Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Trabajo de Fin de Grado

Desarrollo de un método para medir las diferencias de forma entre objetos tridimensionales

Autor: Álvaro Martínez Caballero

Tutor: Ángel Merchán Pérez

Contenido

Capítulo 1

1.Introducción

Todas las células del cuerpo humano, incluyendo las neuronas del sistema nervioso, constan de un núcleo, un citoplasma y una membrana. Las neuronas, en particular, son células que cuentan con varias prolongaciones ramificadas llamadas dendritas, a través de las cuales reciben impulsos nerviosos de otras neuronas, y con una prolongación única, el axón, a través del cual emiten impulsos hacia otras células [1]. (Figura 1.1)

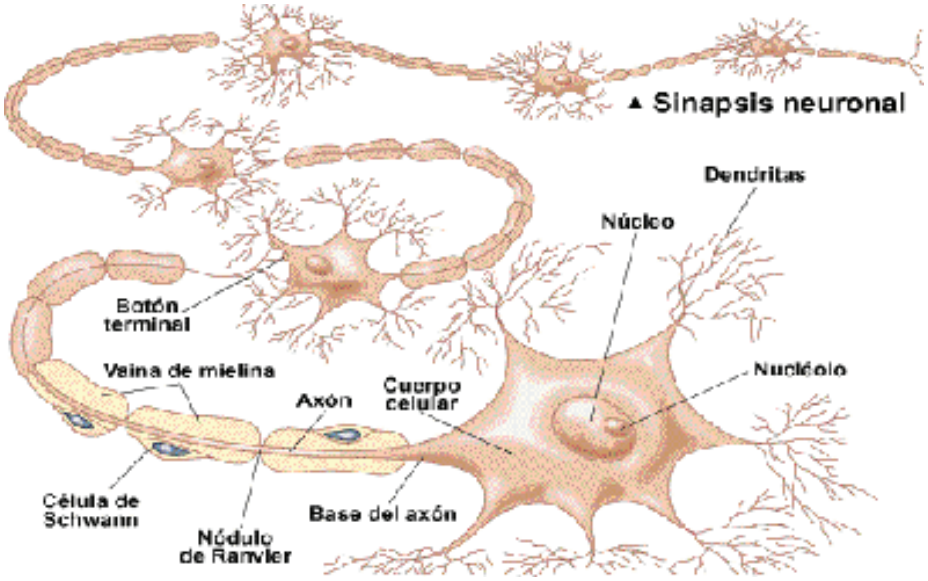


Figura 1.1: Componentes de una neurona y sinapsis

Las sinapsis son los contactos especializados que permiten el paso de información entre las neuronas. No fue hasta 1959, con el uso del microscopio electrónico, que se descubrió que la mayoría de los contactos sinápticos ocurren en las espinas dendríticas, que son unas pequeñas protuberancias que aparecen en la superficie de las dendritas.
Las espinas dendríticas fueron descubiertas por Ramón y Cajal, quien les dio ese nombre porque le recordaban a las espinas de un rosal. Son de gran importancia en el campo de la neurociencia ya que la forma y tamaño de estas espinas influyen en las sinapsis que reciben y por tanto en la transmisión de información. El objetivo del trabajo es ser capaces de medir la diferencia de forma entre un grupo de espinas para comprobar si la forma de estas estructuras está correlacionada con la región cerebral a la que pertenecen.

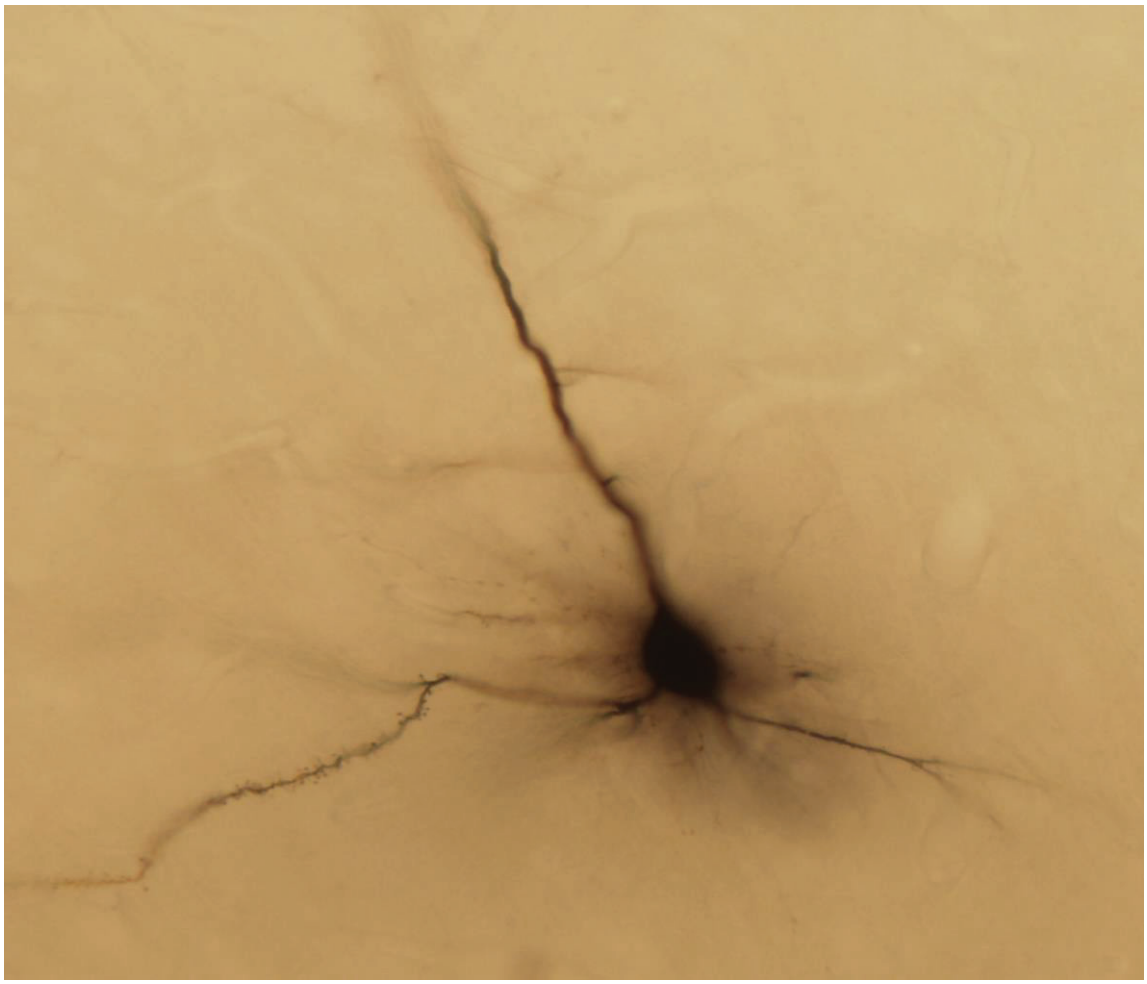


Figura 1.2: Neurona del cerebro de un ratón. Se observa el cuerpo celular y varias dendritas. En la dendrita que aparece a la izquierda se aprecian las pequeñas espinas dendríticas [20].

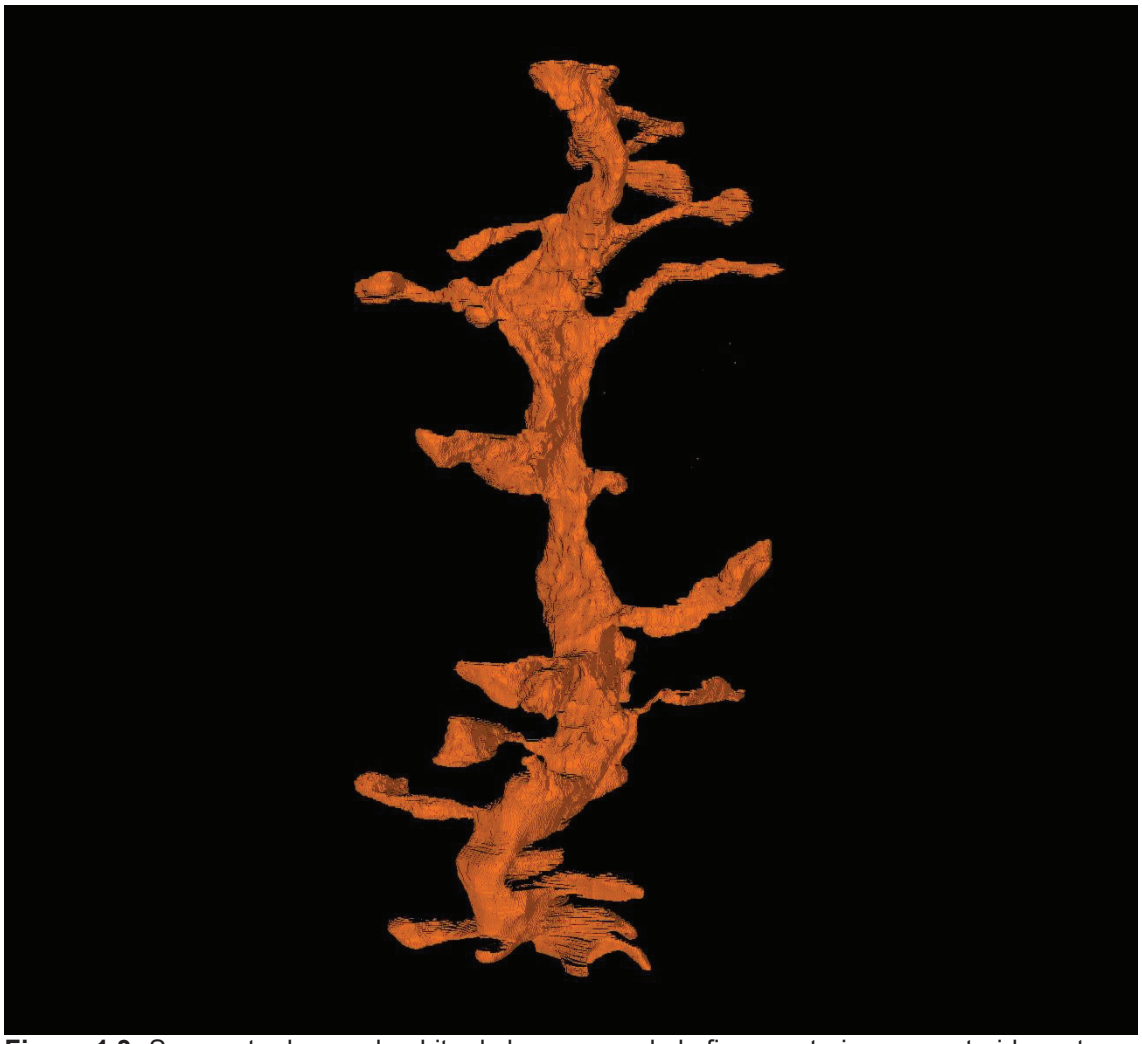


Figura 1.3: Segmento de una dendrita de la neurona de la figura anterior, reconstruido en tres dimensiones a partir de imágenes de microscopía electrónica. Se ve el tallo de la dendrita que recorre la imagen en vertical y varias espinas dendríticas que surgen de él [20]

Habitualmente, las diferencias de forma se evalúan visualmente, de manera cualitativa y subjetiva, por lo que es muy difícil que dos o más investigadores coincidan en sus conclusiones. Nosotros nos proponemos desarrollar un método completamente objetivo, basado en medidas geométricas, que elimine por completo el elemento subjetivo.

Para medir la diferencia de forma utilizaremos la denominada “distancia de Hausdorff”, que consiste en la superposición de dos elementos tridimensionales y medir la distancia entre sus superficies. En un trabajo previo ya se realizó una prueba de concepto en la que se aplicó este método para comparar la forma de espinas dendríticas [21]. En ese estudio se compararon las formas de un pequeño grupo de espinas dendríticas midiendo las distancias de Hausdorff de cada una de ellas con todas las demás. La principal desventaja de este método es su ineficiencia, ya que supone un gran coste computacional si tenemos un gran número de espinas para comparar.

En nuestro caso contamos con 945 espinas, con lo que si comparamos cada una con todas las demás habría que realizar 446.040 mediciones. En el presente trabajo hemos desarrollado un método alternativo en el que compararemos cada espina con tres figuras geométricas de referencia (Figura 1.4) que tendrán una forma parecida a la que tienen las espinas y volumen 1. De esta manera, el número total de medidas se reducirá a 2835.
Las tres figuras de referencia serán una esfera, un cilindro con dos casquetes semiesféricos en sus extremos y un cono con un casquete semiesférico en su base. Comparando las espinas con una esfera tendremos una medida de "esfericidad". Si las comparamos con un cilindro con los extremos redondeados obtendremos una medida de su "simetría axial". Finalmente, al compararlas con un cono delimitado por una semiesfera tendremos una medida de la "asimetría longitudinal".
Al medir las distancias de Hausdorff [2] entre cada una de las espinas y las tres figuras geométricas cada espina queda caracterizada por tres valores, que a su vez pueden usarse para compararlas con otras espinas.

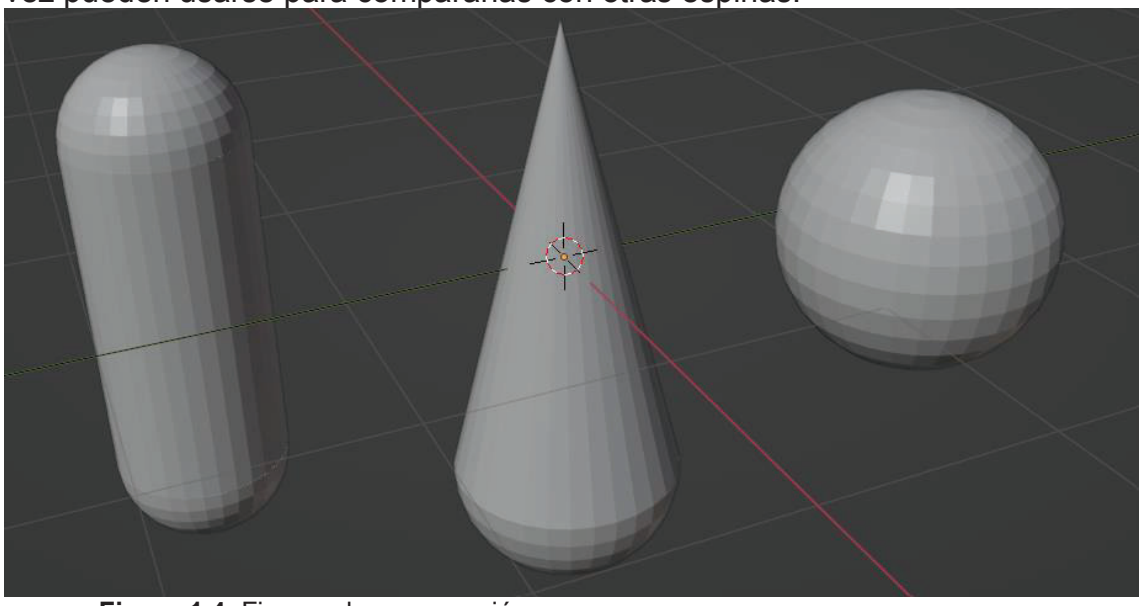


Figura 1.4: Figuras de comparación

Disponemos de un gran número de espinas reconstruidas en 3D a partir de imágenes de microscopía electrónica (Figura 1.5). Conocemos su volumen, longitud y superficie, pero su forma solo ha podido evaluarse de forma cualitativa. Es decir, que la forma la describimos "a ojo". Con este proyecto la forma se podrá caracterizar de una manera objetiva y reproducible.

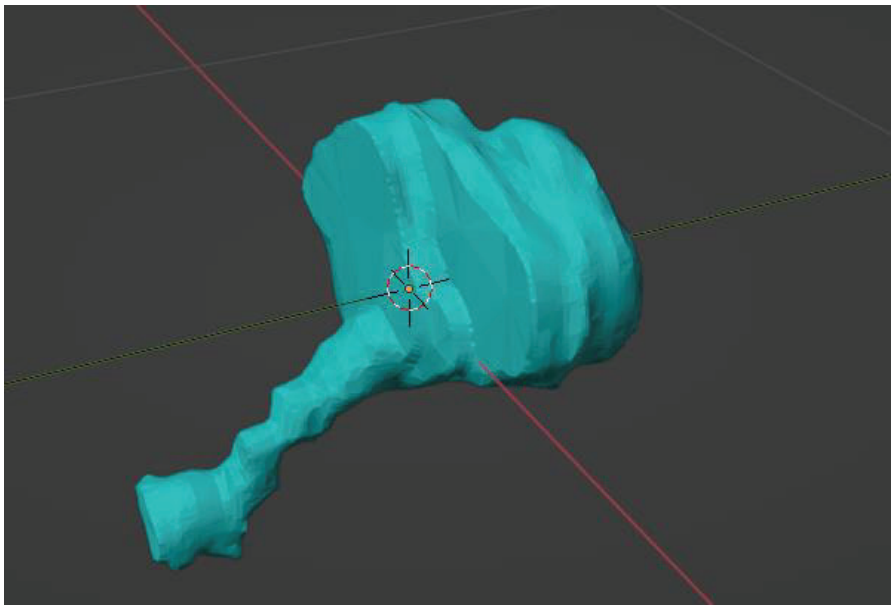


Figura 1.5: Espina dendrítica en 3D

Para poder conseguir la visualización y medición de formas, primero debemos llevar a cabo una serie de requisitos. En primer lugar, tendremos que crear una base de datos que contenga las características e imágenes 3D de cada espina, lo que será útil para el programa de visualización ya que gracias a tener la ruta de cada espina en nuestra base de datos no tendremos que buscarla manualmente.
A continuación, será necesaria la creación de un script que manipule las espinas para poder escalarlas, simplificarlas y suavizarlas. Esto es importante porque para conseguir una medición consistente, hay que llevar a cabo una serie de comprobaciones como pueden ser que las espinas tengan el mismo tamaño y la misma orientación. Seguidamente, crearemos el programa de visualización y el programa de cálculo de la distancia de Hausdorff.

Capítulo 2

2.Tecnologías aplicadas

Con respecto a las tecnologías empleadas para la realización del proyecto, se han utilizado las siguientes:

- El programa *Blender* (programa dedicado al modelaje tridimensional) para el tratamiento y visualización de las espinas en formato .wrl. *Blender* es un programa informático multiplataforma, dedicado especialmente al modelado, iluminación, renderizado, la animación y creación de gráficos tridimensionales. También de composición digital utilizando la técnica procesal de nodos, edición de vídeo, escultura y pintura digital. Se puede descargar desde su página web: https://www.blender.org/ y la versión que se ha utilizado es la 3.9.7.

  Se ha utilizado este programa gracias a su capacidad de crear scripts en Python, esto es importante ya que queremos automatizar la visualización y medición de las espinas. Para crear estos scripts es necesario la utilización de la API **bpy.** *Blender* proporciona sus módulos de Python, como **bpy** y **mathutils**, al intérprete incorporado para que puedan importarse a un script y dar acceso a los datos, clases y funciones de *Blender.*

- Para la creación de la base de datos se ha utilizado el gestor de bases de datos SQLlite. SQLite es una biblioteca en lenguaje C que implementa un motor de base de datos SQL pequeño, rápido, autónomo, de alta confiabilidad y con todas las funciones. SQLite es el motor de base de datos más utilizado en el mundo. SQLite está integrado en todos los teléfonos móviles y en la mayoría de las computadoras y viene incluido dentro de innumerables otras aplicaciones que la gente usa todos los días. Además, se utilizará su módulo **sqlite3** de Python para poder gestionarla.

- Se han utilizado las siguientes librerías de Python:

  - **Numpy:** es el paquete fundamental para la computación científica en Python. Es una biblioteca de Python que proporciona un objeto de matriz multidimensional, varios objetos derivados (como matrices y matrices enmascaradas) y una variedad de rutinas para operaciones rápidas en matrices, que incluyen manipulación matemática, lógica, de formas, clasificación, selección, E/S., transformadas discretas de Fourier, álgebra lineal básica, operaciones estadísticas básicas, simulación aleatoria y mucho más. Se han utilizado las funciones **numpy.array** y **numpy.unravel_index.**

  - **Scipy**: SciPy es una colección de algoritmos matemáticos y funciones de conveniencia construidas sobre la extensión NumPy de Python. Agrega un poder significativo a la

sesión interactiva de Python al proporcionar al usuario comandos y clases de alto nivel para manipular y visualizar datos. Con SciPy, una sesión interactiva de Python se convierte en un entorno de procesamiento de datos y creación de prototipos de sistemas que rivaliza con sistemas como MATLAB, IDL, Octave, R-Lab y SciLab. Utilizaremos la función **directed_Hausdorff** de su módulo **spatial** para calcular la distancia de Hausdorff entre dos conjuntos de puntos**.** Además de **Convex_Hull** y **distance_matrix**.

- **Tkinter:** es el paquete más utilizado para crear interfaces gráficas en Python. Es una capa orientada a objetos basada en Tcl (sencillo y versátil lenguaje de programación open-source) y Tk (la herramienta GUI estándar para Tcl). Utilizaremos esta librería para crear nuestra interfaz gráfica.
- **Openpyxl:** openpyxl es una biblioteca de Python para leer/escribir archivos Excel 2010 xlsx/xlsm/xltx/xltm.

  Nació de la falta de una biblioteca existente para leer/escribir de forma nativa desde Python, el formato Office Open XML. Utilizaremos esta librería para guardar los resultados de nuestras medidas.

Todas estas bibliotecas serán instaladas mediante el comando **pip.**

Capítulo 3

3. Desarrollo de la propuesta

Describiremos en primer lugar la base de datos que nos permitirá acceder a los archivos 3D que representan las espinas dendríticas reconstruidas.

3.1 Diseño e implementación de la base de datos

Las espinas dendríticas reconstruidas en 3D se obtuvieron de seis animales y pertenecen a dos regiones del cerebro distintas: el hipocampo y la corteza somatosensiorial. Las espinas están almacenadas en archivos en formato .wrl. Para poder diseñar la base de datos que vamos a utilizar, en primer lugar, debemos conocer los datos con los que vamos a trabajar. Estos datos describen cada una de las espinas y están recogidos en un archivo .xlsx, que contiene las siguientes columnas:

- Animal_ID: representa el animal al que pertenece la espina.

  Tenemos 6 animales distintos, identificados con los siguientes números: 2, 4, 5, 24, 25 y 28

- Región: representa la región del cerebro a la que pertenece la espina.

  Tenemos 2 regiones: Hippocampus (HC) y Somatosensory (SS)

- Layer: representa la capa dentro de la región.

  Dependiendo de la región tenemos unas capas u otras.

  En Hippocampus tenemos tres capas o layers: Stratum Lacunosum-Moleculare, Stratum Radiatum, Stratum Oriens.

  En Somatosensory tenemos las siguientes capas: I,II,III,IV,Va,Vb,VI

- Spine_ID: representa el identificador de cada espina, el nombre de cada una de ellas está formado por: “Espina”+ ID, siendo ID un número entero.

- Spine_Apparatus: indica si la espina tiene aparato de espina o no. Sólo algunas espinas tienen aparato de espina, que es un orgánulo intracelular, por tanto sus valores pueden ser Yes o No.

- SpApp_volume: representa el volumen del aparto de la espina en el caso de que exista.

- SpApp_area: representa el área del aparto de la espina en caso de que exista.

- Skeleton_ID: El Skeleton es una representación simplificada de cada espina que el software usa para medir su longitud.
- SAS_Synapse: representa la sinapsis que se establecen sobre las espinas, pueden ser de dos tipos: Asymmetric y Symmetric.
- SAS_Type: representa el tipo de sinapsis, pueden ser de dos tipos: Excitatory, Inhibitory.
- Spine_length: representa la longitud de las espinas.
- Spine_Area: representa el área de las espinas.
- Spine_volume: representa el volumen de las espinas.
- Min_neck_diameter: representa el diámetro de la figura en su punto más estrecho.
- SAS_area, SAS_perimeter y SAS_curvature: son medidas correspondientes de la sinapsis.
- Multiple_synapses: representa si una espina tiene más de una sinapsis. Sus valores pueden ser Yes o No.
- Branched_Spine: indica si la espina está ramificada o no. Puede tomar los siguientes valores: No, Spine root + branches y Spine Branch.

Tras conocer los datos con los que vamos a trabajar, el siguiente paso es elegir qué tipo de base de datos vamos a utilizar. Se propusieron los modelos de bases de datos MySQL y SQLlite, y nos decantamos por el uso de una base de datos SQLlite por los siguientes motivos:

1. A diferencia de MySQL, una base de datos SQLite no necesita un servidor, por lo que se integra directamente con la aplicación sin ser necesario instalar la base de datos en algún espacio y luego conectarla con la aplicación.
2. Otro motivo es que SQLite no requiere una instalación, solamente es necesario descargar las bibliotecas de SQLite.
3. Y por último el rendimiento, ya que las operaciones de lectura y escritura son más rápidas en SQLite, casi un 35% más rápido que el sistema de archivos. Otro factor que mejora el rendimiento es que solo carga los datos que necesita, en lugar de leer todo el archivo y mantenerlo en memoria.

Una vez elegido el modelo de base de datos a utilizar, podemos crear la base de datos. Nuestra base de datos constará de una sola tabla, la cual contendrá todos los datos de cada espina. Las columnas de nuestra tabla serán todas las características de las espinas mencionadas anteriormente:

Animal_ID	INT
Región	VARCHAR
Layer	VARCHAR
Spine_ID	VARCHAR
Spine_Apparatus	VARCHAR

SpApp_volume	FLOAT
SpApp_area	FLOAT
Skeleton_ID	VARCHAR
SAS_Synapse	VARCHAR
SAS_Type	VARCHAR
Spine_length	FLOAT
Spine_area	FLOAT
Spine_volume	FLOAT
Min_neck_diameter	DOUBLE
SAS_area	FLOAT
SAS_perimeter	FLOAT
SAS_curvature	FLOAT
Multiple_synapses	VARCHAR
Branched_spine	VARCHAR

Debido que la base de datos utiliza la columna Spine_ID como identificador de cada espina y que varias espinas pueden tener el mismo Spine_ID, debemos modificar la clave primaria para que cada entrada de la tabla sea una entidad única. Para ello utilizaremos como clave primaria una combinación de: Animal_ID, Región, Layer y Spine_ID.

Finalmente, para insertar los datos en la base de datos, lo primero es descargar la librería **sqlite3[3]** para Python, la cual nos permitirá conectarnos y realizar consultas a la base de datos. Además, descargaremos la librería **pandas[4]** que utilizaremos para manipular el archivo de Excel**.** Una vez descargadas, crearemos un script en Python que recorrerá las filas del Excel que contiene los datos de las espinas, y los insertará en la base de datos.

Para poder relacionar cada espina de la base de datos con su imagen 3D, añadimos una nueva columna llamada “Path”, que contendrá la ruta al archivo correspondiente a cada espina. Debido a que las imágenes se encuentran almacenadas en relación a sus características, se ha creado un script mediante el cual consultando las características de cada espina obtenemos su ruta y la insertamos en la tabla. Por ejemplo, viendo el nombre de la espina *HBP29-WT-ID2-HC43-LacMol-3-Espina 13* sabemos que pertenece al animal de ID 2, que se encuentra en la región *Hippocampus* y pertenece al Layer *Stratum Lacunosum-Moleculare.*

Los directorios se clasifican de la siguiente forma: primero dividimos las espinas según su región (Hippocampus o Somatosensory), dentro de cada región se dividen por Animal_ID y por Layer. Como última división, se utilizan los valores de Branched_spine, Spine_Apparatus y Multiple_synapses.

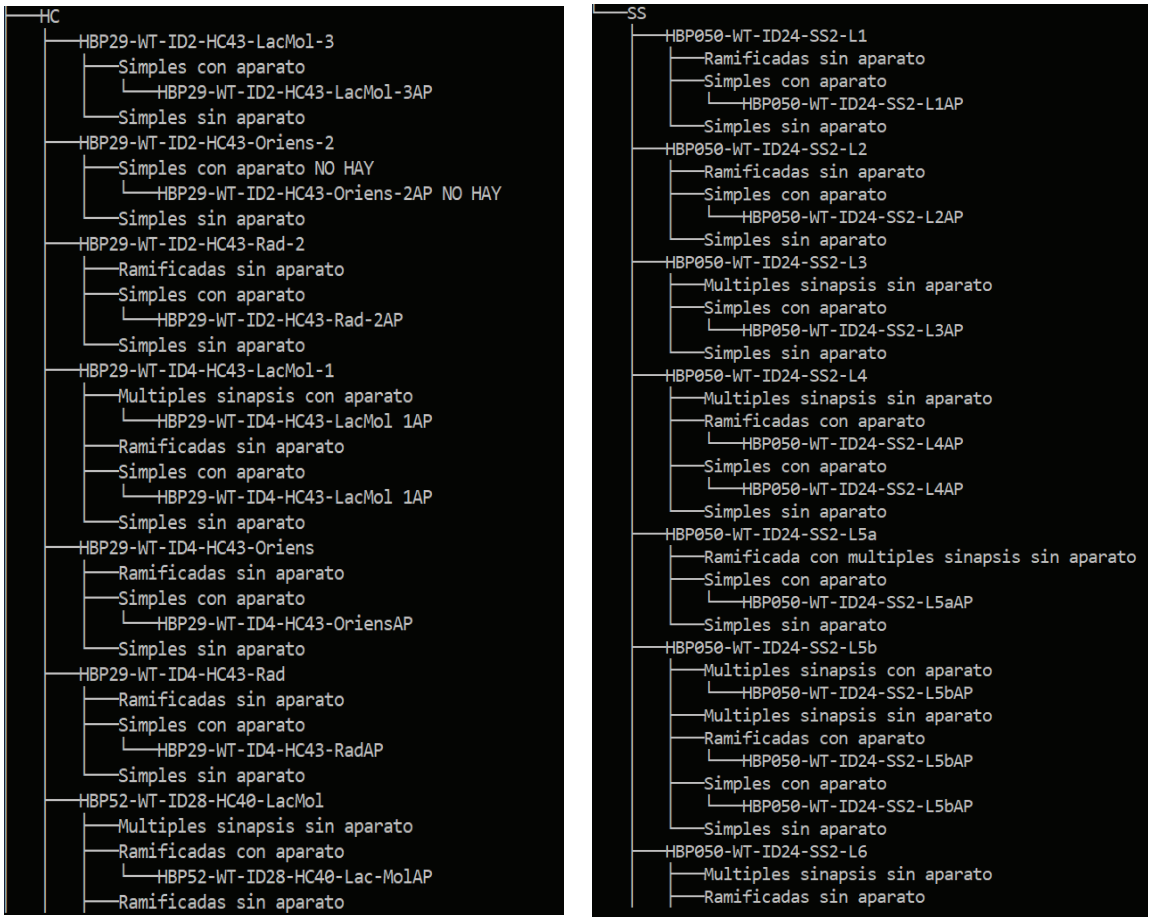


Figura 3.1: Árbol de directorios de Hipocampo (HC) y Somatosensory (SS).

3.2 Visualización de las espinas dendríticas

A continuación, describiremos el método que hemos seguido para visualizar los archivos 3D de manera individual o por grupos.

3.2.1 Simplificación de las espinas

Cada espina es un archivo en formato .wrl que consiste en un gran número de puntos que representan los vértices de una malla tridimensional.

La extensión de archivo WRL es utilizada por VRML (Virtual Reality Modeling Language). WRL es el formato estándar para gráficos vectoriales 3D interactivos diseñados principalmente para aplicaciones WWW. Los archivos WRL contienen una descripción de realidad virtual, que incluye información como el punto de inicio de la vista 3D, coordenadas, objetos, colores, texturas, propiedades de los elementos de la escena, así como objetos y formas de las escenas.

Debido al coste computacional que conlleva realizar cálculos con un gran número de vértices y caras, es necesario buscar una forma de simplificar la espina. Para ello utilizaremos la herramienta ***Decimate[5]*** que nos proporciona Blender para reducir el número de vértices y caras de las espinas. Es esencial que esta reducción no afecte a la forma de nuestra espina, ya que esto alteraría el parámetro con el que luego vamos a trabajar. Contando con esta restricción, hemos comprobado que puede reducirse un 80% el número de vértices y caras de las espinas sin alterar significativamente su forma.

3.2.2 Suavizado de las espinas

Por último, añadiremos un modificador que suavice la espina. *Blender* nos ofrece distintos tipos de suavizado, de los cuales hemos elegido el ***Smooth Laplacian[6]****,* que permite reducir el ruido en la superficie de una malla con cambios mínimos en su forma. Este suavizado es muy útil para objetos que se han reconstruido a partir del mundo real y contienen ruido no deseado.

Aunque con mallas de más de 10.000 vértices puede tardar varios minutos, cabe destacar que esta modificación se realiza únicamente porque mejora el aspecto estético de la espina, pero no se utilizará cuando se calculen las distancias de Hausdorff para no introducir alteraciones innecesarias.

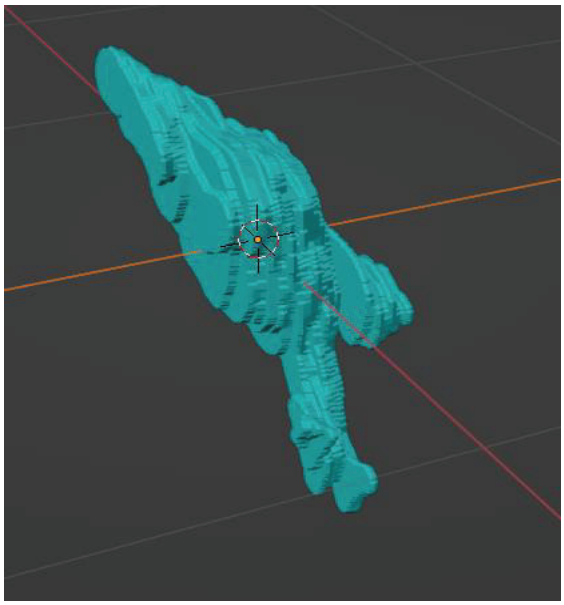


Figura 3.2.2.1: Espina sin suavizar

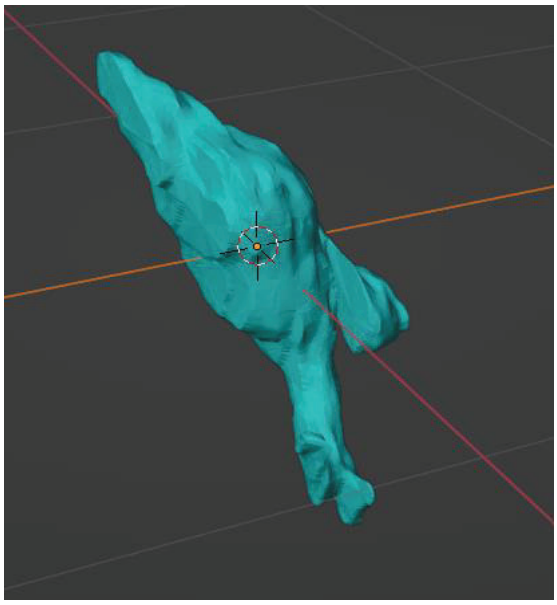


Figura 3.2.2.2: Espina suavizada

3.2.3 Script para la visualización

Este script se utiliza para ver los archivos de las espinas en *Blender*. Se pueden visualizar una o varias espinas, seleccionadas con cualquier criterio elegido por el usuario. Para poder ejecutar un script en *Blender* se ha seguido la siguiente sintaxis:

blender archivo.blend –python script.py --rutas

Una vez recibidas las rutas de las espinas, tendremos que eliminar los dos primeros guiones de nuestro string de rutas mediante las instrucciones:

argv=sys.argv

argv=argv[argv.index("—")+1:]

Para cargar en Blender la imagen de cada espina utilizaremos la función ***bpy.ops.import_scene.x3d[7],*** pasándole como argumento la ruta de la espina. Tras cargar todas las espinas que queremos visualizar, recorreremos todos los objetos de la escena almacenados en el módulo ***bpy.data.objects*** y trabajaremos solo con las espinas, cuyo nombre comienza por "Shape_IndexedFaceSet".

La primera acción será establecer el origen de los objetos con la función ***bpy.ops.object.origin_set (type='GEOMETRY_ORIGIN', center='MEDIAN')[8]***, con esta llamada moveremos la geometría del objeto a su origen. A continuación, modificaremos la propiedad *location* para distribuir las espinas en la escena, de tal forma que solo puede haber 4 espinas a la misma altura.

Para finalizar, escalaremos las espinas con una proporción de 0.001 en cada eje. El escalado se realiza porque las coordenadas de los archivos originales están en nm y aplicando ese escalado las pasamos a micras. A continuación crearemos los modificadores de *Decimate* y *Smooth Laplacian* con las siguientes sentencias:
bpy.ops.object.modifier_add(type='DECIMATE'),bpy.ops.object.modifier_add(type='LAPLACIANSMOOTH')

Aplicaremos el *decimate* del 80%***:***
bpy.context.object.modifiers["Diezmar"].ratio = 0.2

Y al suavizado le aplicaremos un factor_lambda de 1.5 y 10 iteraciones:

bpy.context.object.modifiers["LaplacianSmooth"].lambda_factor = 1.5

bpy.context.object.modifiers["LaplacianSmooth"].iterations = 10

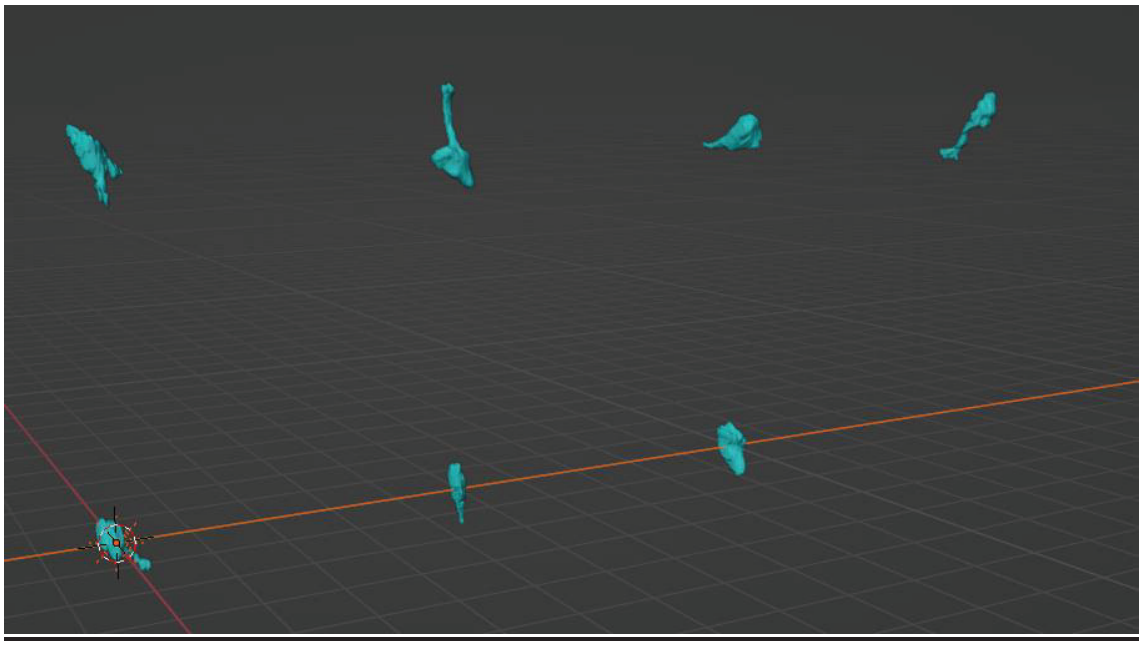


Figura 3.2.3.1: Ejemplo de cómo se representarían 7 espinas en la escena de Blender.

3.3 Medición de las Distancias de Hausdorff

A continuación, describiremos el método geométrico que hemos utilizado para medir de forma objetiva las diferencias de forma entre diferentes objetos. Para comparar la forma de las distintas espinas dendríticas utilizaremos las medidas de Hausdorff. Para hacerlo, necesitaremos primero hacer algunas modificaciones, como escalarlas para que tengan todas el mismo volumen y orientar todas las espinas de la misma manera.

3.3.1 Distancia de Hausdorff

La distancia de Hausdorff mide cuán lejos están uno de otro de dos subconjuntos compactos de un espacio métrico.

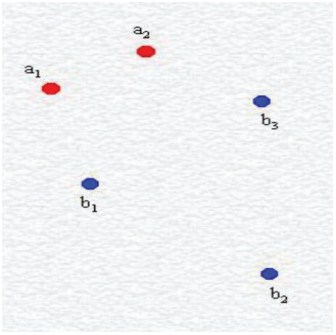
Definición:

$$d_H(X,Y) = \max\{d_1, d_2\}, \qquad \begin{cases} d_1 = \sup_{x\in X} \inf_{y\in Y} \operatorname{dist}(x,y) \\ d_2 = \sup_{y\in Y} \inf_{x\in X} \operatorname{dist}(x,y)\} \end{cases}$$

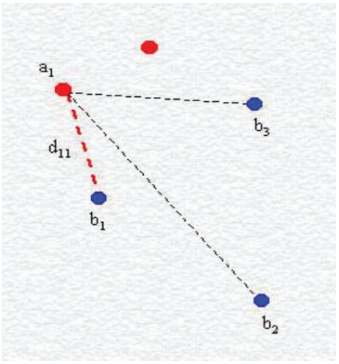
Para calcular esta distancia utilizaremos la función ***directed_Hausdorff[9]*** de la librería ***scipy***. Esta función recibe dos conjuntos de puntos y devuelve la distancia de Hausdorff. Tendremos que llamar a esta función dos veces, la primera para calcular la distancia de la espina a la figura de referencia (esfera, cilindro o cono) y otra para calcular la distancia de la figura a la espina, y quedarnos con la máxima distancia de las dos.

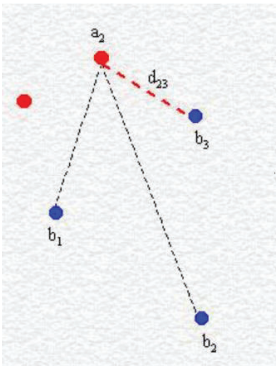
Ejemplo de el cálculo de la distancia de Hausdorff entre dos conjuntos de puntos:

Tenemos los conjuntos de puntos A y B:



Calculamos las distancias de a1 con bj y nos quedamos con la mínima. Haremos lo mismo con a2:





Por lo que d1 será la distancia mayor de estas dos distancias.
Para calcular d2 se seguirá el mismo procedimiento pero calculando la distancia desde el conjunto B al conjunto A.

La distancia de Hausdorff será el máximo entre d1 y d2.

3.3.2 Diseño de las figuras de referencia

Como se ha mencionado en la introducción, las espinas se compararán con tres figuras geométricas, lo que permitirá medir la esfericidad, simetría axial y asimetría longitudinal de la espina.

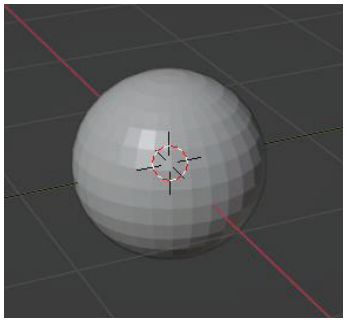
Se han elegido estas tres figuras debido a que su forma es-parecida a la de las espinas dendríticas. Para que las comparaciones sean consistentes, todas las figuras deben tener el mismo volumen, por lo que para conseguir un volumen 1 en las tres figuras el radio se ha calculado de la siguiente forma:

- Esfera: $r = \sqrt[3]{\frac{3}{4*pi}}$
- Cilindro con una altura de cuatro veces el radio y dos semiesferas en los extremos: $r = \sqrt[3]{\frac{1}{\frac{4*pi}{3}+4*pi}}$
- Cono con una altura de cuatro veces el radio de la base y con una semiesfera en la base: $r = \sqrt[3]{\frac{1}{2*pi}}$

Para la creación de la esfera se ha utilizado la función:

bpy.ops.mesh.primitive_uv_sphere_add(segments=32, ring_count=16, radius=r, calc_uvs=True, enter_editmode=False, align='WORLD', location=(0.0, 0.0, 0.0), rotation=(0.0, 0.0, 0.0), scale=(1.0,1.0, 1.0))

Siendo el radio: $r = \sqrt[3]{\frac{3}{4*pi}}$



Para crear el cilindro con dos semiesferas en los extremos, primero se ha creado un cilindro con el radio anterior, a continuación, se ha creado una esfera y se ha divido en dos para poder colocar cada semiesfera en los extremos del cilindro. Función para la creación del cilindro:

bpy.ops.mesh.primitive_cylinder_add(vertices=32, radius=radio, depth=4*0.39, end_fill_type='NGON', calc_uvs=True, enter_editmode=False, align='WORLD', location=(0.0, 0.0, 0.0), rotation=(0.0, 0.0, 0.0), scale=(1.0, 1.0, 1.0))

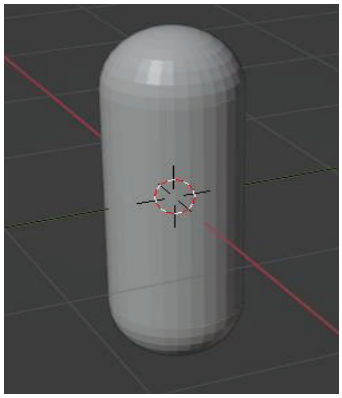
Siendo radio: $r = \sqrt[3]{\frac{1}{\frac{4*pi}{3}+4*pi}}$

Para la creación de la esfera se ha utilizado la función anterior.

Para dividir la esfera por la mitad pasamos al modo edición de *Blender*, después seleccionamos el loop del centro. Esto se hace posicionado el ratón sobre la línea (horizontal en este caso) y con la tecla Alt presionada, se hace clic derecho en el ratón.

Luego presionamos la letra V y esa parte que está seleccionada se cortará.

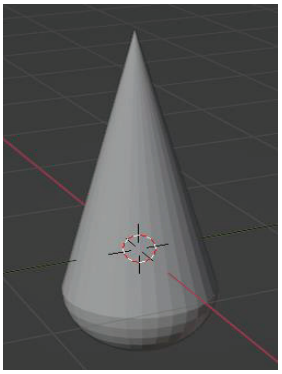
Para seleccionar luego la mitad de la esfera, se selecciona un solo vértice y se presiona letra L [19].



Para la creación del cono con una semiesfera en su base, crearemos el cono con radio $r = \sqrt[3]{\frac{1}{2*pi}}$, utilizaremos la función:

bpy.ops.mesh.primitive_cone_add(vertices=32, radius1=radio, radius2=0.0, depth=4*radio, end_fill_type='NGON', calc_uvs=True, enter_editmode=False, align='WORLD', location=(0.0, 0.0, 0.0), rotation=(0.0, 0.0, 0.0), scale=(1.0, 1.0, 1.0))

La semiesfera se creará y se dividirá de la misma forma que para el cilindro, a diferencia de que en este caso solo nos quedaremos con una.



3.3.3 Configuración de las espinas

Para poder medir las diferencias de forma entre las espinas y las figuras, es importante que no influyan factores como el tamaño, la posición o la orientación de las espinas, por ello las figuras y las espinas deben cumplir una serie de requisitos:

1. Volumen normalizado: todos los objetos que queremos comparar deben tener el mismo volumen para que las diferencias de tamaño no influyan en las diferencias de forma. Para lograr esto, escalaremos todas las espinas para que tengan volumen 1. Para conseguir esto dividiremos las dimensiones de la espina por el siguiente factor, $factor = \sqrt[3]{x * y * z}$.
2. Origen: los centros de masas o centroides de las espinas y figuras deben colocarse en el mismo punto.
3. Orientación: para poder comparar de forma correcta las espinas con el cilindro y el cono, es necesario que estén orientadas de tal forma que encajen de la mejor forma posible con estas figuras. En el caso de la esfera la orientación no influye, siempre que coincidan los centros de masa.

   En primer lugar, orientaremos el eje mayor de la espina para que sea paralelo al eje Z. Para ello calculamos el eje mayor de la espina, es decir, buscamos los dos puntos más alejados de la espina y formamos un vector con ellos. Este vector formará un ángulo con respecto al eje X e Y, rotaremos el vector de tal forma que sea perpendicular a estos ejes. Esto ya sería suficiente para comparar con el cilindro, ya que sus dos extremos son iguales.

   Para el cono necesitamos realizar un paso más, ya que tiene un extremo grueso y otro fino, que deben coincidir con la parte más gruesa y más

fina de la espina. Una vez tenemos el eje mayor de la espina paralelo al eje Z tenemos que comprobar que la parte más gruesa de la espina esté en la parte inferior para poder comparar la espina con la parte más ancha del cono. Con el punto medio del eje mayor, calculado de la siguiente forma $puntomedio = (\frac{x+x1}{2}, \frac{y+y1}{2}, \frac{z+z1}{2})$, y con el centro de masas (punto (0,0,0)), formamos un vector desde el punto medio al centro de masas y dependiendo del ángulo que forma con el eje X indicará si la parte gruesa está hacia arriba o abajo.

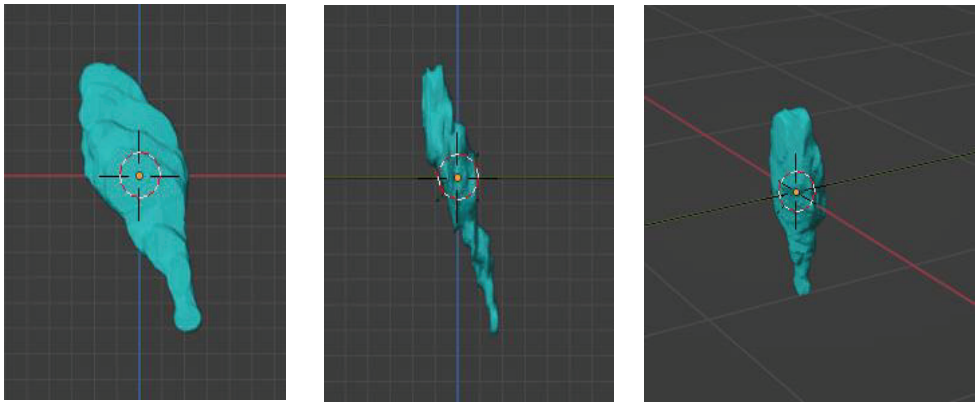


Figura 3.3.3.1: Perspectivas de espina sin rotar

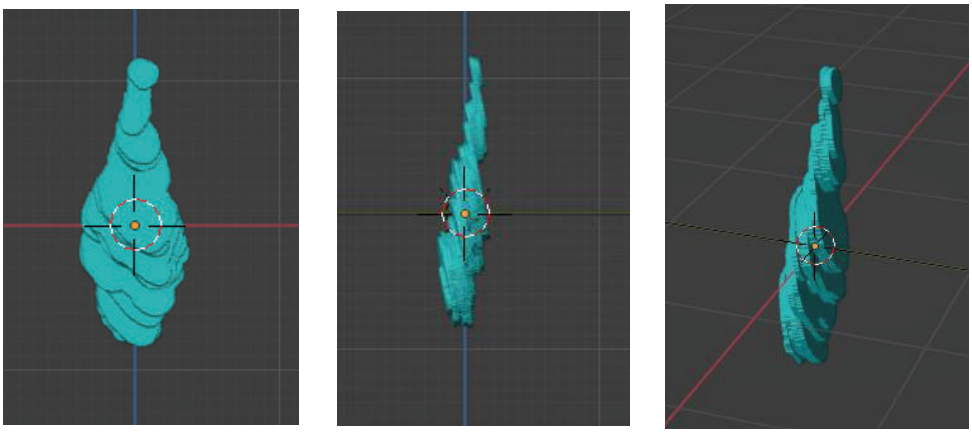


Figura 3.3.3.2: Perspectivas de espina rotada

3.3.3 Script distancias de Hausdorff

Para poder calcular la distancia de Hausdorff de varias espinas al mismo tiempo se ha creado un script que se encargará de ello.

En primer lugar, cargaremos las tres figuras que utilizaremos como referencia para compararlas con las espinas. Para ello utilizamos la función ***bpy.ops.import_scene.obj()[7]*** que nos permite importar objetos, recibiendo como parámetro la ruta de la figura.

Una vez tenemos las tres figuras cargadas, las colocaremos con su centro de masas en el punto (0,0,0) modificando su atributo *location* y utilizando la función:

bpy.ops.object.origin_set(type='ORIGIN_CENTER_OF_MASS',center='MEDIAN')

El siguiente paso será cargar las espinas. Para ello utilizaremos la función ***bpy.ops.import_scene.x3d***, que recibirá como parámetro la ruta de la espina, una vez cargadas, mediante un bucle **for** recorreremos todos los objetos de la escena y nos enfocaremos en los que su nombre contiene la cadena "Shape_IndexedFaceSet", es decir, en los objetos que representan las espinas.

Antes de poder comparar las espinas con las figuras, estas deberán cumplir con unos requisitos. En primer lugar, deben tener el centro de masas colocado en el mismo lugar que las figuras, para conseguir esto utilizaremos las siguientes sentencias:

bpy.ops.object.origin_set(type='ORIGIN_CENTER_OF_MASS',center='MEDIAN')

ob.location=0,0,0

Lo siguiente será aplicar el modificador *decimate* a la espina, para ello primero añadimos el modificador con la siguiente sentencia:

bpy.ops.object.modifier_add(type='DECIMATE')

Y elegimos un *ratio*:

bpy.context.object.modifiers['Diezmar'].ratio=0.2

El siguiente requisito es que las espinas tengan volumen 1, por lo que dividiremos cada dimensión por un mismo factor,

$factor = \sqrt[3]{x * y * z}$.:

dims=ob.dimensions

factor=(dims[0]*dims[1]*dims[2])**(1/3)

ob.dimensions=dims[0]/factor,dims[1]/factor,dims[2]/factor

Por último, hay que orientarlas para que se superpongan lo mejor posible con el cilindro y el cono.

Necesitaremos los vértices que forman la espina, que se pueden obtener con la sentencia ***ob.data.vertices***, siendo ob la espina, pero las coordenadas de estos vértice son locales. Para conseguir las coordenadas globales tendremos que multiplicar estos vértices por la **world_matrix[14]** del objeto. Con la siguiente sentencia multiplicaremos los vértices por la **matrix_world** y los guardaremos en la variable *vértices_espina*:

vértices_espina=[np.array((ob.matrix_world @ v.co)) for v in ob.data.vertices]

Además, necesitaremos saber cuál es el eje mayor y sus dos extremos. Debido al gran número de vértices de la espina, medir las distancias de cada uno con el resto tiene un gran coste computacional, por lo que hemos calculado el convexHull de la espina utilizando la función ***ConvexHull[10]*** del módulo ***spatial*** de ***scipy,*** esta nos devuelve los vértices que pertenecen al casco convexo:

candidates =vertices_espina[spatial.ConvexHull(vertices_espina).vertices]

Para calcular la distancia entre todos los candidatos se ha utilizado la función ***spatial.distance_matrix[11]***, que recibe dos matrices (x e y) y devuelve una matriz que contiene la distancia de cada vector en x a cada vector en y.

dist_mat = spatial.distance_matrix(candidates, candidates)

Por último, hemos calculado los dos puntos más alejados, es decir, los dos puntos que tienen la máxima distancia en dist_mat, para ello hemos utilizado la función ***unravel_index[12]*** de la librería **numpy**, esta función calcula el índice correspondiente en una matriz de un índice lineal. Por ejemplo, en una lista formada por [1,2,3,4] el índice lineal del número 4 es el 3, sin embargo, el índice del número 4 en una matriz 2x2 sería el (1,1).

Esta función recibe dos argumentos, el primero de ellos es una matriz de enteros cuyos elementos son índices en la versión aplanada de una matriz de dimensiones shape.

La función ***argmax()[13]*** devolverá el índice lineal de la distancia mayor.

i, j = np.unravel_index(dist_mat.argmax(), dist_mat.shape)

Teniendo el eje mayor de la espina y sus dos extremos, podemos calcular el ángulo que forman con el eje X e Y para rotar el eje mayor de tal forma que sea paralelo a el eje Z. Primero calcularemos el vector que se forma entre los dos puntos:

punto1=candidates[i].copy()

punto2=candidates[j].copy()

Debido a que en Blender el eje X crece hacia la izquierda cambiamos el valor de la x.

punto1[0]=-punto1[0]

punto2[0]=-punto2[0]

vector=[punto1[0]-punto2[0],punto1[1]-punto2[1],punto1[2]-punto2[2]]

Para calcular el ángulo que se forma con el eje X e Y utilizaremos al formula $angulo = \tan^{-1}(\frac{y}{x})$

angulo_x=math.atan2(vector[2],vector[0])

angulo_y=math.atan2(vector[2],vector[1])

Pasamos los ángulos de radianes a grados:

angulo_x_grados=angulo_x*(180.0/math.pi)

angulo_y_grados=angulo_y*(180.0/math.pi)

Una vez que tenemos los ángulos empezamos a rotar cada eje para que quede paralelo a el eje Z.

En ángulo de giro lo calcularemos de la siguiente forma:

$$giro = |\frac{pi}{2} - |angulo\_x||$$

Para rotar la espina utilizaremos el atributo ***rotation_euler*** el cual contiene la rotación en los tres ejes del objeto, por lo que si queremos rotar por el eje X modificaremos el primer valor(0) de ***rotation_euler****, y para rotar por el eje Y modificaremos el segundo valor(1)*

Si el ángulo es positivo diferenciaremos dos casos, cuando es mayor de 90º y cuando es menor. Cuando es mayor de 90º giraremos en sentido horario y si es menor de 90º giraremos en sentido antihorario.

En caso de que el ángulo sea negativo diferenciaremos también dos casos, cuando es mayor de -90º y cuando es menor de -90º. Cuando es mayor de -90º se gira en sentido antihorario mientras que cuando es menor de -90º se gira en sentido horario.

Tras rotar las espinas, tenemos que actualizar la escena de la siguiente forma:

ob.data.update()

bpy.context.view_layer.depsgraph.update()

La escena se debe actualizar para que los datos de la espina se actualicen, ya que sin esta actualización las coordenadas de los vértices no se habrían modificado tras el giro, por lo que volvemos a guardar los vértices de la espina.

vertices_espina_nuevos=np.array([np.array((ob.matrix_world @ v.co)) for v in ob.data.vertices])

En este punto, ya podríamos comparar la espina con el cilindro y la esfera, pero aun es necesario colocar la parte gruesa de la espina hacia abajo para poder compararlo con el cono.

Para saber si la parte gruesa está hacia arriba o hacia abajo vamos a calcular un vector que ira desde el punto medio del eje mayor hacia el punto (0,0,0) que como se ha mencionado anteriormente es el centro de masas.

Calcularemos de nuevo el eje mayor de la misma forma que antes y con las coordenadas de los dos puntos más alejados calcularemos el punto medio y el vector:

punto_medio=[(punto1[0]+punto2[0])/2,(punto1[1]+punto2[1])/2,(punto1[2]+punto2[2])/2]

vector2=[-punto_medio[0],-punto_medio[1],-punto_medio[2]]

Calcularemos el ángulo con respecto al eje X, y si el ángulo es positivo quiere decir que la parte gruesa esta hacia arriba mientras que si es negativo indicará que esta hacia abajo. En caso de que la parte gruesa esté hacia arriba habrá que realizar un giro de 180º.

angulo_x2=math.atan2(vector2[2],vector2[0])

if angulo_x2*(180.0/math.pi)>0:

    ob.rotation_euler[1]+=math.pi

Y volveremos a actualizar la escena:

ob.data.update()

bpy.context.view_layer.depsgraph.update()

vertices_espina_nuevos=np.array([np.array((ob.matrix_world @ v.co)) for v in ob.data.vertices])

Como ejemplo de cómo se verían estos cambios hechos a las espinas, vamos a medir las distancias de Hausdorff entre estas dos espinas:

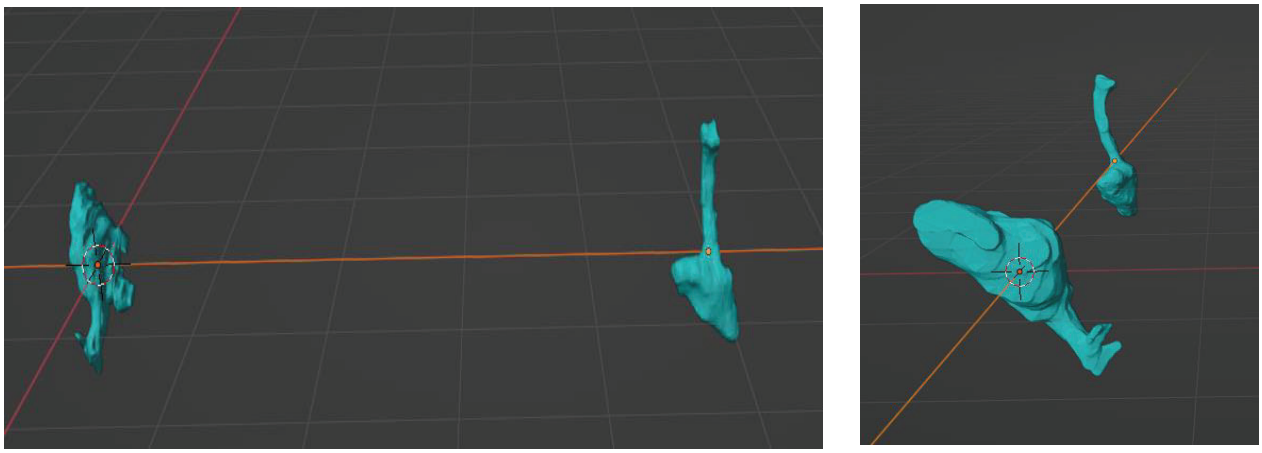


Figura 3.3.1: Espinas sin configurar

Como podemos observar, estas dos espinas no pueden compararse directamente, deben ser configuradas. Como resultado de su configuración quedarían de la siguiente forma:

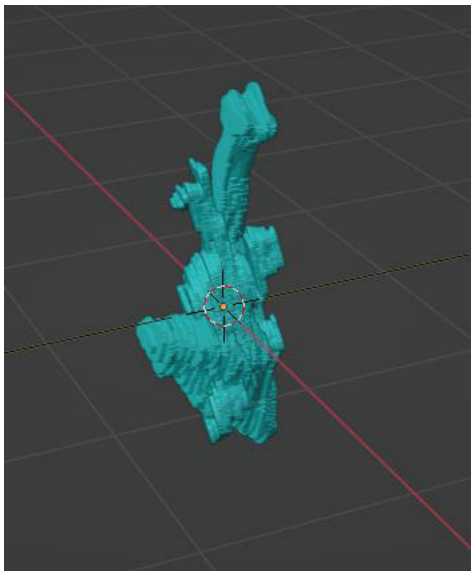


Figura 3.3.2: Espinas configuradas

Una vez configuradas, vemos que ambas tienen colocados sus centroides en el mismo punto y están orientadas con respecto a su eje mayor.

En este punto ya podemos calcular las distancias de Hausdorff. Ya tenemos los vértices de la espina, pero también necesitamos los vértices de las tres figuras, para obtenerlos primero guardaremos cada figura en una variable de la siguiente manera:

cilindro=bpy.data.objects['Cilindro']

esfera=bpy.data.objects['Esfera']

cono=bpy.data.objects['Cono']

Y accederemos a sus vértices y los guardaremos en otra variable:

vertices_cono=np.array([np.array((cono.matrix_world @ v.co)) for v in cono.data.vertices])

vertices_cil=[(cilindro.matrix_world @ v.co) for v in cilindro.data.vertices]

vertices_esfera=[(esfera.matrix_world @ v.co) for v in esfera.data.vertices]

Las coordenadas de los vértices deben ser también globales, por lo que los multiplicaremos por la **matrix_world** de cada objeto.

Ya tenemos todos los datos necesarios para calcular las distancias, por lo que llamaremos a la función ***d_hausdorff*** , la cual recibe dos conjuntos de vértices y retorna la distancia de Hausdorff. Esta función hace uso de la función ***directed_hausdorff*** de la librería **scipy**. A dicha función se le llamará dos veces para calcular primero la distancia de la figura a la espina y posteriormente para calcular la distancia de la espina a la figura, retornando la mayor de estas.

def d_hausdorff(cord_espina,cord_esfera):

h_1=directed_hausdorff(cord_esfera,cord_espina)[0]

h_2=directed_hausdorff(cord_espina,cord_esfera)[0]

return max(h_1,h_2)

Se llamará a esta función tres veces, una para cada figura.

Una vez obtenidas las distancias, las almacenaremos en una matriz (nº espinas x 4), para ello utilizaremos un *array* auxiliar que contendrá el nombre y las tres distancias.

vector_d.append(nombre[index])

vector_d.append(d_esfera)

vector_d.append(d_cilindro)

vector_d.append(d_cono)

Los nombres de las espinas se han obtenido mediante sus rutas:

for r in routes:

r=r.split("\\")

r=r[len(r)-1]

nombre.append(r)

Y añadiremos las tres medidas a la matriz:

matriz_d.append(vector_d)

Esta matriz la copiaremos a una hoja Excel. Para manipular la hoja de Excel utilizaremos la librería ***openpyxl[15]***, crearemos una hoja y le añadiremos 4 columnas: nombre de la espina, Esfera, Cilindro, Cono.

wb = openpyxl.Workbook()

hoja1=wb.active

hoja1.append(("","Sphere","Cylinder","Cone"))

Para añadir las distancias al Excel, se recorrerá la matriz por filas y se irán añadiendo:

for m in matriz_d:

hoja1.append(m)

Para finalizar se guardará el Excel con el nombre “distancias.xlsx”, se guardará siempre en el Escritorio:

ruta_guardado = os.path.join(os.path.join(os.environ['USERPROFILE']), 'Desktop')

Utilizaremos la variable de entorno USERPROFILE, que indica la ruta absoluta del usuario la cual añadiremos con la función ***join*** para unirla con Desktop.

Para guardar el archivo Excel usaremos la función ***save*** de la librería **openpyxl**:

wb.save(ruta_guardado)

Ejemplo de un Excel que contiene la medición de 5 espinas:

	Sphere	Cylinder	Cone
HBP29-WT-ID2-HC43-LacMol-3-Espina 1018.wrl	0,547005	0,836697	1,122169
HBP29-WT-ID2-HC43-LacMol-3-Espina 1033.wrl	0,543728	0,831475	1,148912
HBP29-WT-ID2-HC43-LacMol-3-Espina 1140.wrl	0,432518	0,802068	1,135667
HBP29-WT-ID2-HC43-LacMol-3-Espina 1220.wrl	0,483651	0,81707	1,135928
HBP29-WT-ID2-HC43-LacMol-3-Espina 13.wrl	0,641317	0,848608	1,146344

Figura 3.3.3.3: Excel con medición de 5 figuras

3.4 Interfaz Gráfica

Para poder visualizar las espinas de forma eficiente sin tener que recorrer el árbol de directorios para buscar una por una cada espina, hemos creado una interfaz gráfica que permite seleccionar espinas con respecto a sus características. Con esta aplicación el usuario puede seleccionar las espinas que desee para visualizarlas o medir las distancias de Hausdorff. Para ello, hemos instalado y utilizado una librería de Python llamada ***tkinter[16]***, considerada un estándar para la creación de interfaces gráficas en Python.

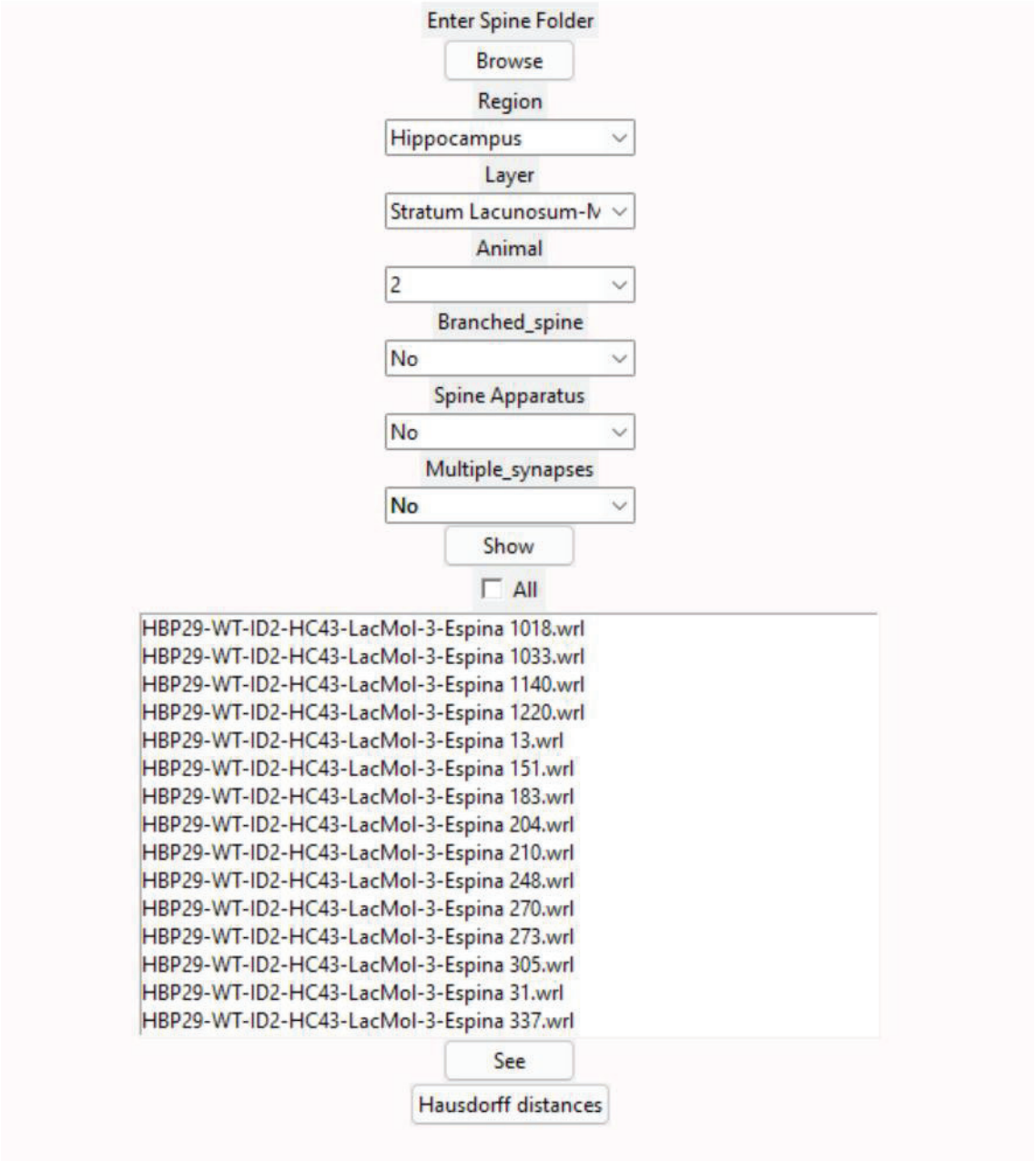


Figura 3.4.1: Interfaz de la aplicación para seleccionar, visualizar y medir las espinas

En primer lugar, crearemos la ventana:

ventana=tk.Tk()

ventana.title("Primera ventana")

ventana.geometry('700x700')

ventana.configure(background='snow')

En esta ventana es donde se añadirán todos los componentes de nuestra interfaz.

Como está aplicación debe ser capaz de funcionar en distintos dispositivos, debemos modificar las rutas a la imagen de cada espina en la base de datos, ya que la ruta varía con respecto al ordenador en el que se ejecute. Esta es la función del botón “Browse”, el cual abre el explorador de archivos mediante la función ***askDirectory()[17]*** para que se elija el directorio donde se encuentran las espinas, una vez seleccionado se hará una consulta a la base de datos para obtener la ruta por defecto y sustituirla por la actual.

Las rutas tienen una parte que varía con respecto a la máquina en la que se ejecuta y otra que no cambia. Teniendo esto en cuenta, podemos realizar una consulta a la base de datos para obtener la ruta de cualquier espina, una vez la tengamos recortamos esa ruta y nos guardaremos la parte variante en una variable llamada “ruta_actual2”.

La nueva ruta la obtendremos como resultado de la llamada a la función askDirectory(), que se encargará de abrir el explorador de archivos y darnos a elegir el directorio que queremos usar. Por lo que teniendo estas dos rutas podemos realizar una consulta para modificar las rutas en la base de datos. El código de la función que modificará la ruta:

```
def config_bd():
    try:
        db=sqlite3.connect(db_path)
        mycursor=db.cursor()
        filename = filedialog.askdirectory()
        filename=filename.replace('/','\\\\')
        consulta="SELECT Path from espinasdendriticas LIMIT 1"
        mycursor.execute(consulta)
        filas=mycursor.fetchall()
        ruta_actual2=filas[0][0]
        raindice=ruta_actual2.find("Espinas")
        ruta_actual2=ruta_actual2[:(raindice-2)]
        consulta=f"UPDATE espinasdendriticas SET
Path=REPLACE(Path,'{ruta_actual2}','{filename}')"
        mycursor.execute(consulta)
```

```
    finally:
        mycursor.close()
        db.commit()
        db.close()
```

A continuación, se elegirían las características de las espinas que queramos visualizar mediante los desplegables, creados con la función ***Combobox[18]***. Dependiendo de los valores elegidos en los desplegables aparecerán unos u otros en los siguientes, es decir, si se elige Hippocampus como Región, los Layers que aparecerán son solo los de Hippocampus.

Cabe destacar que cada desplegable incluye el valor TODOS, en el caso de que no se quiera filtrar por esa característica.

Los Combobox se han creado de la siguiente forma:

```
sinapsiscombo=ttk.Combobox(
    state="readonly",
    values=["No","Yes","TODOS"]
)
simplecombo=ttk.Combobox(
    state="readonly",
    values=["No","Yes","TODOS"]
)
aparatocombo=ttk.Combobox(
    state="readonly",
    values=["No","Yes","TODOS"]
)
regioncombo=ttk.Combobox(
    state="readonly",
    values=["Hippocampus","Somatosensory","TODOS"]
)
animalcombo=ttk.Combobox(
     state="readonly",
    values=[],
    postcommand=animal_changed
)
```

```
layercombo=ttk.Combobox(
    state="readonly",
    values=[],
    postcommand=selection_changed
)
```

Los desplegables de Animal y Layer cambiaran su contenido dependiendo de la Región elegida, para poder cambiar estos valores, asignamos a la variable *postcommand* de ttk.Combobox una función, esta función asignará unos valores u otros dependiendo de la Region. Para el desplegable de Animal usamos la función **animal_changed()**, mientras que para el Layer utilizamos la función **selection_changed().**

```
def animal_changed():

    reg=regioncombo.get()
    if reg=="Hippocampus":
        animalcombo["values"]= ["2","4","28","TODOS"]
    else:
        animalcombo["values"]=["5","24","25","TODOS"]

def selection_changed():

    reg=regioncombo.get()
    if reg=="Hippocampus":
        layercombo["values"]= ["Stratum Lacunosum-Moleculare","Stratum Oriens","Stratum Radiatum","TODOS"]
    else:
        layercombo["values"]=["I","II","III","IV","Va","Vb","VI","TODOS"]
```

Cabe destacar que antes de cada desplegable se ha creado una etiqueta que describa el contenido de cada desplegable:

```
directorio=tk.Label(text="Enter Spine Folder")
region=tk.Label(text="Region")
layer=tk.Label(text="Layer")
```

animal=tk.Label(text="Animal")

simple=tk.Label(text="Branched_spine")

aparato=tk.Label(text="Spine Apparatus")

sinapsis=tk.Label(text="Multiple_synapses")

Una vez elegidas las características, el siguiente paso es hacer "click" en el botón show, que llamará a una función que se encarga de realizar la consulta en la base de datos con las características elegidas previamente. Además, esta función mostrará en la ***ListBox*** los nombres de las espinas que cumplen esas condiciones y guardará las rutas de estas espinas en un vector, el cual será utilizado luego si queremos visualizar o calcular las distancias de Hausdorff. La ***ListBox*** también contendrá una scrollbar en caso de que todas las espinas no quepan en la caja.

La ListBox se ha creado con la función ListBox de la librería tkinter:

espinas=tk.Listbox(width=70,height=15,yscrollcommand=scrollbar.set,selectmode=tk.EXTENDED)

Las variables *width* y *height* determinarán el ancho y largo de la ListBox y la variable *yscrollcommand* añadirá una scrollbar. La scrollbar ha sido creada de la siguiente manera:

scrollbar = ttk.Scrollbar(ventana,orient=tk.VERTICAL)

scrollbar.config(command=espinas.yview)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

El botón show se ha creado utilizando la función Button del módulo ttk:

mostrar=ttk.Button(text="Show",command=lambda:buscar_bd(animalcombo.get(),layercombo.get(),regioncombo.get(),simplecombo.get(),aparatocombo.get(),sinapsiscombo.get()))

Al botón se le asigna el nombre "Show" mediante la variable *text* y se le asigna una función en la variable *command* que debe ejecutar al hacer "click" sobre él. Esta función llamada **buscar_bd** recibe como parámetros los valores elegidos en todos los desplegables, para poder buscar en la base de datos las espinas cuyas características coincidan con las elegidas. El código de la función buscar_bd es el siguiente:

```
def buscar_bd(animal,layer,region,branch_spine,aparato,sinapsis):
    global rutas
    rutas=[]
    espinas.delete(0,tk.END)
    cond_animal="1=1"
```

```
cond_layer="1=1"
cond_branch="1=1"
cond_aparato="1=1"
cond_region="1=1"
cond_sinapsis="1=1"
if layer!= "TODOS":
    cond_layer=f"Layer='{layer}'"
if animal!="TODOS":
    cond_animal=f"Animal_ID={animal}"
if espinas.size()>1:
    espinas.delete(0,tk.END)
if region!="TODOS":
    cond_region=f"Region='{region}'"
if sinapsis=="Yes":
    cond_sinapsis="Multiple_synapses='First of multiple synapses'"
elif sinapsis=="No":
    cond_sinapsis="Multiple_synapses='No'"

if branch_spine=='No':
    cond_branch=f"Branched_spine ='{branch_spine}'"

elif branch_spine!="TODOS":
    cond_branch=f"Branched_spine ='Spine root + branches'"
if aparato!="TODOS":
    cond_aparato=f"Spine_Apparatus='{aparato}'"
```

En primer lugar, creamos una lista que contendrá las rutas de las espinas encontradas. Las variables cond_ contendrán la condición que se añadirá en la consulta a la BD, en caso de que se haya elegido la opción TODOS en el desplegable se añadirá la condición 1=1 por defecto, para que no filtre por esa característica.

Una vez tenemos las condiciones con las que haremos la consulta en la base de datos, nos conectaremos a esta y ejecutaremos la consulta.

```
try:
    db=sqlite3.connect(db_path)
```

```
mycursor=db.cursor()

consulta=f"""SELECT Path  FROM espinasdendriticas WHERE
{cond_animal} and {cond_layer} and {cond_region}

        and  {cond_branch} and  {cond_aparato} and
{cond_sinapsis}"""


mycursor.execute(consulta)
```

La consulta devolverá las rutas de las espinas que han cumplido con las condiciones. De estas rutas obtendremos el nombre de la espina y lo guardaremos en la ListBox:

```
filas=mycursor.fetchall()

    for r in filas:

        rg=recortar_ruta(r[0])

        rutas.append(r[0])

        espinas.insert(tk.END,rg)
```

Guardaremos en la lista “rutas” las rutas completas de las espinas, mientras que en el la ListBox “espinas” guardaremos únicamente el nombre. Este nombre ha sido obtenido como resultado de la llamada a la función **recotar_ruta**, esta función recibe como parámetro la ruta completa y devuelve el nombre de la espina.

```
def recortar_ruta(ruta):
   for i in range(len(ruta)-1, -1, -1):
      if ruta[i]=="\\":
         ruta=ruta[i+1:]
         break
   return ruta
```

recortar_ruta recorre la ruta hacia atrás hasta encontrar la primera \, devolviendo la cadena de texto contenida entre ese \ y el final de la ruta.

Podemos elegir una o varias espinas (Ctrl +”click”) de la **ListBox**, para elegir todas habrá que marcar el **checkbox** llamado “*All”.* Este checkbox ha sido creado con la función Checkbutton:

```
chkValue = tk.BooleanVar()

chkValue.set(False)
```

todos=tk.Checkbutton(text="All",command=select_todos,var=chkValue)

La variable chkValue indicará si el checkbox está marcado o no, al inicio estará desmarcado, por lo que se pone a False.

Al marcar o desmarcar el checkbox se llamará a la función **select_todos,** esta función se encargará de seleccionar o deseleccionar todas las espinas de la ListBox:

```
def select_todos():
    for i in range(espinas.size()):
        if(chkValue.get()):
            espinas.select_set(i)
        else:
            espinas.select_clear(i)
```

Si queremos visualizar las espinas, haremos “click” en el botón “See”, el cual abrirá un archivo vacío en Blender, con el script mencionado en el punto 3.2.3 y con las rutas de las espinas como parámetros. Este botón se crea de la forma ya explicada anteriormente:

visual=ttk.Button(text="See",command=get_rutas)

Ejecutando la función **get_rutas** al ser pulsado:

```
def get_rutas():
    rutas_validas=[]
    index=espinas.curselection()
    for i in index:
        rut=buscar_ruta(espinas.get(i))
        rut[0]='"'+rut[0]+'"'
        rutas_validas.append(rut[0])

    str_rutas=" ".join(rutas_validas)
    command=rf'"blender "{r_vacio}" --python "{r_escalar}" -- {str_rutas}"'
    os.system(command)
```

Esta función buscará las rutas de las espinas seleccionadas en la ListBox mediante la función **buscar_ruta:**

```
def buscar_ruta(nombre):
    return [x for x in rutas if nombre in x]
```

Y guardará sus rutas en la lista “rutas_validas”, cada ruta debe estar entre comillas. Una vez tenemos las rutas de las espinas, la ruta del archivo vacío en Blender y del script de visualización, ejecutamos el comando mediante la función **system** de la librería **os.**

Las variables *r_vacio* y *r_escalar* corresponden con la ruta a un archivo vacío en Blender y el script de visualización respectivamente. Estas dos rutas se han obtenido de la siguiente forma:

fil=os.path.dirname(os.path.abspath(__file__))

Dado que estos dos archivos se encuentran en la misma ruta que la interfaz, guardaremos en la variable *fil* la ruta absoluta de la interfaz, mediante la función **dirname** del módulo **path.** A esta ruta añadiremos el nombre de cada archivo:

r_vacio=os.path.join(fil, "vacio.blend")

r_vacio=r"{}".format(r_vacio)

r_escalar=os.path.join(fil, "scalar.py")

r_escalar=r"{}".format(r_escalar)

Siendo “r_vacio” la ruta al archivo vacio.blend, “r_escalar” la ruta al script explicado en el punto 3.2.3 y “str_rutas” las rutas de las espinas. La función system de la librería **os** ejecutara el comando.

Del mismo modo funcionará el botón “Hausdorff distances”:

distancia=ttk.Button(text="Hausdorff distances",command=calcular_distancias)

La función **calcular_distancias** abrirá un archivo vacío en Blender, con el script mencionado en el punto 3.3.3 con las rutas de las espinas como parámetros.

command=rf'"blender "{r_vacio2}" --python "{r_dis}" -- {str_rutas}"'

Siendo “r_vacio” la ruta al archivo vacio2.blend, “r_dis” la ruta al script explicado en el punto 3.3.3 y “str_rutas” las rutas de las espinas.

3.5 Conclusiones y resultados

Hemos desarrollado una metodología que permite analizar la forma de objetos tridimensionales de manera objetiva. Si comparamos cada una de las espinas dendríticas con todas las demás tendríamos que realizar un número muy alto de medidas. Sin embargo, si comparamos cada espina con tres figuras geométricas de referencia, el número de medidas totales se reduce considerablemente.

La interface gráfica facilita la selección de los grupos de espinas que queremos representar o medir, por lo que las comparaciones de la forma de las espinas pueden realizarse según los criterios que necesite el usuario.

Una vez medidas las espinas que nos interesan, podemos representar la forma de cada espina en un gráfico tridimensional empleando las tres distancias de Hausdorff con respecto a las tres figuras de referencia. En este gráfico, las espinas de forma similar se agruparían a poca distancia, mientras que las que tienen formas muy diferentes estarían muy lejos unas de otras, como en la figura 3.5.1.

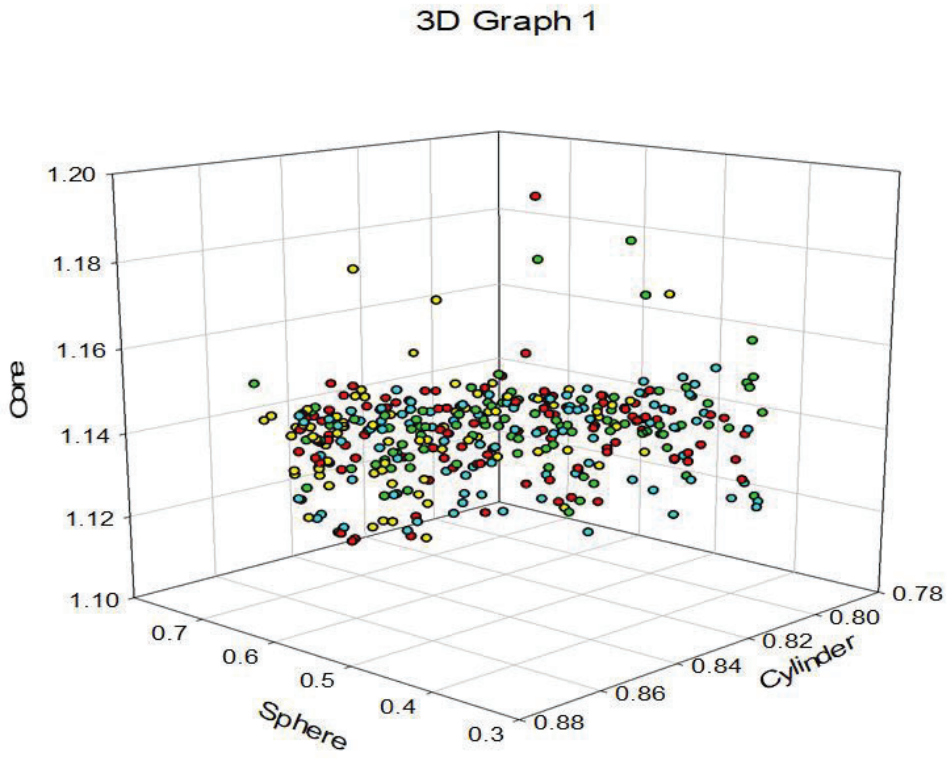
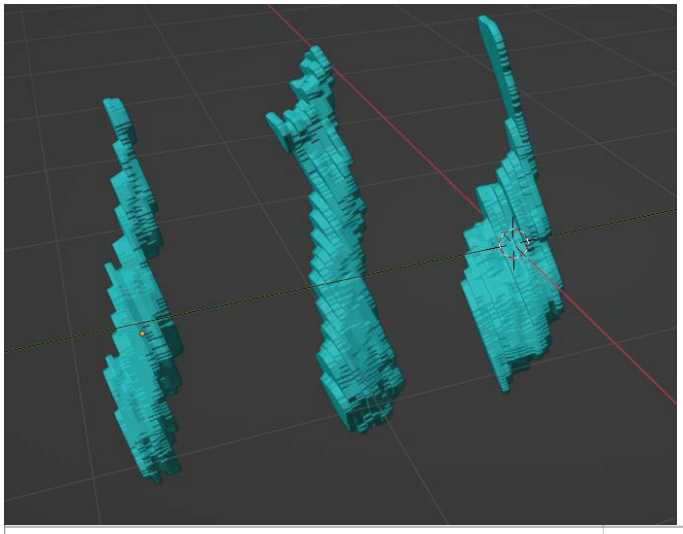


Figura 3.5.1: Gráfico en el que se representan las distancias de Hausdorff de un grupo de espinas dendríticas con respecto a una esfera (Sphere), un cilindro con casquetes semiesféricos (Cylinder) y un cono con casquete semiesférico (Cone). Los colores indican que las espinas dendríticas proceden de cuatro regiones distintas del cerebro.

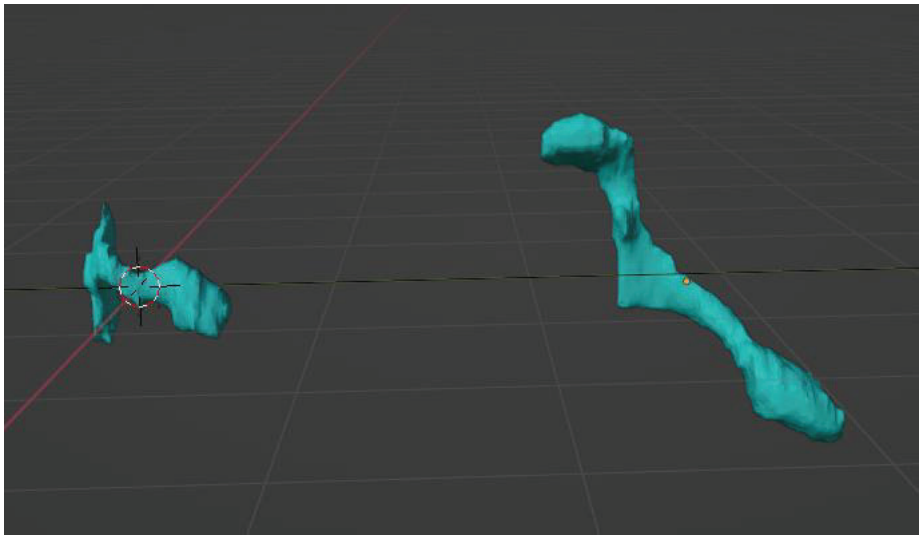
Para poder comprobar que las distancias están calculadas correctamente, hemos elegido 3 espinas parecidas, sus distancias de Hausdorff deben ser muy parecidas:



	Sphere	Cylinder	Cone
HBP29-WT-ID2-HC43-LacMol-3-Espina 1033.wrl	0,543728	0,831475	1,148912
HBP29-WT-ID2-HC43-LacMol-3-Espina 273.wrl	0,559315	0,844089	1,14808
HBP29-WT-ID2-HC43-LacMol-3-Espina 447.wrl	0,570886	0,843997	1,147488

Figura 3.5.2: Espinas parecidas y sus correspondientes medidas

Por lo contrario, entre estas dos espinas con forma distinta las distancias de Hausdorff son más distantes:



	Sphere	Cylinder	Cone
HBP29-WT-ID4-HC43-LacMol 1-Espina 1166.wrl	0,515472	0,807597	1,171416
HBP52-WT-ID28-HC40-Lac-Mol. Espina 874.wrl	0,620174	0,83684	1,149102

Figura 3.5.3: Espinas distintas y sus correspondientes medidas

En definitiva, las conclusiones de nuestro trabajo son las siguientes:

1- Es factible aplicar un método objetivo para medir la forma de objetos biológicos como las espinas dendríticas, que tienen formas muy variables. En nuestro caso hemos implementado las medidas de Hausdorff.

2- Aunque es posible medir las distancias de Hausdorff entre cada espina dendrítica y todas las demás, es mucho más eficiente medir estas distancias entre cada espina y tres figuras geométricas de referencia.

3- Una vez realizadas las medidas, cada espina queda caracterizada por sus tres distancias a las figuras de referencia. De esta manera, un grupo cualquiera de espinas puede representarse en un espacio tridimensional y las posiciones de unas con respecto a otras indicarían las similitudes o diferencias de forma entre ellas (si se encuentran cerca o si están alejadas unas de otras, respectivamente).

4- El procedimiento que hemos desarrollado es completamente objetivo y reproducible, por lo que no se emplean los criterios cualitativos habitualmente utilizados para el análisis de la forma.

4.Análisis de Impacto

Hasta la fecha, cuando pretendemos describir o comparar la forma de cualquier objeto recurrimos a un análisis casi completamente subjetivo y por lo tanto sujeto a sesgos que no podemos cuantificar. La principal aportación de nuestro trabajo es eliminar ese componente subjetivo, analizando la forma mediante un método matemático que da resultados cuantitativos reproducibles.

Además, el método cualitativo tradicional sólo puede realizarse con un número limitado de objetos, ya que los tendríamos que comparar visualmente. Sin embargo, nuestro método puede aplicarse a un gran número de objetos, sin que sea necesario visualizarlos uno por uno. Esto facilita mucho el análisis cuando se dispone de una muestra grande de objetos, como en el caso de las espinas dendríticas.

Aunque en nuestro trabajo hemos usado las espinas dendríticas como modelo, parece claro que el método puede emplearse para medir la forma de cualquier tipo de objeto, siempre que dispongamos de los archivos 3D correspondientes.

Por último, reducir la forma de un objeto tridimensional a tres medidas numéricas tiene la ventaja de que es un método muy eficiente desde el punto de vista computacional, ya que reduce mucho los cálculos necesarios. Por ello es válido como un método que puede aplicarse inicialmente cuando necesitamos estudiar un gran número de estructuras complejas. Sin embargo, también es posible que resulte una simplificación excesiva si necesitamos realizar un estudio en profundidad que detecte diferencias más sutiles. Por todo ello, no damos por completamente terminado el desarrollo de esta metodología, ya que es necesario evaluar sus ventajas e inconvenientes en profundidad, comprobar si puede aplicarse en otros campos de estudio, y realizar las modificaciones necesarias para aumentar su sensibilidad sin perder eficiencia.

A nivel personal la realización de este trabajo ha mostrado las capacidades adquiridas a lo largo de estos cuatro años, como la capacidad de resolver los problemas encontrados de una forma u otra para poder lograr el objetivo y la capacidad de poder aprender tecnologías que desconocía y poder emplearlas de forma correcta.

Bibliografía

[1] I. González-Burgos: Las espinas dendríticas y la memoria: un largo camino por recorrer: https://www.revistaciencia.amc.edu.mx/images/revista/59_4/PDF/05-Espinas_dendriticas.pdf

[2] Wikipedia: Distancia de Hausdorff

https://es.wikipedia.org/wiki/Distancia_de_Hausdorff

[3] docs.python: sqlite3https://docs.python.org/3/library/sqlite3.html

[4] aprendeconalfs.es: Pandas

https://aprendeconalf.es/docencia/python/manual/pandas

[5]Blender API: DecimateModifier

https://docs.blender.org/api/current/bpy.types.DecimateModifier.html

[6] Blender API: LaplacianSmoothModifier

https://docs.blender.org/api/current/bpy.types.LaplacianSmoothModifier.html

[7] Blender API: import_scene

https://docs.blender.org/api/current/bpy.ops.import_scene.html

[8] Blender API: Object

https://docs.blender.org/api/current/bpy.ops.object.html

[9] Scipy API: directed_hausdorff

https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.directed_hausdorff.html

[10] Scipy API: convexHull

https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.ConvexHull.html

[11] Scipy API: distance_martrix

https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance_matrix.html

[12] Numpy API: unravel_index

https://numpy.org/doc/stable/reference/generated/numpy.unravel_index.html

[13] Numpy API: argmax

https://numpy.org/doc/stable/reference/generated/numpy.argmax.html

[14]b3d.interplanety:bounding-box

https://b3d.interplanety.org/en/bounding-box-2/

[15] Openpyxl

https://openpyxl.readthedocs.io/en/stable/

[16] Tkinter

https://docs.python.org/es/3/library/tkinter.html

[17] Tkinter: dialog

https://docs.python.org/3/library/dialog.html

[18]Tkinter: Combobox

https://docs.python.org/3/library/tkinter.ttk.html#tkinter.ttk.Combobox

[19] foro3d.com: crear una semiesfera

https://www.foro3d.com/f24/crear-semiesfera-en-blender-104950.html

Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
Fecha/Hora	Wed Jun 01 21:44:24 CEST 2022
Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
Numero de Serie	561
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)