UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

VISUALIZADOR WEB DE RECONSTRUCCIONES TRIDIMENSIONALES DE TEJIDO HEPÁTICO PARA EL ESTUDIO DE LA ESTRUCTURA TISULAR

Informe de Memoria de Título para optar al título de:
Ingeniero Civil Biomédico

POR

Boris Gonzalo Vega Burgos

Profesor(es) Guía:
Pamela Guevara A.
Fabián Segovia M.

agosto 2022
Concepción (Chile)

Agradecimientos

A mis papás por su constante apoyo durante esta etapa y por sacrificar tanto por permitirme la oportunidad de estudiar.

A mis amigos, en especial la Stefi y la Valu por su apoyo cada vez que estaba mal o necesitaba consejos, la Fran y Patrick por su compañia en este proceso, el cris y el fabo por sus consejos y ayuda.

A mis profesores Pamela Guevara y Fabián Segovia, por su guiarme y aconsejarme durante todo el proceso.

Resumen

El análisis histológico a través de técnicas convencionales 2D nos ha ayudado enormemente a entender la estructura y función de diversos tejidos. Sin embargo, la interpretación de planos o cortes transversales omite información importante de la arquitectura tisular. Recientes avances en microscopía y análisis de imágenes han permitido generar reconstrucciones 3D con resolución sub-celular de los tejidos. Lamentablemente, su utilización está restringida a usuarios avanzados que cuenten con *hardware* de alta potencia y *software* especializado.

Para democratizar el acceso a estas reconstrucciones, se generó un flujo de trabajo automatizado que optimiza los modelos geométricos, logrando así su implementación en una plataforma web de visualización. Esta se basa en *WebGL*, tecnología que permite visualizar, interactuar y extraer información de las estructuras. Como prueba de concepto, se utilizaron reconstrucciones 3D de tejido hepático generadas con el *software Motion Tracking*. Para asegurar la compatibilidad de los modelos y mejorar su rendimiento se utilizó el software de código abierto *Blender*. Se automatizó un proceso de optimización, que reduce el consumo de recursos, pero mantiene los modelos fieles a la morfología original. El proceso de optimización redujo la geometría de los modelos en un 90 % de la original, con un error en la morfología medio de 0.4 micras. Los modelos optimizados redujeron el consumo de memoria RAM del navegador en un 68 %, asegurando su compatibilidad con navegadores presentes en equipos de bajos recursos (e. g. celulares). Esto permite al visualizador mantener una tasa de cuadros por segundo estable que permite interactuar de manera fluida con los modelos. Este método no presenta barreras económicas, ya que sólo requiere de un navegador web y *software* de código abierto. Además, permite a usuarios inexpertos cargar nuevos modelos a la plataforma web.

Se espera que este tipo de herramientas sea utilizada ampliamente para facilitar la comprensión de las estructuras tisulares, funcionando como un complemento a la histología tradicional.

Tabla de Contenidos

Índice de Figuras

Siglas

2D 2 dimensiones

3D 3 dimensiones

API API es una interfaz de programación de aplicaciones (del inglés *application programming interface*)

BaSiC Herramienta para corrección de fondo y sombras en imágenes de microscopía óptica (del inglés *BaSiC image correction method*)

FBX FBX es un formato de archivo de imágenes propietario de Autodesk (del inglés *Autodesk FBX Interchange File*)

GLTF GLTF es el formato estándar para escenas y modelos tridimensionales (del inglés *Graphics Language Transmission Format*)

LSM Imagen de microscopio de escaneo láser (del inglés *Laser scanning microscopy image*)

MC Algoritmo de marcha de cubos (del inglés *Marching Cubes algorithm*)

MT Motion Tracking (del inglés *Motion Tracking*)

ROI Regiones de interés (del inglés *Regions of interest*)

STL STL es un formato de archivo de imágenes para estereolitografía (del inglés *Stereolithography File*)

TIFF TIFF es un formato de archivo de imágenes con etiquetas (del inglés *Tagged Image File Format*)

WBNS Corrección de fondo y eliminación de ruido basado en Wavelet para imágenes de microscopía de fluorescencia (del inglés *Wavelet-based background and noise subtraction for fluorescence microscopy images*)

1 Introducción

1.1 Introducción general

Comprender la interacción entre células en los tejidos es un proceso complicado al utilizar los métodos tradicionales. Un clásico ejemplo es la histología, disciplina que a través de la observación por microscopía óptica de cortes de tejido, permite inferir el estado organizacional de los tejidos y sus principales estructuras. Estos cortes, al ser extremadamente delgados (5-10 μm), solo entregan información en 2 dimensiones (2D) del tejido a estudiar, limitando el proceso de aprendizaje a la observación e interpretación de planos o cortes transversales. Esto es de principal importancia en el caso de tejidos que poseen complejas estructuras en 3 dimensiones (3D), siendo el tejido hepático un excelente ejemplo. El año 2015 se publicó el *software* Motion Tracking (del inglés *Motion Tracking*, MT), en el cual se implementaron una serie de algoritmos para pre procesamiento y segmentación de tejidos en 3D. MT ha sido exitosamente utilizado para entender el tejido hepático [1], a través de la generación de modelos en 3D altamente precisos de los que se puede obtener información estructural, estadística y de organización de cada componente del tejido [1].

Hasta ahora, las reconstrucciones 3D de tejido hepático solo han sido utilizadas con el fin de entender principios básicos de la estructura y función tisular. En este proyecto se propone generar las herramientas que permitan acercar estos modelos 3D al público general.

El uso de visualizaciones en 3D permite mejorar el proceso de aprendizaje, facilitando la comprensión de sistemas tisulares complejos como es el tejido hepático. Por tal motivo se propone desarrollar un *software* de visualización, generando una experiencia inmersiva e interactiva, utilizando datos generados por el *software Motion Tracking*, del que se extraen modelos 3D precisos de las diferentes estructuras presentes en el tejido hepático, además de información asociada relevante, con el fin de entregar una experiencia educativa completa.

1.2 Objetivos

1.2.1 Objetivo general

Desarrollar un sistema de visualización para navegador web, que utilizando modelos 3D de tejido hepático obtenidos del software *"Motion Tracking"*, genere un sistema interactivo para el aprendizaje de biología celular e histología.

1.2.2 Objetivos específicos

- Extraer modelos 3D del tejido hepático utilizando el software *"Motion Tracking"*.
- Implementar un método automatizado para optimizar los modelos 3D, asegurando su funcionalidad en un navegador web.
- Diseñar y desarrollar página web con el visualizador, que permita desplegar información relevante respecto a los modelos utilizados.

1.3 Alcances y limitaciones

El *software* de visualización a crear debe mantener modelos 3D fieles a los extraídos para investigación. Para esto se diseñó un sistema específico de optimización, por lo que en otro tipo de estructuras podría no tener los mismos resultados.

Para obtener los mallados se requiere utilizar *software* especializado en la extracción de mallados en 3D desde imágenes de microscopía. Además luego deben modificarse los mallados para su compatibilidad y rendimiento óptimo en navegador web.

La obtención de los mallados requiere de herramientas de alta exigencia computacional, generando archivos de gran tamaño y alta resolución. Estos archivos requieren de altos tiempos de procesamiento y memoria para ser visualizados, por lo que son editados a través de métodos de optimización, donde parte de la geometría original se pierde. Se utilizan herramientas de análisis para mantener este margen en el mínimo posible, intentando asegurar un rendimiento óptimo con una estructura fiel al mallado original.

Se decidió trabajar con una plataforma web, para eliminar barreras de entrada a la información

presentada, ya que los datos originales requieren de computadores de alta capacidad. Se consideró aplicarlo en realidad aumentada a través de aplicaciones móviles, pero también genera una barrera por razones de compatibilidad, donde no todo equipo es compatible con la tecnología. En cambio, los navegadores web son accesibles desde cualquier tipo de computador, además de dispositivos móviles, con poca exigencia de hardware. Esto trae consigo una menor disposición de recursos para visualizar las imágenes por lo que el proceso de optimización de los mallados debe ser eficaz.

Se trabajó en conjunto al laboratorio *Cell and Tissue Arquitecture Lab* del profesor co-tutor Fabián Segovia. Alumnos tesistas del laboratorio procesan las imágenes de microscopía y obtienen los mallados 3D que luego se utilizan en el proceso de optimización desarrollado. La herramienta se ha desarrollado para su utilización futura dentro del laboratorio, por lo que el feedback de los integrantes del lab fue escencial en el desarrollo de las herramientas, asegurando facilidad en su utilización y agregando características que se consideren necesarias por el grupo.

1.4 Metodología

Se comienza con la extracción de mallados 3D de tejido hepático. Estos corresponden al hígado de un ratón de entre 6 a 9 semanas de vida. El proceso requiere de un preprocesamiento de las imágenes extraídas por microscopía donde se utilizó el *software Fiji* [2] para realizar corrección de fondo y normalización de intensidad de píxeles. Luego se cambia el formato para su compatibilidad con el *software Motion tracking*, este posee las herramientas necesarias para el proceso de segmentación de las estructuras de interés en el tejido, generando los mallados 3D de éstos, su etiquetado y su exportación en formato de extensión .stl. STL es un formato de archivo de imágenes para estereolitografía (del inglés *Stereolithography File*, STL). Luego se importan al software de edición Blender para su optimización y exportación en los formatos necesarios. Para la aplicación móvil se utilizó el formato de extensión .fbx, este es un FBX es un formato de archivo de imágenes propietario de Autodesk (del inglés *Autodesk FBX Interchange File*, FBX) compatible con la plataforma de desarrollo *Unity* [3]. En cambio para la visualización web se utilizó el formato en extensión .glb que corresponde al GLTF es el formato estándar para escenas y modelos tridimensionales (del inglés *Graphics Language Transmission Format*, GLTF).Este permite un rápido procesamiento en el navegador para su renderizado en *WebGL*.

1.5 Temario

El temario del presente informe es el siguiente:

- Capítulo 2: Se presentan estudios de histología hepática, procesos de extracción de imágenes de microscopía y su procesamiento. Además, estándares en el desarrollo de páginas web.
- Capítulo 3: Se detallan trabajos previos en el desarrollo de *software* de visualización de estructuras de interés biológico.
- Capítulo 4: Se describe el proceso de edición de mallados 3D, con las bases necesarias para crear un algoritmo de simplificación de la geometría. Además de una prueba de concepto en base a una aplicación de realidad aumentada.
- Capítulo 5: Se detalla el proceso de desarrollo del visualizador web, utilizando los estándares web descritos en el capítulo 2.
- Capítulo 6: Se señalan las conclusiones de la investigación y el trabajo futuro a desarrollar.

2 Revisión bibliográfica

2.1 Introducción

A continuación se presenta un estudio de la terminología a utilizar en este trabajo. Se profundiza en las áreas de histología, en especial del tejido hepático, además de la visualización de estructuras biológicas.

Luego se realiza una revisión de trabajos previos, con el objetivo de conocer el estado actual del desarrollo de *software* de visualización y comprender las metodologías usadas por diversos autores, así como sus limitaciones y trabajo futuro.

2.2 Tejido hepático

2.2.1 Función del hígado

El hígado es un órgano de vital importancia ya que cumple las funciones de detoxificación, metabolismo y síntesis de proteínas debido a su conexión directa al sistema circulatorio [4]. La bilis producida por los hepatocitos es transportada a través de los canalículos biliares. Esta interviene en procesos de digestión, asociados al proceso de metabolismo, mientras que la sangre se transporta por los sinusoides, transportando nutrientes [5].

2.2.2 Estructura del hígado

El tejido hepático tiene varios modelos de organización, donde ninguno puede considerarse exclusivo al resto. En este caso se analizó el modelo clásico, que define una unidad funcional: el lóbulo hepático [4].

Esta unidad corresponde a una estructura poligonal, normalmente un hexágono, donde el eje central corresponde a la vena central, mientras que en cada vértice se encuentra la triada porta, que comprende la vena y arteria hepática, además del ducto biliar, ver figura 2.1. Entre estas estructuras se encuentran hepatocitos, células estrelladas y células de Kupffer. Conectando la vena central a la porta

están los sinusoides, estructuras que atraviesan todo el tejido en diferentes direcciones a través de los hepatocitos, conectando ambas venas. También se tienen los canalículos biliares, que cruzan entre los hepatocitos recogiendo la bilis producida que es transportada hacia el ducto biliar [5].

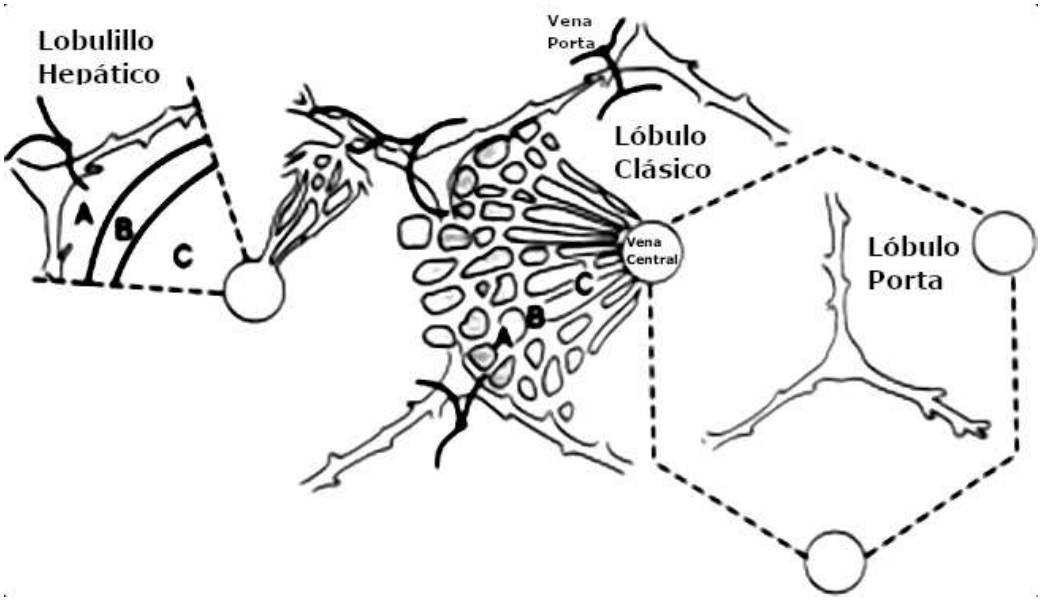

Fig. 2.1: Unidad funcional del hígado, lobulo y lobulillo hepático, se puede observar el modelo clásico, correspondiente al hexágono con la vena central y el modelo primario, que corresponde al de la izquierda, donde solo se considera una sección del hexágono que contiene un vértice. Figura adaptada desde [5].

De este modelo se puede extraer un submodelo, el lobulillo hepático, el cual corresponde a una sección del lóbulo hepático, una especie de cono, que comprende la vena central y la extensión de tejido hasta llegar a un vértice con la triada porta [5]. De este modelo serán extraídas las imágenes a utilizar en este tejido.

2.2.3 Estructuras presentes en el hígado

- Hepatocitos: Forman alrededor del 80 % del volumen del tejido hepático. Son células polares, sus caras basales están en contacto con los sinusoides, los cuales transportan sangre entre las venas central y porta. En cambio, sus caras apicales forman los canalículos biliares, donde secretan la bilis transportada hasta el ducto biliar [6].

- Sinusoides: Son canales que transportan la sangre desde las venas porta hacía la vena central, forman una red compleja de forma radial alrededor de la vena central.

- Canalículos biliares: Son canales que transportan la bilis secretada por los hepatocitos, forman una red que la conduce hasta los ductos biliares.

- Células de Kupffer: Son parte de la pared de los sinusoides. Se encargan de la eliminación de partículas tóxicas y sustancias extrañas en la sangre proveniente de la vena porta. Corresponden a macrófagos anclados a la superficie endotelial interna del sinusoide, por lo que se exponen directamente a la sangre que pasa a través de estos [5].

- Células estrelladas: Corresponden a lipocitos presentes en la zona externa del endotelio sinusoidal, almacenan grasas y vitamina A. Poseen una gran cantidad de microtúbulos y microfilamentos asociados a la pared del sinusoide, en especial con regiones nerviosas, por lo que se estima que participan en la regulación del flujo sanguíneo en los sinusoides. [5].

2.3 Imágenes de microscopía fluorescente a partir de secciones gruesas tejido hepático

El proceso de obtención de imágenes de microscopía requiere el corte de trozos transversales del hígado del sujeto, los cuales deben ser de un grosor mayor al normalmente usado, para mantener la arquitectura de las estructuras intacta. Se aplican tinciones fluorescentes para destacar zonas de interés al momento de realizar las capturas. Estas pueden ser mejoradas al realizar procesos de transparencia en el tejido, las que permiten que los láseres del microscopio puedan recorrer el grosor completo de las muestras [7].

Se obtuvieron imágenes de tejido hepático de ratón, con edad entre 6 a 9 semanas de vida. El hígado fue cortado en secciones de 100 micrómetros de altura. A estas se aplican tinciones fluorescentes [1]:

- DAPI, tinción fluorescente que se adhiere a la región adenina-timina del ácido desoxirribonucleico, lo que permite su uso para la obtención de imágenes de núcleos celulares.

- FLK-1, tinción fluorescente que destaca los sinusoides.

- CD13, tinción fluorescente que destaca los canalículos biliares.

- Faloidina, tinción fluorescente que destaca las membranas celulares.

- F4/80, tinción fluorescente que destaca macrófagos, por lo que se utiliza para detectar las células de Kupffer.

- Desmina, tinción fluorescente que destaca las células estrelladas.

La muestra es aclarada mediante SeeDB, un agente aclarante basado en una solución saturada de fructosa en agua. Este, a diferencia de otros agentes, permite un aclarado rápido, sin mayor deformación de la muestra y compatible con las tinciones recién mencionadas [7]. Este aclarado mejora la obtención de imágenes en secciones gruesas de tejidos, generando una clara diferencia entre la tinción de estructuras de interés y el resto del tejido, facilitando la posterior segmentación.

Para capturar toda la zona de interés del hígado, es necesario tomar varias capturas del tejido, formando 2 secciones o más que conforman toda el área a estudiar. Para poder unir estas secciones se considera un porcentaje de sobrelapado entre ellas y se utilizan herramientas diseñadas para el *software Fiji*, que alinean este sobrelape y generan la imagen final.

2.4 Extracción de mallados 3D de tejido hepático

Parte de la investigación requirió aprender el pipeline de extracción de mallados 3D, donde se utilizan los *software Fiji* y *Motion Tracking*. Se segmentan las Regiones de interés (del inglés *Regions of interest*, ROI), como núcleos,hepatocitos, canalículos biliares y sinusoides, del tejido hepático de un ratón de 6 a 9 semanas de vida. El procesamiento se completó hasta la segmentación de venas grandes. Se decidió utilizar modelos ya segmentados, dado que el pipeline está en constante evolución y procesos como la clasificación de núcleos requieren de entradas manuales, consumiendo demasiado tiempo.

2.4.1 Datos y códigos utilizados

- 4 Imágenes de microscopía láser confocal en formato Imagen de microscopio de escaneo láser (del inglés *Laser scanning microscopy image*, LSM), asociado al tipo de microscopio de marca ZEISS. Con 5 canales para los marcadores fluorescentes utilizados, que destacan estructuras tisulares de interés. La resolución de cada imagen es de 752x752 píxeles. Cada una posee más de 300 planos, donde cada vóxel es de 300 nanometros. Ver figura 2.2.
- *Scripts* para el *software* Fiji, contienen procesos para el alineado y unión de las imágenes.
- *Scripts* para el *software Motion Tracking*, contienen los procesos necesarios para realizar la segmentación.
- 5 Archivos de mallados en formato STL previamente segmentados. Corresponden a las venas porta y central, canalículos biliares, sinusoides, hepatocitos y núcleos.

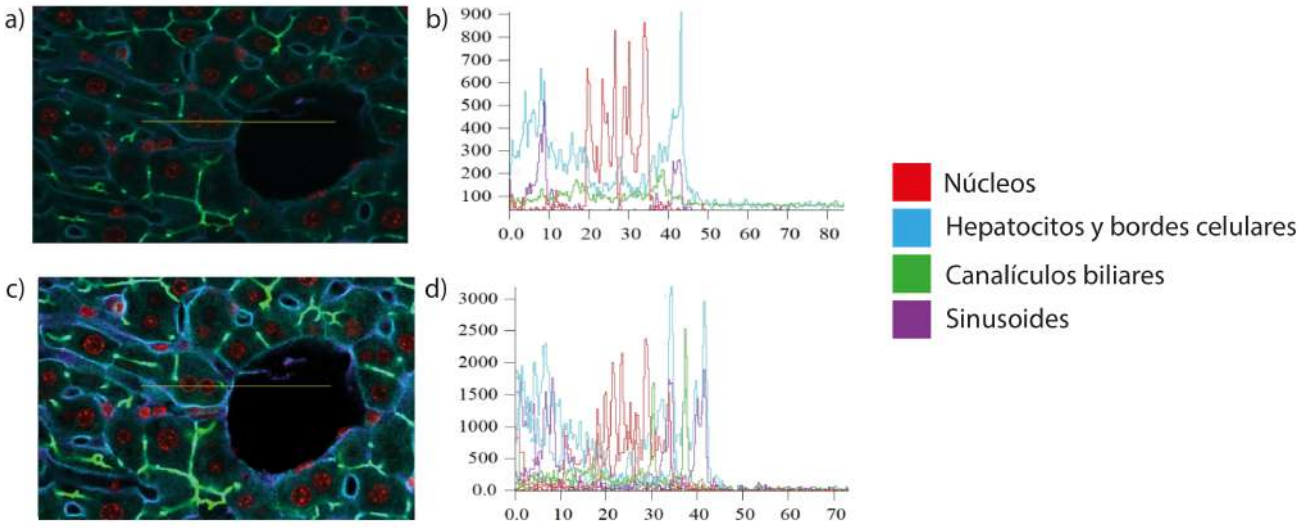

Fig. 2.2: Sección cercana a la vena central. En rojo se observan núcleos, en verde canalículos biliares, sinusoides en magenta y en cyan los hepatocitos. La línea amarilla en a) y c) corresponde a la extracción de la intensidad de píxeles de las imágenes, realizado en el *software Motion Tracking*, b) y d) son la gráfica entregada por el software. a) Imagen original, previa a cualquier procesamiento. b) Gráfico de intensidad de píxeles de la imagen a), se tiene una diferencia de intensidad de alrededor de 800 entre la sección izquierda de la imagen, donde se encuentran los hepatocitos, contra la sección derecha con el interior de la vena central. c) Imagen normalizada, luego de aplicar el método Herramienta para corrección de fondo y sombras en imágenes de microscopía óptica (del inglés *BaSiC image correction method*, BaSiC) en el *software Fiji*. Se observa una intensidad de píxel mayor en el borde de las estructuras frente al fondo, con una diferencia superior a la presente en la imagen original. d) Gráfico de intensidad de píxeles de la imagen c). Se observan intensidades de píxel mayores que las presentes en b). Solo en la sección izquierda, el análisis de la zona interior de la vena central es más cercano a 0 que b).

2.4.2 Mallados tridimensionales

Un mallado corresponde a un grupo de vértices, líneas y caras, que dan forma a un objeto tridimensional. Los vertices son puntos en el espacio, las líneas son la unión de vértices y las caras son superficies delimitadas por líneas.

Las caras pueden formarse por diferentes tipos de polígonos, normalmente triángulos, polígonos de cuatro lados o los denominados *N-gons*, que forman figuras complejas de varios lados y configuraciones. Los renderizadores de imágenes 3D suelen trabajar con figuras de mallados triangulares.

2.4.3 Método BaSiC

Herramienta para corrección de fondo y sombras en imágenes de microscopía óptica (del inglés *BaSiC image correction method*, BaSiC) . Está diseñada para solucionar la presencia de una atenuación de la intensidad de brillo entre el centro y los extremos de imágenes de microscopía óptica, problema que se ve aumentado por la necesidad de utilizar marcadores fluorescentes. Esto afecta la calidad visual de las imágenes obtenidas, pero especialmente altera el análisis de tejidos gruesos.

El método fue creado en base a la ecuación de sombreado 2.4.1, donde considera el efecto de S(x) y D(x), que corresponden al cambio en iluminación a través de la imagen y el plano oscuro respectivamente.

$$I^{meas}(x) = I^{true}(x) \times S(x) + D(x) \tag{2.4.1}$$

El algoritmo genera una matriz de medidas, que luego es descompuesta en una matriz de menor rango y una matriz dispersa. La primera es generada en base a una suma escalar de S(x) y D(x), aplicando constantes de suavizado para regular su dispersión en el dominio de Fourier.

Los parámetros son determinados automáticamente por el algoritmo, con la capacidad de adaptarse a diferentes imágenes [8]. Finalmente, se determinan los valores necesarios para corregir los niveles de intensidad de cada imagen. Aplicando el proceso inverso de 2.4.1 se forma la imagen final, generando una apariencia homogénea y con la gran ventaja de ser utilizable dentro del *software Fiji* [2], que permite importar imágenes de microscopía en muchos formatos, asegurando compatibilidad.

2.4.4 Algoritmo WBNS

Al igual que con BaSiC, esta herramienta fue creada con el fin de corregir los niveles de intensidad en imágenes de microscopia óptica que utilizan fluorescencias. En este caso se aplica la transformada discreta wavelet de Haar.

La información original de la imagen queda dividida en 2 tipos de componentes, de alta frecuencia, llamados coeficientes de detalles y los de baja frecuencia, llamados coeficientes de aproximación.

Luego se aplican wavelets de filtro pasa alto y bajo, donde la información obtenida desde el filtro pasa bajo continúa la operación. El proceso se repite según la cantidad de píxeles de la imagen original y el usuario tiene control sobre el nivel donde se realizará el corte entre fondo e imagen.

El resultado del proceso se resta a la imagen original y luego se aplica la transformada inversa de wavelet para obtener la imagen final [9].

2.4.5 Algoritmo de marcha de cubos

Esta herramienta de construcción de mallados 3D de alta resolución, fue creada específicamente para la construcción de modelos 3D de estructuras anatómicas, utilizando imágenes de tomografía computarizada y resonancia magnética. Este tipo de imágenes corresponde a cortes en 2D de la anatomía por analizar [10].

El algoritmo selecciona 8 píxeles, 4 de un corte y 4 del corte adyacente, generando un cubo entre ellos. Luego determina cómo la superficie se intersecta con el cubo para luego crear los triángulos que coinciden con la intersección. Después avanza a los siguientes 8 píxeles, de aquí el nombre de cubos de marcha.

Dada la geometría del cubo, existen 256 posibles intersecciones, que se consideran para de optimizar la computación y no hacer el cálculo de cada una de ellas. El Algoritmo de marcha de cubos (del inglés *Marching Cubes algorithm*, MC) posee 14 patrones generados gracias a las características simétricas del cubo, por lo que el algoritmo solo selecciona índices asignados a cada patrón pre almacenado, para generar la superficie. Para finalizar, se calcula la normal de cada triángulo generado, información utilizada por algoritmos de renderizado [10].

2.4.6 Preprocesamiento de imágenes

Para mejorar la segmentación se utilizaron *scripts* creados para el *software Fiji*, que aplican los algoritmos descritos en la sección de metodología. Estos corrigen las problemáticas discutidas en puntos anteriores, asociadas a la técnica de microscopía fluorescente.

2.4.6.1 Normalización y corrección de fondo en ejes X e Y

Se ejecutó el primer *script* que aplica el método BaSiC para corregir la diferencia en la intensidad de píxeles provocadas por la iluminación del láser del microscopio. Este se aplica en cada imagen por separado, pero considerando el valor de normalizado común entre las 4 (ver figura 2.2). Esta normalización solo es aplicada en los ejes X e Y.

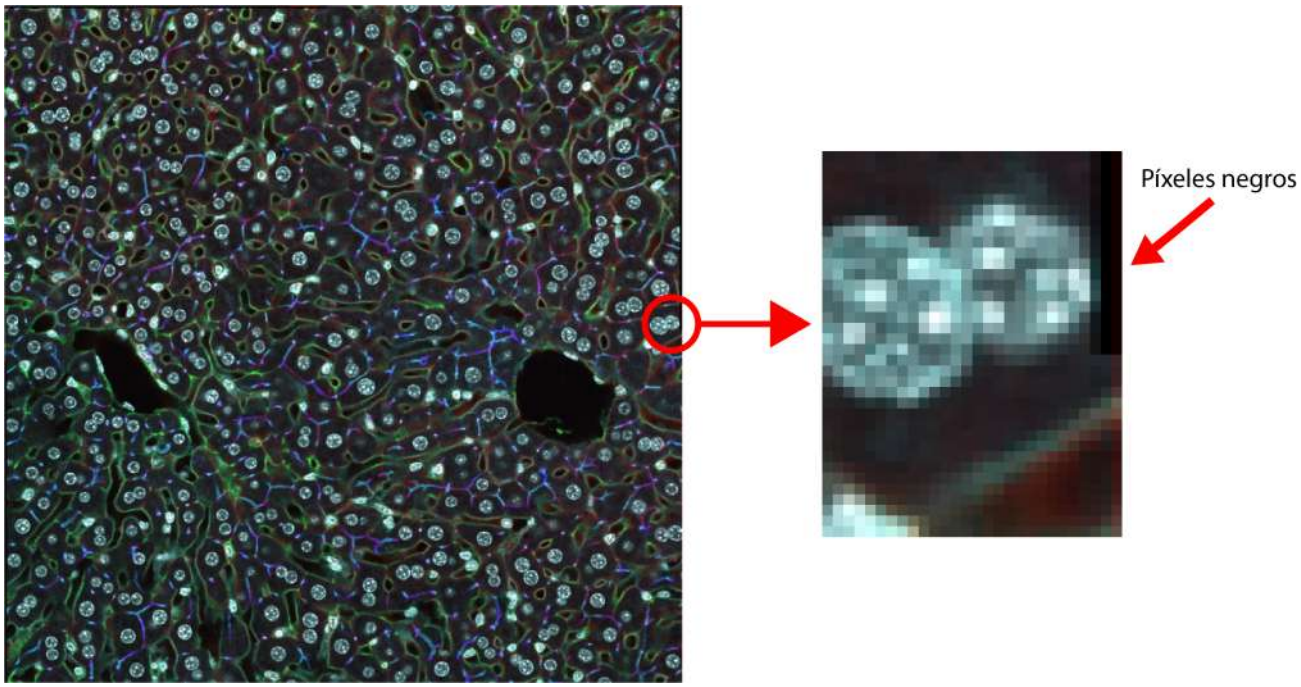

Fig. 2.3: Unión de las imágenes originales por el método de *stitching* aplicado en *Fiji*, con colores asignados a cada canal para diferenciarlos. Se pueden observar 2 zonas oscuras que corresponden a las venas central y porta, además de los píxeles negros generados en los bordes de la imagen.

El proceso de normalización en el eje Z se realizará en *Motion Tracking* ya que parte la segmentación es dependiente de la imagen producida por este *software*. El resultado es almacenado en TIFF es un formato de archivo de imágenes con etiquetas (del inglés *Tagged Image File Format*, TIFF), el cual es compatible con el *software Motion Tracking*.

2.4.6.2 Unión de imágenes

Dado el tamaño del tejido de estudio, la zona de interés se encuentra dividida en 4 imágenes correspondientes a cada cuadrante de la imagen completa a utilizar. Para unir los cuadrantes se utiliza un proceso denominado *stitching*, el cual en base a un 10 % de sobrelape entre cada cuadrante, alínea las 4 imágenes originales uniéndolas en una imagen que abarca un trozo de tejido entre una vena central y una vena porta. El proceso requiere mover los píxeles de las imágenes en base al alineado de este sobrelapado. Por ello, se generan píxeles negros en los extremos de la imagen, además de planos con cuadrantes también negros, dada la diferencia en altura de partes del tejido. Ver figura 2.3.

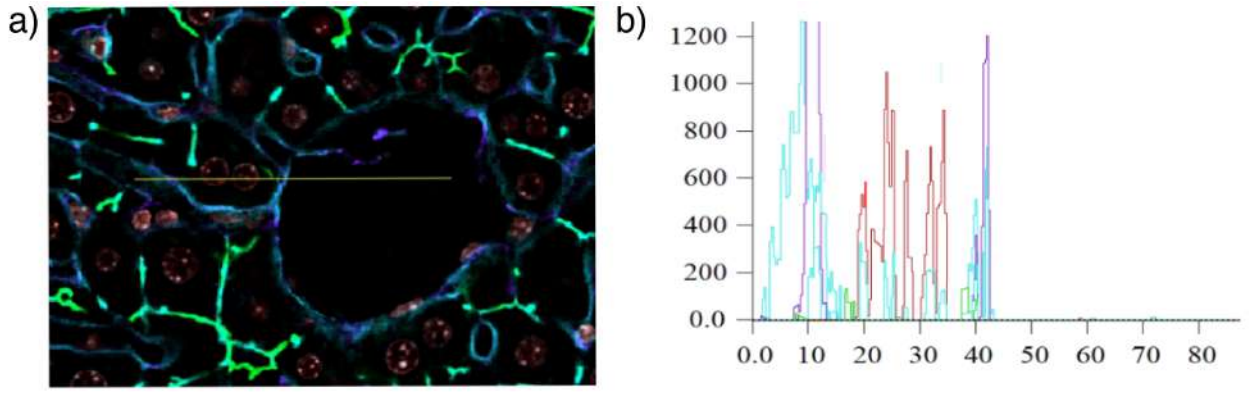

Fig. 2.4: a) Imagen luego de aplicar la corrección de fondo por WBNS, se observa cómo se elimina gran parte de los artefactos en las células que proceden de las fluorescencias aplicadas. b) Gráfico de intesidad de píxel de la imagen a). WBNS mantuvo solo las secciones con intensidad de píxel alta, observado en la sección izquierda del gráfico, generando un gran contraste en la zona de las venas central y porta, denotado por las intensidades 0 de la sección derecha, que corresponde al interior de la vena central. Esto mejora los procesos aplicados por *scripts* de segmentación a utilizar más adelante.

2.4.6.3 Importado de imagen a *Motion Tracking*

Fiji es capaz de importar una variedad de formatos propietarios de archivos de microscopía como el formato LSM utilizado. Se utiliza una herramienta del *software Motion Tracking* la que importa las imágenes directamente desde *Fiji*, solucionando problemas de compatibilidad por formatos.

Se debe realizar un alineado manual de los canales de la imagen en el eje Z. Para ello se utiliza el canal de fluorescencia DAPI + Faloidina como referencia para alinear los 4 canales restantes.

Para eliminar los píxeles negros generados en los procedimientos anteriores, se realiza un corte de la imagen, donde se eliminan los píxeles extremos y los cortes en profundidad con información incompleta. La imagen generada se almacena para la segmentación de la vena central y porta.

2.4.6.4 Aplicación de WBNS y marcado de regiones de interés

Se aplica el algoritmo Corrección de fondo y eliminación de ruido basado en Wavelet para imágenes de microscopía de fluorescencia (del inglés *Wavelet-based background and noise subtraction for fluorescence microscopy images*, WBNS) [9], usando el *software Fiji*. Este aplica una corrección de fondo en base la a transformada de wavelet. WBNS permite editar parámetros de tamaño de ventana y nivel de ruido. En este caso se utilizó una ventana de valor 10 y 2 niveles de ruido y fue aplicado en cada canal de la imagen. Los canales resultantes deben volver a unirse utilizando *Fiji*.

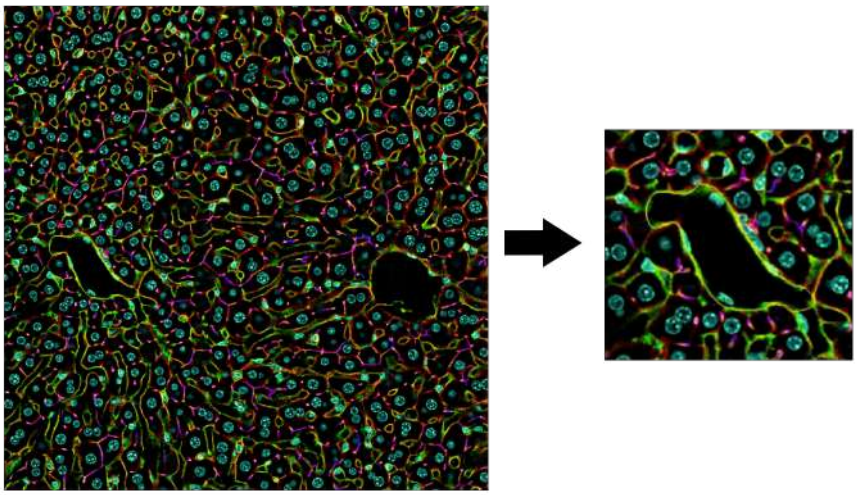

Fig. 2.5: Corte de imagen con ROI dibujada alrededor de la vena porta. Al realizar un acercamiento a la vena porta, se puede observar el marcado de la ROI por el borde de la estructura.

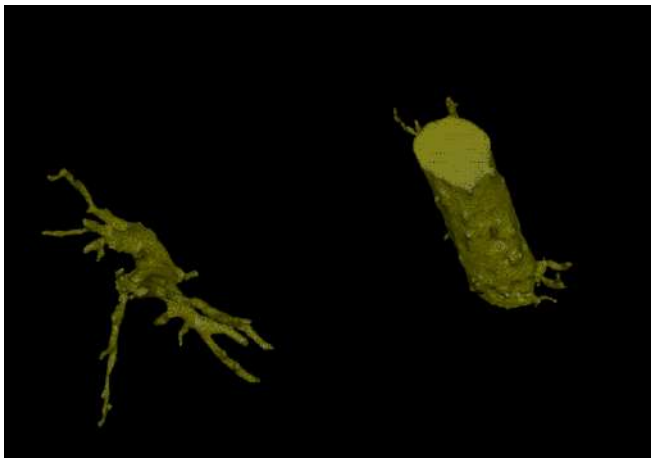

Fig. 2.6: Mallados 3D generados al triangular la segmentación de venas porta y central en el *software Motion Tracking*.

Para la segmentación se requiere delimitar la ROI de la vena porta. Ver figura 2.5. El proceso es manual y requiere dibujar una línea alrededor de la pared de la vena cada 20 a 30 cortes, los cortes intermedios son generados automáticamente por MT.

2.4.6.5 Segmentación de venas

El primer *script* genera los mallados 3D de la vena central y porta, funciona de manera inversa a una segmentación común, ya que se segmenta el fondo de la imagen. Por esto se utiliza la imagen previa a la aplicación de WBNS, ya que la corrección de fondo reduce también la intensidad de píxel entre la membrana y núcleos de los hepatocitos, generando errores en la segmentación.

El proceso aplica el algoritmo MC que genera el mallado 3D. Además se incluyen procesos de suavizado de malla y eliminación de artefactos en base a un umbral de tamaño. Este umbral tuvo que ajustarse para evitar eliminar la vena porta que en esta imagen tiene un menor tamaño a la central. Ver figura 2.6.

2.5 Estándares web

Los estándares web son las tecnologías usadas para crear aplicaciones web. Estas han sido creadas por grupos de estándares, que definen y dan soporte a lenguajes de programación y herramientas. Las 3 principales herramientas son lenguajes de programación, que trabajan en conjunto para generar una página web, *HTML, CSS* y *Javascript*. Además para otras funciones como el renderizado 3D nace *OpenGL*, el cual fue adaptado para su ejecución efectiva en la web generando *WebGL*.

Luego se integran herramientas de estándares abiertos, donde en vez de grupos o empresas, son definidos por comunidades. El código abierto permite una cooperación entre los usuarios y desarrolladores, donde han nacido herramientas que simplifican la utilización de los estándares web, como lo es *Three.js*, que facilita el uso de *WebGL*.

2.5.1 OpenGL

Es una API es una interfaz de programación de aplicaciones (del inglés *application programming interface*, API) utilizada para el renderizado de gráficos en 2D y 3D. Posee la ventaja de ser multilenguaje y multiplataforma, por lo que es la API más utilizada en la industria [11].

Nace con el objetivo de unificar las API de diferentes tarjetas gráficas, evitando que programadores requieran de manejar lenguajes específicos para el equipamiento disponible.

2.5.2 WebGL

Es una API multiplataforma para el renderizado de graficos 3D en navegadores web. Está basada en *OpenGL*, y es compatible con lenguajes como *HTML* y *JavaScript* que son comunes en el desarrollo web [12].

WebGL es compatible con gran parte de los navegadores web, en especial los más populares como *Google Chrome, Opera, Mozilla* y *Safari*, donde las compañias dueñas de estos son parte de *Khronos*

consortium's WebGL Working Group, por lo que se asegura su compatibilidad.

Las ventajas que presenta utilizar *WebGL* son:

- Al basarse en *OpenGL* se vuelve familiar con los lenguajes de programación de gráficas 3D.
- Compatibilidad multiplataforma y multinavegador
- Integración con el lenguaje *HTML*, permitiendo interacción en web.
- Aceleración por hardware del renderizado de gráficas 3D en web

2.5.3 Three JS

Es una biblioteca de *Javascript*, creada para facilitar lo más posible el renderizado de contenido 3D en una página web. Utiliza *WebGL* para renderizar, pero simplifica su utilización con funciones para generar las escenas, luces, sombras, materiales y texturas, sin tener que crearlas desde 0 en *WebGL*.

Es un proyecto de código abierto, que se encuentra disponible a través de *GitHub*. Por esto ha logra integrar muchas funciones y mantenerse compatible con navegadores web actuales.

2.5.4 HTML

HTML o *HyperText Markup Lenguage*, es un lenguaje que describe de manera semántica la forma y apariencia que debe poseer una página web, al generar una estructura y agregar marcas a cada sección, por esto se denomina lenguaje de marcado. Originalmente se diseñó para describir documentos científicos, pero con los años se extendió y modificó, ganando la capacidad de representar incluso aplicaciónes.

Su desarrollo se encuentra a cargo de WHATWG (*Web Hypertext Application Technology Working Group*), que busca generar avances en la web desarrollando estándares.

2.5.5 CSS

CSS o *Cascading Style Sheets*, es un lenguaje que describe el renderizado de documentos estructurados, como los creados por lenguajes de marcado. Se utiliza en conjunto a *HTML* para definir una

página web, donde *HTML* ordena el contenido y *CSS* genera el orden y apariencia de la página al momento de renderizarla. Su desarrollo y mantención se encuentra a cargo de *CSS Working Group*.

2.6 Uso de visualizaciones científicas

El estudio de la biología ha crecido de la mano de métodos de visualización, debido a la necesidad de observar y representar estructuras complejas para su análisis, enseñanza y discusión. La ilustración científica comenzó a mano, pero hoy en día los métodos computacionales son más eficientes y capaces de recrear células y tejidos.

En esta área de la ciencia, la ilustración comenzó con figuras sencillas como flechas, círculos y cilindros, representando células, conexiones y movimientos. Al pasar a *software*, se implementaron métodos similares, que pierden gran parte de la precisión respecto a la forma real de la estructura, pero se mantienen populares, donde la representación 2D estática sigue siendo la de más uso.

Se pueden diferenciar 2 métodos para el renderizado de estas imágenes: fotorrealista y no-fotorrealista, donde el primero incluye mayor cantidad de características como texturas, colores y se añaden distintas fuentes de luz para incluir sombreados. En cambio el renderizado no-fotorrealista toma mayores libertades artísticas, con la utilización de colores simples y sin mayor uso de texturas para destacar diferencias entre estructuras. Además, mantiene gran parte de la geometría, omitiendo detalles finos [13].

Al momento de utilizar estas ilustraciones para explicar procesos y estructuras se requiere de la opinión del usuario, en especial de la persona a cargo de entregar la información. Un estudio presenta guías a seguir para diseñar herramientas de aprendizaje, en específico en genética pero sigue dentro del área de la biología [14], estas son:

- Mantener un equilibrio entre lo artístico y la precisión.
- Crear materiales accesibles a una gran audiencia.
- Incluir la complejidad y el caos natural en el estílo estético.
- Considerar los principios del aprendizaje multimedia.
- Una animación debe considerar pausas naturales
- Incluir fuentes, asegurando la credibilidad del material

- Crear materiales como una herramienta completa y además una versión modular.
- Mostar más que decir
- Identificar problemas en base a trabajo colaborativo

Estas guías se generaron a partir de un taller realizado por profesores, utilizando animaciones moleculares. Se recolectaron datos desde la preparación de la clase hasta que fue completada [14].

3 Trabajos previos

Se investigó sobre trabajos en el área de visualización de estructuras de interés en distintas áreas de la biología. Muchos avances se han realizado en especial dentro de la neurosciencia, por lo que aplicando metodologías ya probadas se puede realizar la analogía al sistema que se quiere implementar. Gran parte de los trabajos coinciden en el uso de renderizadores como *WebGL* y en la importancia de asegurar un rendimiento óptimo al reducir la geometría de representaciones tridimensionales.

3.1 Journey to the centre of the cell

Software creado con el objetivo de mejorar el proceso de aprendizaje y visualización, utilizando modelos 3D en un ambiente de realidad virtual, ver figura 3.1. Su diseño se basó en 3 principios, crear representaciones artísticas de conocimiento científico, visualización directa de datos o un híbrido que tome datos de distintas fuentes [15].

Las imágenes utilizadas son un modelo de células de cáncer de mama. Estas fueron procesadas en cortes de 50 nanómetros. Una limitante encontrada corresponde al ruido asociado a las imágenes de microscopía con fluorescencia, complicando la segmentación correcta de imágenes, además de métodos para destacar la información importante de cada imagen.

Para eliminar el ruido se usó una combinación de imágenes de microscopía de barrido e imágenes de microscopía óptica de fluorescencia. La microscopía de barrido se utilizó para obtener los modelos, mientras que la información relevante de las estructuras se obtuvo desde microscopía óptica de fluorescencia. El *software 3Dmod* genera un mallado 3D desde la información previamente separada según estructura a segmentar. Se generan regiones de interés manuales cada 10 cortes y el resto es generado por interpolación lineal. Las mallas fueron importadas al *software Autodesk Maya*, donde se optimizó la topología y se crearon las texturas para ser importados en la plataforma de desarrollo *Unity* [3] para crear el sistema de interacción [15].

El diseño del *software* en realidad virtual consideró 2 áreas a explorar: una región externa de la célula y una interna, además de un espacio de entrada. Este último se creó para evitar una experiencia no grata en primeros usuarios de realidad virtual, donde la sobrecarga sensorial, la distinta disposición al mareo y el manejo de los controles puede generar complicaciones. Instrucciones e información es entregada mediante audio mientras se recorren las áreas diseñadas.

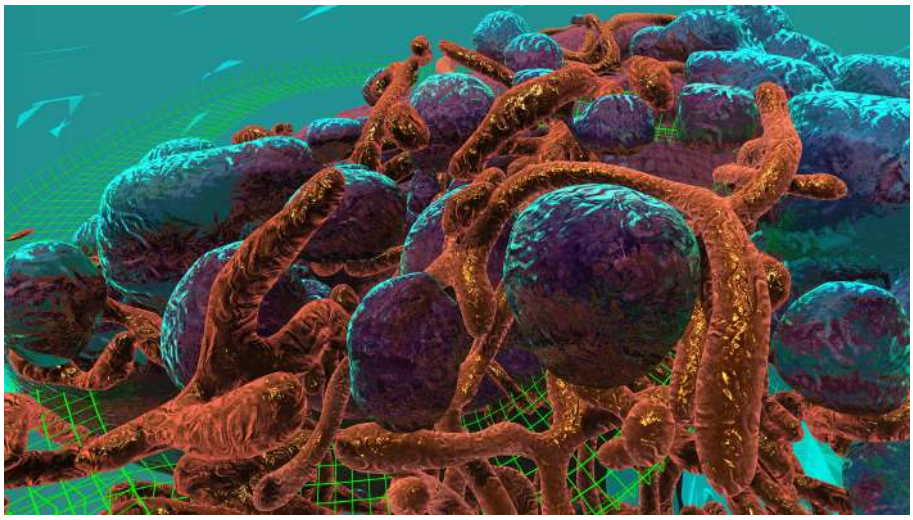

Fig. 3.1: Captura *ingame* del *software* "Journey to the centre of the cell", se observan estructuras segmentadas desde la visión del jugador en primera persona, con un casco de realidad virtual.

3.2 Nanoscape

Aplicación de escritorio o realidad virtual, que espera entregar una experiencia inmersiva de estudio, mejorando la compresión de la escala de estructuras celulares y sus interacciones, frente a los métodos de estudio tradicional con observación 2D de imágenes [16]. El *software* se enfoca en la visualización de estructuras proteícas en la membrana celular, con modelos generados en base a segmentación de la base de datos *RSCB Protein Data Bank* (PDB). Corresponden a imágenes de proteínas superficiales de células asociadas al cáncer de mama, ver figura 3.2. Incluye animaciones de la interacción entre las estructuras, creadas con *Autodesk Maya*, al generar un esqueleto interior a los modelos, utilizando un *plugin* específico *Molecular Maya*, este esqueleto permite interacciones físicas y movimiento. *Nanoscape* fue creado por el motor *Unity* [3], el cual ensambla y crea los espacios además de la interactividad con el usuario.

Debido a la alta cantidad de triángulos en los mallados 3D originales, se remodelaron las estructuras, tomando cierto nivel de libertad artística a modo de reducir la cantidad de polígonos presentes en los mallados. Así se mantiene en niveles estables la carga computacional, ya que la animación y ambiente posee una gran cantidad de modelos interactuando entre sí. Este proceso fue realizado en el *software Zbrush* especial para escultura y modelado en 3D [16].

Gran parte de las proteínas analizadas poseen formas no definidas, que normalmente son omitidas y se asume un modelo estándar para representar cada tipo de proteína. El *software* ,en cambio, considera una gran cantidad de variaciones encontradas en la base de datos utilizada, con animaciones creadas para cada tipo. Un aspecto de la célula no fue realizado como el objetivo original, la membrana. Dada

Fig. 3.2: Captura *ingame* del *software* "Nanoscape", se observan estructuras proteícas sobre la membrana celular, además de proteínas flotando en el exoplasma.

la alta cantidad de fosfolípidos y su constante movimiento la carga computacional sería extremadamente alta. Por ello se decidió generar una malla a la cual se aplicó una textura con la representación de movimiento constante de los fosfolípidos [16].

3.3 BrainBrowser: distributed, web-based neurological data visualization

BrainBrowser es una biblioteca de *Javascript* de código abierto, que permite utilizar herramientas de visualización 3D para neuroimagenología en la web. Permite manipular y analizar datos en tiempo real a través del navegador web, utilizando tecnologías como *WebGL* y *HTML*.

Las funciones se programaron de manera modular, integrando la utilización de *Three.js* para el manejo de *WebGL*. El objetivo es generar un sistema de buen rendimiento, que no requiera extensiones o software que instalar, ser dinámico e interactivo, que pueda adaptarse al uso de distintos laboratorios y que sea de código abierto[17].

Los archivos de mallados son cargados por las API nativas de *Javascript*, donde se cargan archivos en formatos comunes de neuromagenología como *OBJ, Freesurfer ASC, MNI OBJ* y *JSON*.

Cuando esta herramienta fue desarrollada *WebGL* poseía un límite de vértices a renderizar delimitado por 16 bits. Se utilizó la herramienta *WebWorkers* para lograr renderizar el listado completo de vértices de una malla. El límite correspondía a 65536 vértices, mientras que un tamaño común utilizado es de 560674 vértices. Luego la información se pasa por funciones de *Three.js* para preparar

el renderizado[17].

El sistema permite observar modelos de superficie y volumen con la función de aplicar mapas de colores que pueden modificarse a gusto por el usuario, además de una vista de *wireframe*, cambios de opacidad para aplicar transparencia en los mallados y movimientos de la cámara mediante el *mouse*[17].

3.4 Fiberweb: Diffusion Visualization and Processing in the Browser

Visualizador enfocado en imágenes de resonancia magnética de difusión, que incluye un sistema de procesamiento de datos en la web, que permite realizar tractografías directamente en el navegador.

Para la visualización utiliza sistemas basados en *WebGL*, incluyendo la biblioteca *Three.js* para simplificar el desarrollo del visualizador. En especial, asegura la mejora y mantención del sistema, ya que la comunidad de *Three.js* lo mantiene en constante actualización. Además, es un lenguaje de programación de menor nivel al usado en otros visualizadores para este tipo de datos, permitiendo mayor manipulación de los datos para asegurar un rendimiento más alto [18].

El renderizado permite ver las estructuras en 3D y 2D, además de poder seleccionar partes de la anatomía para aplicar el proceso de tractografía. Este genera en tiempo real *streamlines* que corresponden a fibras del cerebro, representadas por líneas generadas por vectores en 3D [18].

La cantidad de *streamlines* afecta el rendimiento en zonas de alta densidad, dada la cantidad de puntos a renderizar en el espacio, generando bajas en la tasa de cuadros por segundo.

3.5 Discusión

Se confirma la necesidad de incluir un sistema de optimización de los mallados 3D , dado que los trabajos analizados comparten esta característica para alcanzar un rendimiento aceptable. Además, coinciden en el uso de *software* de edición 3D para realizar estos procesos. Los trabajos dedicados al desarrollo web coinciden en el uso de renderizadores basados en *WebGL* y la utilización de *Javascript* junto a la biblioteca *Three.js* que facilita la programación de los elementos necesarios para renderizar los mallados 3D.

4 Edición de mallados 3D

4.1 Introducción

En base a lo observado en los trabajos previos de tejido hepático, se confirma la necesidad de modificar la geometría obtenida en los mallados 3D. El objetivo es reducir la cantidad de polígonos presentes en los modelos, pero mantener fidelidad a la estructura segmentada. Se utilizó el software de edición de código abierto *Blender* [19], ya que posee las herramientas necesarias, además de poder integrar *scripts* en Python. Además se usó el *software Meshlab*, que permite realizar el análisis comparativo entre los mallados. Finalmente, se creó una extensión para el *software Blender* que simplifica la aplicación del método de optimización, facilitando el uso futuro de la herramienta.

4.2 Análisis morfológico de los mallados

El objetivo del proceso de optimización considera la reducción de geometría para asegurar un rendimiento óptimo, siendo el objetivo principal alcanzar niveles estables de cuadros por segundo en el visualizador, pero manteniendo la forma original de las estructuras segmentadas. Este último punto requiere de aplicar métodos de comparación de mallados, con el fin de encontrar el nivel de diferencia entre ellos. Este proceso se utilizó para ajustar los procesos decimación de geometría.

4.2.1 Distancia de Hausdorff

Para comparar los mallados se utilizó la distancia de Hausdorff. Esta permite comparar puntos en el espacio, pero no solo basándose en su distancia euclidiana, la que solo permite saber qué tan lejos está un punto del otro, sino que considera una unión entre los puntos, generando un grupo cerrado. Al comparar 2 grupos cerrados de datos se aplica un *Delta* de expansión, donde la distancia de Hausdorff corresponde al máximo valor del mínimo *Delta* tal que la expansión del primer grupo de datos, contenga al segundo y al mismo tiempo, la expansión del segundo grupo de datos, contenga al primero. Ver ecuación 4.2.1. La aplicación en mallados 3D se realiza considerando los vértices como los grupos de puntos a comparar, ya que las líneas forman el grupo cerrado de datos, cumpliendo las condiciones para utilizar este método.

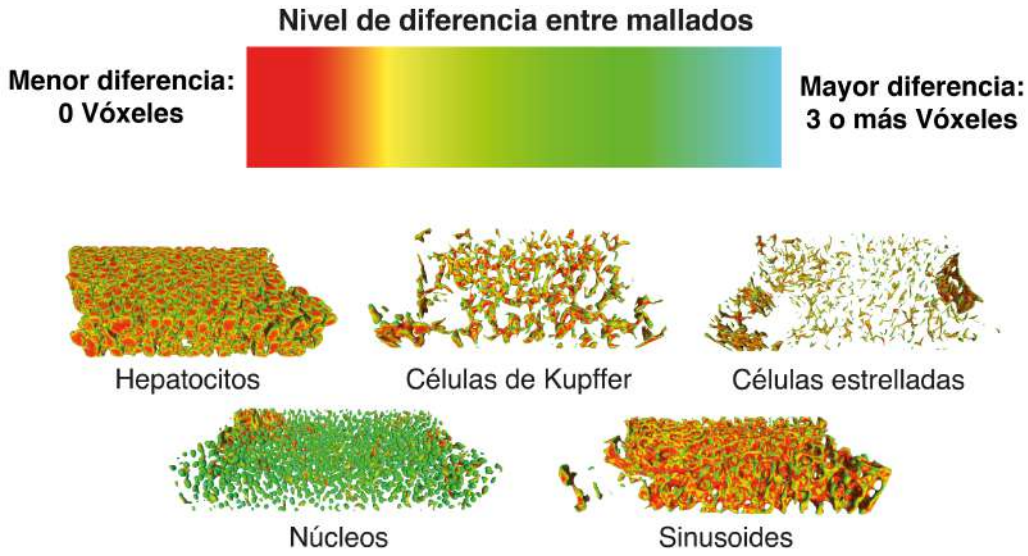

Fig. 4.1: Resultados obtenidos a partir del análisis por distancia de Hausdorff en el software Meshlab. El software permite aplicar un mapa de colores según los valores obtenidos, donde el rojo equivale a los errores más bajos y los tonos verde y azul a los errores más altos. Se observa como en todas las estructuras se mantiene el error cercano a los tonos amarillos y rojos, excepto en los núcleos.

$$d_H(X,Y) = max\{inf(d(X,Y), inf(d(Y,X))\} \tag{4.2.1}$$

4.2.2 Metro: Aplicación de la distancia de Hausdorff

La herramienta *Metro* permite evaluar 2 mallados triangulares a diferente nivel de detalle, osea, con diferente cantidad de triángulos que forman la misma estructura. Esta no necesita conocer el método utilizado para alterar el mallado, si no que directamente evalúa la diferencia entre las 2 mallas en base a un error de aproximación basado en el cálculo de la distancia de Hausdorff [20].

El algoritmo *Metro* se encuentra integrado en el *software Meshlab*. Este permite aplicar las funciones de manera más sencilla. Además acepta los formatos utilizados, facilitando su importación y análisis. Se utilizan los vértices del mallado a comparar como entrada para el *sampling*, este realiza el cálculo del error de aproximación y guarda un valor de calidad en cada vértice. Los valores de calidad permiten aplicar un mapa de color, dando un análisis visual de los resultados obtenidos [21]. Ver figura 4.1.

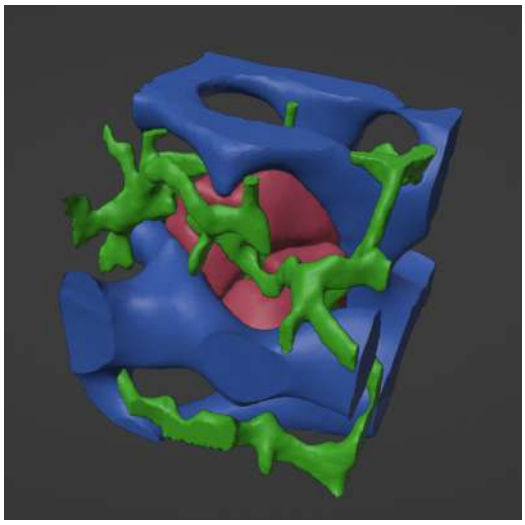

Fig. 4.2: Sección del tejido con los 2 hepatocitos seleccionados. Presenta la aplicación de un material simple de un color base, donde los hepatocitos están en rosado, sinusoides azul, canalículos biliares verde y núcleos en rojo.

4.3 Exploración inicial

Para explorar el uso de las herramientas de manera rápida se decidió trabajar con una sección pequeña de la segmentación. Esto permitió acelerar los procesamientos utilizados, dado que se poseía menor cantidad de datos. Se seleccionaron 2 células adyacentes, dado que se encontró un hepatocito de 1 núcleo junto a uno de 2 núcleos, que es una característica propia de este tipo de célula. Ver figura 4.3.

Dejando sólo visible la geometría de cada mallado se ajustó la posición de estos para evitar sobrelape de las estructuras. La segmentación posee alrededor de un 1 % de error con el que se produce un sobrelape entre mallas, pero en el proceso de importación al *software Blender* el mallado de los sinusoides alteró su posición original, por lo que esta visión de geometría permitió ajustar manualmente hasta quedar lo más cercano a su posición original. El sobrelape restante fue eliminado mediante operatorias booleanas entre los mallados, donde la zona de intersección fue eliminada de la malla de los hepatocitos. Esta decisión se basó en las adaptaciones que posee la membrana de los hepatocitos frente a las estructuras aledañas. Ver figura 4.2.

Se creó un material simple de un color para aplicar en la superficie del mallado, con el objetivo de mejorar la identificación de cada estructura. Este material se vuelve importable al *software Unity* al modificar el formato del mallado desde STL a FBX, que almacena información de texturas y materiales a diferencia del formato original. Ver figura 4.2.

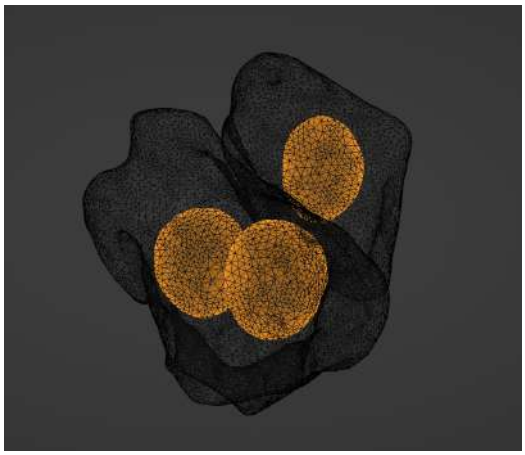

Fig. 4.3: Vista de la geometría de células seleccionadas. Se omite la textura para observar los núcleos segmentados al interior de las células. Se tiene un hepatocito mono nuclear y uno binuclear.

Del estudio del proceso de edición, remodelado y herramientas disponibles, se diseñó un procedimiento para alterar la geometría de las estructuras, con tal de mejorar su apariencia con la menor cantidad de polígonos posibles. Se comienza con la herramienta de subdivisión [22], que aumenta la cantidad de triángulos, esto para aplicar un suavizado inicial y preparar la malla para una futura reducción de geometría.

Se continúa con la herramienta de retopología [22], la cual modifica el polígono utilizado en la malla, se cambia de triángulos a quads, figuras de 4 vértices que permiten representar fácilmente geometrías tubulares. Ver figura 4.3. Se decidió el uso de quads dado el proceso de modelación de mallas con mínima cantidad de polígonos, donde se aprovecha la geometría rectangular para marcar bordes de estructuras y su facilidad de representar tubos. En esta geometría se aplica un segundo suavizado, donde luego se aplica la herramienta de decimación, que reduce la cantidad de polígonos en la malla [22]. Esto permitió mantener la forma original de la estructura segmentada, con una mejora estética en la superficie del mallado. Luego se vuelve a aplicar la herramienta de retopología para volver a geometría triangular y un último suavizado. Ver figura 4.6

4.4 Proceso de optimización de mallados 3D

Desde la experiencia inicial donde se trabajó con secciones pequeñas de los mallados, se diseñó un proceso de optimización pero aún sin analizar su efectividad ni precisión. Además, la utilización de una sección pequeña no permite observar el verdadero efecto al momento de cargar todo un tejido en el navegador web. Para integrar una métrica comparativa se utilizó el *software Meshlab*, el cual permite analizar la morfología de los mallados, entregando el nivel de diferencia entre estos. Además con un

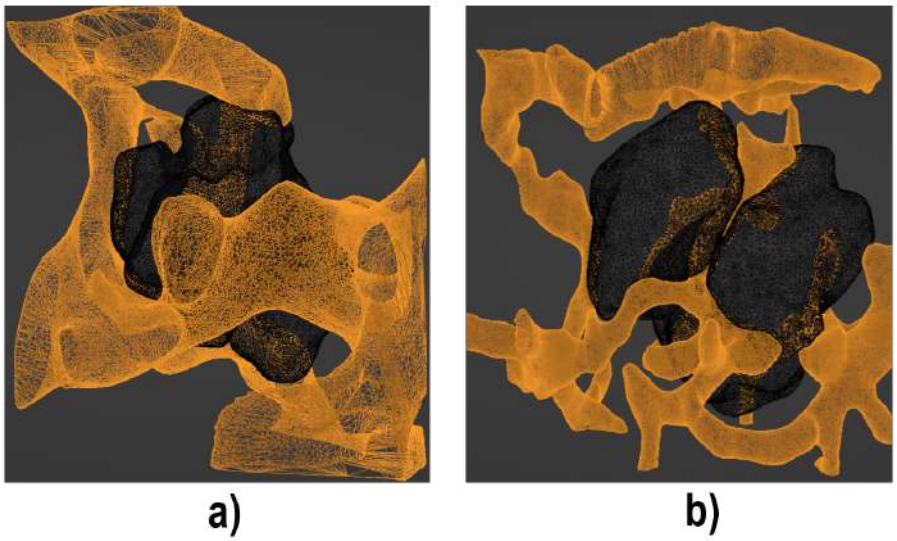

Fig. 4.4: a) Vista de la geometría de hepatocitos en negro, sinusoides en naranjo. La malla de los sinusoides se tuvo que ajustar a la posición mostrada para coincidir con los hepatocitos. b) Vista de la geometría de hepatocitos en negro, canalículos biliares en naranjo. se destaca la presencia de la zona apical de ambos hepatocitos en el centro, donde se genera un espacio para el paso de un canalículo biliar.

prototipo de visualizador web se pueden observar las métricas de consumo de memoria y cuadros por segundo que alcanza la visualización. Estos 2 factores fueron utilizados de manera de minimizar el error generado por la optimización, maximizando los cuadros por segundo obtenido y reduciendo el consumo de memoria RAM en comparación al mallado original, pero ahora con secciones completas del tejido analizado.

El diseño del proceso de optimización fue realizado de manera iterativa, donde cada paso realizado fue documentado, luego se exportó el mallado de cada sección del tejido, se analizó su diferencia morfológica frente al mallado original. Finalmente se cargó en un prototipo del visualizador web para ver su efectividad en la reducción de consumo de recursos. Ver figura 4.8.

La integración de la visualización web requirió ajustar la posición de los mallados en el espacio, para tener siempre una misma distancia hacia la cámara que renderiza el mallado posteriormente. Por ello se agregó inicialmente una función que centra la geometría en el origen y la alínea con los ejes X, Y y Z. Así, dentro del visualizador web solo se coloca la cámara apuntando al origen, permitiendo ver cualquier tejido renderizado.

La visualización también considera aspectos estéticos, por lo que para quitar imperfecciones de los mallados se considera un proceso de suavizado, el que altera la posición de los vértices, generando superficies más uniformes en la malla.

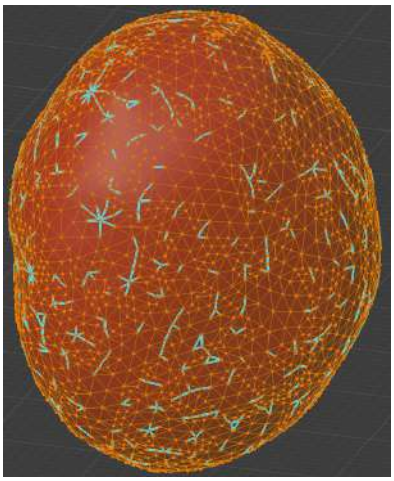

Fig. 4.5: Mallado 3D del núcleo posterior al proceso de edición. En azul se observan aristas de la geometría original, en naranjo, las aristas creadas por el proceso de retopología luego del decimado.

Al observar las estructuras del tejido, podemos ver la gran cantidad de canales presentes, donde sinusoides y canalículos biliares componen gran parte del tejido, y determinan la forma de secciones basales y apicales de los hepatocitos. Esto implica que mucha geometría depende de formas tubulares. Así se consideró agregar un proceso de retopología, dado que secciones cilíndricas son de fácil representación por geometría cuadrangular, y existen algoritmos de reducción de geometría específicos para este tipo de polígonos. Se consideró posible reducir la geometría de estas zonas, sin afectar mayormente su morfología al realizar este proceso, esto permite agregar otra capa de reducción de datos sin influir de gran manera en la forma del mallado. El proceso se completa con una decimación, que une vértices según una razón entregada al aplicar la función. Para obtener resultados óptimos de uso de memoria y reducción del tamaño del archivo generado, se aplicó una decimación que elimina un 70 % de la geometría. Este proceso genera polígonos de distinto número de caras, por lo que se vuelve necesario aplicar un proceso de retopologia triangular. La que permite un mejor renderizado. Finalmente, se aplica un suavizado, para mejorar la estética luego de tantas alteraciones sobre la malla. Ver figura 4.7.

La optimización general consiste de los siguientes pasos:

- Alineación con el origen: para asegurar la correcta visualización se debe mantener la cámara apuntando a la estructura, por ello se mantiene la misma posición para toda estructura optimizada, centrando el origen en el centro de la estructura.
- Cambiar el *Blender* a modo edición: permite acceder a las herramientas necesarias para el procesamiento.
- Suavizado: se altera la posición de los vértices, reduciendo puntas y extrusiones de la malla.

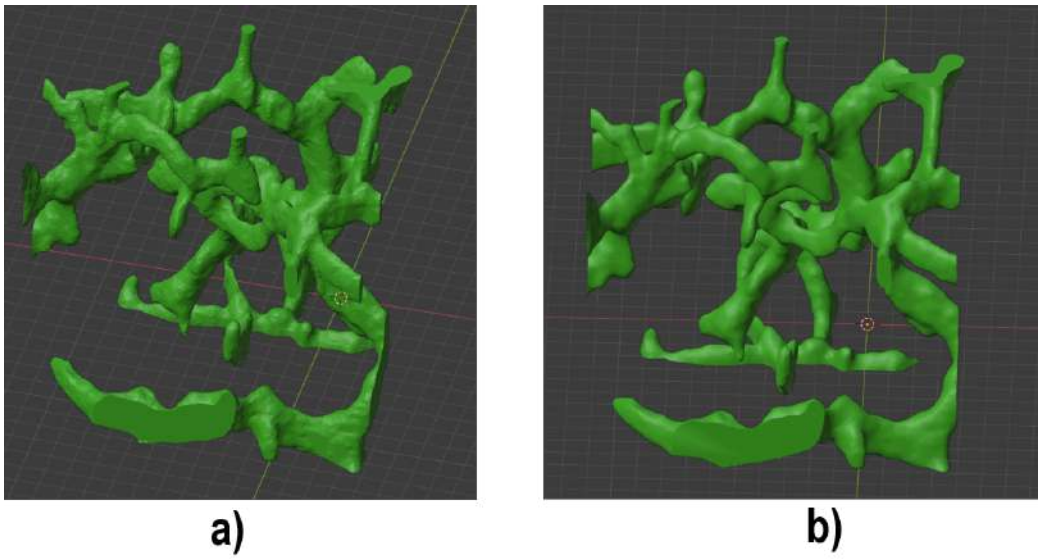

Fig. 4.6: a) Mallado de sección de canalículos biliares previo a proceso de edición. Se observa superficie rugosa con triángulos visibles. b) Mallado de sección de canalículos biliares luego del proceso de edición. La estructura mantiene su forma pero con una superficie limpia y geometría no visible.

- Retopología: Se modifica la geometría del mallado, cambiando de triangular a cuadrangular en varias secciones.
- Subdivisión inversa: algoritmo de reducción de geometría, basado en el sistema de subdivisión, por lo que solo se aplica en geometrías cuadrangulares.
- Decimación: algoritmo de reducción de geometría, colapsa aristas vecinas, funciona en base a cualquier tipo de geometría, pero puede generar triángulos, cuadriláteros o n-gons.
- Triangulación: proceso de retopología que genera una malla totalmente triangular.
- Suavizado final: se aplica un último suavizado para compensar las alteraciones que realizan los procesos de reducción de geometría.
- Cambiar el *Blender* a modo objeto: carga el mallado como un objeto completo, simplificando la interacción con la interfaz.

Una modificación importante fue la eliminación del proceso de subdivisión. Este se utilizó para aplicar suavizados que mejoren la estética del mallado, pero su aplicación aumentaba demasiado la cantidad de triángulos de la malla, por lo que se aplicaron suavizados sobre la malla sin alterar. Se mantuvo el proceso de retopología cuadrangular, usado ya que gran parte de las mallas tiene una forma tubular o se ve afectada por estructuras tubulares. Un cilindro es fácilmente representado por

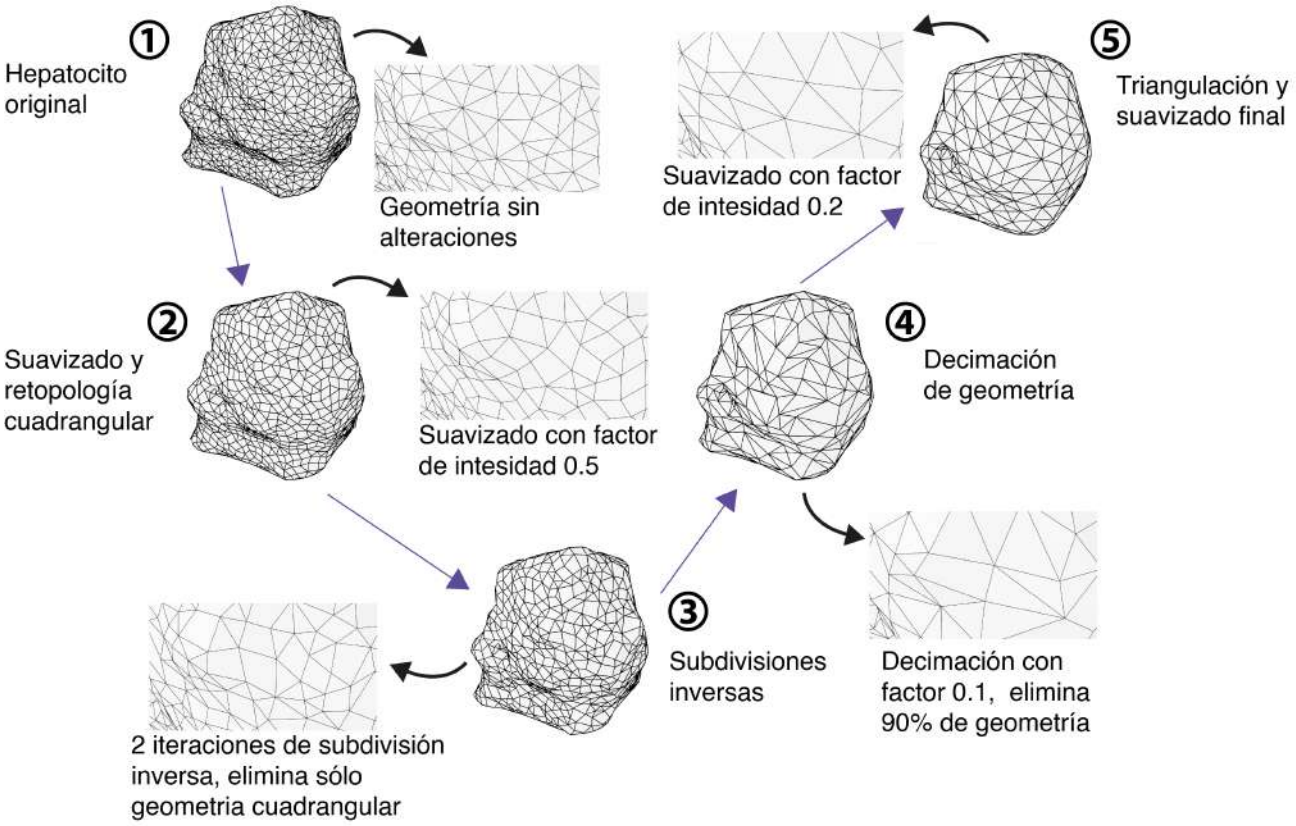

Fig. 4.7: Se describe el proceso de optimización de los mallados, agregando un acercamiento a la malla para ver observar mejor los cambios en la geometría. Siguiendo las lineas azules se ve el avance entre cada paso del optimizado, comenzando con el proceso de retopología cuadrangular, luego la subdivisión inversa, sigue una decimación y se completa con una retopología triangular y suavizado leve.

geometría cuadrangular, por lo que reducir la cantidad de polígonos que lo representa no debería afectar en gran manera su forma. Luego es necesario volver a geometría triangular, ya que estos son los polígonos más óptimos para los sistemas de renderizado.

El proceso fue diseñado bajo datos que no presentaban 2 estructuras: Células estrelladas y células de Kupffer. A pesar de esto, el proceso de optimización obtuvo resultados similares que con el resto de las estructuras, por lo que demostró una buena robustez. Ver figura 4.9.

4.5 Optimización de núcleos

La utilización del análisis morfológico permitió observar un problema al momento de optimizar los núcleos de las células. Estos poseían alrededor del doble del error que el resto de estructuras. Por ello, aplicando la misma metodología, se diseñó un proceso de optimización especifico para estas

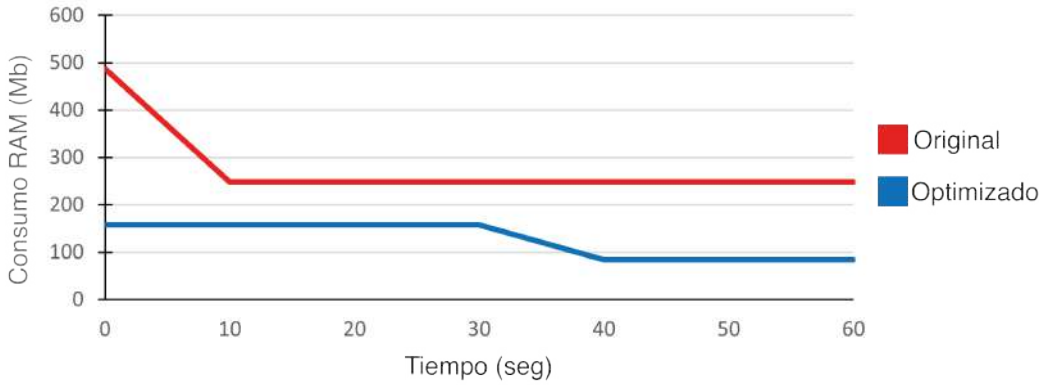

Fig. 4.8: Comparación del uso de memoria RAM. Se cargó el mallado de los hepatocitos de un sujeto en su versión original, indicado en rojo y luego en su versión optimizada indicada en azul. Se observa un alto consumo en el momento de la carga del archivo, el cual decae con el tiempo. El modelo optimizado reduce el nivel de consumo de memoria en todo momento.

estructuras.

Se eliminaron 2 pasos del proceso de optimización, en base a los resultados del análisis morfológico, el suavizado inicial de la estructura y el proceso de retopología cuadrangular. El mallado de los núcleos posee una alta cantidad de triángulos solo por la gran cantidad de núcleos presentes en el tejido. El tamaño de estos es pequeño, por lo que no se componen de demasiada geometría. Al suavizar estas estructuras con tan pocos triángulos se pierde mucho volumen, que genera una mayor distancia entre la malla original y la optimizada, aumentando el nivel de error. El segundo proceso fue aplicado específicamente para estructuras tubulares o afectadas por estas y los núcleos no coinciden con ninguna de las 2 descripciones. La geometría esférica es mal representada por geometría cuadrangular, en especial cuando posee pocos polígonos o se reduce la cantidad de polígonos que la componen. Eliminar este factor de la optimización fue la mayor reducción del error obtenido por el proceso original, ver figura 4.9.

4.6 Creación de extensión para Blender

El proceso de edición de los mallados requiere de experiencia con el *software*, donde se deben manejar las herramientas utilizadas, métodos de importación y exportación, ajustes de variables y la misma navegación dentro de *Blender*. Por esto se creó una extensión, la cual permite realizar los procesamientos de manera automática, necesitando la menor cantidad de clicks posibles del usuario.

Blender, al ser un *software* de código abierto posee herramientas para la creación de *scripts*, los

	Hepatocitos		Células estrelladas		Células de Kupffer	
Optimización	Ligera	Intensa	Ligera	Intensa	Ligera	Intensa
Número de triángulos	729591	233021	92617	29499	117160	38044
Tamaño de archivo (mb)	35.6	11.4	4.5	1.4	5.7	1.8
Tasa de cuadros por segundo	40	60	60	60	60	60
Vóxeles de error promedio	0.279574	0.435242	0.347339	0.535522	0.251062	0.378912

	Sinusoides		Venas		Núcleos	
Optimización	Ligera	Intensa	Ligera	Intensa	Ligera	Específica
Número de triángulos	321025	103589	15714	4408	206256	206256
Tamaño de archivo (mb)	15.6	5	0.7	0.2	10	10
Tasa de cuadros por segundo	45	60	60	60	60	60
Vóxeles de error promedio	0.188659	0.288395	0.351846	0.457790	0.546795	0.215123

Fig. 4.9: Comparación entre modelos de optimización creados. Se observa cómo la optimización intensa genera archivos de menor tamaño y mayores tasas de cuadros por segundo que la ligera, pero con un error aún menor a 1 vóxel. Frente a los mallados originales, la optimización ligera reduce el tamaño del archivo en un 70 % en cambio la intensa permite reducir hasta un 95 %. La optimización de núcleos se modificó y se observa una mejora en el error que genera en las mallas de núcleos.

cuales pueden ser integrados a una extensión facilitando su ejecución. El *software* se basa en el lenguaje de programación *Python*, con bibliotecas propias para la ejecución de acciones dentro del programa [22].

Al tener un proceso definido, se realizó un *script* que aplique el método diseñado. Además, se consideraron opiniones de usuarios del laboratorio, para integrar más funciones que faciliten su experiencia.

Se creó un panel, el cual posee las siguientes herramientas:

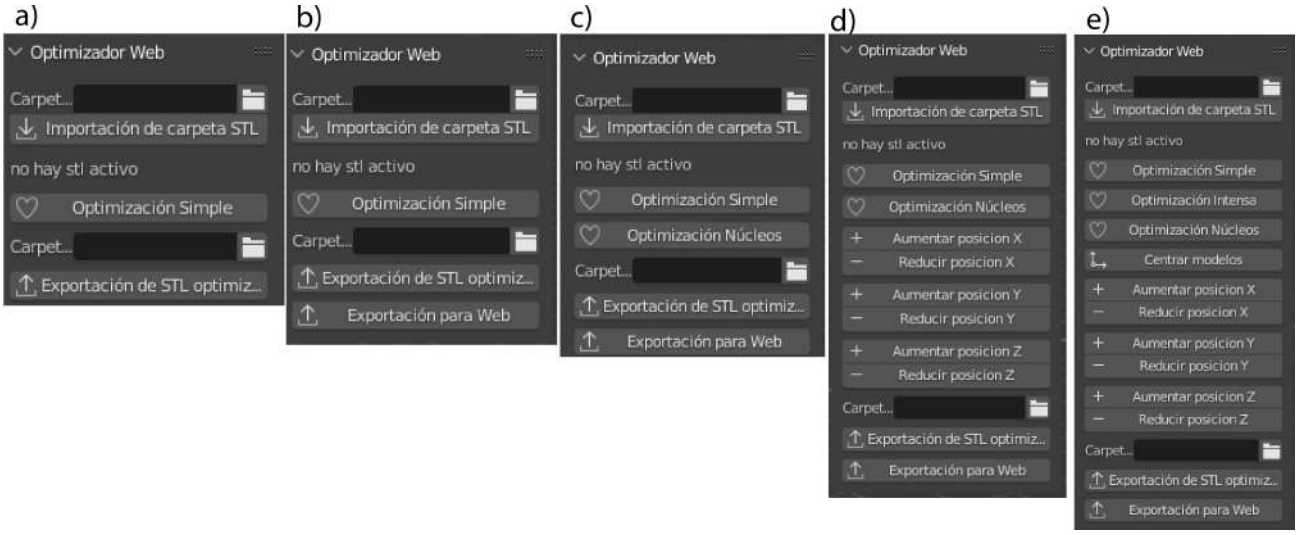

Fig. 4.10: Capturas de la evolución del menú de la extensión para el software Blender. a) Versión inicial, sólo exportado en formato STL y proceso único de optimizado. b) Segunda iteración, integrado de exportado en formato GLTF. c) Tercera iteración, integrado de sistema de optimización para núcleos. d) Cuarta iteración, cambio de nombre en botones para aclarar su uso, integración de herramientas de alineación manual de estructuras. e) Versión final, integración de botón de centrado automático y proceso de optimización intensa.

- Selección de carpeta de importación, dado que los modelos del tejido siempre son almacenados dentro de una carpeta con el nombre del sujeto.
- Botón de importación, se simplifica el proceso de importación al crear un comando que importa todos los archivos del formato correspondiente.
- Botón de recarga, en caso de encontrar errores, o querer volver a importar el modelo original, se integró la función de reimportar el modelo seleccionado en el *software*.
- Botón de optimización web, este procesa todos los mallados seleccionados, generando su versión de menor geometría en solo un click.
- Botón de optimización de núcleos, este posee un proceso de optimización específico para núcleos, ya que el proceso general produce un error mayor en estas estructuras.
- Botones de alineación, se integró esta función para ajustar errores en procesos de alineación que sucede en algunos mallados.
- Selección de carpeta de exportación.
- Botón de exportación en STL, se mantiene una opción para exportar en el formato original, debido a que genera archivos de menor tamaño.

- Botón de exportación web, se generan archivos en formato GLTF, este es necesario para su posterior visualización web.

La programación fue realizada de manera modular, con el fin de poder integrar nuevas funciones fácilmente en caso de ser necesario. Esto ocurrió con el formato GLTF, el cual fue integrado posteriormente a la exportación STL. Este fue necesario al momento de comenzar a diseñar el visualizador web. También sucedió con la optimización de núcleos, que fue diseñada y añadida luego de integrar el análisis morfológico.

4.7 Prueba de concepto: Aplicación móvil de realidad aumentada

Para comprobar los límites de la plataforma de desarrollo *Unity* [3], se diseñó una aplicación móvil de realidad aumentada, que despliega el mallado de los hepatocitos. Se utilizaron las herramientas:

- *AR Foundation*, para utilizar *scripts* en lenguaje C++ para crear la interacción entre cámara y modelos a visualizar.
- *ARCore XR Plugin*, sistema que permite la funcionalidad de realidad aumentada en sistemas operativos *Android*. Necesario para crear la aplicación móvil.
- *XR Plugin Management*, que mejora la interacción entre las herramientas dedicadas a realidad aumentada y virtual. Permite asegurar compatibilidad entre los *plugin* utilizados.

Desde *AR Foundation* se utilizó su cámara AR junto al sistema de *Raycast Manager*, el cual, definiendo un origen y dirección, genera un rayo. Luego se detecta un impacto con superficies de este rayo para realizar la proyección de un modelo 3D. Se creó un *script* que utiliza esta herramienta, donde el punto de impacto del rayo es representado por un cursor, y sobre el cual se proyecta el mallado de los hepatocitos apenas se confirma la presencia de un plano horizontal. Ver figura 4.12.

La aplicación requiere compatibilidad específica con *ARCore*, lo cual limita la cantidad de dispositivos disponibles para su instalación. Esto no depende de la versión de sistema operativo *Android* sino de componentes presentes en la electrónica del equipo.

Unity permite importar el formato FBX, por lo que los materiales generados se reconocen por el *software*. Se agregó una animación simple, la cual rota el mallado alrededor del eje Z, para observar de mejor manera la malla. Este proceso requiere la integración de un controlador de animación, que controla la posición del mallado en el tiempo. Ver figura 4.13.

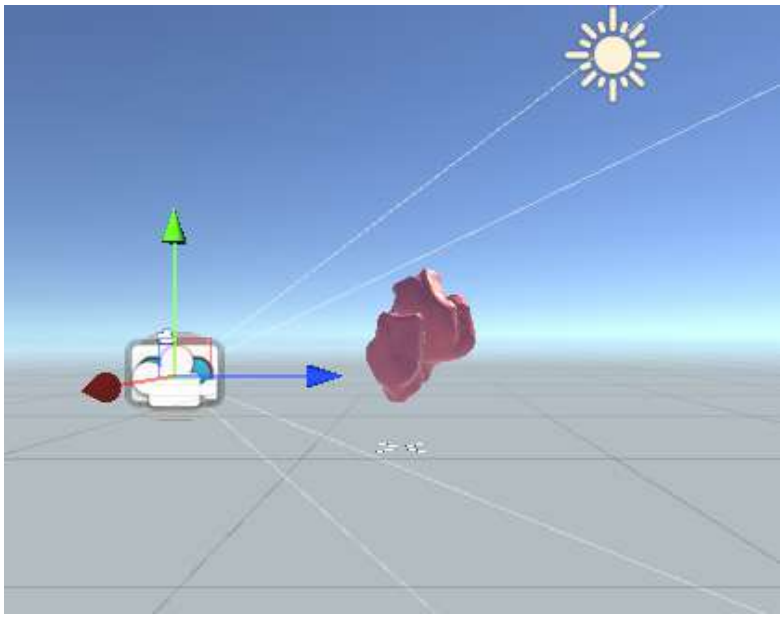

Fig. 4.11: Captura del *software Unity*, para prueba de realidad aumentada. Se observan los límites de visión de la cámara, con el mallado de los hepatocitos en el centro del área de visión.

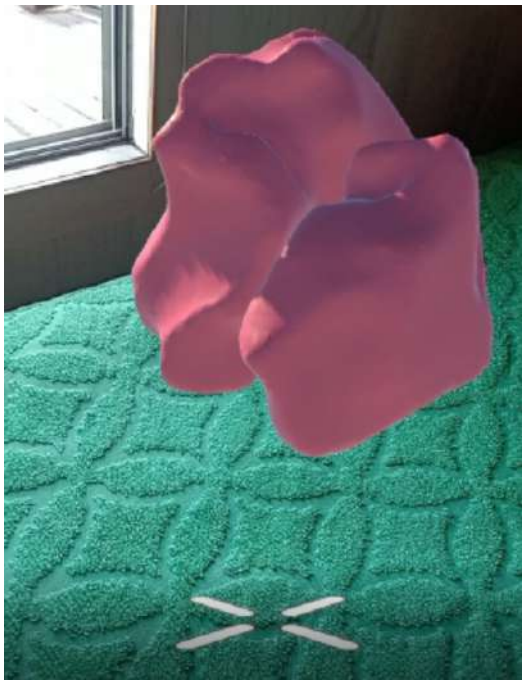

Fig. 4.12: Captura de aplicación móvil de realidad aumentada en funcionamiento. Se comprueba el funcionamiento de la aplicación, donde en el fondo se observa una superficie verde y una ventana, a la que apunta la cámara del celular, sobre la que proyecta el modelo de los hepatocitos.

Fig. 4.13: Captura del modelo del hepatocito en el menú del *software Unity*. Los componentes desplegados a la izquierda muestran el material importado correctamente desde *Blender*, la geometría del mallado y finalmente el componente que controla la animación.

5 Desarrollo de visualizador web

5.1 Introducción

Parte de los objetivos es generar herramientas de fácil acceso, por lo que se decidió concentrar el trabajo en el desarrollo de un visualizador web. Dado que aplicaciones móviles o realidad virtual generan grandes barreras económicas debido al equipo necesario para su uso. En cambio, un navegador web es compatible con cualquier dispositivo móvil o de escritorio, requiriendo solo de conexión a internet. Para generar este tipo de herramientas se deben manejar los lenguajes de programación centrales del desarrollo web *HTML, CSS* y *Javascript*, además de utilizar los sistemas de renderizado de modelos 3D, como *WebGL*.

5.2 Visualizador web de tejido hepático

El desarrollo del visualizador requiere de programación en los 3 lenguajes bases de una página web *HTML, CSS* y *Javascript*. Se utilizó el editor de código *Visual Studio Code*, ya que permite mantener organizados los archivos, además de presentar una extensión que permite generar un servidor local para la ejecución del visualizador, ahorrando la necesidad de un *host* para los prototipos de este.

El trabajo con *WebGL* se simplificó mediante el uso de la biblioteca de *Javascript Three.js*, permitiendo integrar los componentes necesarios en menos líneas de código. Así se puede concentrar el trabajo en la integración de interactividad y herramientas necesarias para mejorar la experiencia con el visualizador.

Junto al equipo de trabajo del laboratorio se definieron funciones imprescindibles para el visualizador web:

- Presentar datos relevantes de las estructuras visualizadas.
- Alterar el color de cada mallado a preferencia del usuario.
- Cambiar visibilidad de objetos en pantalla, para aislar estructuras.
- Cargar varios mallados de distintos sujetos.

- Cambiar los ángulos de visión de los mallados.

Se trabajó en base a 3 archivos con distinto lenguaje de programación:

- HTML: Archivo principal, aquí se definen los componentes de la web, su nombre, bibliotecas utilizadas y se colocaran los datos de cada estructura. Además esta coordina los archivos de código.
- *Javascript*: Define las funcionalidades de cada sección, carga los archivos y renderiza los mallados.
- CSS: Define la forma y posición de los elementos en la web. Además se aprovecharon características que definen la posición para realizar acciones dentro de la web.

Durante el desarrollo del visualizador se generaron varios prototipos, los cuales integran gradualmente las funciones indicadas. De esta manera se puede comprobar y recibir *feedback* de cada característica integrada. Esto fue necesario ya que la herramienta debe ser utilizable y actualizable por personas con poca experiencia en programación.

5.3 Primer prototipo

Se comenzó con *HTML*, donde se declaran las estructuras del visualizador. En este archivo se importan las herramientas a utilizar, como *Three.js, stats.js* que permite observar el rendimiento de la aplicación, ver figura 5.1, *GLTFLoader.js*, con la cual se importan los mallados, *OrbitControls.js*, que permite interactuar con la cámara. Esta última permite cumplir una de las funciones solicitadas por el laboratorio.

Utilizar bibliotecas requiere alterar sus archivos originales, ya que requieren de modificar una ruta. Para asegurar la compatibilidad en caso de actualizaciones se descargaron las bibliotecas. La otra opción es trabajar directamente desde los archivos en la web, pero cabe la posibilidad de que con actualizaciones se altere el funcionamiento de las funciones utilizadas y el visualizador requiera modificarse. Al adjuntar las bibliotecas en la carpeta que almacena el código del visualizador, se asegura utilizar siempre la misma versión, evitando problemas de compatibilidad a futuro.

Se decidió generar una barra lateral con botones para el control del visualizador, con una sección para elegir los modelos desplegados y luego integrar las herramientas como cambio de color del mallado, nivel de transparencia o cambiar de mallado desplegado. El resto del espacio es considerado

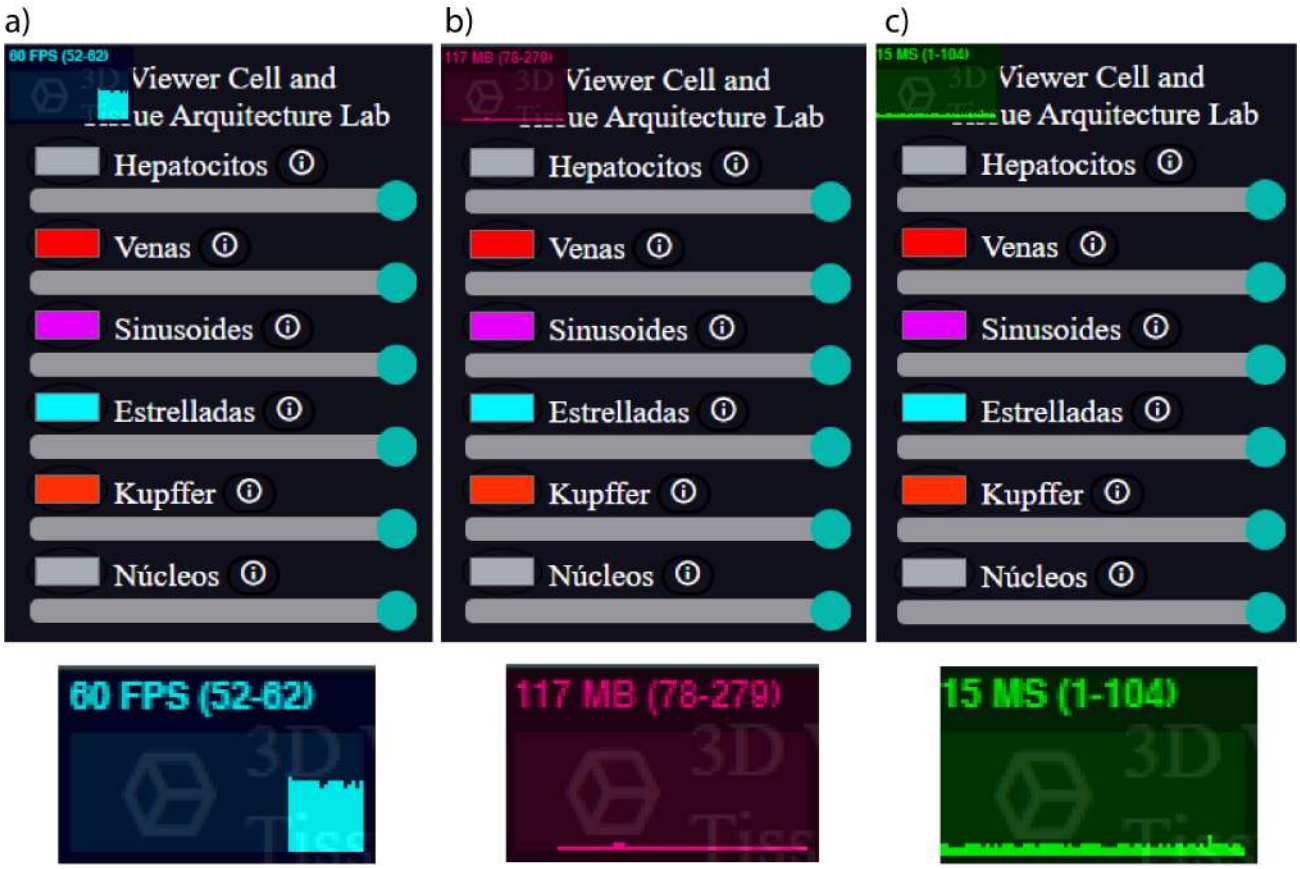

Fig. 5.1: Capturas del visualizador web utilizando stats.js. En la esquina superior izquierda del navegador se encuentra la información entregada por esta biblioteca, permite observar en a) la tasa de cuadros por segundo (FPS *frames per second*), en b) la memoria RAM utilizada y en c) los tiempos de respuesta del visualizador en milisegundos.

como *canvas* donde se renderiza el mallado a visualizar. Los botones y funciones fueron definidos pero sin utilidad todavía. Ver figura 5.2 sección a).

EL siguiente proceso corresponde a trabajar con *CSS*, donde se ajustaron los tamaños de fuentes, estilos de botones, generando cambios de color al momento de acercar el *mouse* a estos. Así se destaca que poseen una función al ser presionados, haciendo más intuitivo el uso del visualizador. Para esto se utilizan los ID o tags asignados en HTML, así cada elemento es modificado por separado o toda una clase de elementos pueden recibir ciertas características.

Se decidió mantener un fondo blanco en el área de renderizado, de manera que haga contraste con los mallados y se puedan aplicar sombras para facilitar la diferencia entre estructuras. En cambio, se tomó una tonalidad azul oscuro y letras blancas para la barra lateral.

Finalmente se escribió el código en *Javascript*, que define las funciones de la página web. Se

comenzó con integrar el sistema de carga de los mallados 3D. Para esto fue cargada la biblioteca *GLTFLoader.js* en HTML, las funciones que integra permiten importar los mallados previamente optimizados en el software Blender, utilizando el formato GLTF que acelera los procesos de carga.

Luego se creó una cámara, esta requiere indicar una posición y orientación en el espacio. Se le inicia apuntando al origen, dado que el proceso de optimización centra los mallados en el origen del espacio. Además, se cargaron modelos de gran tamaño para conseguir una distancia promedio a la que colocar la cámara. Así, cualquier objeto cargado se encuentra en el área de visión de la cámara por defecto.

Se declara el renderizador basado en *WebGL*, donde el tamaño del *canvas* se generó de manera de poder ser modificado. Los cambios en el tamaño de ventana fueron considerados al crear una función que los detecta y altera el tamaño del renderizador. Así se evitan problemas al cambiar de resolución, o alterar el tamaño de la ventana del navegador web, donde el visualizador siempre se ajustará al tamaño de esta. Se asignaron distancias de renderizado, esto indica que a cierto valor de distancia que toma la cámara, se deja de observar el mallado o parte de este.

El espacio donde se renderiza la imagen requiere de luces y sombras, por lo que es necesario integrar variables de iluminación para observar los modelos. Sin declarar un sistema de iluminación todo el mallado se observa en negro, a pesar de poseer una textura o color asignado.

Se probaron 2 métodos para generar la iluminación:

El primero declara luces direccionales en cada eje del espacio y generando una luz hemisférica o tipo sol en una posición del espacio. Esto tiende a problemas debido a la distancia desde el origen al cual colocarla. Al poseer objetos de distinto tamaño, se complica el proceso de decidir una distancia correcta para la iluminación, especialmente al declarar varios puntos de luz.

Se obtuvieron mejores resultados con el segundo método, usando luces hemisféricas o luces tipo sol, las que iluminan gran parte del espacio, por lo que a pesar de colocarlas a gran distancia del mallado, lo logran iluminar por completo, además poseen la capacidad de generar sombras, mejorando la visualización de los mallados.

Se integró la funcionalidad de ajustar la posición de la cámara con entradas generadas por el *mouse*. Esto fue aplicado con la biblioteca previamente cargada en HTML *OrbitControls.js*. Se puede realizar un cambio en el acercamiento al girar la rueda del *mouse*, también cambiar la orientación en que se observa el mallado al presionar la rueda, y al realizar un click sobre el *canvas*, se puede cambiar la posición de la cámara. Este sistema es intuitivo, ya que posee asignaciones de teclas presentes en

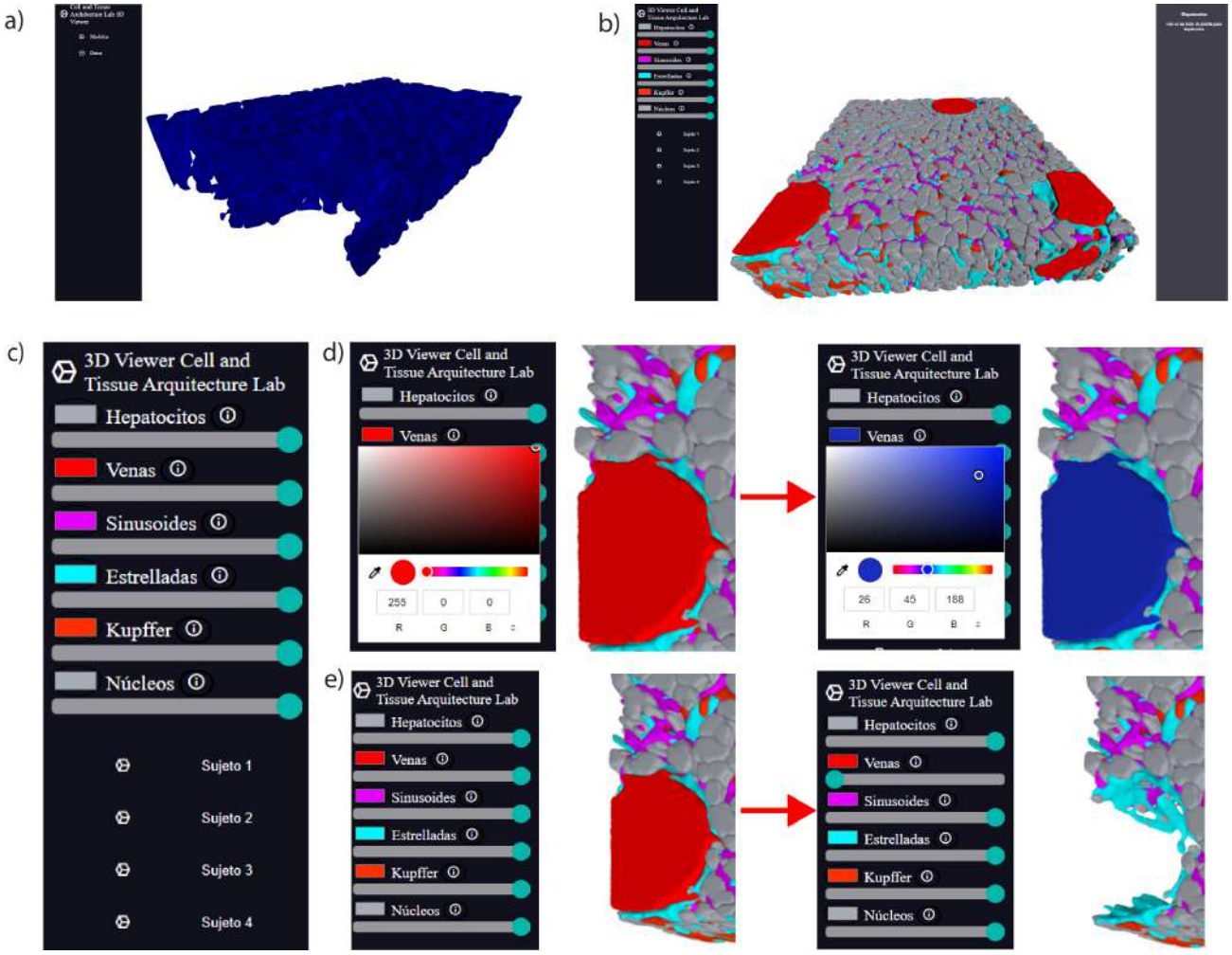

Fig. 5.2: Capturas de funciones del visualizador web. a) Imagen del primer prototipo del visualizador, el cual es capaz de cargar mallados pero no alterar sus colores ni transparencia. Tampoco tienen funcionalidad los botones de la barra lateral. b) Imagen del tercer prototipo, con botones funcionales, un mallado con material aplicado y el despliegue de información de la estructura. c) Imagen de la barra lateral, se observan los distintos botones generados para controlar color, transparencia, modelo visualizado y datos de la estructura. d) Ejemplo del botón de cambio de color. Se modifica el color por defecto de las venas, de un tono rojo a azul, mediante el menú desplegable que genera el botón. e) Ejemplo de la barra deslizante de transparencia. Se modifica la transparencia de las venas al mover la barra, el efecto puede ser gradual, pero se observa en sus extremos, con las venas visibles y luego invisibles al no ser renderizadas.

distintos tipos de visualizaciones.

Con esto se cumple el objetivo de poder modificar los ángulos de visión de los mallados y se preparan las bases para generar las funciones restantes. Además, esta versión se utilizó para conseguir la tasa de cuadros por segundo obtenidos por las diferentes optimizaciones. Ver figura 4.9.

5.4 Segundo prototipo

Con los botones ya definidos en HTML durante el primer prototipo, se les integró funcionalidad a través de *Javacript*. Así, al presionar uno de estos botones se elimina el mallado renderizado y se carga el correspondiente al botón. Este proceso generó problemas, se observó que el uso de memoria aumentaba cada vez que se realizaba un cambio de mallado. Se requirió mejorar la función de carga de mallados, ahora con la capacidad de limpiar por completo la memoria al momento de cargar otro modelo. Este proceso no viene por defecto en *Three.js*, ya que hay casos donde es conveniente mantener modelos precargados. En el visualizador, mantener otros modelos precargados genera un mayor consumo de memoria, por lo que se integró la funcionalidad de eliminar la malla anterior por completo. Esto requiere de mayor tiempo de carga al cambiar de mallado, pero menor consumo de memoria, asegurando la tasa de cuadros por segundo estable obtenida por la optimización de los mallados.

Inicialmente las texturas y material eran aplicados dentro del software *Blender*, pero debido a que parte del objetivo requiere automatizar los procesos con pocas entradas del usuario, se decidió generar los materiales y texturas directamente en *Three.js*. En este caso se utilizó un material tipo *Lambert*, este permite proyectar sombras pero no brillos sobre el material. Esto lo hace más eficiente que materiales como el *Standard* o *Phong*, que son reactivos frente a la cantidad de luz y por lo tanto consumen mayor cantidad de recursos.

Además, definir el material en *Javascript* permite simplificar el proceso de optimización en Blender. Generar un material, textura y color de manera automatizada se vuelve complicado para el usuario, donde se deben recibir entradas para el tipo de material, color y mallado en cual aplicar.

En cambio, *Three.js* posee funciones para generar sus propios mallados simples, materiales y texturas. Estos materiales pueden aplicarse directamente sobre los mallados cargados. Se crearon funciones que en base al orden de carga de los archivos, el cual es estandarizado desde su exportación en *Blender*, asigna los materiales y colores determinados para cada tipo de estructura.

5.5 Tercer prototipo

El sistema de generación de material y textura en el prototipo anterior fue la base para generar nuevas herramientas, ya que al definirse dentro de HTML y *Javascript*, es posible alterar sus variables en base a entradas del usuario. Así se integraron 2 herramientas necesarias para el visualizador: Cambio de color y cambio de transparencia de los mallados. Ver figura 5.2.

Al crear los materiales se les asignó la característica de transparencia. Así es posible modificar esta variable en otra función. El dato que se extrae proviene del archivo HTML, ya que se crearon barras deslizantes para cada estructura. Así se extrae el dato de transparencia desde HTML y se modifica en *Javascript*. Al ejecutar esta función en el ciclo de animación, se observan estos cambios de manera instantánea. Al usarse una barra se puede alterar la transparencia de manera gradual, permitiendo una experiencia más personalizada, donde se pueden observar estructuras a través de otras, en vez de un botón que directamente deje de renderizar uno de los mallados.

El color de cada estructura fue definido por el equipo de trabajo del laboratorio. Para coincidir con las representaciones utilizadas en investigación, se genera una lista de colores por defecto, correspondiente a las definidas por el laboratorio, pero se integró la función para ser modificada, de la misma manera que la transparencia. Para esto se utiliza HTML que posee un sistema de entrada de color donde el usuario puede seleccionar cualquier color del espectro. Ver figura 5.2.

Todas las funciones fueron creadas modularmente, para que replicarlas o agregar nuevas estructuras sea sencillo para usuarios futuros, donde solo se debe copiar una nueva función y luego alterar su nombre y archivos de origen. Estas consideraciones de diseño se aplicaron para que el visualizador pueda ser mantenido y actualizado por miembros del laboratorio. Así usuarios sin conocimientos de informática puedan a través de instrucciones simples, integrar nuevos mallados o funciones al visualizador.

5.6 Versión final

Al ya cumplir con las características esenciales, se realizaron cambios estéticos y ajustes de funcionalidad para mejorar la experiencia del usuario. Para esto se incluyeron íconos dentro de los botones, ya que normalmente se asocia el símbolo con la función.

Se realizaron cambios en CSS, donde inicialmente se configuraron los tamaños de los botones, barras y datos en base a píxeles. Esto genera problemas al visitar el visualizador en equipos de distinta resolución, ya que al poseer más píxeles en pantalla o menos, la escala del visualizador sigue siendo constante, viéndose más pequeño de lo diseñado o más grande. Por ello, se utilizaron otras unidades métricas disponibles en CSS como *rem*, la que se ajusta según el tamaño de fuente de la página, *vh* y *vw* que se ajustan según el tamaño de la ventana. Así, secciones como la de datos se mantienen ajustadas según el texto ingresado, y secciones como las barras y botones se ajustan a la pantalla.

Esta versión fue subida a *GitHub*, la que permite tener la página funcional en internet, solo con la

condición de no utilizar archivos de un tamaño mayor a 100MB. Al utilizar la optimización intensa diseñada en *Blender*, hasta los archivos de mayor tamaño alcanzan valores menores a 100MB, por lo que se puede mantener la página en línea sin problemas.

5.7 Consideraciones de diseño para uso futuro

Al igual que la automatización del sistema de optimizado de los mallados, también se consideran métodos para simplificar el uso del visualizador web para usuarios sin experiencia. Esto para que el laboratorio pueda seguir utilizando la herramienta para mostrar sus resultados.

Para esto se programó de manera modular, generando funciones que cargan los archivos directamente con interacción en la web, en vez de por programado manual. Para agregar nuevos mallados solo se debe generar una nueva lista de rutas de archivos, y copiar la plantilla agregada para generar un nuevo botón, ya que las funciones y forma se asocian a los tags generados en HTML. También la carga de archivos se genera al presionar botones, por lo que al copiar la plantilla se generan todas las variables necesarias para tener funcional un nuevo mallado, solo cambiando nombres y rutas de archivo, en vez de requerir que el usuario deba programar.

6 Conclusiones

6.1 Discusión

En la literatura analizada, las visualizaciones coinciden en el uso del *software Unity* para desarrollar aplicaciones. También en el uso de *Three.js* para el desarrollo web. Se requiere de edición de los mallados para reducir la cantidad de polígonos y optimizar el uso de recursos del visualizador. Se comprueba la necesidad de tomar libertades artísticas dentro del proceso, controlando el tamaño de las estructuras a presentar, limitando el ángulo de visión o aplicando colores no realistas con el fin de destacar zonas. Se consideró este aspecto en la utilización de colores con alto contraste entre estructuras, además de suavizar la superficie de la malla para darle un acabado con mejor estética.

El *software Fiji* y *Motion Tracking* presentan alta precisión en la segmentación de estructuras, pero una gran curva de aprendizaje. El uso de segmentaciones extraídas por expertos en el desarrollo de la aplicación en realidad aumentada permitió comprobar limitantes de las plataformas de desarrollo en base a los mallados objetivo, dado que la segmentación propia no alcanza la precisión que posee un usuario experto. Así, se continuó trabajando para el visualizador web, donde, utilizando datos y mallados segmentados por usuarios avanzados, se obtienen mejores representaciones y datos más precisos. Luego, el proceso de optimización permite que estos modelos puedan manejarse dentro del navegador web a tasas de cuadros por segundo estables y con un consumo de memoria RAM aceptable para equipos de bajo rendimiento.

Los sistemas de renderizado utilizados permiten mantener niveles bajos de uso de memoria RAM, en especial la integración de funciones de eliminación de información no usada, evitando acumulación de memoria al utilizar la herramienta web. Así, el visualizador permite una experiencia fluida al usuario.

Cumplir con las funciones que se decidieron esenciales permite una experiencia interactiva con el visualizador, donde el usuario tiene control sobre los mallados, su tamaño, ángulo de visión, color y transparencia.

La segmentación utilizada corresponde a un gran avance científico, ya que permite extraer información del tejido invisible a los métodos antiguos. La creación de este visualizador genera un método interactivo que incluye estos datos relevantes de los tejidos lo que permitiría mejorar el proceso de

aprendizaje en histología. Además, le entrega otra utilidad a los datos generados por investigaciones y en base a una herramienta de fácil acceso, reduciendo las barreras de entrada al conocimiento científico.

6.2 Conclusiones

Se desarrolló un visualizador web para mallados 3D de tejido hepático, obtenidos desde imágenes de microscopía mediante el *software Motion Tracking*. El trabajo requirió de incluir un proceso de optimización de los mallados 3D, el cual se automatizó mediante el *software Blender* para simplificar su utilización.

La extracción de mallados 3D fue realizada para los procesos iniciales, donde se aplicaron los prototipos de optimización de mallados. Luego se decidió en conjunto al equipo del *Cell and Tissue Arquitecture Lab*, utilizar mallados segmentados por miembros del laboratorio. Esto permitió utilizar mallados extraídos por usuarios con mayor expertiz. Además, lograron incluir más estructuras, como las células estrelladas y células de Kupffer.

Se dedicó tiempo a la automatización de los procesos realizados, esto permitió obtener una herramienta de optimizado de mallados que no requiere conocimiento del *software Blender*, además de un proceso simplificado para cargar los mallados al visualizador web.

El desarrollo del visualizador web tomó más tiempo, debido a la falta de experiencia con los lenguajes de programación necesarios. Aún así se lograron cargar los mallados junto a las características determinadas imprescindibles por el equipo del *Cell and Tissue Arquitecture Lab*. Así, el visualizador posee un nivel aceptable de interactividad para el usuario, al ser capaz de personalizar color, transparencia, tamaño de las estructuras observadas y la opción de desplegar información relevante de las estructuras renderizadas.

Se aplicaron técnicas de optimización de memoria en el visualizador web, que en conjunto al sistema de optimización de mallados lograron que sea ejecutable en equipos de bajo rendimiento. Los resultados obtenidos generan una tasa de cuadros por segundo estable, de 30 cuadros por segundo en equipos de bajo rendimiento y de 60 cuadros por segundo en equipos de alto rendimiento.

El desarrollo de este visualizador web entrega una herramienta importante para la difusión y aprendizaje de biología celular e histología. Además, democratiza el acceso a información científica, al sólo requerir de acceso a internet y un navegador web. Sumado a esto, la automatización de los procesos

necesarios para integrar nuevos mallados al visualizador simplifica su utilización, facilitando el uso futuro al no requerir de expertiz en los lenguajes de programación y *software* utilizados.

6.3 Trabajo futuro

Se podría realizar una evaluación de la plataforma web como herramienta pedagógica. Esto permitiría demostrar la efectividad del uso de visualizaciones 3D de estructuras en el proceso de aprendizaje de estas.

También se podría generar un flujo de trabajo para integrar nuevos mallados al visualizador, para que miembros futuros del laboratorio puedan aprender a utilizar la herramienta de optimización y web.

Otra mejora sería integrar sistemas de compresión de archivos. Esto podría mejorar las velocidades de descarga, necesitando menor velocidad en la conexión de internet para utilizar el visualizador, pero se debe evaluar cómo afecta los procesos de carga al momento de descomprimir los mallados.

Además se podría mejorar la compatibilidad del visualizador con dispositivos móviles. Para ello, se requiere alterar el diseño mediante CSS con metodologías que mantengan la plataforma funcional dentro de navegadores de escritorio y de dispositivos móviles.

Por último, se puede programar un renderizador específico para este tipo de estructuras, con el objetivo de lograr un mayor rendimiento en dispositivos móviles y equipos de bajo costo. Este proceso requiere analizar la geometría específica de los mallados usados, además de luego generar una comparación entre los renderizados, considerando la memoria RAM, tiempos de carga y varios dispositivos para obtener una comparación válida.

Bibliografía

[1] H. Morales-Navarrete, F. Segovia-Miranda, P. Klukowski, K. Meyer, H. Nonaka, G. Marsico, M. Chernykh, A. Kalaidzidis, M. Zerial, and Y. Kalaidzidis, "A versatile pipeline for the multi-scale digital reconstruction and quantitative analysis of 3d tissue architecture," *eLife*, vol. 4, Dec. 2015. [Online]. Available: https://doi.org/10.7554/elife.11214

[2] J. Schindelin, I. Arganda-Carreras, E. Frise, V. Kaynig, M. Longair, T. Pietzsch, S. Preibisch, C. Rueden, S. Saalfeld, B. Schmid, J.-Y. Tinevez, D. J. White, V. Hartenstein, K. Eliceiri, P. Tomancak, and A. Cardona, "Fiji: an open-source platform for biological-image analysis," *Nature Methods*, vol. 9, no. 7, pp. 676–682, jun 2012.

[3] U. Technologies, "Unity game engine," [Online] Available:https://unity.com/es.

[4] "XXIX. the anatomy and physiology of the liver," *Philosophical Transactions of the Royal Society of London*, vol. 123, pp. 711–770, Dec. 1833. [Online]. Available: https://doi.org/10.1098/rstl.1833.0031

[5] R. S. Mccuskey, "The hepatic microvascular system in health and its response to toxicants," *The Anatomical Record: Advances in Integrative Anatomy and Evolutionary Biology*, vol. 291, no. 6, pp. 661–671, 2008. [Online]. Available: https://doi.org/10.1002/ar.20663

[6] H. Elias, "A re-examination of the structure of the mammalian liver. II. the hepatic lobule and its relation to the vascular and biliary systems," *American Journal of Anatomy*, vol. 85, no. 3, pp. 379–456, nov 1949.

[7] M.-T. Ke, S. Fujimoto, and T. Imai, "SeeDB: a simple and morphology-preserving optical clearing agent for neuronal circuit reconstruction," *Nature Neuroscience*, vol. 16, no. 8, pp. 1154–1161, Jun. 2013. [Online]. Available: https://doi.org/10.1038/nn.3447

[8] T. Peng, K. Thorn, T. Schroeder, L. Wang, F. J. Theis, C. Marr, and N. Navab, "A BaSiC tool for background and shading correction of optical microscopy images," *Nature Communications*, vol. 8, no. 1, jun 2017.

[9] M. Hüpfel, A. Y. Kobitski, W. Zhang, and G. U. Nienhaus, "Wavelet-based background and noise subtraction for fluorescence microscopy images," *Biomedical Optics Express*, vol. 12, no. 2, p. 969, jan 2021.

[10] W. E. Lorensen and H. E. Cline, “Marching cubes: A high resolution 3d surface construction algorithm,” *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 163–169, aug 1987.

[11] O. Wiki, “Main page — opengl wiki,,” 2018, [Online; accessed 20-April-2022]. [Online]. Available: http://www.khronos.org/opengl/wiki_opengl/index.php?title=Main_Page&oldid=14430

[12] W. P. Wiki, “Main page — webgl public wiki,,” 2020, [Online; accessed 20-April-2022]. [Online]. Available: http://www.khronos.org/webgl/wiki_1_15/index.php?title=Main_Page&oldid=2621

[13] D. S. Goodsell and J. Jenkinson, “Molecular illustration in research and education: Past, present, and future,” *Journal of Molecular Biology*, vol. 430, no. 21, pp. 3969–3981, Oct. 2018. [Online]. Available: https://doi.org/10.1016/j.jmb.2018.04.043

[14] K. Patterson, B. Terrill, B.-S. Dorfman, R. Blonder, and A. Yarden, “Molecular animations in genomics education: designing for whom?” *Trends in Genetics*, vol. 38, no. 6, pp. 517–520, Jun. 2022. [Online]. Available: https://doi.org/10.1016/j.tig.2022.03.003

[15] A. P. Johnston, J. Rae, N. Ariotti, B. Bailey, A. Lilja, R. Webb, C. Ferguson, S. Maher, T. P. Davis, R. I. Webb, J. McGhee, and R. G. Parton, “Journey to the centre of the cell: Virtual reality immersion into scientific data,” *Traffic*, vol. 19, no. 2, pp. 105–110, Nov. 2017. [Online]. Available: https://doi.org/10.1111/tra.12538

[16] S. R. Kadir, A. Lilja, N. Gunn, C. Strong, R. T. Hughes, B. J. Bailey, J. Rae, R. G. Parton, and J. McGhee, “Nanoscape, a data-driven 3d real-time interactive virtual cell environment,” *eLife*, vol. 10, Jun. 2021. [Online]. Available: https://doi.org/10.7554/elife.64047

[17] T. Sherif, N. Kassis, M.-Ã. Rousseau, R. Adalat, and A. C. Evans, “BrainBrowser: distributed, web-based neurological data visualization,” *Frontiers in Neuroinformatics*, vol. 8, Jan. 2015. [Online]. Available: https://doi.org/10.3389/fninf.2014.00089

[18] L.-P. Ledoux, F. C. Morency, M. Cousineau, J.-C. Houde, K. Whittingstall, and M. Descoteaux, “Fiberweb: Diffusion visualization and processing in the browser,” *Frontiers in Neuroinformatics*, vol. 11, Aug. 2017. [Online]. Available: https://doi.org/10.3389/fninf.2017.00054

[19] B. O. Community, “Blender - a 3d modelling and rendering package,” Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. [Online]. Available: http://www.blender.org/

[20] G. Ranzuglia, M. Callieri, M. Dellepiane, P. Cignoni, and R. Scopigno, “Meshlab as a complete tool for the integration of photos and color with high resolution 3d geometry data,” in *CAA 2012*

Conference Proceedings. Pallas Publications - Amsterdam University Press (AUP), 2013, pp. 406–416. [Online]. Available: http://vcg.isti.cnr.it/Publications/2013/RCDCS13

[21] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, "MeshLab: an Open-Source Mesh Processing Tool," in *Eurographics Italian Chapter Conference*, V. Scarano, R. D. Chiara, and U. Erra, Eds. The Eurographics Association, 2008.

[22] B. D. Team, *Blender 3.0 Manual*, [Online] Available: https://docs.blender.org/manual/en/latest/index.html.