principios de 2009. Se diseñó orientado a la creación de aplicaciones para Internet, principalmente Web, porque la programación de software para servidores era el tipo de desarrollo que hacía el autor en aquella fecha [37].

Node es una plataforma construida encima del entorno de ejecución Javascript de Chrome para construir fácilmente rápidas y escalables aplicaciones de red. Node.js usa un modelo de E/S no bloqueante dirigido por eventos que lo hace ligero y eficiente, perfecto para aplicaciones data-intensive en tiempo real.

Node provee un entorno de ejecución para un determinado lenguaje de programación y un conjunto de librerías básicas, o módulos nativos, a partir de las cuales se pueden crear aplicaciones orientadas principalmente a las redes de comunicación, aunque una parte de estas librerías permite interactuar con componentes del sistema operativo a través de funciones que cumplen con el estándar POSIX. Básicamente este estándar o familia de estándares define las interfaces y el entorno, así como utilidades comunes, que un sistema operativo debe soportar y hacer disponibles para que el código fuente de un programa sea portable (compilable y ejecutable) en diferentes sistemas operativos que implementen dicho estándar. En este caso, Node facilita fun-

ciones para manejo de archivos, Entrada/Salida, señales y procesos conformes a las características establecidas por POSIX [38].

A.3. Protocolos

Los protocolos designan el conjunto de reglas que rigen el intercambio de información a través de una red de computadoras. HTMoL necesita comunicación en red, por lo tanto, utiliza el Protocolo de transferencia de Hipertexto, HTTP, ya que en él se basa toda la Web y el Protocolo de Control de Transferencia, TCP, para la transmisión de datos como un flujo continuo, o stream. A continuación se detallan los protocolos utilizados.

A.3.1. Protocolo de transferencia de Hipertexto (Hypertext Transfer Protocol)

El Protocolo de transferencia de Hipertexto (HTTP) es un protocolo a nivel de aplicación usado en la comunicación de sistemas distribuidos y relacionados entre sí mediante hipermedia. Es un protocolo de tipo cliente-servidor y solicitud-respuesta, en el que mediante un URI un cliente envía una solicitud a un servidor y este envía de vuelta una respuesta. Existen varios métodos (también llamados verbos) que definen el tipo de solicitud, siendo los más comúnmente usados GET, POST, PATCH y DELETE.

A.3.2. Protocolo de control de transferencia (Trasfer Control Protocol)

EL protocolo de control de transferencia (TCP) es uno de los principales protocolos de la capa de transporte del modelo TCP/IP. En el nivel de aplicación, posibilita la administración de datos que vienen del nivel más bajo del modelo, o van hacia él, (es decir, el protocolo IP). Cuando se proporcionan los datos al protocolo IP, los agrupa en datagramas IP, fijando el campo del protocolo en seis (para que sepa con anticipación que el protocolo es TCP). TCP es un protocolo orientado a conexión, es decir, que permite que dos máquinas que están comunicadas controlen el estado de la transmisión. TCP elimina la duplicación de datos, asegura que estos se vuelvan a ensamblar en el mismo orden en que se enviaron y reenvía información cuando se pierde un datagrama [40].

A.4. Comunicación cliente-servidor

En esta sección se explica la manera en la que se realiza la comunicación entre el cliente y el servidor de la aplicación. Para lograr la comunicación entre el cliente y

el servidor se utilizó la tecnología de los Websockets. En las siguientes secciones se detalla el uso de los Websockets en HTMoL.

A.4.1. WebSockets

Existen tecnologías que permiten al servidor enviar datos al cliente en el mismo momento que detecta que hay nuevos datos disponibles. Se conocen como "Push" o "Comet". Uno de los trucos más comunes para crear la ilusión de una conexión iniciada por el servidor se denomina Long Polling (sondeo largo). Con el Long Polling,vel cliente abre una conexión HTTP con el servidor, el cual la mantiene abierta hasta que se envíe una respuesta. Cada vez que el servidor tenga datos nuevos, enviará la respuesta (otras técnicas implican Flash, solicitudes XHR de varias partes y los denominados html files). El Long Polling y las otras técnicas funcionan bastante bien. Se utilizan todos los días en aplicaciones como el chat de Gmail.

Sin embargo, todas estas soluciones comparten un problema: el exceso del HTTP, lo que no las hace aptas para aplicaciones de baja latencia. Por ejemplo los juegosm-multijugador de tiro en primera persona del navegador o cualquier otro juego online con un componente en tiempo real.

Los Websockets (WS) son una tecnología avanzada que permite establecer una sesión de comunicación interactiva entre el navegador Web del usuario y el servidor. Con esta tecnología es posible enviar mensajes al servidor y recibir respuestas por eventos sin tener que consultar al servidor para una respuesta.

La tecnología Websocket proporciona un canal de comunicación bidireccional sobre un único socket TCP. En la figura A.6 se observa el funcionamiento general de un Websocket, en donde el cliente inicia la conexión a través de una petición HTTP (handshake) en la cual se establecen los parámetros de la comunicación. Cuando la comunicación se establece se deja de utilizar el protocolo HTTP y se pasa a utilizar el WebSocket.

Una vez que se tiene una conexión a través de Websockets, se mantiene una conexión persistente entre el cliente y el servidor, y ambas partes pueden enviar datos en cualquier momento. Así también cualquiera de las dos partes puede cerrar la conexión.

En HTMoL se utilizan los Websockets para establecer la comunicación entre el cliente y el servidor cuando el usuario quiere reproducir una trayectoria de dinámica molecular que se encuentra en el servidor. El servidor abre un puerto y permanece escuchando. El cliente crea un nuevo Websocket con la URL del servidor e inicia la conexión para la transmisión de datos.

A.5. Tansmisión de trayectorias

En la sección anterior se explicó la manera en la que se comunica el cliente con el servidor. La comunicación que se realiza es utilizada para la transferencia de datos binarios desde el servidor hacia el cliente, los cuales contienen información de las

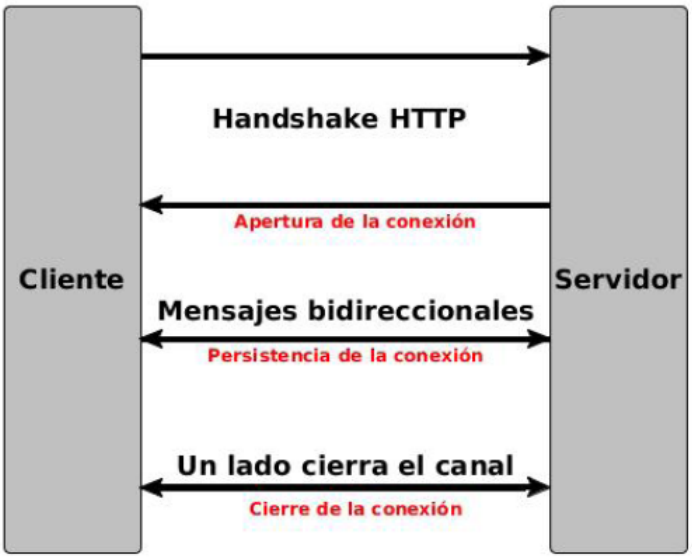

Figura A.6: Representación de un websocket

coordenadas de cada uno de los átomos, que componen una molécula, en cada uno de los frames que contiene una trayectoria de dinámica molecular. En esta sección se describe la forma en la que estos datos se transmiten del servidor al cliente para su posterior procesamiento y visualización. La transferencia se realiza utilizando las funciones de la plataforma Node y haciendo uso de los Websockets.

A.5.1. Transferencia de streams

De todas las herramientas con las que cuenta el desarrollador de Node, una de las más potentes es sin duda la transferencia de los streams.

Un stream en Node es un objeto que encapsula un flujo binario de datos y provee mecanismos para la escritura y/o lectura en él. Por tanto, pueden ser readable, writable o ambas cosas. Cuando se tiene una conexión establecida entre el cliente y el servidor, el cliente envía un mensaje, en formato JSON, hacia el servidor con la ruta del archivo que está solicitando. El servidor verifica la existencia del archivo y responde con el tamaño en bytes del archivo en caso de existir o con un mensaje de error en caso contrario. Si el archivo existe, el cliente envía otro mensaje solicitando el número de átomos de la trayectoria solicitada. El servidor abre el archivo y lee del byte cuatro al byte siete que son los que representan el número de átomos en la trayectoria (formato XTC) y los envía al cliente. El cliente verifica que esta cantidad coincida con el número de átomos de la estructura estática, en caso de coincidir, envía un nuevo mensaje solicitando una parte del archivo de la trayectoria en bytes, a lo cual el servidor responde leyendo los datos solicitados y enviándola en pequeños flujos de información. Por defecto Node envía 65536 bytes (64kb), este es el tamaño de los streams que son enviados al cliente. Sin embargo, este parámetro se puede cambiar

para aprovechar mejor los recursos de la red.

Los streams que se envían son creados con la función CreateReadStream de Node, el cual devuelve un objeto ReadStream de 64kb, esto es, regresa un bloque de información que se puede leer.

El cliente recibe los datos enviados por el servidor, los almacena en un Buffer y los comienza a procesar, cuando el Buffer se queda vacío el cliente solicita más datos al servidor y repite las operaciones anteriores hasta que el servidor termine de enviar la cantidad de datos solicitada por el cliente.

A.5.2. Apertura de puertos

Los *routers* o el servicio que nos proporciona la DNS para tener la conexión a Internet por lo general tiene habilitado el puerto 80 con los protocolos *tcp* y *udp*. Es necesario activar nodejs en otro puerto diferente en donde esté montado nuestro servidor de publicación de nuestra página, que por lo general es *Apache*. Para esto necesitamos decirle a nuestro proveedor de DNS que active este otro puerto con estos protocolos, si no es así, no podrá inicializar el proceso del *streamming* en nuestro servidor.

A.6. Procesamiento de datos binarios

Todos los streams que recibe el cliente son objetos de tipo Buffer, así que si se desea ver su representación en texto se tendrán que convertir de formato. En el caso de nuestra aplicación se tienen que realizar una serie de cálculos para convertir los datos binarios en datos de trayectorias moleculares. Para manipular la información binaria se hace el uso de los objetos ArrayBuffer de JavaScript y sobre estos se hace una serie de operaciones que se describen más adelante.

A.6.1. ArrayBuffer

Un buffer (implementedo por el objeto ArrayBuffer) es un objeto representando un segmento de datos; no tiene formato de referencia, y no ofrece mecanismo para acceso a su contenido. Para acceder a la memoria contenida en un buffer, es necesario usar una vista. Una vista provee un contexto (que es, un tipo de dato, desplazamiento de inicio, y número de elementos) que vuelven los datos en un arreglo tipado real.

El ArrayBuffer es un tipo de datos que es usado para representar un buffer de datos binario de tamaño fijo genérico. No se puede manipular directamente el contenido de un ArrayBuffer; en su lugar, se crea una vista de arreglo tipado o una DataView que representa al buffer en un formato específico, y se usa eso para leer y escribir los contenidos del buffer.

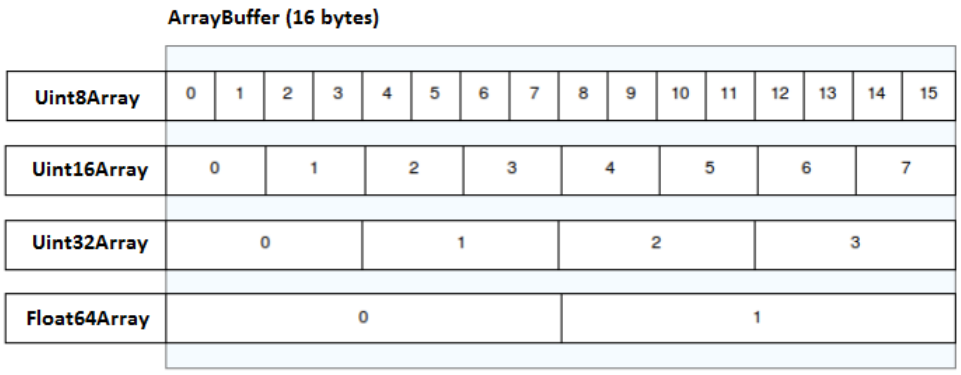

Figura A.7: Representación de arreglos tipados

A.6.2. Arreglos tipados

Las vistas de arreglo tipado tienen nombres autodescriptivos y proveen vistas para todos los tipos numéricos usuales como Int8, Uint32, Float64, etcétera. Hay una vista de vector tipado especial, el Uint8ClampedArray. Restringe los valores entre 0 y 255. Esto es útil para procesamiento de datos de Canvas, por ejemplo.

En la figura A.7 se muestra una representación de las distintas vistas que se pueden utilizar en un objeto ArrayBuffer de 16 bytes.

A.6.3. Interfaz DataView

La DataView es una interfaz de bajo nivel que provee una API getter/setter para leer y escribir datos en el buffer arbitrariamente. Esto es útil cuando hay que lidiar con diferentes tipos de datos, por ejemplo. Las vistas de arreglo tipado están en el orden de bytes nativo de la plataforma. Con una DataView se puede controlar el orden de bytes. Es big-endian por defecto y puede ser cambiado a little-endian en los métodos getter/setter.

A.6.4. Procedimientos

Las estructuras descritas en las secciones anteriores son utilizadas para manipular la información binaria que llega desde el servidor, el proceso que se realiza es el que se muestra en la figura A.8.

Cuando llega el primer stream al cliente, este es recibido por un WebWorker, lo primero que hace la aplicación es una validación, comparando los primeros 4 bytes con el número 1995, el cual es denominado “número mágico” para los archivos de trayectorias de dinámica molecular con extensión xtc, es decir, si los primeros 4 bytes del archivo representan al número 1995 entonces el archivo es del tipo de xtc, de lo contrario es otro tipo de archivo. También se comparan los siguientes 4 bytes que representan la cantidad de átomos en la molécula. Si las comparaciones anteriores

se cumplen el proceso continua, de lo contrario se envia un mensaje con el error correspondiente.

El siguiente paso es leer el encabezado de un frame, en donde se obtienen datos como es el paso de simulación, el tiempo de simulación, la precisión y el número de bytes necesarios para calcular las coordenadas del frame actual. En esta parte también se encuentran algunos parámetros que se utilizan después para obtener las coordenadas. Al finalizar esta etapa se envía solo la cantidad necesaria de bytes para calcular las coordenadas atómicas en el frame.

Posteriormente se hace una serie de operaciones a nivel de bits para obtener las coordenadas, las cuales son almacenadas en un arreglo tipado. Si el arreglo tiene datos suficientes para comenzar con la reproducción de la trayectoria, los datos son enviados al hilo principal para iniciar la renderización mientras el WebWorker sigue procesando más información. Si hay más datos para calcular otro frame se repite el procedimiento de cálculo de coordenadas, de lo contrario el cliente solicita más datos al servidor. El hilo principal almacena los datos recibidos del WebWorker en su propio buffer, si el buffer llega a un mínimo el hilo principal pide más datos al WebWorker repitiendo el proceso hasta que se llegue al final del archivo.

Apéndice B

Estructura del código y principales algoritmos de HTMoLv3

En el presente anexo se integran los principales algoritmos usados, la jerarquía de directorios y archivos y la lógica y el flujo de la codificación desarrollada en HTMoLv3.

B.1. Algoritmos de HTMoLv3

Los algoritmos que se muestran a continuación se implementaron para mantener la mínima sincronización entre la GPU y el CPU. El algoritmo 1 procesa los átomos seleccionados mostrándolos en un color verde. Para esto a cada átomo que se va a procesar primeramente se checa si ya estaba seleccionado, si es así ya no lo procesa y pasa al siguiente. En caso de que no estuviera seleccionado entonces se pone su propiedad *seleccionado* en true, después de esto se realiza un cálculo para obtener su posición relativa en el arreglo de JavaScript *ColorTotal*, guardándola en una variable llamada *mul*, esto para evitar realizar 1,156 multiplicaciones dentro del loop for que recorre la cantidad de números flotantes de colores de cada esféra de 16 latitudes y 16 longitudes. El número que se ingresa en este arreglo de la línea 8 a la 11 representa la configuración del color verde en formato RGBA (0101). Después de esto de la línea 15 a la 21 se define si el bloque al que pertenece este átomo ya está agregado al arreglo de bloques a procesar. Si no es así lo agrega con el método *push* de JavaScript en la línea 24. Después de que se terminan de procesar todos los átomos a seleccionar, se ingresa al buffer de bloques en la GPU los arreglos que contiene átomos que fueron modificados esto se hace de la línea 29 a la 34. De esta manera sólo se procesan los bloques necesarios y se mantiene mínima la sincronización del CPU con la GPU. Este algoritmo toma como variables globales el arreglo *ColorTotal* y el contexto de WebGL denotado por la variable *gl*.

Algoritmo 1 Procesar selección

Entrada: Conjunto de átomos a seleccionar: AtomosSeleccionados.
Salida: Buffers de color a modificar de la GPU: ArrayBloques.

```
1: ArrayBloques = [ ]
2: para i = 0; Hasta i < AtomosSeleccionados.length hacer
3:   atom = AtomosSeleccionados[i]
4:   si atom.seleccionado=falso entonces
5:     atom.seleccionado=cierto
6:     mul = (atom.PosBloque − 1) ∗ nColor
7:     para j = 0; Hasta j < nColor hacer
8:       ColorTotal[atom.PosBloque − 1][mul + j] = 0
9:       ColorTotal[atom.PosBloque − 1][mul + j + 1] = 1
10:      ColorTotal[atom.PosBloque − 1][mul + j + 2] = 0
11:      ColorTotal[atom.PosBloque − 1][mul + j + 3] = 1
12:      j = j + 4
13:    fin para
14:    agregar =cierto
15:    para j = 0; Hasta j < ArrayBloques.length hacer
16:      si (atom.PosBloque − 1) = ArrayBloques[j] entonces
17:        agregar =falso
18:        break
19:      fin si
20:      j = j + 1
21:    fin para
22:    si agregar =cierto entonces
23:      agregar =falso
24:      ArrayBloques.push(atom.BloqueSolid − 1)
25:    fin si
26:  fin si
27:  i = i + 1
28: fin para
29: para i = 0; Hasta i < ArrayBloques.length hacer
30:   gl.bindBuffer(ColorBuffer[ArrayBloques[i]])
31:   gl.bufferData(newFloat32Array(ColorTotal[ArrayBloques[i]]))
32:   gl.bindBuffer(null)
33:   i = i + 1
34: fin para
```

B.2. Jerarquía de directorios

La Figura B.1 nos muestra la jerarquía de directorios utililizada en HTMoLv3.0; Los directorios contenidos son: *fonts*, *js*, *node_modules*, *php*, *styles*, *pdb* y *trajectories*.

Los archivos codificados en JavaScript se encuentran ubicados en la carpeta llamada *js*. Estos archivos están nombrados de tal forma que se comprenda su contenido, por ejemplo el archivo *Buffers.js* contiene la información relativa a los buffers requeridos para los objetos ingresados en la escena. El archivo *Main.js* contiene las funciones y los objetos principales. El archivo *ButtonsFunctions.js* contiene las funciones requeridas para los botones. El archivo *camera.js* contiene la configuración inicial de la cámara. Dentro de la carpeta node_modules se encuentran los archivos necesarios para instalar node.js en ubuntu. Dentro de la carpeta fonts y styles se encuentran los archivos necesarios para darle un estilo y color a la interfáz. Dentro de la carpeta xtcfiles se encuentra los archivos de dinámica molecular, estos son los archivos con formato dcd y xtc. En la carpeta pdb se encuentran los archivos pdb usados para realizar pruebas y ejemplos. Además de estas carpetas se encuentra el archivo HTMoL.html, el cuál corresponde a nuestra página web principal y otro archivo llamado example.html correspondiendo a un ejemplo de una página web que contiene en un apartado a HTMoL. También se encuentra en nuestra carpeta principal un archivo llamado server.js, este archivo es el que se ejecuta para hacer que HTMoL funcione como servidor y poder enviar datos por medio de streamming.

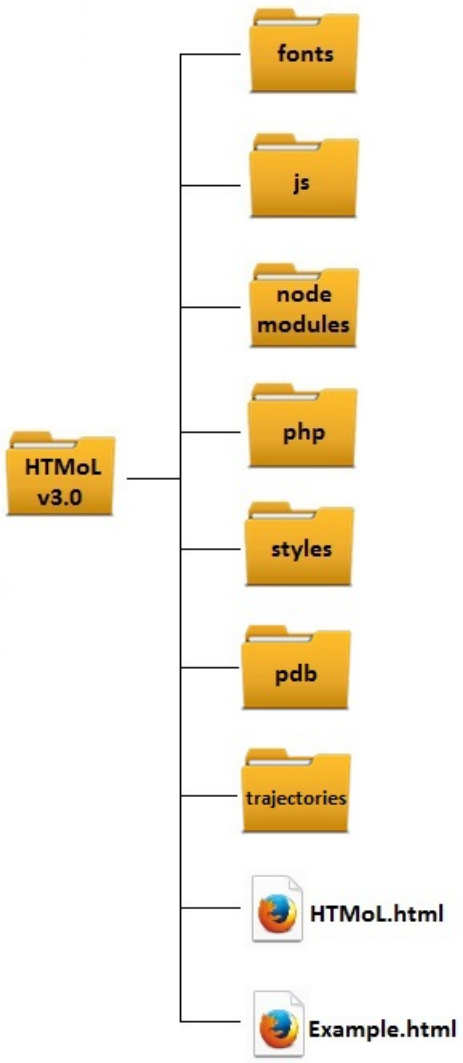

Figura B.1: Jerarquía de archivos de HTMoLv3.0